EN1094 Laboratory Practice

Department of Electronic and Telecommunication Engineering
University of Moratuwa
Sri Lanka

Contents

Part I

Orientation

Workshop 1: Introduction to Laboratory Instruments

Objective: To develop the ability to use the common components and instruments in the laboratory..

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

1. Identify resistors, capacitors and read their values
2. Measure values of resistors using the multi-meter
3. Use the oscilloscope to display and measure the characteristics of electrical signals

Equipment Required:

1. DC power supply
2. Digital multi-meter
3. Analog multi-meter
4. Protoboard
5. Oscilloscope
6. Signal generator
7. Spectrum analyzer

Components Required:

1. Resistors: 330 Ω, 10 kΩ, 100 kΩ, 1 MΩ
2. Capacitors: 4.7 μF, 220 μF, 10 pF, 100 nF
3. Diodes: 1N4001, Zener diode, Red LED
4. Transistor: 2N2222

1.1 Identifying Resistors and Capacitors

Task 1. *Read the values of the provided resistors using the colour code, including tolerances. Record the values in Table 1.1.*

Label	1st band	2nd band	3rd band	4th band	Value	Tolerance
R_1						
R_2						
R_3						
R_4						

Table 1.1: Resistor details

Task 2. *Read the values of the provided capacitors. Identify their types (ceramic or electrolytic) and record in Table 1.2. You may use non-standard abbreviations "Ce" for ceramic and "El" for electrolytic when recording the type.*

Label	Printed value/identifier	Capacitance	Type
C_1			
C_2			
C_3			
C_4			

Table 1.2: Capacitor details

Label	Analog MM reading	Analog MM scale	Digital MM reading	Digital MM scale
R_1				
R_2				
R_3				
R_4				

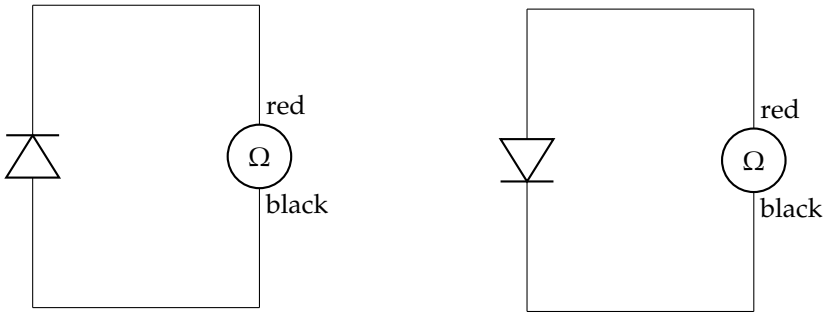
Table 1.3: Resistance values measured using the multi-meter

1.2 Measuring Resistance Using the Multi-meter

Task 3. *Measure the exact resistance values of the provided resistors using the multi-meter ohm scale. Record your observations in Table 1.3.*
***Hint**: Fix the resistors on the protoboard to hold them still when measuring.*

The resistance of a forward biased diode (R_f) is low and the resistance of a reversed biased diode (R_r) is high. By observing R_f and R_r, it is possible to test a diode and determine whether it is working or not.

Task 4. *Identify the diode pins and fix the diode on the protoboard. Use the analog multi-meter as an ohmmeter (refer Figure 1.1), and measure and record R_f and R_r. Is the diode working properly?*



R_f (Forward Resistance) R_r (Reverse Resistance)

Figure 1.1: Measuring forward and reverse resistances of a diode using the analog multi-meter

A digital multi-meter in the *Diode Mode* can be used to check whether a diode is working. In contrast to the resistance, in this mode, the multi-meter will display the bias voltage drop across the diode.

Task 5. *Use the digital multi-meter in the Diode Mode to check the provided diode. Draw the setup under each configuration (forward biased and reverse biased, similar to that of Figure 1.1) and record the displayed voltage drop.*

1.3 Measuring DC Bias Voltages of a BJT

Task 6. *Identify the Emitter (E), Base (B) and Collector (C) pins of the BJT and draw the pin configuration.*

Forward Biased		Reversed Bias	
R_{BE}	R_{BC}	R_{BE}	R_{BC}

Table 1.4: Junction resistances of a BJT

Task 7. *Fix the transistor on the protoboard and measure the resistances of the B-E (R_{BE}) and the B-C (R_{BC}) junctions separately, in forward biased and reversed biased configurations. Record your observations in Table 1.4.*

Task 8. *Construct the circuit given in Figure 1.2 on the protoboard. The transistor is 2N2222. Apply +5 V to V_{cc} while the Emitter terminal is grounded.*

a) Apply 0 V to V_{in} and measure V_{CE} using the multi-meter. Is the LED ON or OFF?

b) Apply +5 V to V_{in} and measure V_{CE} using the multi-meter. Is the LED ON or OFF?

c) Measure and record I_B and I_C using the analog and the digital multi-meter respectively.

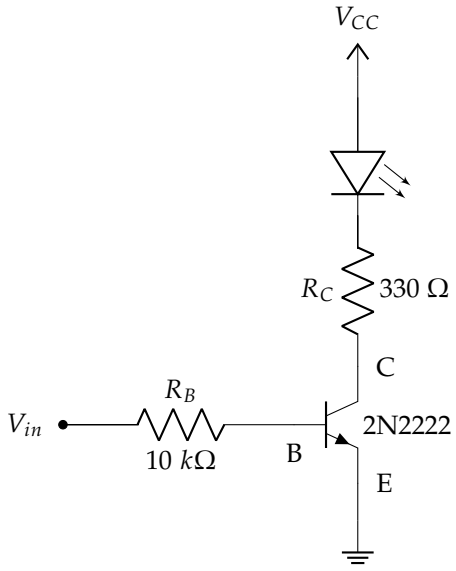


Figure 1.2: Transistor as a switch

1.4 Observing Signals Using the Oscilloscope

Task 9. *Connect the signal generator across a 10 $k\Omega$ resistor as in Figure 1.3. Apply a sinusoidal signal with a peak-to-peak voltage (V_{pp}) of 1 V and frequency 1 kHz. Connect the oscilloscope probes of channel 1 and observe the signal. Plot V_{in} vs t.*

Task 10. *Measure and record the peak-to-peak amplitude of the observed signal using the **Cursors** of the oscilloscope.*

Task 11. *Change the input (V_{in}) to a sinusoidal signal with frequency 150 kHz and plot V_{in} vs t. Measure the period of the signal using the **Cursors**.*

Task 12. *Change the input (V_{in}) to a sinusoidal signal with a 100 mV peak-to-peak amplitude, a frequency of 150 kHz, and a 1 V DC offset. Observe the signal using the oscilloscope with **AC coupling** and **DC coupling**, separately. Plot the two waveforms on the same plot. You may use different amplitude scales to fit the waveform within the observable area.*

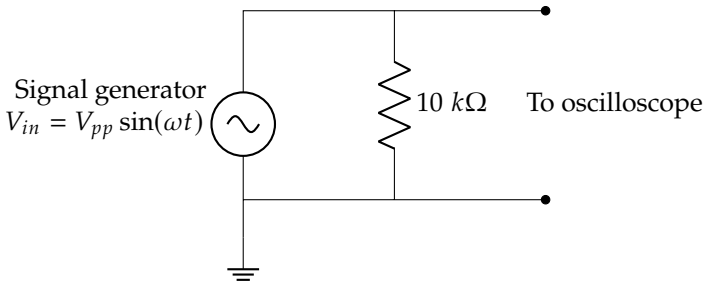


Figure 1.3: Connecting signal generator to a load resistor

Task 13. *Change the input (V_{in}) to a square signal with 100 mV peak-to-peak amplitude, a frequency of 150 kHz, a* 0 *V DC offset and a* 30% *duty cycle. Measure the Root Mean Square voltage of the signal using the* ***Measure*** *function of the oscilloscope.*

Task 14. *RMS voltage of a square waveform can be calculated as*

$$V_{RMS} = V_{pp}\sqrt{D}$$

where V_{pp} is the peak-to-peak amplitude and D is the duty cycle expressed as a fraction. Calculate the RMS voltage for the square wave using the above relationship and comment on the agreement with the direct measurement from the oscilloscope.

♣ The End ♣

Workshop 2: Introduction to MATLAB and Simulink

Objective: To start using MATLAB as a computational and simulation tool for Engineering applications

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

1. Use MATLAB for Matrix operations
2. Manipulate control structures in MATLAB programming efficiently
3. Utilize in-built functions in MATLAB to solve problems
4. Construct user-defined functions appropriately
5. Represent data in a graphical form using MATLAB
6. Simulate simple systems using Simulink
7. Understand the wide variety of applications of MATLAB as a tool

Equipment Required:

1. A Personal Computer
2. MATLAB software or MATLAB online

2.1 Introduction to MATLAB

MATLAB is a tool for numerical computation and visualization. It can be used for tasks such as math computation, algorithm development, modeling, simulation, prototyping, data analysis, exploration, visualization, scientific and engineering graphics, and application development (Graphical User Interface building). In the laboratory practice module you will be using MATLAB mainly for the telecommunication laboratories. This orientation session will guide you through the basics of MATLAB.

2.2 Installing MATLAB

MATLAB is an extremely useful tool when studying telecommunications. However, it should be noted that MATLAB is not a free software. If you have MATLAB already installed in your computer or if you have a MATLAB license you can use the desktop version for practicals. Otherwise, you can use the trial version of the MATLAB online (https://uk.mathworks.com/products/matlab-online.html) version. A Guide for setting up MATLAB online will be provided to you.

2.3 MATLAB Layout

Figure 2.1 shows the layout of MATLAB online version (The layout of MATLAB desktop version is similar). The layout mainly consists of,the following sections.

- Command Window
- MATLAB Editor
- Command History
- Current Folder
- Workspace

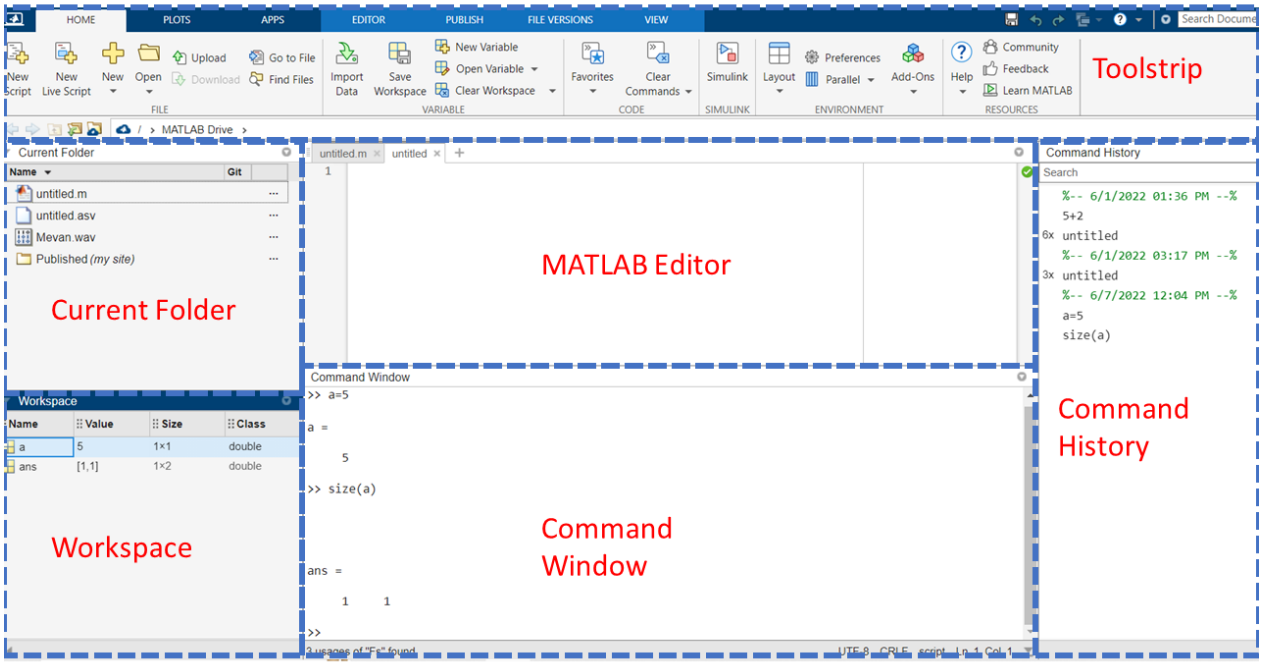


Figure 2.1: The MATLAB Layout

- Toolstrip

Command Window: Area used to execute simple commands or operations directly (Adding two numbers, operations with matrices etc).
MATLAB Editor: Used to write MATLAB scripts or MATLAB functions. Script and functions can contains several operations and commands.
Workspace: Displays the details of all the variables defined in command window and script area sections in the current session.
Command History: Displays all the commands executed in command window and script area sections in the current session.
Current Folder: Shows the current folder. You can use this section to change the current directory.

2.4 Matrices and Operations in MATLAB

All MATLAB variables are multidimensional arrays, irrespective of the type of data. Typically, we will encounter only two-dimensional arrays (also known as matrices) in MATLAB. For example a real number is represented as a 1×1 matrix. The variable names are,

- case sensitive
- can contain up to 31 characters
- must start with a letter.

2.4.1 Scalars, Vectors, and Matrices

All of the above data types are represented as matrices in MATLAB.
Scalar: A scalar is simply a real/complex number. A scalar is represented as a 1×1 matrix. Type in the following command in the command window to define the scalar a and set it's value to 5.

$$>> a = 5 \tag{2.1}$$

Observe the output. You can suppress the output of a command by simply typing ";" after the command (Use $>> a = 5;$). The size(A) function returns the shape of the matrix A. Type in >> size(a) to obtain the shape of the matrix.

Vectors: Vectors can be either row vectors or column vectors. An example column vector c is given below

$$c = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix}. \tag{2.2}$$

An example row vector r is shown below.

$$r = \begin{bmatrix} 1 & 2 & 3 & 4. \end{bmatrix}. \tag{2.3}$$

In MATLAB the above column vector is represented as a 4×1 matrix, whereas the above row vector is represented as a 1×4 matrix. To define the column vector c, input,

$$>> c = [1; 2; 3; 4] \tag{2.4}$$

To define the row vector r, input,

$$>> r = [1, 2, 3, 4] \tag{2.5}$$

or

$$>> r = [1\ 2\ 3\ 4] \tag{2.6}$$

Confirm the shapes of each vector using the size command.

There are different ways of declaring vectors. For example a vector having elements between 0 and 10 with intervals of 0.1 can be defined as below.

$$>> c1 = (0 : 0.1 : 10) \tag{2.7}$$

A vector having 100 equally spaced elements from 0 to 2π can be defined as below.

$$>> c1 = \text{linspace}(0, 2 * \text{pi}, 100) \tag{2.8}$$

Matrices: Let us define the matrix A given by,

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \tag{2.9}$$

in MATLAB. We can use the command,

$$>> A = [1\ 2\ 3; 4\ 5\ 6; 7\ 8\ 9] \tag{2.10}$$

2.4.2 Indexing in Matrices

The matrix indices begin from 1 (note that in many programming languages the indexing begins with 0). In addition indices must be positive integers within the range of the dimensions of the matrix.

Try out the following commands for matrix A, that was declared earlier, and understand the results.

1. $>> A(2, 3)$ - Returns the scalar at location $(2, 3)$ of the matrix as a 1×1 matrix (scalar).
2. $>> A(:, 3)$ - Returns the third column of the matrix as a 3×1 matrix (column vector).

3. >> $A(6)$ - Returns the sixth element of the matrix as a 1×1 matrix (Elements are counted column-wise as 1, 4, 7, 2, 5, 8).

4. >> $A(1,:)$ - Returns the first row of the matrix as a 1×3 matrix (row vector).

5. >> $A(2:3, 1:2)$ - Returns the sub-matrix of matrix A with rows ranging from 2 to 3 and columns ranging from 1 to 2.

Indexing in vectors is similar. You can use commands >> $c(3)$ and >> $r(3)$ to obtain the third elements of the vectors c and r.

2.4.3 Matrix Concatenation

Matrix concatenation arises in various applications. Two or more matrices can be concatenated. For example consider the matrices,

$$X = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}, Y = \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix}, Z = \begin{bmatrix} 9 & 10 & 11 \\ 12 & 13 & 14 \end{bmatrix}. \tag{2.11}$$

These matrices can be concatenated row-wise to form the matrix T where,

$$T = \begin{bmatrix} 1 & 2 & 5 & 6 & 9 & 10 & 11 \\ 3 & 4 & 7 & 8 & 12 & 13 & 14. \end{bmatrix}. \tag{2.12}$$

We can accomplish this in MATLAB using,

$$>> T = [X\ Y\ Z]; \tag{2.13}$$

Also we can obtain the matrix M by concatenating X and Y column-wise as,

$$M = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \\ 7 & 8 \end{bmatrix}. \tag{2.14}$$

This can be accomplished in MATLAB using,

$$>> M = [X; Y]; \tag{2.15}$$

Note that when performing concatenation, special attention must be given to the dimension of the matrices. For instance, >> $M = [X; Y; Z]$ will generate an error since the matrix Z has 3 columns.

2.4.4 Arithmetic Operations

The basic arithmetic operations that can be performed in MATLAB are listed below.

- + Addition
- - Subtraction
- * Multiplication
- $\wedge$ Power
- ' Transpose for a real matrix
- \ left division (>> $x = A\backslash b$ is the solution of $Ax = b$)
- / right division (>> $x = A/b$ is the solution of $xA = b$)

Dimensions must agree when performing matrix addition, subtraction etc. For matrices * performs matrix multiplication (not the element-wise multiplication). Hence, the multiplied matrices should have proper dimensions.

We can also mix scalars and matrices in arithmetic operations. For example $a + b$, where a is a matrix and b is a scalar will add b to each element of a. Similarly, $a * b$ will multiply each element of a by b.

To perform element-wise operations between matrices, we precede the operators by '.'

- $a.*b$ multiplies each element of a by the respective element of b
- $a./b$ divides each element of a by the respective element of b
- $a.\backslash b$ divides each element of b by the respective element of a
- $a.\wedge b$ raises each element of a by the respective element of b.

Task 1. *If* $A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$, *use MATLAB to obtain the following.*

1. $A + 3$
2. $A - 3$
3. $A * 3$
4. $A/3$
5. $A.\wedge 2$

Task 2. *If* $A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$, *and* $B = \begin{bmatrix} 1 & 1 & 1 \\ 2 & 2 & 2 \\ 3 & 3 & 3 \end{bmatrix}$, *use MATLAB to obtain the following.*

1. $A * B$
2. $A.*B$
3. $A./B$
4. $A.\backslash B$
5. $A.\wedge B$

2.4.5 Relational and Logic Operators in MATLAB

The relational operators are used to perform comparisons between MATLAB variables. Following are the major relational operators.

- == Equal
- = Not equal to
- < Strictly smaller
- > Strictly greater
- <= Smaller than or equal to
- >= Greater than or equal to

For two equal dimensional matrices A and B, >> A == B returns a logical matrix C with the same dimensions as A and B. Each entry of this matrix indicates whether the corresponding entries of A and B are equal. Other operators are similar.

The logical operators perform element-wise logical operations. >> $A\&B$ and >> $A|B$ perform element-wise AND and element-wise OR operations between matrices A and B, respectively.

Task 3. *Apply the above operations to the two matrices defined in task 2.*

2.5 Control Structures

So far we defined matrices and performed simple operations using MATLAB. In addition to this knowledge, writing a MATLAB program for an application requires the use of control structures. Under control structures we will examine conditionals (or selection) and loops.

2.5.1 Conditionals

Conditionals are used to execute one or more statements if a particular condition is met. In programming languages conditionals are implemented using "If" statements.

The syntax of an "If" statement is given below.

```
if (condition_1)
   MATLAB commands
elseif (condition_2)
   MATLAB commands
elseif (condition_3)
   MATLAB commands
else
   MATLAB commands
```

Task 4. *Type the following commands in the MATLAB command window and observe the results.*

```
value1 = 5;
value2 = 10;
if value1 > value2
 value3 = 1
elseif value1 == value2
 value3 = 0
else
 value3 = −1
end
```

```
value1 = 5;
value2 = 10;
if value1 > value2
 disp('Greater than')
elseif value1 == value2
 disp('Equal to')
else
 disp ('Less than')
end
```

disp() is a built-in function of MATLAB, that can be used to display texts. There are more advanced built-in functions to do this as well. For the time being, we shall use disp(). You will be introduced to such built in functions later in this workshop.

2.5.2 Loops

Loops are used to repeatedly execute the same set of statements on all the objects within a sequence (for example executing the same set of statements on all the objects of a vector). We will study the "For" and the "While" loops.

Given below is the syntax of a "For" loop.

```
for controlVariable = start : step : stop
   MATLAB commands
end
```

Task 5. *In order to understand the "For" loop, execute the following MATLAB commands.*

```
for i = 1:2:10
 i
end
```

```
total = 0;
for i = 1:2:10
 total = total + i;
end
total
```

Task 6. *We can implement the above code using a while loop as follows. Execute the following MATLAB code.*

```
total = 0;
i = 1
while i<10
 total = total + i;
 i = i + 2;
end
total
```

A *break* statement can be used within a loop to stop the loop's execution after a particular condition is met.

```
total = 0;
for i = 1:2:10
 total = total + i;
 i
 if(total>20)
    break;
 end
end
total
```

Similarly, a *continue* statement implemented within a loop will check for a particular condition. But instead of completely stopping the execution of the loop, it will stop the execution of the commands after the continue statement of the iterations in which the condition is met. To observe the difference between "break" and "continue", run the following commands.

```
total = 0;
for i = 1:2:10
 if(i == 7)
    continue;
 end
```

```
 total = total + i;
end
total
```

```
total = 0;
for i = 1:2:10
 if(i == 7)
    break;
 end
 total = total + i;
end
total
```

2.6 MATLAB Scripts and Functions

When problems become complicated and require re–evaluation, entering commands on the MATLAB command window is not practical. The solution is to write functions/scripts as m-files in MATLAB, and reuse them wherever necessary.

2.6.1 MATLAB Scripts

Scripts are collections of commands executed in sequence, when called. They are saved with an extension ".m".

On the home tab of the MATLAB main window, there is an icon named "New Script". Click on it and open the MATLAB editor. Type the following commands on the editor window.

```
% This is a comment in MATLAB
array = [1 2 3 4 5 6 7 6 5 4 3 2 1];
numOfElements = length(array);
total = 0;
for i = 1:numOfElements
 total = total + array(1,i);
end
total
```

Save the script as myFirstScript.m (You can give any name. Only the extension should be .m). Type "myFirstScript" in command window and enter. You will observe the answer for total. Every time you want to execute the same set of commands, you can save the common set of commands as a MATLAB script by a certain name, and just call out that name on the command window.

2.6.2 MATLAB Functions

A Function is a 'black box' that communicates with workspace through input(s) and output(s). Figure 2.2 illustrates the basic components of a MATLAB functions.

There are two types of MATLAB functions. There are built in MATLAB functions and the user defined functions.

Built-in Functions

Built-in functions are the functions that are defined in MATLAB and can be readily used without implementation. There are thousands of Built-in functions in MATLAB. They can be used as tools to write MATLAB programs very conveniently and efficiently. Examples for built-in functions are min, max, mean, size, and inv functions. For example min function can be used to find the minimum element of a vector.

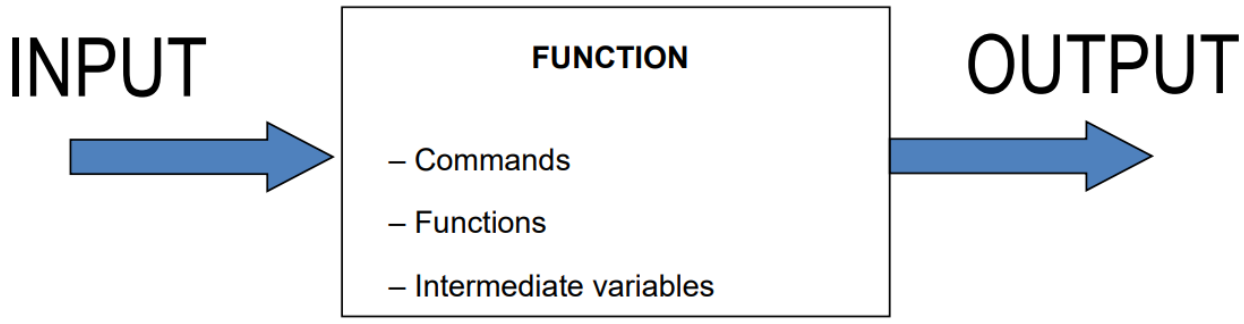


Figure 2.2: The Components of a MATLAB Function

Task 7. *Find the operations performed by the other functions mentioned. Explore different types of MATLAB functions and their functionalities.*

Using the MATLAB **help** feature is one of the most effective ways of learning MATLAB. On the *home* window of MATLAB, there is an icon named *help*. Click on it and get the Product help window. You can search for any Built-in function in this help window.

You can also type "»help *functionName*" on the MATLAB command window to get the details of known built-in functions. For example you can type "»help sin" to get the details about sin function.

User defined functions

User defined functions are the custom functions defined by the users for their own applications. Similar to MATLAB scripts, such functions are written in the MATLAB editor. General syntax of a user defined function is given below.

```
function output(s) = function_name(inputs)
   MATLAB commands
```

The user defined functions are written in separate ".m" files. The “function_name” must match the file name. In other words, the ".m" file containing the function should be saved by the name function_name.m.

Implement the following function in a new ".m" file.

```
function x = impedance (r,c,l,w)
%Impedance function calculates Xc, Xl, Z(magnitude) and Z(angle) of the RLC
%connected in series.
if nargin ~= 4
 error('Please input 4 arguments')
end
x(1) = 1/(w*c);
x(2) = w*l;
Zt = r + (x(2) – x(1))*i;
x(3) = abs(Zt);
x(4) = angle(Zt);
```

Task 8. *Now that you know how to write MATLAB scripts as well, in order to call the function you have just defined, write the following script in a separate ".m" file and run it on the MATLAB command window. You can use any name for your script.*

```
R = input("Enter R: ");
C = input("Enter C: ");
L = input("Enter L: ");
W = input("Enter w: ");
```

```
Y = impedance(R, C, L, W)
```

You can call user defined functions within a script or within another user defined function. Directly calling them from the MATLAB command window is also possible.
Note: Make sure that all your scripts and functions are saved in the same folder, before you execute them. You should set that folder as your current work folder as well. Instructors will help you in this regard.

Task 9. *Complete the following recursive function to generate the n-th term of the Fibonacci sequence. In the Fibonacci sequence, the zeroth term is zero and the first term is 1. Every term which follows is equal to the sum of the previous two terms.*

```
function fn = fibonacci (n)
%Generates the nth Fibonacci number
%Input = n
%Output = F(n)
if n < 0
 error('n less than zero')
elseif n == 0
 fn = 0;
elseif n == 1
 fn = 1;
else
 %Complete the function by entering the %relevant MATLAB commands here
end
```

2.7 Plotting

Visualizing data is an important aspect in Engineering. Graphs are used to get a vivid idea about the data and information that you can obtain using certain systems and applications. MATLAB supports plotting of graphs in 2D as well as in 3D. We shall first learn on how to plot in 2D. The following example will give you a good understanding about plotting in 2D.

Imagine we need to plot the graphs of sin(x) and cos(x).

```
close all;
clear all;
clc;
x = 0:0.01:2*pi;
y1 = sin(x);
y2 = cos(x);
plot (x,y1,'r-');
hold on;
plot (x,y2,'g--');
xlabel('Angle in radians');
ylabel('y1 and y2');
title('My first graph in Matlab');
legend('sin(x)','cos(x)');
```

Functionality of each of the built-in functions should be easy to understand. The instructors in the labs will help you if you have any issues. The main command to know is plot(). plot(x,y) will simply plot y against x. plot(x,y,LineSpec) can be used to specify the style of the curve (color, dotted/ dashed etc.). Visit the plot function's help page and find more about it. To learn about 3D plotting in MATLAB, use the help window as well as online materials (https://uk.mathworks.com/help/matlab/visualize/creating-3-d-plots.html). The main functions that you should be familiar with are meshgrid, mesh and surf.

2.8 Self-Learning MATLAB Exercises

Instructors/Lecturers will help you to go through these tasks. You are expected to finish these at home. The idea of these tasks is to enhance your knowledge on MATLAB as well as to familiarize you with different applications of MATLAB.

2.8.1 Use of in-built functions and vectorization

MATLAB is optimized for operations involving matrices and vectors. Consider a MATLAB inbuilt-function f which has a scalar input and a scalar output. If we need to calculate f for each entry of a matrix X we can use $f(X)$. This is known as vectorization. For example if need to calculate the sine value of each entry of the matrix

$$X = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}, \tag{2.16}$$

we can use $\sin(X)$.

Task 10. *Generate a 2D array (with the name randomArray and dimensions (5,10)) comprising of random integers which are uniformly distributed in the region 50 to 100 using the built in functions in MATLAB (You can use rand function).*

1. *Calculate the sine value of each of the elements in "randomArray" by*
   (a) *vectorization.*
   (b) *using a for loop.*
2. *Use the tic-toc timer in MATLAB to measure the time elapsed in each case.*
3. *Rename the script file as "task10_<index>.m" and submit to the link on Moodle.*

2.8.2 Sound Processing in MATLAB

Save the sound file anthem.wav in your current work folder. To read the sound file into MATLAB as a matrix, use the following command.

```
[anthem,fs]=audioread('anthem.wav');
```

To play it back,

```
sound(anthem,fs)
```

The array named anthem comprises the numerical values which represent the amplitudes of the sound file. We can play around those numbers and do lots of processing, just by doing various matrix operations and arithmetic operations. If you are interested, you can experiment and learn.

2.8.3 User defined functions

Task 11. *Write a simple MATLAB script (in an m-file) to perform the following task. Input 2 arrays (2D) with the same dimensions and compare the corresponding elements of those arrays. If equal, output 1, and if not, output 0. The output array should be a binary array with the same dimensions as that of the input arrays. If the input arrays do not have the same dimensions, output an error message.*
Rename the script file as "task11_<index>.m" and submit to the link on Moodle.

2.8.4 Image Processing

MATLAB can be used for image processing and Machine vision applications. To get an idea about such applications, execute the following commands and observe the results. Those who are interested can learn more by experimenting and self-study.

```
image = imread("someImage.jpg"); %someImage.jpg should be saved in your work folder.
imshow(image)
grayImage = rgb2gray(image);
imshow(grayImage)
imTemp = image;
imTemp(:,:,2) = 0;
imTemp(:,:,3) = 0;
imshow(imTemp)
```

2.9 Introduction to Simulink

Simulink is a program for simulating signals and dynamic systems. As an extension of MATLAB, Simulink adds many features specific to the simulation of dynamic systems while retaining all of MATLAB's general purpose functionality.

You can access Simulink by clicking on the Simulink icon on the home tab of the MATLAB main window.

2.9.1 Simulink Layout

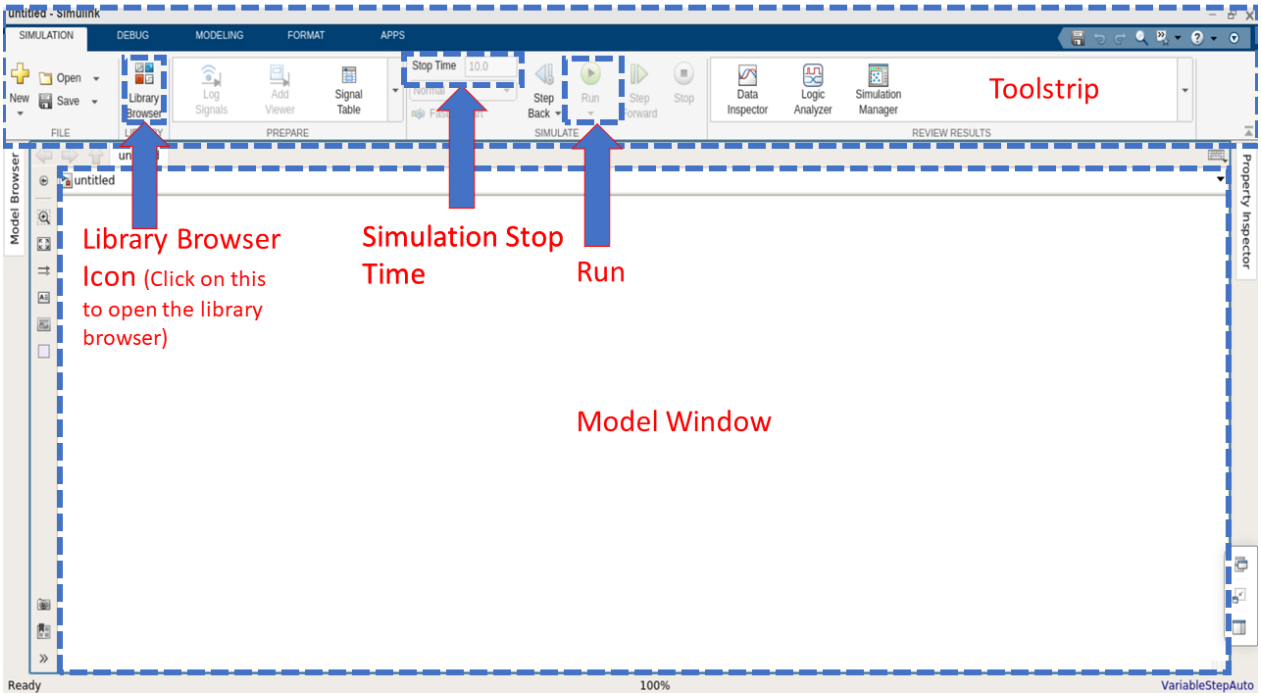


Figure 2.3: The Simulink Layout

Figure 2.3 shows the Simulink Layout. To construct a Simulink model, you first copy blocks from the Simulink Library Browser to the Model window. As shown in Figure 2.3 Simulink Library Browser can be opened by the Simulink Library Icon located in the toolstrip.

2.9.2 Constructing and simulating a simple model in Simulink

In this section you will develop a simple Simulink model to integrate a sine wave. Both the original wave and the integrated wave will be displayed for comparison purposes.

Figure 2.4 shows the block diagram of the completed model. We will analye the step by step process of buiding the above model.

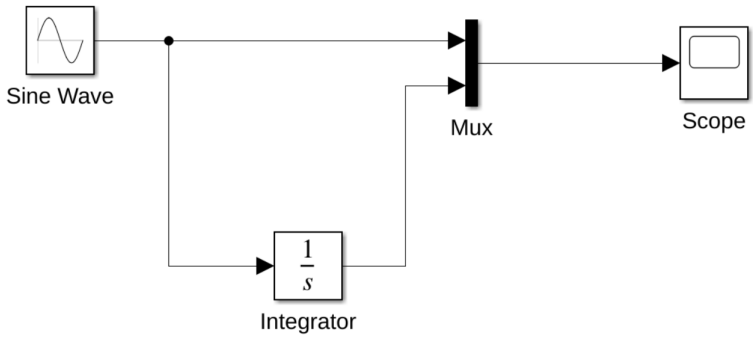


Figure 2.4: The Simulink Model

1. Select *File* > *New* > *Model* to construct a new model.
2. To create the simple model shown in Figure 2.4, you need four blocks:
   - Sine Wave — To generate an input signal for the model
   - Integrator — To process the input signal
   - Scope — To visualize the signals in the model
   - Mux — To multiplex the input signal and processed signal into a single scope

   You can add each block using the following steps.
   - Select the **Simulink/Sources** library in the Simulink Library Browser. The Simulink Library Browser will then display the Sources library. Select the **Sine Wave** block in the Simulink Library Browser, then drag it to the model window. A copy of the Sine Wave block appears in the model window. Click on the Sine Wave block to change it's properties.Change the Frequency to 3 rad/s and the Sample time to 0.001.
   - Select the **Simulink/Sinks** library in the Simulink Library Browser. Select the **Scope** block from the Sinks library, then drag it to the model window. A Scope block appears in the model window.
   - Select the **Simulink/Continuous** library in the Simulink Library Browser. Select the **Integrator** block from the Continuous library, then drag it to the model window. An Integrator block appears in the model window.
   - Select the **Simulink/Signal Routing** library in the Simulink Library Browser. Select the **Mux** block from the Sinks library, then drag it to the model window. A Mux block appears in the model window.
3. Make the connections as Figure 2.4. The connection between Sine Wave block and Mux block is somewhat different from the other three. Because the output port of the Sine Wave block already has a connection, you must connect this existing line to the input port of the Integrator block. The new line, called a branch line, carries the same signal that passes from the Sine Wave block to the Mux block. To weld a connection to an existing line:
   - Position the mouse pointer on the line between the Sine Wave and the Mux block.
   - Press and hold the Ctrl key, then drag a line to the Integrator block's input port.

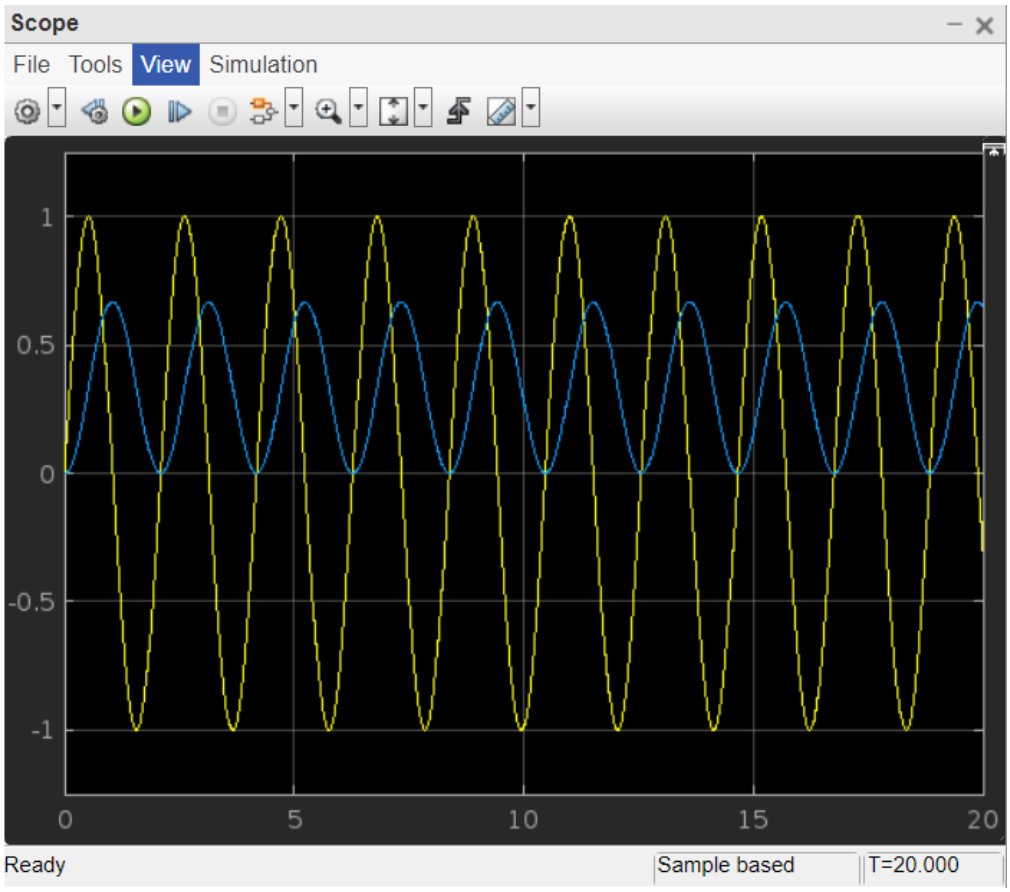


Figure 2.5: The Sine Wave and the Integration of the Sine Wave

4. Set the simulation stop time to 20s (shown in Figure 2.3).

5. Now simulate your sample model and observe the results. To run the simulation click *Run* button shown in Figure 2.3.

6. Double-click the **Scope** block in the model window. The Scope window should display the simulation results as shown in Figure 2.5.

2.10 Self-Learning Simulink Exercises

Task 12. *Study following Simulink blocks using Scope:*

- *Signal generator*
- *Digital clock*
- *Step*
- *Uniform random number*
- *Pulse generator*
- *Sine wave function*

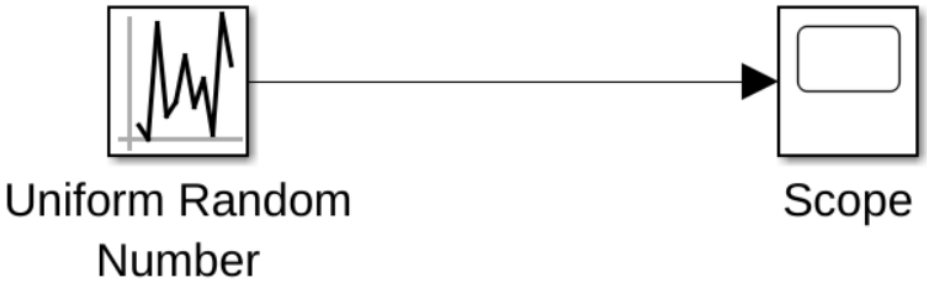


Figure 2.6: The Uniform Random Number Block

For example the use of uniform random number is shown in Figure 2.6.

Select the **Simulink/Sources** library in the Simulink Library Browser. Select the **Uniform Random Number** block in the Simulink Library Browser, then drag it to the model window. Click on the Uniform Random Number block and change the Minimum to -2 and the Sample time to 0.3.

Set the simulation time to 20s and Run the simulation. You will observe a graph similar to Figure 2.7.

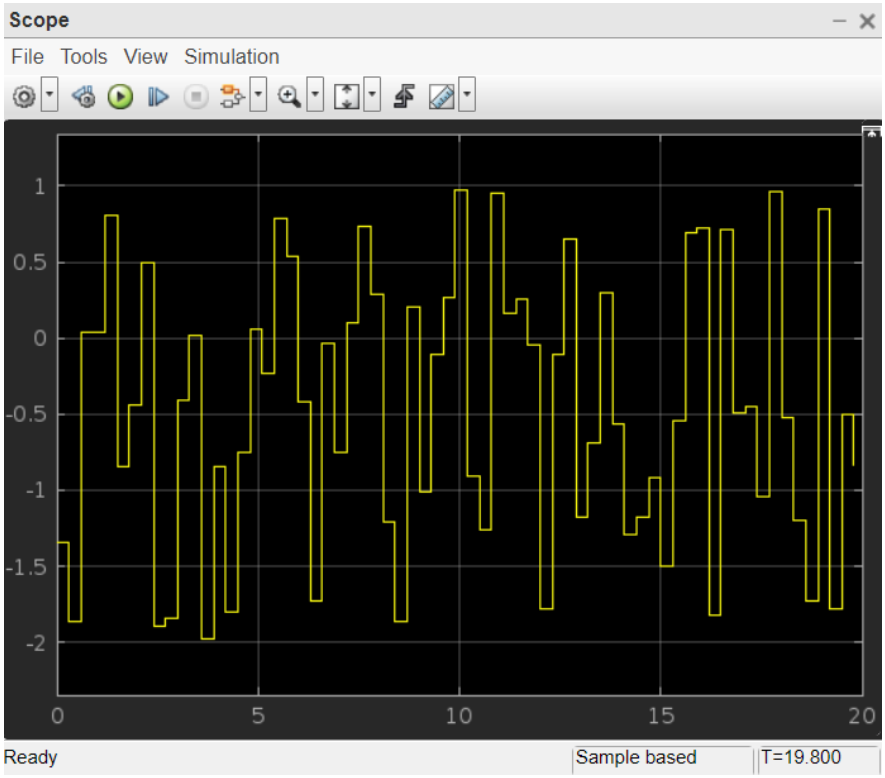


Figure 2.7: The Plot of the Uniform Random Numbers

Task 13. *Follow the steps below to design a multiplier with a square-wave input.*

1. *Start a Simulink session.*
2. *Copy the pulse generator block by dragging it to your model (The block can be found in Simulink/Sources). Specify the parameters as follows.*
   - *Period: 1*

- *Pulse width: 50%*
- *Amplitude: 0.7*

3. *Next, multiply the output of the signal generator by a gain. Gain block can be found in the Simulink/Math Operations library. Set the gain to 0.9 from the properties of the block.*
4. *Use the Scope block to analyze the output.*
5. *Connect the blocks.*
6. *Set the simulation stop time to 10s and run the simulation.*
7. *Save the model as "task13_<index>.slx" and submit to the link on Moodle.*

♣ The End ♣

Workshop 3: OMNET++ Simulation Software

Objective: To become familiar with the OMNET++ environment for the simulation of communication networks.

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

1. Get a basic understanding of the OMNET++ simulation platform.
2. Simulate a simple communication network using OMNET++

Equipment Required:

1. A Personal Computer
2. OMNET++ (most recent version)

3.1 Introduction

Communication networks allow many users to communicate with each other irrespective of geographical location and run many other applications and obtain services. The resources of the communication network are shared by all its users. Protocols allow users in diverse geographical locations, using different types of hardware and software to communicate with each other over a network using a standard set of rules.

Simulation tools are commonly used to study the performance of communication networks and protocols in different situations as well as to design new protocols. OMNET++ (**O**bjective **M**odular **Net**work **T**estbed in C++), which we will use in this lab, is a well-known open-source simulation tool for communications networks used extensively by the academic community. In this session, you will first get familiar with the OMNET++ simulation environment, and then use it to implement a very simple communication network.

Use the learning resources at https://docs.omnetpp.org to learn more about OMNET++. In particular, to learn OMNET++ in detail, see https://doc.omnetpp.org/omnetpp/manual/. This session is an adaptation of the initial parts of the *Tictoc* tutorial. After this introductory session, please complete the *Tictoc* tutorial available at https://docs.omnetpp.org/tutorials/tictoc/.

3.2 Setting up the OMNET++ simulation environment

Step A: Follow following video (upto 6:55) to download and install OMNET++ on your computer.
https://www.youtube.com/watch?v=PfAWhrmoYgM&list=PLaBPUIXZ8s4AwAk5EelikvvyG4EzX2hpx&index=2

The process of getting OMNET++ ready to run on your computer involves downloading and installing the software, configuring and building it, and then verifying your installation.

Once you have completed the installation step, when you look into the new omnetpp-6.0 directory, you should see directories named `doc, images, include, tools`, etc., and files named `mingwenv.cmd, configure, Makefile`, and others. In the doc directory, you will find the document `InstallGuide` which will help you further on the installation process.

At the end of this step, you will have launched OMNET++ and will see the Welcome Screen. Close this screen for now.

Step B: Next, follow the instructions at https://inet.omnetpp.org/Installation.html to install the INET framework.

The INET Framework is the standard protocol model library of OMNeT++. INET contains models for the Internet protocol stack and many other protocols and components. Several other simulation frameworks take INET as a base, and extend it into specific directions, such as vehicular networks (Veins, CoRE), overlay/peer-to-peer networks (OverSim), or LTE (SimuLTE).

3.3 Creating and running your first OMNET++ project

Let us begin with a "network" that consists of two nodes. The nodes will do something simple: one of the nodes will create a packet, and the two nodes will keep passing the same packet back and forth. We'll call the nodes `tic` and `toc`. There is a 100ms propagation delay between `tic` and `toc`.
Step A: Create the project Tictocfirst using the guidelines provided below.

- Start the OMNeT++ IDE (Integrated Development Environment) by typing `omnetpp` in your terminal (Command window).
- Once in the IDE, choose *New -> OMNeT++* Project from the menu.
- A wizard dialog will appear. Enter `tictocfirst` as project name, choose Empty project when asked about the initial content of the project, then click *Finish*. An empty project will be created, as you can see in the *Project Explorer*.
- A wizard dialog will appear. Enter tictocfirst as project name, choose Empty project when asked about the initial content of the project, then click Finish. An empty project will be created, as you can see in the Project Explorer.

This project will hold all files that belong to your simulation. You need to create three files the `NED` file, the `.cc` file and the `.ini` file as described below:

- **The NED file:** OMNeT++ uses the NED (NEtwork Description) language to define components and to assemble them into larger units like networks. NED lets the user declare simple modules, and connect and assemble them into compound modules. The user can label some compound modules as *networks*; that is, self-contained simulation models. Channels are another component type, whose instances can also be used in compound modules. We start implementing our model by adding a NED file. To add the file to the project, right-click the project directory in the *Project Explorer* panel on the left, and choose *New -> Network Description File* (NED) from the menu. Enter `tictocfirst.ned` when prompted for the file name.

  Switch into *Source* mode to enter text into your NED file. Your NED file `tictocfirst.ned` contents should be as below.

```
simple Txc1first
{
gates:
input in;
output out;
}
network Tictocfirst
{
@display("bgb=390,315");
submodules:
tic: Txc1first {
@display("p=59,221");
}
toc: Txc1first {
@display("p=310,61");
}
connections:
tic.out --> {  delay = 100ms; } --> toc.in;
tic.in <-- {  delay = 100ms; } <-- toc.out;
}
```

The first block in the file declares `Txc1first` as a simple module type. Simple modules are atomic on NED level. They are also active components, and their behavior is implemented in C++. The `Txc1first` simple module type is represented by the C++ class `Txc1first`. Two instances of `Txc1first`, named `tic` and `toc` will pass the same messages between each other with a delay of 100ms. The name of this small network is `Tictocfirst`.

After you have created your NED file, switch back to *Design* mode to see the network you have created. The view of the two modes is shown in Figure 3.1.

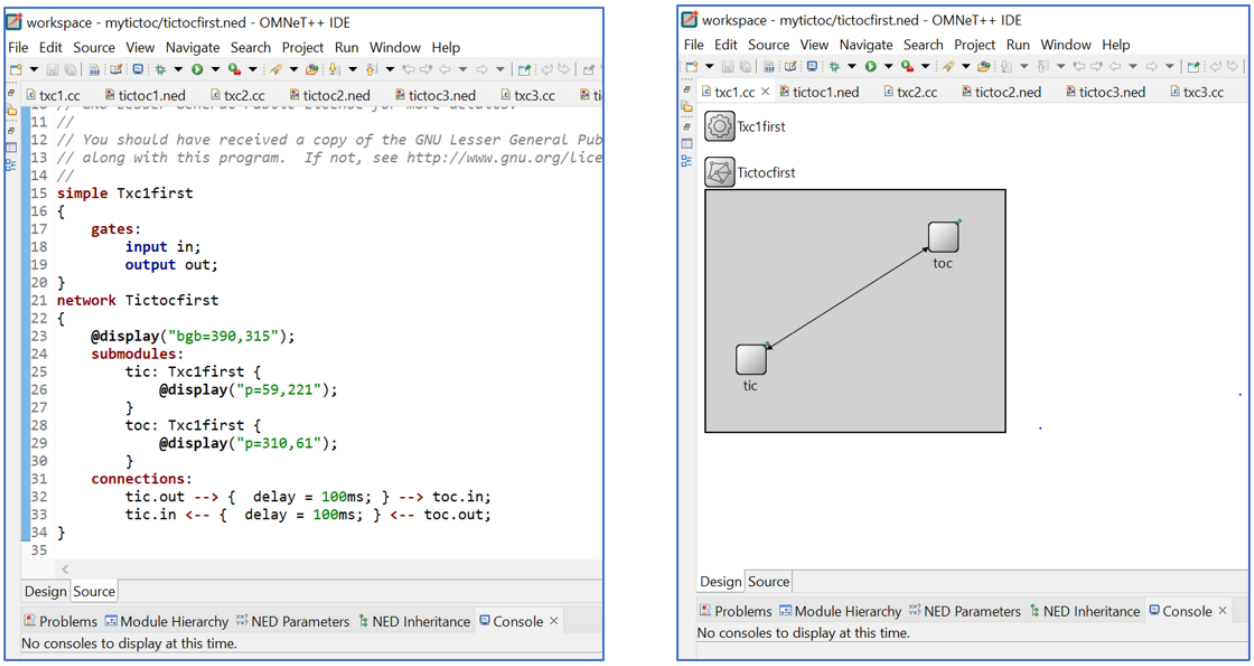


Figure 3.1: Network design with the NED file

- **The C++ file:** `Txc1first` class needs to subclass from OMNeT++'s `cSimpleModule` class, and needs to be registered in OMNeT++. This is done in the `txc1first.cc` file. We now need to implement the functionality of the `Txc1first` simple module in C++. Create a file named `txc1.cc` by choosing *New -> Source File* from the project's context menu (or *File -> New -> File* from the IDE's main menu), and enter the following content:Your `txc1first.cc` file contents should be as below. Note the comments in the code for further clarity.

```
#include <string.h>
#include <omnetpp.h>

using namespace omnetpp;

/**
 * Derive the Txc1first class from cSimpleModule. In the Tictocfirst network,
 * both the 'tic' and 'toc' modules are Txc1first objects, created by OMNeT++
 * at the beginning of the simulation.
 */
class Txc1first : public cSimpleModule
{
  protected:
    // The following redefined virtual function holds the algorithm.
    virtual void initialize() override;
    virtual void handleMessage(cMessage *msg) override;
};
```

```
// The module class needs to be registered with OMNeT++
Define_Module(Txc1first);

void Txc1first::initialize()
{
  if (strcmp("tic", getName()) == 0) {
//The 'ev' object works like 'cout' in C++. We add log statements to Txc1 so that it //prints what it is doing.
    EV << "Sending initial message\n";
    cMessage *msg = new cMessage("tictocMsg");
    send(msg, "out");
  }
}

void Txc1first::handleMessage(cMessage *msg)
{
  // msg->getName() is name of the msg object, here it will be "tictocMsg".
  EV << "Received message '" << msg->getName() << "', sending it out again\n";
  send(msg, "out");
}
```

We redefine two methods from `cSimpleModule`: `initialize()` and `handleMessage()`. They are invoked from the simulation kernel: the first one only once, and the second one whenever a message arrives at the module.

In `initialize()` we create a message object (`cMessage`), and send it out on gate out. Since this gate is connected to the other module's input gate, the simulation kernel will deliver this message to the other module in the argument to `handleMessage()` – after a 100ms propagation delay assigned to the link in the NED file. The other module just sends it back (another 100ms delay), so it will result in a continuous ping-pong.

Messages (packets, frames, jobs, etc) and events (timers, timeouts) are all represented by `cMessage` objects (or its subclasses) in OMNeT++. After you send or schedule them, they will be held by the simulation kernel in the "scheduled events" or "future events" list until their time comes and they are delivered to the modules via `handleMessage()`.

We add log statements to `Txc1first` so that it prints what it is doing. OMNeT++ provides a sophisticated logging facility with log levels, log channels, filtering, etc. that are useful for large and complex models, but in this model we'll use its simplest form `EV`, which is an object that works like `cout` in C++.

- **The .ini file:** To be able to run the simulation, we need to create an `omnetpp.ini` file. This file tells the simulation program which network you want to simulate (as NED files may contain several networks). Your `omnetpp.ini` file contents should be as below.

```
[General]
network = Tictocfirst
```

The name you have given for the network in your NED file is `Tictocfirst`, which appears here.

Step B: After you have created the three files, follow the steps below to compile and run your first project.

- Launch the simulation by selecting `tictocfirst.ini` (in either the editor area or the *Project Explorer*) as shown in Figure 3.2, and pressing the *Run* button.

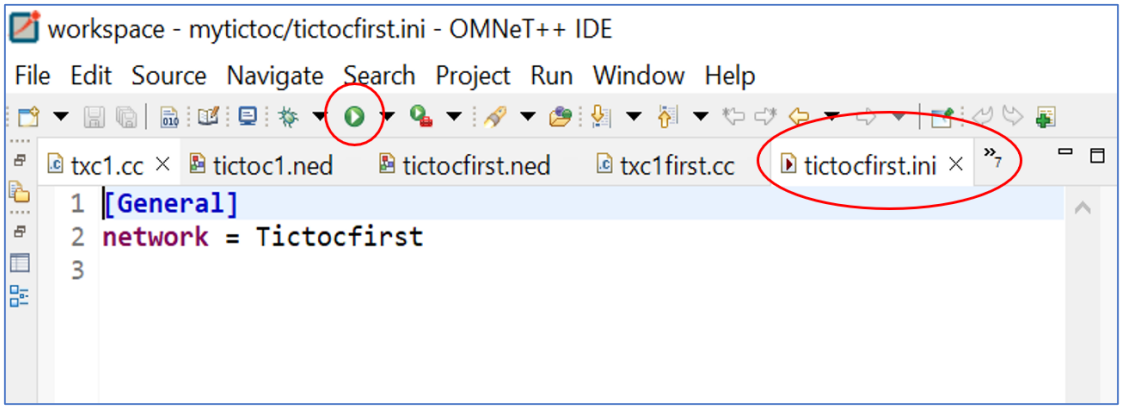


Figure 3.2: The OMNET++ IDE

- After successfully launching your simulation, you should see a new GUI window appear, similar to the one in Figure 3.3. The window belongs to `Qtenv`, the main OMNeT++ simulation runtime GUI. You should also see the network containing *tic* and *toc* displayed graphically in the main area.

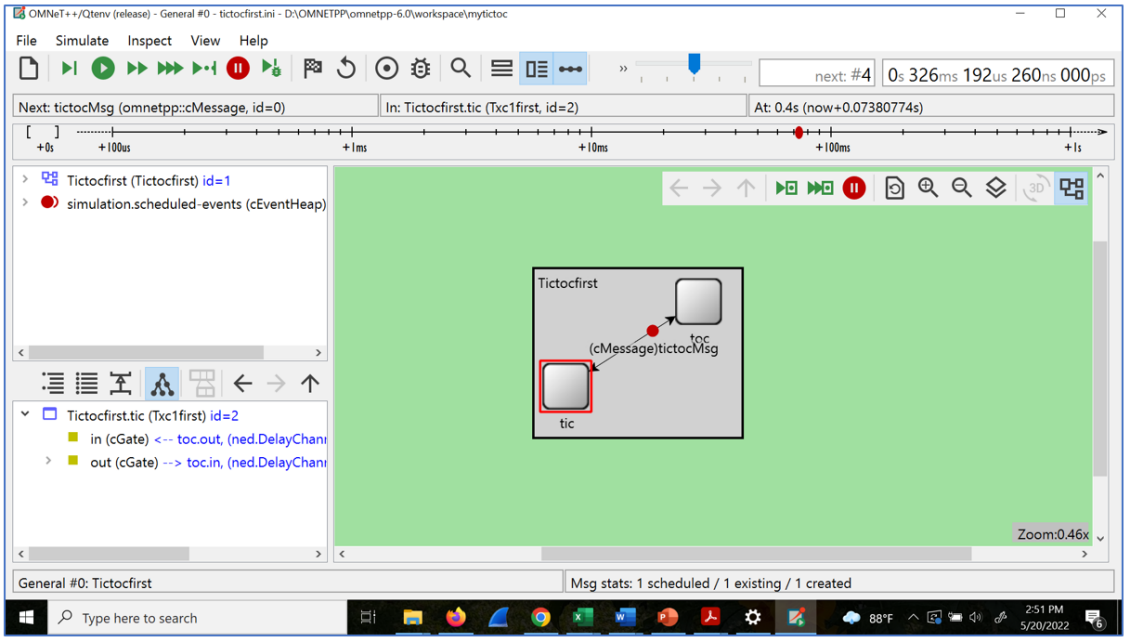


Figure 3.3: The Qtenv runtime GUI

- Press the *Run* button on the toolbar to start the simulation. What you should see is that `tic` and `toc` are exchanging messages with each other.

The main window toolbar displays the current simulation time. This is virtual time, it has nothing to do with the actual (or wall-clock) time that the program takes to execute. Actually, how many seconds you can simulate in one real-world second depends highly on the speed of your hardware and even more on the nature and complexity of the simulation model itself.

You can play with slowing down the animation or making it faster with the slider at the top of the graphics window, single-step through it or run in express mode using with the tools on the top tool bar. You can stop the simulation by hitting the STOP button on the toolbar. You can exit the simulation program by clicking its *Close* icon or choosing *File -> Exit*.

3.4 Observing simulation results

In this section we will learn two simple methods of observing simulation results.

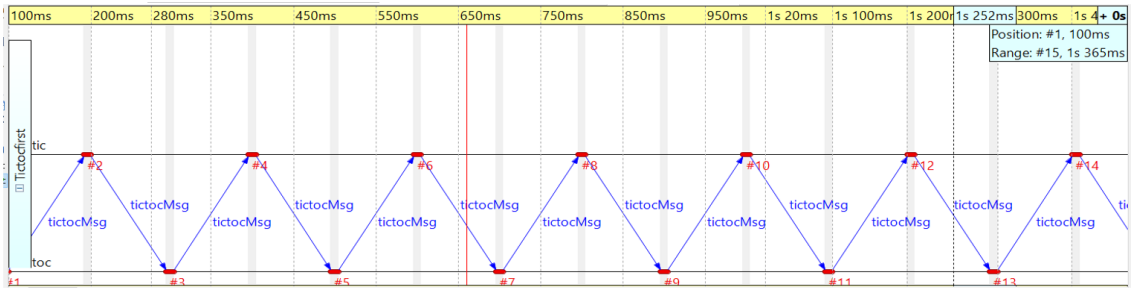


Figure 3.4: Message exchanges

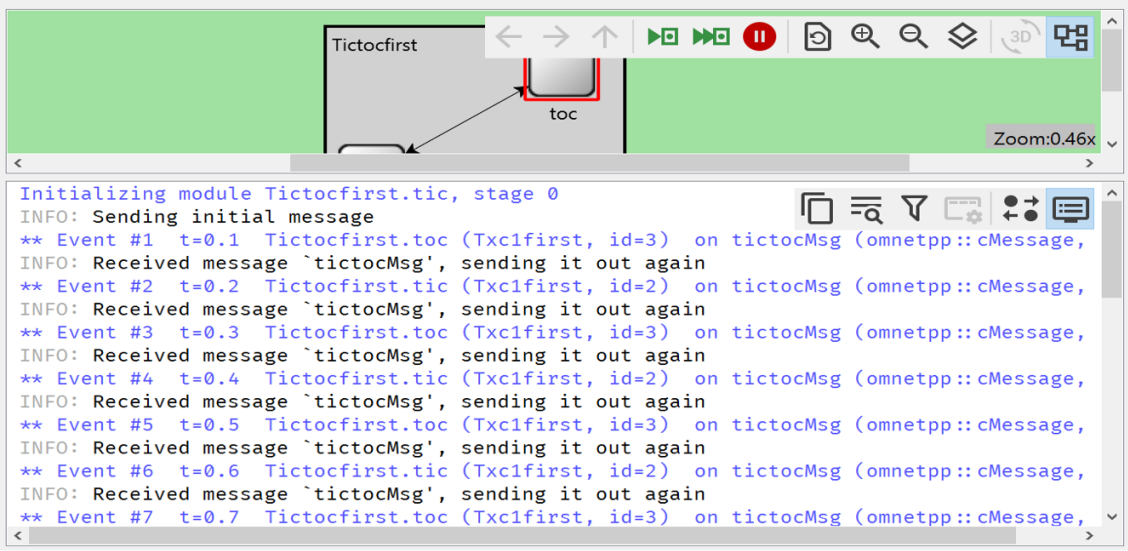


Figure 3.5: Simulation logs

- Observing message exchanges: Run the simulation again. Use the *Record* button in the `Qtenv` graphical runtime environment after launching, to record the message exchanges during the simulation into an *event log file*. After you exit the simulation, the log file can be analyzed later with the *Sequence Chart* tool in the IDE. The `results` directory in the project folder contains the `.elog` file. Double-clicking on it in the OMNeT++ IDE opens the Sequence Chart tool, and the event log tab at the bottom of the window. You will see a chart similar to Figure 3.4. You can also open separate output windows for `tic` and `toc` by right-clicking on their icons and choosing *Component log* from the menu.
- Observing simulation logs: When you run the simulation in the OMNeT++ runtime environment, the output similar to Figure 3.5 will appear in the log window. This is the simplest form of logging available in OMNET++.

3.5 Warm-up exercises with OMNET++

The first OMNET++ simulation that you just completed is adapted from the *Tictoc tutorial* available at https://docs.omnetpp.org/tutorials/tictoc/

Follow the complete *Tictoc tutorial*, which contains a series of simulation models numbered 1 through 16. The models are of increasing complexity. They start from the basics and introduce new OMNeT++ features or simulation techniques in each step. The additions are well commented, so the model sources can help you build up a good working knowledge of OMNeT++ in a short time.

♣ The End ♣

Part II

Signals Circuits and Systems

Workshop 1: Signals in Time Domain

Objective: To analyze continuous-time and discrete-time signals in time domain.

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

1. Plot basic continuous-time and discrete-time signals in time domain
2. Creating a simple audio effect

Equipment Required:

1. A personal computer.
2. Python and NumPy.
3. A headphone set (to be brought by the student)

Components Required: None.

1.1 Real CT Sinusoid

Consider the following general sinusoidal signal:

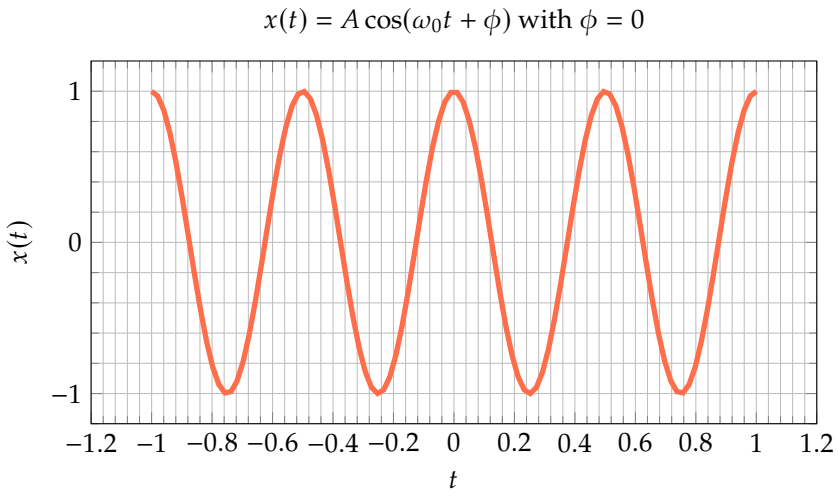
$$x(t) = A\cos(\omega_0 t + \phi) \tag{1.1}$$

where A is the amplitude, $\omega_0 = 2\pi f_0$ is the angular frequency and ϕ is the phase. We can use the following Python code to plot such a signal: Let's import required packages first

```
import numpy as np              # import numpy to handle numerics
import matplotlib.pyplot as plt     # import matplotlib to hadle figures

%matplotlib inline
```

Example 1. *Plot of* $x(t) = A\cos(\omega_0 t + \phi)$ *with* $\phi = 0$.

```
fs = 44100 # 44,100-Hz sampling frequency
ts = 1/fs
t = np.arange(-1., 1., ts) # A linearly-spaced array with step ts
A = 1.
f = 2 # 2 Hz
w0 = 2*np.pi*f
phi = 0.
xt = A*np.cos(w0*t + phi)
fig, ax = plt.subplots()
ax.plot(t,xt)
plt.show()
```



Task 1. *Use the following code snippet to generate plots for*

1. $A = 1$, $\omega_0 = 2\pi$, $\phi = 0$, *i.e.*, $x_1(t) = \cos(2\pi t)$.
2. $A = 0.5$, $\omega_0 = 2\pi$, $\phi = 0$, *i.e.*, $x_2(t) = 0.5\cos(2\pi t)$.
3. $A = 1$, $\omega_0 = 2\pi$, $\phi = \pi/2$, *i.e.*, $x_3(t) = -\sin(2\pi t)$.
4. $A = 1$, $\omega_0 = 4\pi$, $\phi = 0$, *i.e.*, $x_4(t) = \cos(4\pi t)$.

```
fs = 44100 # 44,100–Hz sampling frequency
ts = 1/fs
t = np.arange(–1., 1., ts) # A linearly–spaced array with step ts
f = 2 # 2 Hz
fig, axes  = plt.subplots(2,2, sharex='all', sharey='all', figsize=(18,18))
# Your code goes here
xt1 =
xt2 =
xt3 =
xt4 =

axes[0, 0].plot(t,xt1)
axes[0, 0].set_xlabel('$t$')
axes[0, 0].set_ylabel('$xt1$')
axes[0, 0].grid(True)
axes[0, 0].title.set_text('$\cos(2\pi t)$')

# Your code goes here


plt.show()
```

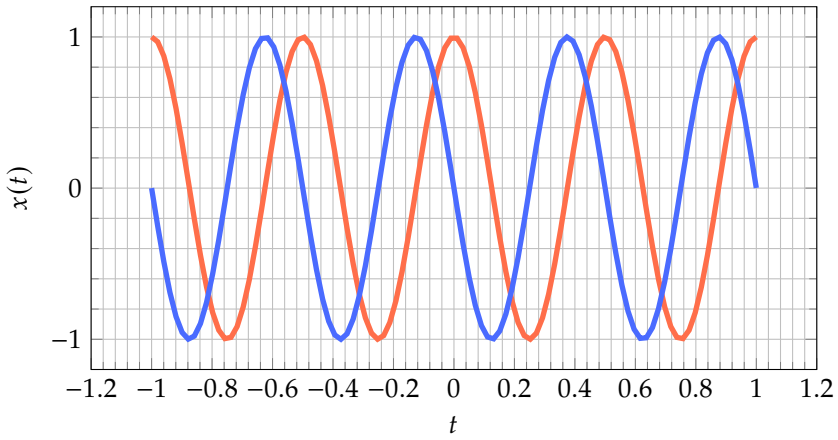
Among $x_1(t)$, $x_2(t)$, $x_3(t)$, and $x_4(t)$, identify the odd and even signals.

Example 2. *The following example illustrates plotting two waves in the same plot.*

```
fs = 4.4e4 # sampling frequency
ts = 1/fs
t = np.arange(–1., 1., ts)
f = 2 # 2 Hz
```

```
w0 = 2*np.pi*f
phi = 0.
x1t = np.cos(w0*t + phi)
phi = np.pi/2
x2t = np.cos(w0*t + phi)
fig, ax = plt.subplots()
ax.plot(t,x1t, label='$\cos(4\pi t)$')
ax.plot(t,x2t, label='$\cos(4\pi t + \pi/2)$')
ax.set_xlabel('$t$')
ax.set_ylabel('$x(t)$')
plt.legend(loc='lower right')
ax.grid(True)

plt.show()
```

Listing 1.1: Plot of $x(t) = A\cos(\omega_0 t + \phi)$ with $\phi = 0$ and $\phi = \pi/2$



1.2 Real CT Exponential (Graded)

Write a Python code to plot the signal $x(t) = Ce^{\alpha t}$ with $C = 1.0$ and $\alpha = 1.2$. Hint: np.exp gives the element-wise exponent of a NumPy array.

```
fs = 4.4e4 # sampling frequency
ts = 1/fs
t = np.arange(−1., 1., ts)

# Your code goes here
```

Comment on the effect due to the sign of α on the signal.

1.3 Growing Sinusoidal Signal (Graded)

Write a Python code to plot the signal with $x(t) = Ce^{rt}\cos(\omega_0 t + \theta)$

1. $C = 0.15$, $r = 2$, and $\theta = 0$.
2. $C = 0.15$, $r = -2$, and $\theta = \pi/2$.

```
fs = 4.4e4 # sampling frequency
ts = 1/fs
t = np.arange(-1., 1., ts)
f = 5 # 5 Hz

# Your code goes here
```

1.4 Real DT Exponential (Graded)

Use the following code snippet plots the discrete-time signal $x[n] = C\alpha^n$ for

1. $C = 1$, $\alpha = 1.1$
2. $C = 1$, $\alpha = 0.92$
3. $C = 1$, $\alpha = -1.1$
4. $C = 1$, $\alpha = -0.92$

```
n = np.arange(-10,10,1)
fig, axes  = plt.subplots(2,2, sharex='all', sharey='all', figsize=(18,18))
C = 1.0
alpha = 1.1
xn1 = C*alpha**(n)

# Your code goes here


axes[0, 0].stem(n,xn1)
axes[0, 0].set_xlabel('$n$')
axes[0, 0].set_ylabel('$1.1^{n}$')
axes[0, 0].grid(True)

# Your code goes here

plt.show()
```

1.5 DT Sinusoids (Graded)

Use the following code snippet plots the discrete-time signal $x[n] = C\alpha^n$ for

1. $x_1[n] = A\cos(\omega_0 n)$ with $A = 1$ and $\omega_0 = 2\pi/12$.
2. $x_2[n] = A\cos(\omega_0 n)$ with $A = 1$ and $\omega_0 = 2\pi/6$.
3. $x_3[n] = A\cos(\omega_0 n)$ with $A = 1$ and $\omega_0 = 8\pi/31$.
4. $x_4[n] = A\cos(\omega_0 n)$ with $A = 1$ and $\omega_0 = 1/1.5$.

```
n = np.arange(−32,32,1)
fig, axes  = plt.subplots(4,1, sharey='all', figsize=(18,18))

# Your code goes here
```

Which of the signals are periodic? State the period for the periodic signals. (Graded)

1. $x_1[n]$
2. $x_2[n]$
3. $x_3[n]$
4. $x_4[n]$

1.6 General Complex Exponential Signals (Graded)

Plot the real part of $x[n] = C\alpha^n$ with $C = |C|e^{j\theta}$, $\alpha = |\alpha|e^{j\omega_0}$ for following C, α, and ω_0 values. Hint: $\text{Re}\{x[n]\} = |C||\alpha|^n \cos(\omega_0 n + \theta)$

1. $C = 1, \alpha = 1.1, \omega_0 = 2\pi/12$
2. $C = 1, \alpha = 0.92, \omega_0 = 2\pi/12$

```
n = np.arange(−12,12,1)
fig, axes  = plt.subplots(2,1, sharex='all', sharey='all', figsize=(18,18))
theta = 0.0

# Your code goes here
```

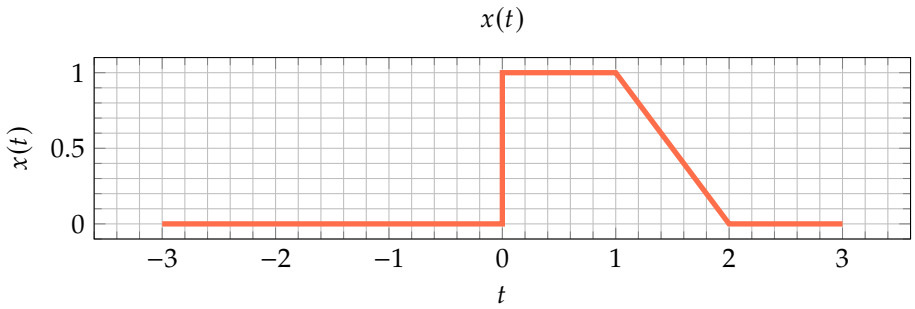
What will happen if α is negative? (Graded)

1.7 Transformation of the Independent Variable (Graded)

The following code plots a signal $x(t)$.

```
def x(t):
    if (t < 0.):
        return 0.
    elif (t < 1.):
        return 1.
    elif (t < 2.):
        return 2. − t
    else:
        return 0.

fs = 4.4e4 # 44,000−Hz sampling frequency
ts = 1/fs
t = np.arange(−3.5, 3.5, ts) # A linearly−spaced array with step ts
fig, axes  = plt.subplots(7,1, sharey='all', figsize=(10,15))

# x(t)
axes[0].plot(t, [x(t_) for t_ in t])
axes[0].set_xlabel('$t$')
```

```
axes[0].set_ylabel('$x(t)$')
axes[0].grid(True)

# Your code goes here

plt.show()
```



Update the above code to plot following functions:

1. $x(t-1)$
2. $x(t+1)$
3. $x(-t)$
4. $x(-t+1)$
5. $x(3t/2)$
6. $x(3t/2+1)$

1.8 Observing a Signal in Frequency Domain

The following code plots a signal and its frequency domain representation. Note that the non-zero values are actually delta functions in frequency, which we will learn later.

```
fs = 100# 100−Hz sampling frequency
ts = 1/fs
t = np.arange(−1., 1., ts) # A linearly−spaced array with step ts
fig, axes  = plt.subplots(2,1, figsize=(10,10))


f = 2 # 2 Hz
omega0 = 2*np.pi*f


xt = 0.75 + 2.*np.cos(omega0*t) + 1.*np.cos(3*omega0*t)

Xf = np.fft.fft(xt)
freq = np.fft.fftfreq(t.shape[−1], d=ts)

axes[0].plot(t,xt)
axes[0].set_xlabel('$t$ in s')
axes[0].set_ylabel('$x(t)$')
valsubrange = np.concatenate((np.arange(0,21,1), np.arange(−1,−21,−1)))
freqsubrage = np.concatenate((np.arange(0,21,1), np.arange(−1,−21,−1)))
axes[1].stem(freq[freqsubrage], Xf.real[valsubrange]/len(t))
```

```
axes[1].set_xlabel('$f$ in Hz')
axes[1].set_ylabel('$\mathfrak{Re}(X(j\omega))$')
plt.xticks(np.arange(-10,11))
plt.show()
```

Interpret the frequency domain representation. [Graded]

1.9 Simple Audio Effects

First, let's install "pyaudio". Please run following commands:

```
!apt install libasound2-dev portaudio19-dev libportaudio2 libportaudiocpp0 ffmpeg
```

```
!pip install pyaudio
```

The following code will play the audio file with the extension ".wav":

```
from IPython.display import Audio # playing the echo wav file
Audio('audio_file.wav')
```

The following code reads and plays a wave file. Create a simple audio effect of your choice. This will need pyaudio to be installed and ffmpeg in the path.

```
import numpy as np
import pyaudio
import wave
from IPython.display import Audio

# import utility

CHUNK = 8820 # 44100 = 1 s

wf = wave.open("power_of_love.wav", 'r')

p = pyaudio.PyAudio()
nchannels=wf.getnchannels()


stream = np.array(np.zeros(nchannels), dtype=np.int16) # init stream
data = wf.readframes(CHUNK)

dtype = '<i2' # little-endian two-byte (int16) signed integers

sig = np.frombuffer(data, dtype=dtype).reshape(-1, nchannels)
signal_chunk = np.asarray(sig)

delayed = np.zeros(signal_chunk.shape, dtype=dtype)

i=0
alpha = 1.0
while data != '' and signal_chunk.shape[0] == CHUNK and i<120:
    i+=1
```

```
    modified_signal_chunk = alpha*signal_chunk + (1. - alpha)*delayed
    modified_signal_chunk_int16 = modified_signal_chunk.astype(np.int16)
    stream = np.vstack((stream, modified_signal_chunk_int16)) # append modified to stream
    # byte_chunk = modifed_signal_chunk_int16.tobytes()
    # stream.write(byte_chunk)
    delayed = signal_chunk
    data = wf.readframes(CHUNK)
    sig = np.frombuffer(data, dtype=dtype).reshape(-1, nchannels)
    signal_chunk = np.asarray(sig)


stream = stream[1:] # pop stream init
byte_stream = stream.tobytes() # np array to bytes
p.terminate()
wf.close()

wfo = wave.open("power_of_love_eco.wav", 'wb') # writing the bot stream to a output wav file
wfo.setnchannels(nchannels)
wfo.setsampwidth(wf.getsampwidth())
wfo.setframerate(wf.getframerate())
wfo.writeframes(byte_stream)
wfo.close()

Audio('power_of_love_eco.wav')
```

Write down the changes that you made to the code. [Graded]

Workshop 2: Signal Analysis in Frequency Domain

Objective: To analyze continuous-time and discrete-time signals in frequency domain.

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

1. Analyze and find the frequency components in a continuous time periodic signal.
2. Synthesize a periodic signal using the Fourier series.
3. Identify the difference between ideal filters and actual filters.
4. Use filters in simple applications.

Equipment Required:

1. A personal computer.
2. Python with NumPy, SciPy, snd Matplotlib

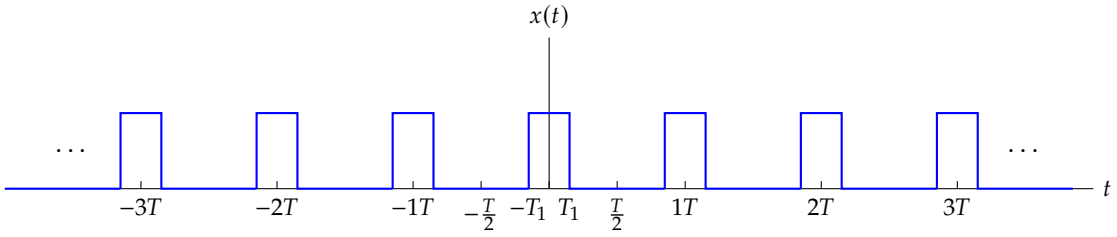
Components Required: None.

2.1 Fourier Series Approximation

The Fourier series analysis equation for a continuous time periodic signal $x(t)$, having the period T, angular frequency ω_0, $(= 2\pi/T)$ can be given as follows.

$$a_k = \frac{1}{T}\int_T x(t)e^{-jk\omega_0 t}\,dt \tag{2.1}$$

Find the Fourier Series coefficients for the square wave given in Figure 1. [Graded] Hint: Take $T_1 = \frac{T}{4}$ (Please note: Include an image of your work.)



Let's import required packages first.

```
# Imports
import numpy as np
import matplotlib.pyplot as plt
from scipy.fftpack import fft,fftshift,ifft
from scipy import signal
```

Q: Taking $A = 1$ V, $T = 1s$ complete the function $a(k)$ to return the Fourier series coefficients of the square wave for given any integer value of k. [Graded]

The Fourier series synthesis equation is given as

$$x(t) = \sum_{k=-\infty}^{\infty} a_k e^{jk\omega_0 t}. \tag{2.2}$$

The following is the equation for the Fourier series approximation of original periodic signal with N number of harmonics.

$$x(t) \simeq x_N(t) = \sum_{k=-N}^{N} a_k e^{jk\omega_0 t}. \tag{2.3}$$

```
# Square pulse
def square(t):
    if t % 1 < 0.25 or t % 1 > 0.75:
        s = 1
    elif t % 1 == 0.25 or t % 1 == 0.75:
        s = 0.5
    else:
        s = 0
    return s

# Fourier series coefficients
def a(k):
    # Your code goes here
    a_k = 1

    return a_k
```

Q: Complete the function fs_approx(t,N) to return the value of a Fourier series approximated periodic signal, at any given time. (Graded)

```
def fs_approx(t, N):
    # Your code goes here
    x_t = 0

    return x_t
```

Q: Update the python script according to the following guidelines. [Graded]

1. Create the array t with equally spaced 1000 elements in the interval $[-2.5, 2.5]$
2. Use the square(t) function to fill the array x with the values of square wave at each time instant in the array t.
3. Use the function fs_approx(t,N) to fill the array y with the function values of the Fourier series approximated square wave.

```
# Fourier series approximation of the square wave
x = []
y = []
N = 5  # CHANGE HERE

time = <----> # Your code goes here
for t in time:
    # Your code goes here
    <------------
    ------------>
```

Q: Write a Python script to plot the original signal, $x(t)$ and the approximated signal, $x_N(t)$ in the same figure for $N = 5$. [Graded] Q: Plot the original signal, $x(t)$ and the approximated signal, $x_N(t)$ in the same figure for $N = 50$. [Graded] Q: Comment on your observations. (i.e. for $N = 5$ and $N = 50$)

2.2 Fourier Series Coefficients [Graded]

Create the two arrays k and ak with integers in the interval $k = -20, \ldots, 20$ and the Fourier series coefficients of the square wave for each k value in the array, respectively. Use stem() function to plot the Fourier series coefficients against k. Q: Plot normalized Fast Fourier Transform (FFT) coefficients in X_{norm} vs k with stem() function. Use set_xlim() function to limit the x-axis to the interval $[-20, 20]$. [Graded]

```
N = 200
t = np.linspace(0, 1-1/N, N)
x = []
for i in t:
  x.append(square(i))

# Obtaining FFT coefficients
X = fftshift(fft(x))
X_norm = X.real/N
k = np.linspace(-N/2, N/2-1, N)

# plotting fft coefficients
# Your code goes here
<------------
------------->
```

Q: Comment on the observations from the above codes. [Graded]

2.3 Ideal Filters and Actual Filters

In this section, we will observe the filtering operation of ideal filters and actual filters by passing a waveform containing sinusoids with different frequencies through the filters.

Q: Complete the function $x(t)$ to return the function value given in the following equation

$$x(t) = a_1 \sin(\omega_1 t) + a_2 \sin(\omega_2 t) + a_3 \sin(\omega_3 t) \tag{2.4}$$

where $a_1 = 0.75$, $a_2 = 1$, $a_3 = 0.5$, $\omega_1 = 100\pi$, $\omega_2 = 400\pi$, $\omega_3 = 800\pi$ [Graded]

```
# Creating 3 sinusoidal signals
# Your code goes here
w1 = <----->
w2 = <----->
w3 = <----->
a1 = <----->
a2 = <----->
a3 = <----->
fs = 4095
ws = 2*np.pi*fs

def x(t):
  # Your code goes here
  x_t =
  <------------
  ------------->
  return x_t
```

Q: Write a python code to plot the waveform in time domain. Limit the x-axis to the interval $[0, 0.04]$. [Graded]

```
time = np.linspace(0,1,fs+1)
xt = [x(t_) for t_ in time]

# Plotting the input signal in time domain
# our code goes here
<------------
------------>
```

Q: Complete the python code for plotting ***absolute*** value of the Fourier transform of $x(t)$, that is X_ω against the angular frequency ω. Execute the cell and sketch the result. [Graded]

```
Xw = fft(xt, 4096)*2*np.pi/fs
Xw = fftshift(Xw)
k = np.arange(1,4097)
w = k/4096*ws - ws/2

# Plotting the input signal in frequency domain
fig, ax = plt.subplots()
# Your code goes here
<------------
------------>

ax.set_title('Frequency Response of the Input signal')
ax.set_xlabel('Angular frequency -'+r' $\omega$ (rad/s)')
ax.set_ylabel('Magnitude')
ax.set_xticks(np.arange(-1200*np.pi, 1200*np.pi+1,400*np.pi))
ax.set_xticklabels([str(i)+(r'$\pi$' if i else '') for i in range(-1200,1210,400)])
ax.set_xlim(-1000*np.pi, 1000*np.pi)
ax.set_yticks([0,np.pi/2,np.pi])
ax.set_yticklabels([0,r'$\pi$/2',r'$\pi$'])
plt.grid()
```

An ideal filter with following frequency response can be used to obtain the sinusoid with the angular frequency of $\omega_2 = 400\pi$, as the output waveform.

$$H(j\omega) = \begin{cases} 1 & \omega_{c1} < |\omega| < \omega_{c2} \\ 0 & otherwise \end{cases} \tag{2.5}$$

Here, ω_{c1} and ω_{c2} are cutoff frequencies and taken as the mid points of the impulses.

Q: Complete the function, ideal_filter(w) to output the $H(j\omega)$. [Graded]

```
# Ideal filter
wc1  = (w1+w2)/2
wc2  = (w2+w3)/2

def ideal_filter(w):
  # Your code goes here
  gain = 1
  <------------
  ------------>
  return gain
```

Q: Use the ideal_filter(w) function to fill the list H_{0w}, with the ideal filter value for each element in w. Complete the following code and sketch the output below. [Graded]

2.3.1 Ideal Filter: Part A

```
k = np.arange(1,4097)
w = k/4096*ws - ws/2
# Your code goes here
H0w  = <----->

# Simulation of Filtering
Y0w  = np.multiply(Xw,H0w)

# Obtaining the time domain signal
y0t  = ifft(fftshift(Y0w*fs/(2*np.pi)))

# Ideal filter frequency response (magnitude)
fig, axes  = plt.subplots(3,1, figsize=(18,18))
axes[0].plot(w,H0w)
axes[0].set_title('Frequency Response of the Ideal Filter')
axes[0].set_xlabel('Angular frequency -'+r' $\omega$  (rad/s)')
axes[0].set_ylabel('Magnitude')
axes[0].set_xticks(np.arange(-1200*np.pi, 1200*np.pi+1,200*np.pi))
axes[0].set_xticklabels([str(i)+(r'$\pi$' if i else '') for i in range(-1200,1210,200)])
axes[0].set_xlim(-1000*np.pi, 1000*np.pi)
axes[0].grid()

# Frequency response of the ideal filter output (magnitude)
axes[1].plot(w,abs(Y0w))
axes[1].set_title('Fourier Transform of the Output Signal')
axes[1].set_xlabel('Angular frequency -'+r' $\omega$  (rad/s)')
axes[1].set_ylabel('Magnitude')
axes[1].set_xticks(np.arange(-1200*np.pi, 1200*np.pi+1,200*np.pi))
axes[1].set_xticklabels([str(i)+(r'$\pi$' if i else '') for i in range(-1200,1210,200)])
axes[1].set_xlim(-1000*np.pi, 1000*np.pi)
axes[1].set_yticks([0,np.pi/2,np.pi])
axes[1].set_yticklabels([0,r'$\pi/2',r'$\pi$'])
axes[1].grid()

# Output signal in time domain
axes[2].plot(time,np.real(y0t))
axes[2].set_title('Output Signal in time domain')
axes[2].set_xlabel('Time (s)')
axes[2].set_ylabel('Amplitude')
axes[2].set_xlim(0, 0.04)
axes[2].grid()
```

2.3.2 Ideal Filter: Part B

Execute the bellow cells and observe the output.

```
# Actual Filter
b, a  = signal.butter(5, [2*wc1/ws, 2*wc2/ws], 'bandpass', analog=False)
ww, h = signal.freqz(b, a, 2047)
ww    = np.append(-np.flipud(ww), ww)*ws/(2*np.pi)
h     = np.append(np.flipud(h), h)

# Filtering
y     = signal.lfilter(b,a,xt)
```

```
# Obtaining the frequency response of the output signal
Y    = fft(y,4096)*2*np.pi/fs
Y    = fftshift(Y)
```

```
# Actual filter frequency response (magnitude)
fig, axes  = plt.subplots(3,1, figsize=(18,18))
axes[0].plot(ww, abs(h) )
axes[0].set_xlabel('Angular frequency –'+r' $\omega$  (rad/s)')
axes[0].set_ylabel('Magnitude')
axes[0].set_title('Frequency Response of the Actual Filter')
axes[0].set_xticks(np.arange(–1200*np.pi, 1200*np.pi+1,200*np.pi))
axes[0].set_xticklabels([str(i)+(r'$\pi$' if i else '') for i in range(–1200,1210,200)])
axes[0].set_xlim(–1000*np.pi, 1000*np.pi)
axes[0].grid()

# Frequency response of the actual filter output (magnitude)
axes[1].plot(w,abs(Y))
axes[1].set_title('Fourier Transform of the Output Signal')
axes[1].set_xlabel('Angular frequency –'+r' $\omega$  (rad/s)')
axes[1].set_ylabel('Magnitude')
axes[1].set_xticks(np.arange(–1200*np.pi, 1200*np.pi+1,200*np.pi))
axes[1].set_xticklabels([str(i)+(r'$\pi$' if i else '') for i in range(–1200,1210,200)])
axes[1].set_xlim(–1000*np.pi, 1000*np.pi)
axes[1].set_yticks([0,np.pi/2,np.pi])
axes[1].set_yticklabels([0,r'$\pi$/2',r'$\pi$'])
axes[1].grid()

# # Output signal in time domain
axes[2].plot(time,np.real(y))
axes[2].set_title('Output Signal in time domain')
axes[2].set_xlabel('Time (s)')
axes[2].set_ylabel('Amplitude')
axes[2].set_xlim(0, 0.04)
axes[2].grid()
```

Q: Comment on your observations in Part - A and Part - B. [Graded]

2.4 Removing Power Line Noise in an ECG Signal

The electrocardiogram (ECG) is a biomedical signal which gives electrical activity of heart. An ECG signal is characterized by six peaks and valleys, which are traditionally labeled P, Q, R, S, T, and U, as shown in Figure 2. ECG has frequency range from 0.5 Hz to 80 Hz and power line interference, mainly coming from electromagnetic interference by power line, introduces 50-Hz frequency component in the ECG signal. This is a major cause of corruption of ECG. In this section, we will design a simple filter to remove the power line interference from an ECG signal. The three main components of an ECG signal are the P wave (depolarization of the atria) the QRS complex (depolarization of the ventricles) and the T wave,(repolarization of the ventricles), as shown in Fig. 2.1

Task 1. *Write a python script to read the data in the file ecg_signal.csv and fill the list ecg with the data.*

```
# Reading the ECG data
ecg = []
# EDIT HERE
```

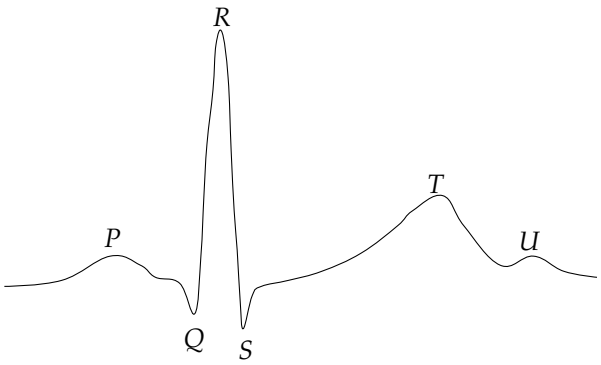
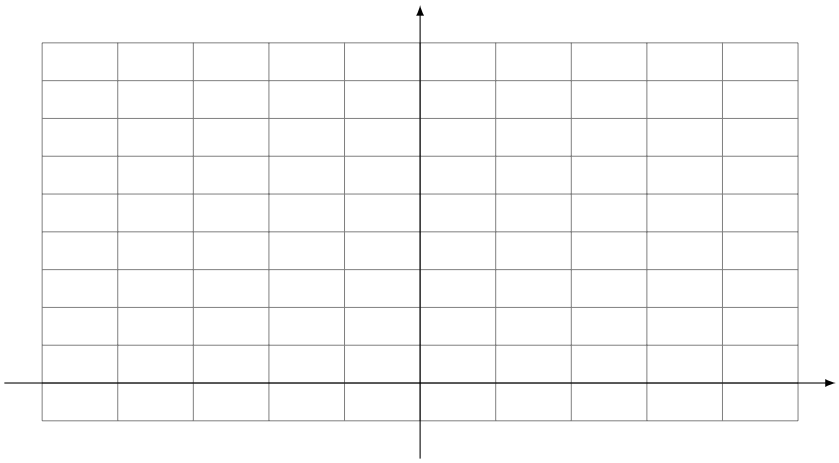


Figure 2.1: PQRS Components of ECG

```
duration = 10 # seconds
T = duration/len(ecg)
Fs = 1/T

# Obtaining the fourier transform
F = fftshift(fft(ecg))
fr = np.linspace(−Fs/2, Fs/2, len(F))
```

Task 2. *Plot the absolute value of the Fourier transform with respect to frequency. Limit the x-axis to the interval* $[-100, 100]$. *Sketch the output.*



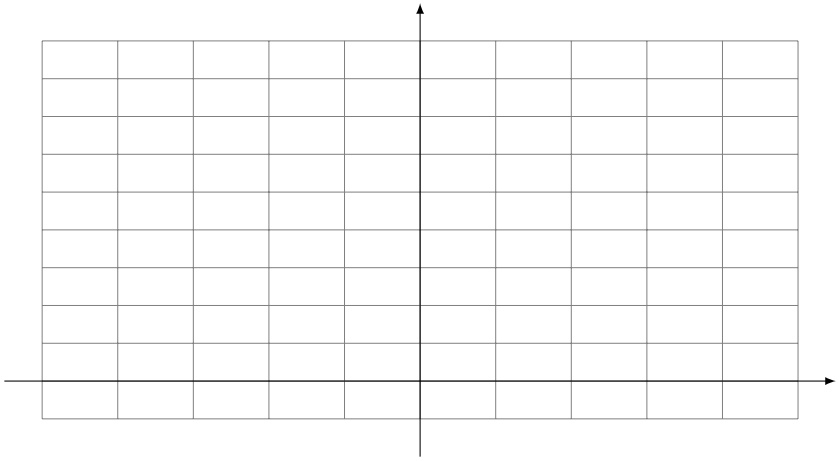
Task 3. *What type of filter that can be used to remove the noise at* 50 Hz*?*

Task 4. *Edit the code below with the correct name of the filter selecting from the table given below. Execute the cell and sketch the frequency response of the filter.*

```
# Designing the filter
f1 = 49
f2 = 51
filter_type = '' # EDIT HERE
b, a = signal.butter(2, [2*f1/Fs, 2*f2/Fs], filter_type , analog=False)

# Obtaining the frequency response of the filter
```

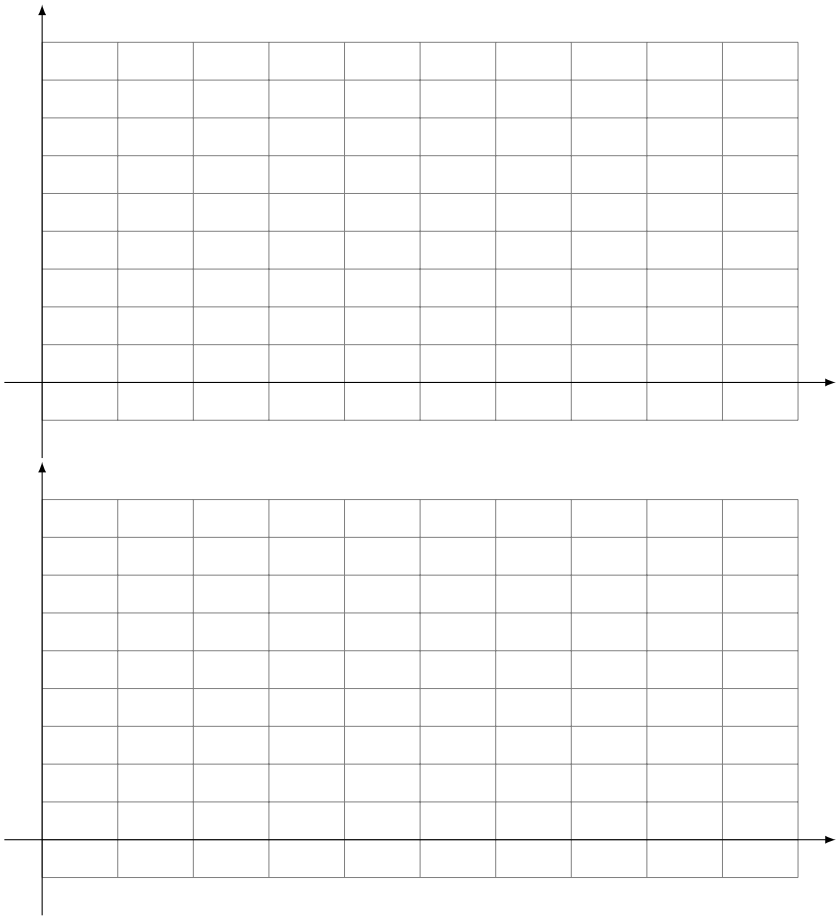
```
ww, h = signal.freqz(b, a, 2047)
ww = np.append(-np.flipud(ww), ww)
h = np.append(np.flipud(h), h)

# Plotting the frequency response
fig, ax = plt.subplots(figsize=(10,6))
ax.plot(ww*Fs/(2*np.pi), abs(h) )
ax.set_title('Frequency Response of the Actual Filter')
ax.set_xlabel('Frequency [Hz]')
ax.set_ylabel('Magnitude')
ax.set_xlim(-100,100)
ax.grid()
```

Filter type	Name to be used in code
Low-pass filter	'lowpass'
Band-pass filter	'bandpass'
High-pass filter	'highpass'
Band-stop filter	'bandstop'



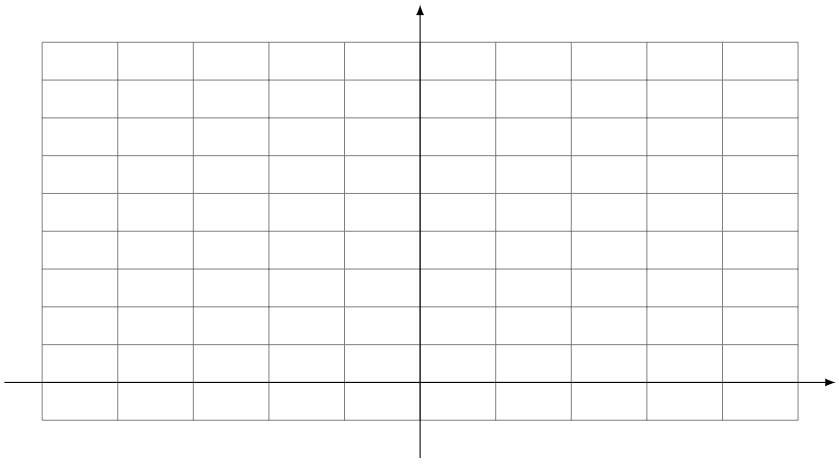
Task 5. *Edit the given below to plot the input and the output waveforms vs time. Use the subplots function to plot the graphs in two axes in the same figure. Limit the x-axis to the interval* $[0, 3]$. *Sketch the result.*

```
time = np.arange(T, duration+T, T)

# Filtering the ECG wavefoem
output = signal.lfilter(b, a, ecg)
```



Task 6. *Complete the code in below to plot the absolute value of Fourier transform of the output waveform with respect to the frequency. Limit the x axis to the interval* $[-100, 100]$. *Execute the cell sketch the output.*

```
F = fftshift(fft(output))
```



Workshop 3: Linear, Time-Invariant Systems

Objective: To analyze linear, time-invariant systems.

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

1. Analyse continuous time systems using the convolution integral.
2. Analyse discrete time systems using the convolution sum.

Equipment Required:

1. A personal computer.
2. Python with NumPy, SciPy, adn Matplotlib

Components Required: None.

3.1 Continuous-Time Systems: Convolution Integral

We have

$$\boxed{y(t) = \int_{-\infty}^{\infty} x(\tau)h(t-\tau)d\tau}$$

which is referred to as the convolution integral or the superposition integral. This corresponds to the representation of a continuous-time LTI system in terms of its response to a unit impulse.

$$y(t) = x(t) * h(t).$$

A continuous-time LTI system is completely characterized by its impulse response—i.e., by its response to a.single elementary signal, the unit impulse $\delta(t)$.

3.1.1 Implementing Convolution Using Numerical Integration

Let $x(t)$ be the input to an LTI system with unit impulse response $h(t)$, where

$$x(t) = e^{-at}u(t), a > 0$$

and

$$h(t) = u(t).$$

We wish to compute the output

$$y(t) = x(t) * h(t)$$

obtained when $x(t)$ is fed to the system represented by $h(t)$. Fig. 3.1 shows $x(t)$, $h(t)$ and related signals.

For $t < 0$, the product $x(\tau)$ and $h(t-\tau)$ is zero, consequently $y(t)$ is zero.

For $t > 0$,

$$x(\tau)h(t-\tau) = \begin{cases} e^{-a\tau}, & 0 < \tau < t, \\ 0, & \text{otherwise.} \end{cases}$$

$$y(t) = \int_0^t e^{-a\tau}d\tau = -\frac{1}{a}e^{-a\tau}\Big|_0^t$$
$$= \frac{1}{a}(1 - e^{-at})$$

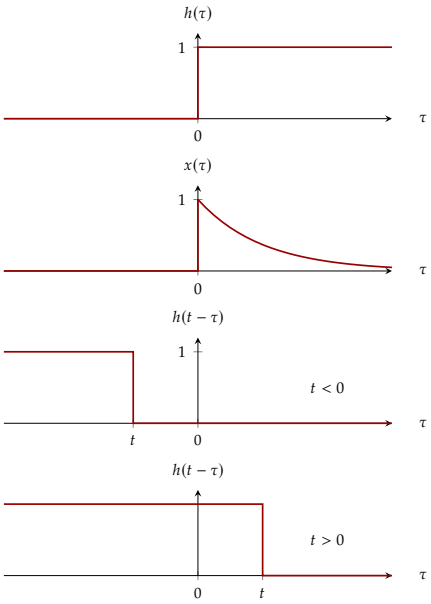


Figure 3.1: Calculation of convolution integral for the example

Thus for all t,

$$y(t) = \frac{1}{a}(1 - e^{-at})u(t)$$

Fig. 3.2 shows graph of the output.

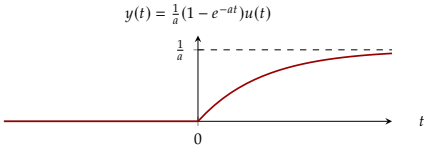


Figure 3.2: Response

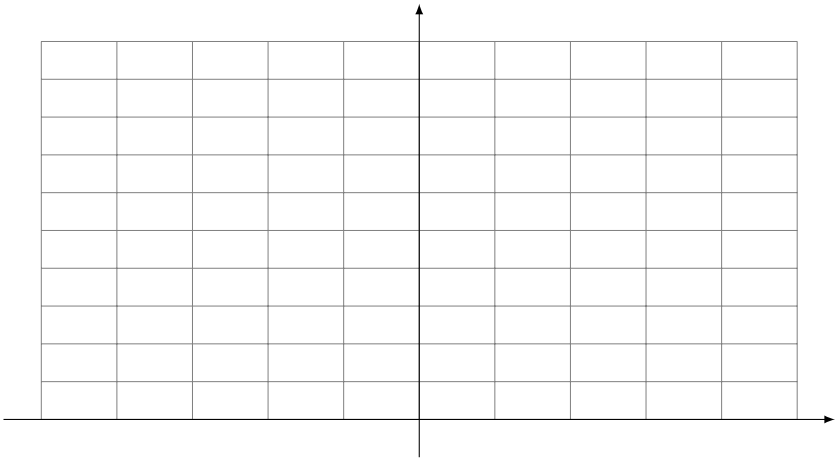
The listing 3.1 shows the computation of the convolution integral to compute $y(t)$. Note that we have to use numerical integration.

```
from scipy import integrate
import numpy as np
import matplotlib.pyplot as plt
h = lambda t: (t > 0)*1.0
x = lambda t: (t > 0) * np.exp(-2*t) # a = -2
Fs = 50  # Sampling frequency for the plotting
T = 5    # Time range
t = np.arange(-T, T, 1/Fs)  # Time samples

plt.figure(figsize=(8,3))
plt.plot(t, h(t), label='$h(t)$')
plt.plot(t, x(t), label='$x(t)$')
plt.xlabel(r'$t$')
plt.legend()

# Plotting
t_ = 1 # For illustration, choose some value for t
```

```
flipped = lambda tau: h(t_ − tau)
product = lambda tau: x(tau)∗h(t_ − tau)
plt.figure(figsize=(8,3))
plt.plot(t, x(t), label=r'$x(\tau)$')
plt.plot(t, flipped(t), label=r'$h(t − \tau)$')
plt.plot(t, product(t), label=r'$x(\tau)h(t −\tau)$')

# Computing the convolution using integration
y = np.zeros(len(t))
for n, t_ in enumerate(t):
    product = lambda tau: x(tau) ∗ h(t_ − tau)
    y[n] = integrate.simps(product(t), t) # Actual convolution at time t

plt.plot(t, y, label=r'$x(t)\ast h(t)$') # Plotting the output y
plt.xlabel(r'$t$')
plt.legend()
```

Listing 3.1: Computing the Convolution Integral

Task 1. *Sketch the output of the listing 3.1 above and comment.*



3.1.2 Convolving with a Signal Composed of Impulse Functions

We approximated impulse function $\delta(t)$ as

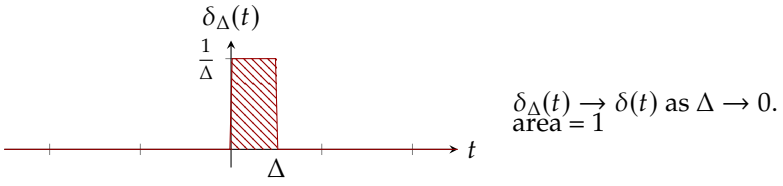


Figure 3.3: Approximation of $\delta(t)$.

Consider the following simple approximate implementation of $\delta(t)$.

```
fs = 1000  # Sampling frequency for the plotting
delta = lambda t: np.array([fs/10 if 0 < t_  and t_ < 1/(fs/10) else 0.0 for t_ in t])
```

Listing 3.2: A Simple Implementation of the Impulse Function

Task 2. *Use Simpson's rule integration (as shown in Listing 3.1), obtain the value of*

$$\int_{-\infty}^{\infty} \delta(t)dt.$$

Consider

$$x(t) = e^{-at}u(t), a > 0,$$

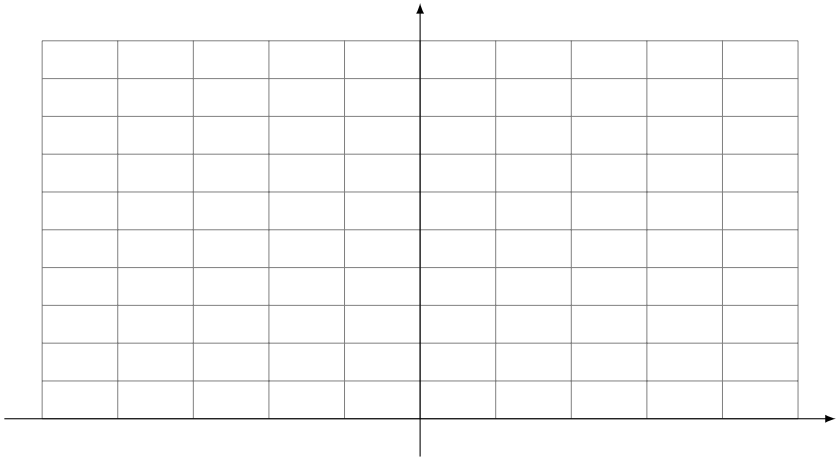
and

$$h(t) = \delta(t+2) + \delta(t-1).$$

Task 3. *Write an expression for*

$$y(t) = x(t) * h(t).$$

Task 4. *Compute $y(t)$ in a similar fashion as in Listing 3.1 and sketch.*



3.2 Discrete-Time Systems: Convolution Sum

Using the convolution we can express the response of an LTI system to an arbitrary input in terms of the system's response to the unit impulse. An LTI system is completely characterized by its response to a single signal, namely, its response to the unit impulse.

The convolution of the sequence $x[n]$ and $h[n]$ is given by

$$y[n] = \sum_{k=-\infty}^{\infty} x[k]h[n-k], \tag{3.1}$$

which we represent symbolically as

$$y[n] = x[n] * h[n] \tag{3.2}$$

Fig. 3.4 shows $x[n]$ and $h[n]$ to be convolved. Fig. 3.5 illustrates how the convolution is implemented. Note

Figure 3.4: $x[n]$ and $h[n]$ for convolution implementation.

that there can be an overlap between $x[k]$ and $h[n-k]$ only when $n \in [-5, 5]$. Therefore, length of the array

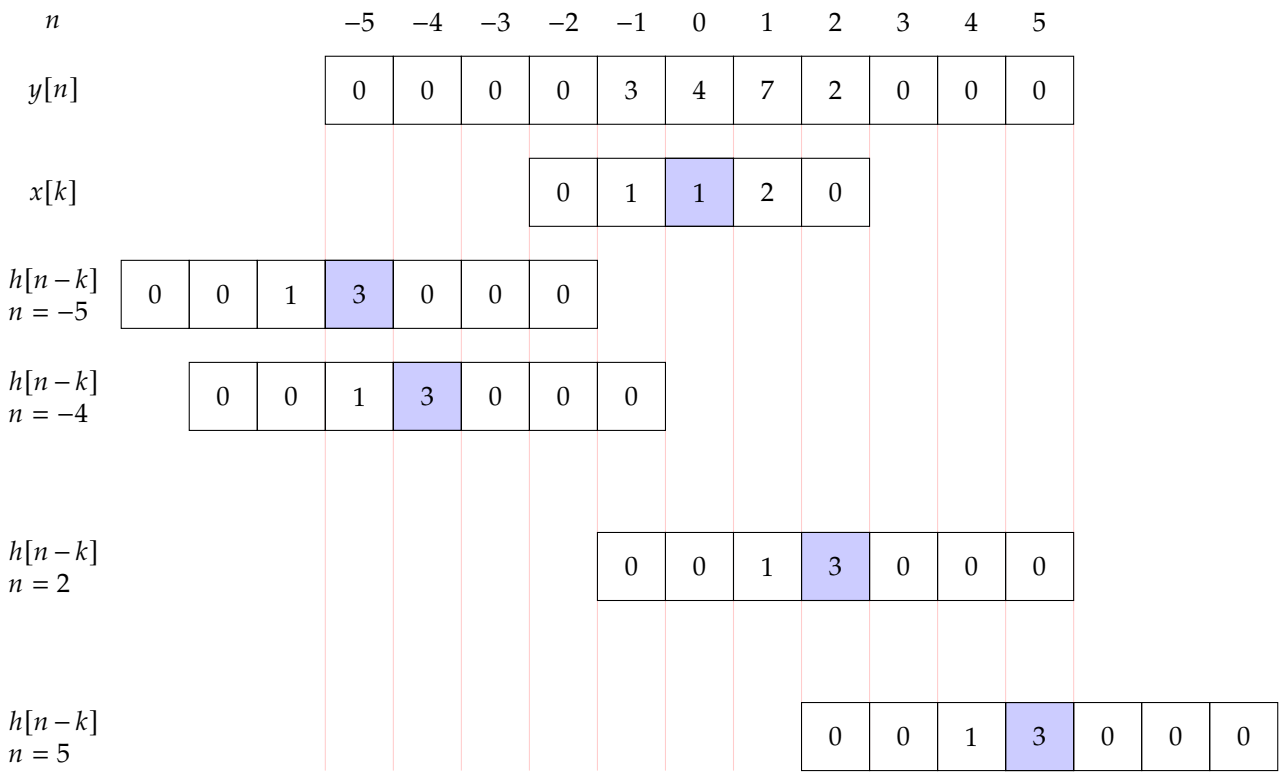


Figure 3.5: Illustration of convolution implementation.

for $y[n]$ is len(x) + len(h) − 1. We must carefully estimate the indices of lower and upper limits of overlapping region for each x and h. The listing 3.3 shows the actual implementation of the convolution sum using NumPy arrays. Note that we cannot have negative indices. Therefore, the implementation has a difference from Fig. 3.5.

```
x = np.array([0, 1, 1, 2, 0])
h = np.array([0, 0, 0, 3, 1, 0, 0])
hr = np.flip(h)
xo = 2
ho = 4
y = np.zeros(len(x) + len(h) − 1)
for n in range(len(y)):
   xkmin = max(0, n − len(h) + 1)
   xkmax = min(len(x), n + 1)
   hkmin = max(0, len(h) − n −1)
   hkmax = min(len(h), len(x) + len(h) −n −1)
   y[n] = np.sum(x[xkmin:xkmax]*hr[hkmin:hkmax])
   print("y[{0}] =  x[{1}:{2}]*h[{3}:{4}] = {5}".format(n, xkmin, xkmax, hkmin, hkmax, y[n]))
```

Listing 3.3: Implementation of DT convolution.

Task 5. *Sketch the output of the listing 3.1 above and comment.*

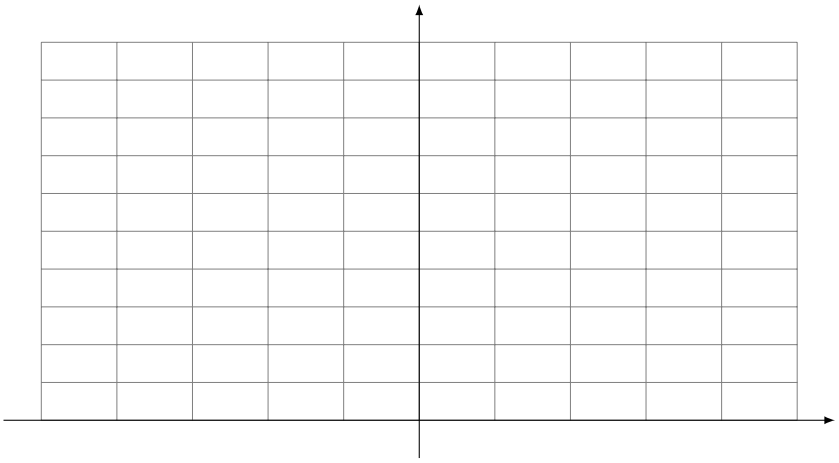


Fig. 3.6 shows $x[n]$ and $h[n]$ to be convolved.

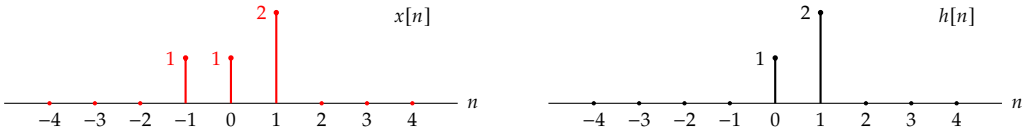


Figure 3.6: $x[n]$ and $h[n]$ for convolution.

Task 6. *Study the listing 3.3 and use a for-loop to compute the convolution sum at each* $n \in [-4, 4]$

Task 7. *Use scipy.signal.convolve*[1] *to compute the above convolution. Describe the effect of modes full, valid, and same.*

3.3 An Application in Audio Signal Filtering

The following code 3.4 is for reading a .wav file and generating the filter coefficients of a so-called finite impulse response (FIR) filter. We can consider these coefficients as the impulse response of the filter and compute the filtered output using convolution. It also plots the frequency response of the filter. Note that frequencies are normalized. The signal can be written to disk as a .wav file as shown at the end of the code snippet.

```
from scipy import signal
import numpy as np
import matplotlib.pyplot as plt
data, samplerate = sf.read('audio_file.wav')

nyquist = samplerate//2
fc = 2000/nyquist
n = 121
b = signal.firwin(n, fc, pass_zero=True)
w, h = signal.freqz(b)
import matplotlib.pyplot as plt
fig, ax1 = plt.subplots()
ax1.set_title('Digital filter frequency response')
ax1.plot(w, 20 * np.log10(abs(h)), 'b')
ax1.set_ylabel('Amplitude [dB]', color='b')
ax1.set_xlabel('Frequency [rad/sample]')
ax2 = ax1.twinx()
angles = np.unwrap(np.angle(h))
```

[1] https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.convolve.html

```
ax2.plot(w, angles, 'g')
ax2.set_ylabel('Angle (radians)', color='g')
ax2.grid()
ax2.axis('tight')
plt.show()


% Your code here for convolution.

sf.write('audio_file_filtered.wav', np.vstack((ch1, ch2)).T + data, samplerate)
```

Listing 3.4: Filtering using convolution.

Task 8. *Noting that data may have a pair of channels (stereo) use convolution to filter the audio signal.*

Task 9. *Creatively achieve various filtering effects.*

3.4 Convolution Sum in 2-D

In 2-D, as applicable in image processing, convolution sum with a kernel $h[m,n]$ with non-zero values in $(m,n) \in ([-a,a],[-b,b])$ is

$$(h*x)[m,n] = h[m,n]*x[m,n] = \sum_{s=-a}^{a}\sum_{k=-b}^{b} h[s,t]x[m-s,n-t].$$

Consider the "image"

$$x[m,n] = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

and the filtering kernel

$$h = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}.$$

Note that there in only one pixel in the image which is non-zero.

Task 10. *Convolve the image $x[m,n]$ with filter $h[m,n]$. Hint: Use signal.convolve2d.*

Task 11. *Interpret the above result.*

3.5 Application: Using Convolution to Filter an Image

Fig. 3.7 shows an image of a set of Allen keys. In this section, we will filter this image with a filtering kernel called the Sobel horizontal kernel. We can read an image using the following snippet in 3.5.

```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
x = mpimg.imread('allenkeys.png')
fig, ax = plt.subplots(1,2)
ax[0].imshow(x, cmap='gray')
```

Listing 3.5: Image reading.

Figure 3.7: Allen keys.

Task 12. *Filter this image with the kernel*

$$h = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}.$$

Workshop 4: RLC Circuits

Objective: To understand transients and frequency responses of RLC circuits.

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

1. Understand transient behaviour of RLC circuits
2. Understand how the voltage and current varies across RLC circuits and their dependency on the frequency
3. Understand resonance in RLC circuits

Equipment Required:

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

Components Required:

1. Resistors - 10 $k\Omega$ (1 No.), 2.2 $k\Omega$ (1 No.), 1 $k\Omega$ (1 No.), 100 Ω (1 No.)
2. Capacitors - 47 μF (1 No.), 100 nF (1 No.)
3. Inductors - 10 mH (1 No.)

4.1 Introduction

RLC circuits consist of resistors, capacitors and inductors. Unlike resistors, inductors and capacitors are sensitive to the variations in current and voltage. Hence, they show frequency-sensitive behaviours. Consequently, the input-output relationship of an RLC circuit will depend on the input frequency, among the other factors. In this experiment, we will observe how different configurations of resistors, capacitors and inductors respond to different input signals.

4.2 Pre-Lab

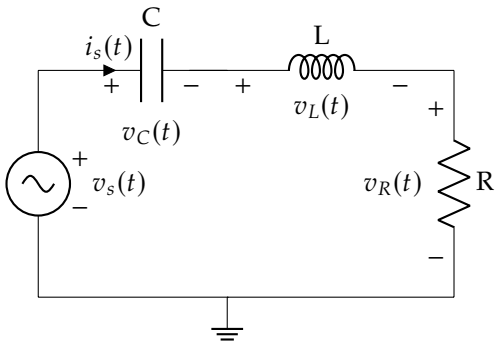


Figure 4.1: RLC filter

Task 1. *Consider the RLC circuit illustrated in Figure 4.1. Let the source frequency be f. Using the relationships, $v_C(t) = \frac{1}{j2\pi fC} i_C(t)$, $v_L = j2\pi fL\ i_L(t)$ and $v_R(t) = R\ i_R(t)$, obtain an expression for $v_R(t)$ in terms of $v_s(t)$, R, L, C and f.*

Task 2. *Calculate the magnitude of $v_R(t)$ in terms of $|v_s(t)|$, R, L, C and f.*

Task 3. *From the result obtained in Task 2, find the frequency $\hat{f}$ for which the $|v_R(t)|$ is maximized.*

Task 4. *Calculate the phase of $v_R(t)$ in terms of $\angle v_s(t)$, R, L, C and f. Obtain the value of $\angle v_R(t)$ when $f = \hat{f}$.*

4.3 Transients

Transient responses are the changes in the output observed immediately after a sudden change in the input. They die out slowly allowing the system to settle down to an equilibrium state. Under this section, we will observe the transient response of a capacitor and an inductor for a step signal (a sudden increase/decrease of voltage).

4.3.1 Transient Response of a Capacitor

Task 5. *Construct the circuit illustrated in Figure 4.2 on the breadboard.*

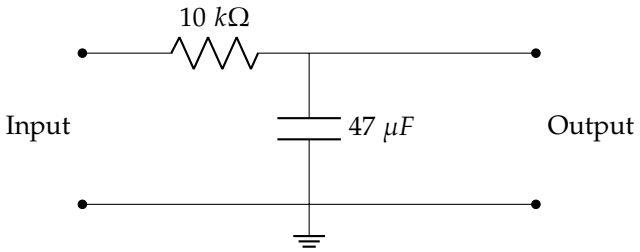


Figure 4.2: Circuit for observing the transient response of a capacitor

Task 6. *Apply a $0.1\ Hz$, $5\ V$ peak-to-peak square wave with a DC offset of $2.5\ V$ to the input using the signal generator. Observe the input and output waveforms from the oscilloscope (set the time scale of the oscilloscope to $2.5\ s$). Draw one period of the output wave.*

4.3.2 Transient Response of an Inductor

Task 7. *Construct the circuit illustrated in Figure 4.3 on the breadboard.*

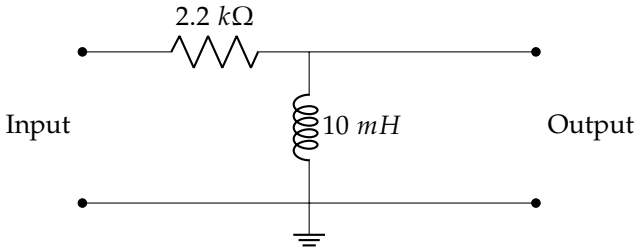


Figure 4.3: Circuit for observing the transient response of an inductor

Task 8. *Apply a $10\ kHz$, $2\ V$ peak-to-peak square wave with a DC offset of $1\ V$ to the input using the signal generator. Observe the input and output waveforms from the oscilloscope (set the time scale to $2.5\ \mu s$). Draw one period of the output wave.*

4.4 RC Filters

RC filters consist of capacitors and resistors. They better suit for driving high impedance loads. In this section we will observe their magnitude and phase responses for sinusoidal inputs.

4.4.1 RC Low-pass Filter

Task 9. *Construct the circuit illustrated in Figure 4.4.*

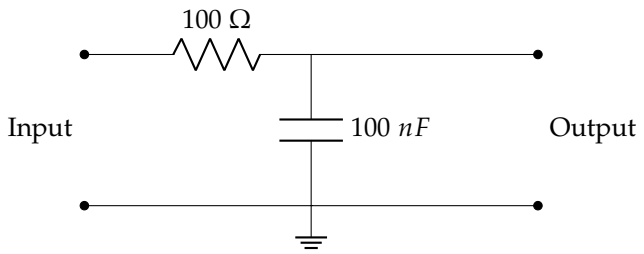


Figure 4.4: RC low-pass filter

Task 10. *Connect the two probes of the oscilloscope to the input and the output of the above circuit. Provide a* 2 V *peak-to-peak sinusoidal input signal with following frequencies and record the input and the output amplitudes along with the voltage gains in Table 4.1 given below.*

Frequency	Input amplitude	Output amplitude	Voltage gain
1 kHz			
10 kHz			
100 kHz			

Table 4.1: RC low-pass filter magnitude response

Task 11. *Observe and comment on how the phase difference between the input and the output waveforms changes with the frequency.*

Task 12. *Measure the phase difference between the input and output waveforms when the input frequency is* 100 kHz *(use* ***cursors*** *in the oscilloscope). Show your calculations.*

4.4.2 RC High-pass Filter

Task 13. *Construct the circuit illustrated in Figure 4.5.*

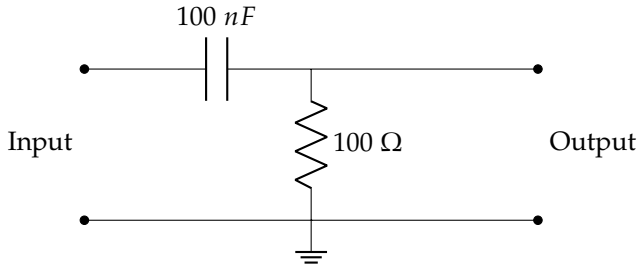


Figure 4.5: RC high-pass filter

Frequency	Input amplitude	Output amplitude	Voltage gain
1 kHz			
10 kHz			
100 kHz			

Table 4.2: RC high-pass filter magnitude response

Task 14. *Connect the two probes of the oscilloscope to the input and the output of the above circuit. Provide a* 2 V *peak-to-peak sinusoidal input signal with following frequencies and record the input and the output amplitudes along with the voltage gains in Table 4.2 given below.*

Task 15. *Use the* ***math*** *function in the oscilloscope to observe the voltage across the capacitor, when the input frequency is* 10 kHz*. Draw all the three waveforms (to the same scale) on the same plot.*

Task 16. *Observe and comment on how the phase difference between the input and output waveforms changes with the frequency.*

4.5 RL Filters

RL filters consist of inductors and resistors. They better suit for driving low impedance loads. In this section we will observe their magnitude and phase responses for sinusoidal inputs.

4.5.1 RL Low-pass Filter

Task 17. *Construct the circuit illustrated in Figure 4.6.*

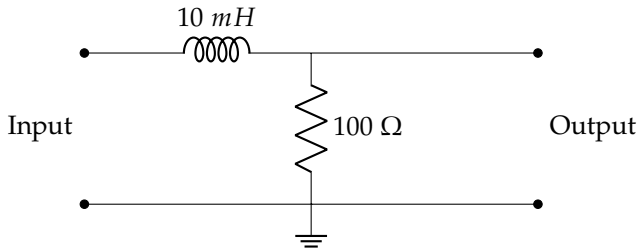


Figure 4.6: RL low-pass filter

Task 18. *Connect the two probes of the oscilloscope to the input and the output of the above circuit. Provide a* 2 V *peak-to-peak sinusoidal input signal with following frequencies and record the input and the output amplitudes along with the voltage gains in Table 4.3 given below.*

Frequency	Input amplitude	Output amplitude	Voltage gain
1 kHz			
10 kHz			
100 kHz			

Table 4.3: RL low-pass filter magnitude response

Task 19. *Observe and comment on how the phase difference between the input and output waveforms changes with the frequency.*

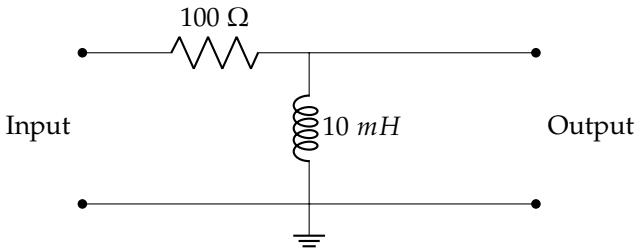


Figure 4.7: RL high-pass filter

4.5.2 RL High-pass Filter

Task 20. *Construct the circuit illustrated in Figure 4.7.*

Task 21. *Connect the two probes of the oscilloscope to the input and the output of the above circuit. Provide a* 2 *V peak-to-peak sinusoidal input signal with following frequencies and record the input and the output amplitudes along with the voltage gains in Table 4.4 given below.*

Frequency	Input amplitude	Output amplitude	Voltage gain
1 kHz			
10 kHz			
100 kHz			

Table 4.4: RL high-pass filter magnitude response

Task 22. *Observe and comment on how the phase difference between the input and output waveforms changes with the frequency.*

4.6 RLC Filters

RLC circuits act as cascaded RL, RC circuits and hence show combined behaviours including resonance. This section focuses on observing their magnitude and phase responses for sinusoidal inputs.

Task 23. *Construct the circuit illustrated in Figure 4.8 on the breadboard.*

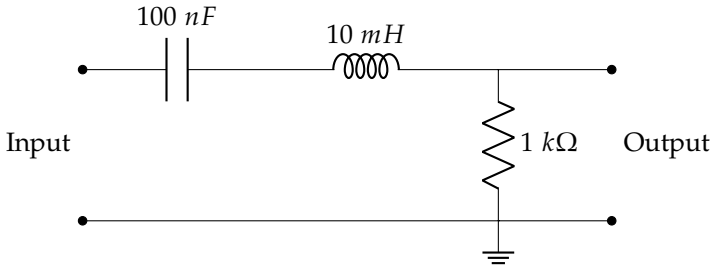


Figure 4.8: RLC filter

Task 24. *Connect the two probes of the oscilloscope to the input and the output of the above circuit. Provide a* 2 *V peak-to-peak sinusoidal input signal with following frequencies and record the input and the output amplitudes along with the voltage gains in Table 4.5.*

Task 25. *Suggest a method to find out the resonance frequency of the above circuit using the result of Task 4.*

Task 26. *Record the resonance frequency,* $\hat{f}$*. Calculate the theoretical value of* $\hat{f}$ *from the result obtained in Task 3.*

Frequency	Input amplitude	Output amplitude	Voltage gain
1 kHz			
10 kHz			
100 kHz			

Table 4.5: RLC filter magnitude response

Task 27. *Verify the above result by observing the magnitude variation of the output while varying the input frequency within a small range around* $\hat{f}$.

♣ The End ♣

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;

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

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

$$A_V = \frac{Z_L}{a_{11}Z_L + a_{12}}. \tag{5.4}$$

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.*

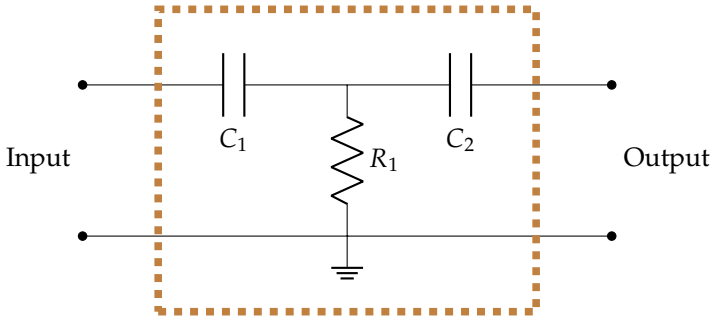


Figure 5.2: Two-port network I

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.*

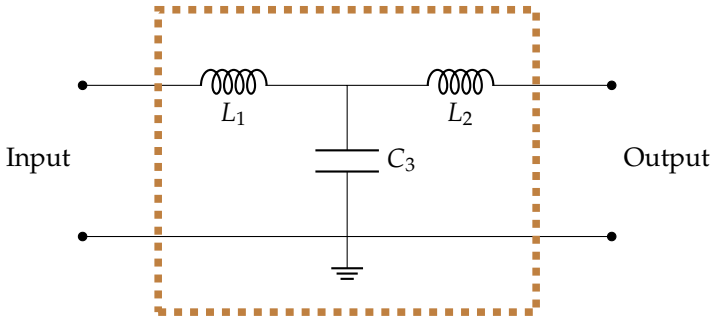


Figure 5.3: Two-port network II

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.*

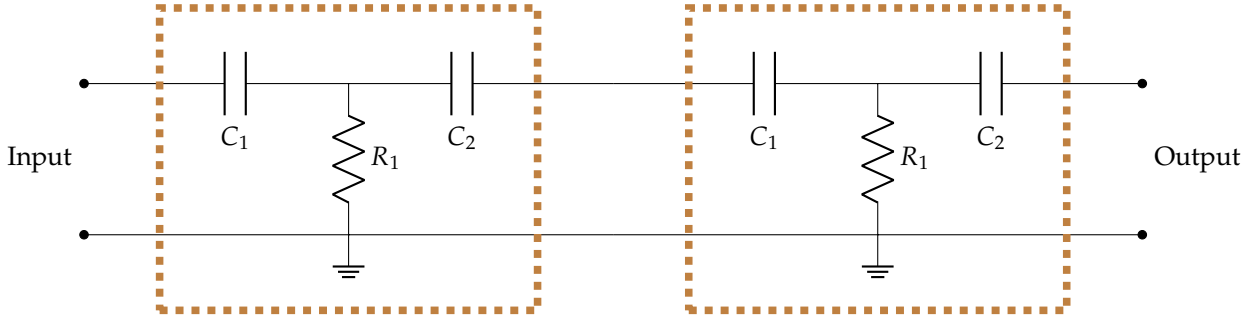


Figure 5.4: Cascaded two-port network

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 ♣

Workshop 6: Realization of Basic Waveforms

Objective: To understand methods of realizing basic waveforms.

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

1. Understand how to synthesize signals in practice
2. Understand practical aspects related to the impulse response of a linear, time-invariant system

Equipment Required:

1. Signal generator
2. Oscilloscope
3. DC power supply
4. Digital multimeter
5. Breadboard and wires

Components Required:

1. NE5532 Op-amp IC (1 No.)
2. Resistors - 100 $k\Omega$ (1 No.), 10 $k\Omega$ (1 No.), 2.2 $k\Omega$ (1 No.), 1 $k\Omega$ (2 Nos.)
3. Capacitors - 220 nF (1 No.), 100 nF (1 No.)

6.1 Introduction

Operational amplifiers (op-amps) can be used to synthesize a wide range of signals with varying shapes, frequencies and amplitudes. Such signals are extremely useful as reference signals or inputs to other systems. In this experiment, you will be synthesizing different wave forms starting from a square wave, which is a relatively easy waveform to generate (e.g.: from an astable multivibrator). Furthermore, you will be able to observe the extent to which the theoretical models of the synthesizers agree with the practical implementations.

6.2 Pre-Lab

Task 1. *Consider the integrator circuit illustrated in Figure 6.1. Obtain the exact relationship between $V_o(t)$ and $V_i(t)$ in terms of R, R_f, C_f and t, where t denotes the time (you may assume ideal op-amp characteristics).*

Task 2. *Show that when $R_f \to \infty$, $V_o(t) - V_o(0) \approx -\frac{1}{C_f R}\int_0^t V_i(\tau)d\tau$.*

Task 3. *Consider the differentiator circuit illustrated in Figure 6.2. Prove that the output voltage $V_o(t)$ is given by $V_o(t) = -R_f C\frac{dV_i(t)}{dt}$, where t denotes the time (you may assume ideal op-amp characteristics).*

Task 4. *Consider the RC filter illustrated in Figure 6.3a. The impulse response $h(t) = \frac{1}{RC}e^{-\frac{t}{RC}}u(t)$ of the filter is illustrated in Figure 6.3b ($u(t)$ denotes the unit step function). A rectangular pulse $s(t, \Delta)$ with unit area is illustrated in Figure 6.3c. Noting that the convolution operation can be written as $h(t) * s(t, \Delta) = \int_{-\infty}^{\infty} h(\tau)s(t - \tau, \Delta)d\tau$, draw a rough plot of $\varphi(t, \Delta) = h(t) * s(t, \Delta)$ against t.*

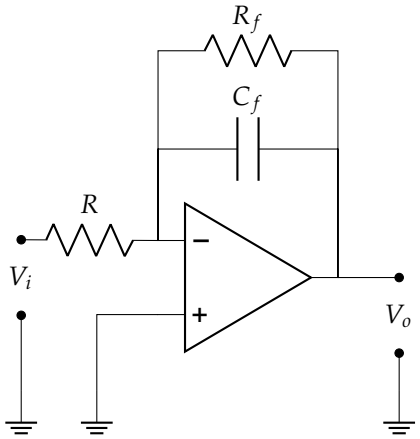


Figure 6.1: Integrator

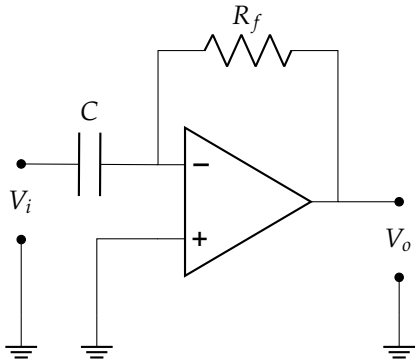
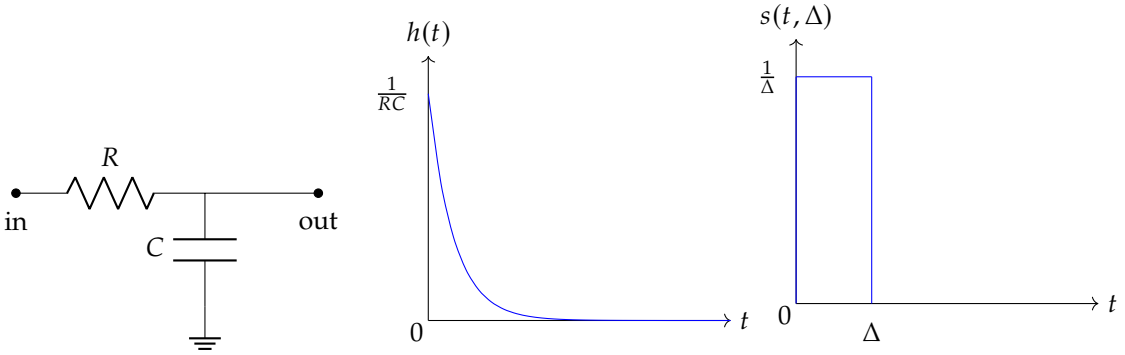


Figure 6.2: Differentiator



(a) RC filter (b) Impulse response of the RC filter (c) Rectangular pulse $s(t, \Delta)$

Figure 6.3: RC filter and related waveforms

Task 5. *Note that the pulse illustrated in Figure 6.3c can be generated as the difference of two unit step functions, given as*

$$s(t, \Delta) = \frac{u(t) - u(t - \Delta)}{\Delta}.$$

We can write the unit step response of the RC filter illustrated in Figure 6.3a as $\hat{h}(t) = \left(1 - e^{-\frac{t}{RC}}\right) u(t)$. *Using the above facts and assuming that the filter is linear and time-invariant, derive the filter output* $\varphi(t, \Delta)$ *obtained when* $s(t, \Delta)$ *is given as the input.*

Task 6. *Find* $\phi(\Delta) = \max_t \varphi(t, \Delta)$ *and draw a rough plot of* $\phi(\Delta)$ *against* Δ.

Task 7. *Show that* $\lim_{\Delta \to 0} \phi(\Delta) = \frac{1}{RC}$.

6.3 Synthesis by Integration

Task 8. *Construct the integrator illustrated in Figure 6.1 using the NE5532 op-amp IC with* $R = 2.2\ k\Omega$, $R_f = 100\ k\Omega$ *and* $C_f = 100\ nF$. *Connect a* $10\ k\Omega$ *load resistor to the output. Provide a dual supply of* $\pm 12\ V$ *using the DC power supply.*

Task 9. *Provide a* $200\ Hz, 1\ V$ *peak-to-peak triangle wave with a* $0\ V$ *DC offset to the input of the integrator using the function generator. Observe both the input and the output on the same screen using the oscilloscope. Draw the waveforms in the space provided.*

Task 10. *Theoretically justify the observations made under Task 9.*

Task 11. *Decide and provide an appropriate waveform to the integrator, in order to generate a triangle waveform with a mean of* $0\ V$ *and a frequency of* $200\ Hz$ *at the output (any amplitude is acceptable). Draw both the input and the output waveforms in the space provided. Specify all the properties and their corresponding values that you have configured on the function generator in order obtain the above waveform.*

6.4 Synthesis by Differentiation

Task 12. *Using the NE5532 op-amp, and taking* $C = 100\ nF$ *and* $R_f = 1\ k\Omega$, *construct the circuit illustrated in Figure 6.2. Provide a* $\pm 15\ V$ *supply voltage to the op-amp using the DC power supply.*

Task 13. *Provide a* $200\ Hz, 5\ V$ *peak-to-peak square wave with a* 50% *duty cycle and a* $2.5\ V$ *offset to the input of the differentiator. Draw both the input and the output waveforms in the space provided.*

Task 14. *Theoretically justify the observations made under Task 13.*

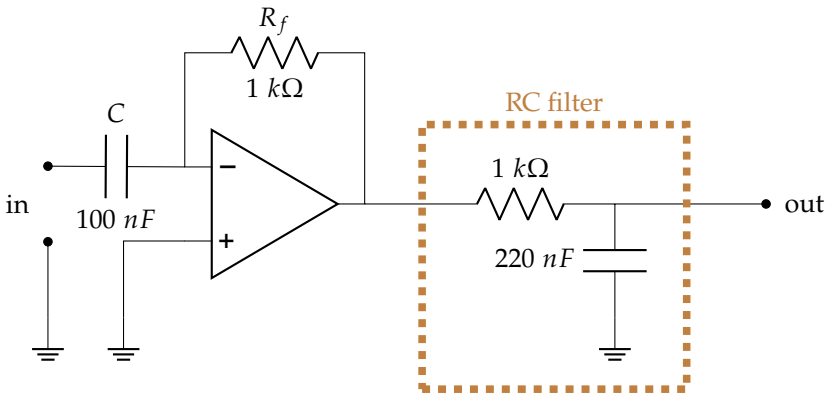


Figure 6.4: RC filter attached to the differentiator output

Task 15. *Extend the differentiator by attaching an RC filter to the differentiator output as shown in Figure 6.4.*

Task 16. *While providing the input mentioned in Task 13 to the differentiator, observe the input and the output of the RC filter and draw the waveforms (to the same scale) in the space provided.*

Task 17. *Comment on the agreement between the observed RC filter output and the theoretical curve obtained in Task 4.*

Task 18. *Disconnect the differentiator output from the filter and provide a square waveform of* $200Hz$ *and constant area, with the properties mentioned in Table 6.1 to the input of the filter. Observe both the input and the output using the oscilloscope. Measure and note down the pulse duration and the peak voltage of the filter output corresponding to each case.*

Pulse width	Amplitude	Offset	Pulse duration	Peak output voltage
10%	0.5 V	0.25 V		
5%	1 V	0.5 V		
2%	2.5 V	1.25 V		
1%	5 V	2.5 V		

Table 6.1: RC filter response for a rectangular pulse

Task 19. *Plot the peak output voltage against the pulse duration recorded in Table 6.1 and comment on the agreement with the theoretical results obtained in Task 6 and Task 7.*

♣ The End ♣

Part III

Electronics

Workshop 1: Building a Simple Audio System using its Building Blocks

Objective: To design and implement a simple audio amplifier using common electronic components and verify its operation

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

1. Implement voltage buffers using bipolar junction transistors
2. Design voltage amplifiers with controlled gain using operational amplifiers
3. Implement active low pass and active high pass filters using operational amplifiers
4. Explain the relationship between tones and frequency components of an audio signal

Equipment Required:

1. Audio Source (PC, Smartphone etc.)
2. Digital Oscilloscope
3. Signal Generator
4. DC power supply
5. Digital multi-meter
6. Breadboard and wires

Components Required:

1. Electronic Module: Pre-amplifier, Tone-Controller and Power Amplifier
2. 3.5mm Audio Jack (2 Nos.)
3. Resistors - 15kΩ (1 Nos.), 18kΩ(1 Nos), 22kΩ (1 Nos.)
4. 4Ω 3W Speaker
5. Jumper clips (4 Nos.)

1.1 Introduction

An amplifier is an active electronic device designed to amplify the voltage/current/power of a signal. In this experiment, you will build a simple audio amplifier and in the process, you will learn about voltage buffers, voltage amplifiers, active filters and power amplifiers. As shown in figure 1.1, the block diagram of the audio system, the original audio signal from the audio source is first fed into a *pre-amplifier*. Audio sources such as microphones typically produce a weak AC signal. The purpose of the pre-amplifier is to amplify this weak AC signal into a more noise-tolerant output signal which is strong enough to process further. In order to mitigate issues related to source loading, the pre-amplifier incorporates a buffer, thereby making it independent of the signal source. After the pre-amplifier, the signal is passed through a *tone controller*, which allows one to increase or diminish the effects of different tones, such as bass and treble, in the audio signal. Then, as the final stage of the audio amplifier, we have the *power amplifier*. The power amplifier amplifies the power of the processed input audio signal to a level that is high enough to drive speakers or headphones.

Figure 1.1: Audio Amplifier System - Block Diagram

In this experiment, you will find an electronic module consisting of **three** stages among your set of components.

- Stage 1: Pre-amplifier - Implemented using a NE5532 dual low-noise operational amplifier IC and a C828 NPN Silicon Epitaxial Planar transistor (*buffer*)
  Note that; *here the pre-amplifier is designed with both inverting and non-inverting configurations for demonstration purposes only. P1 and P2 switches enable separate analysis of each configuration. In real systems, the pre-amplifier is implemented as a non-inverting amplifier.*

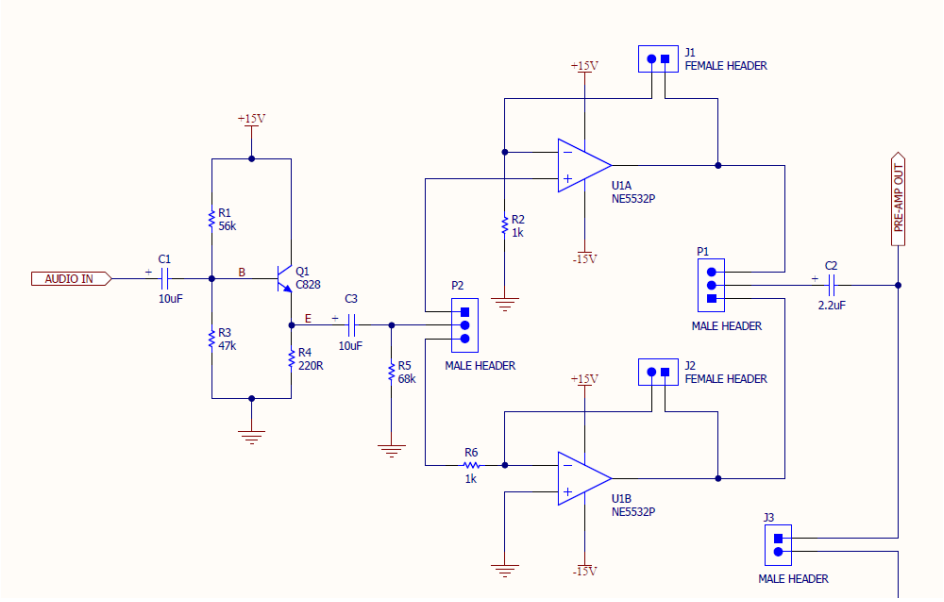


Figure 1.2: Schematic diagram of stage 1 of the Electronic Module

- Stage 2: Tone Controller - Implemented using a NE5532 dual low-noise operational amplifier IC

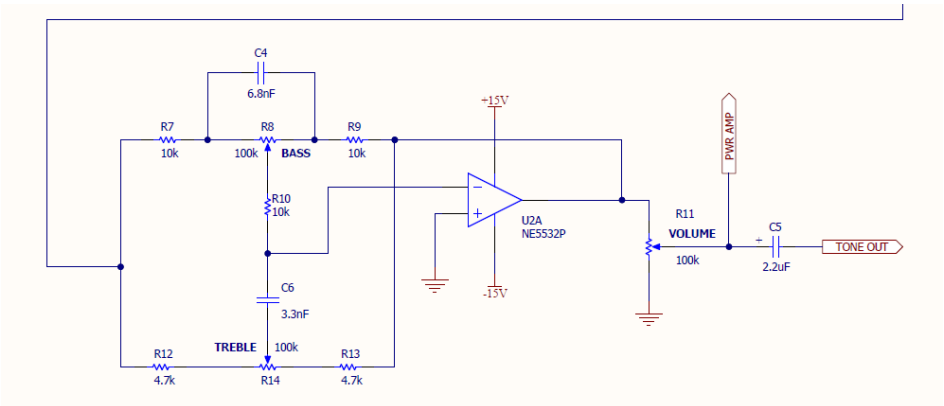


Figure 1.3: Schematic diagram of stage 2 of the Electronic Module

- Stage 3: Power Amplifier - Implemented using a TDA2003 power amplifier IC

1.2 Pre-amplifier

The schematic diagram of the pre-amplifier is shown in figure 1.4. Compare the schematics of the pre-amplifier (figure 1.4) and Stage 1 of the Electronic Module (figure 1.2) to identify the roles of J1, J2 and J3 in the module.

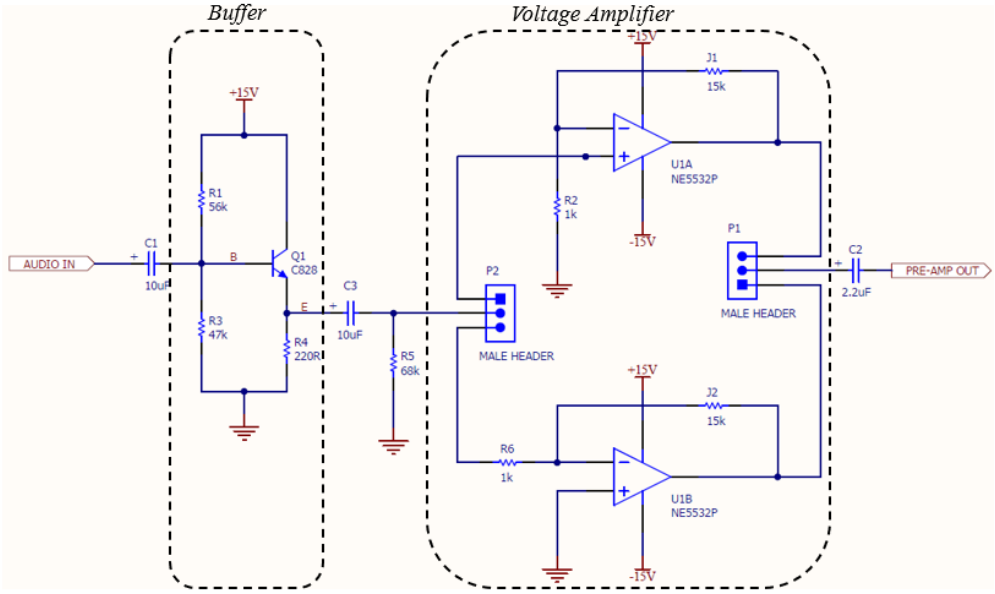


Figure 1.4: Schematic diagram of the pre-amplifier

Task 1. *Consider the buffer of the pre-amplifier (figure 1.4). Name the transistor configuration used and briefly explain its application.*

Task 2. *Give power to the electronic module using the DC power supply. Set the voltage and current limits for both channels in the DC power supply as specified below.*

- *Voltage Limit: 15 V, Current Limit: 0.4 A*

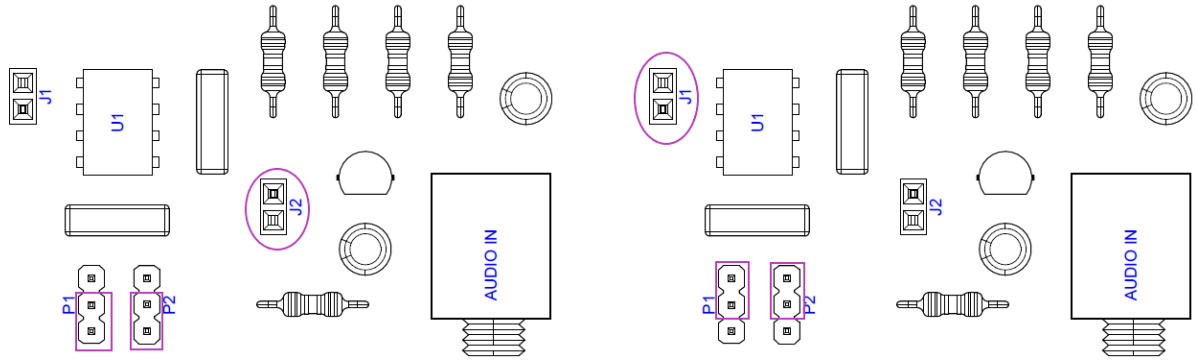
Note that; If the current goes beyond the specified value, inform an instructor immediately.

Task 3. *Configure a 1kHz 1Vpp Sinosoidal waveform on the signal generator, give that waveform as the input signal to the powered-up electronic module. Connect two Jumper clips (to P1 and P2) and connect a 15kΩ resistor (J2) as shown in the configure 1 of the assembly drawing of the electronic module (figure 1.5a). Observe the output of the pre-amplifier using the oscilloscope and plot both input and output wave forms in the same graph.*

Task 4. *Is the pre-amplifier designed as an inverting amplifier or a non-inverting amplifier? (considering the input waveform and the output waveform observed in task 3).*

Task 5. *Calculate the theoretical voltage gain(according to the values given in the schematic diagram(figure 1.4)) and the practically observed voltage gain(considering the input waveform and the output waveform observed in task 3) of the pre-amplifier.*

Task 6. *Are these two values the same? Explain the reasons for your answer.*



(a) Configure 1 (b) Configure 2

Figure 1.5: Assembly Drawing of the Pre-amplifier of electronic module

Task 7. *Choose a resistor value so that the amplifier has a gain of eighteen (18). Connect the resistor to J2. Observe the output of the pre-amplifier and plot both input and output wave forms in the same graph. Calculate the practically observed voltage gain.*

Task 8. *Connect two Jumper clips (to P1 and P2) and connect a 15kΩ resistor (J1) as shown in the configure 2 (figure 1.5b). Observe the output of the pre-amplifier and plot both input and output wave forms in the same graph.*

Task 9. *Calculate the practically observed voltage gain of the pre-amplifier. (considering the input waveform and the output waveform observed in task 8)*

Task 10. *Observe whether there is any difference between the answers obtained in task 4 and task 8. If yes, give reasons for this.*

1.3 Tone Controller

The schematic diagram of the tone controller is shown in figure 1.6.

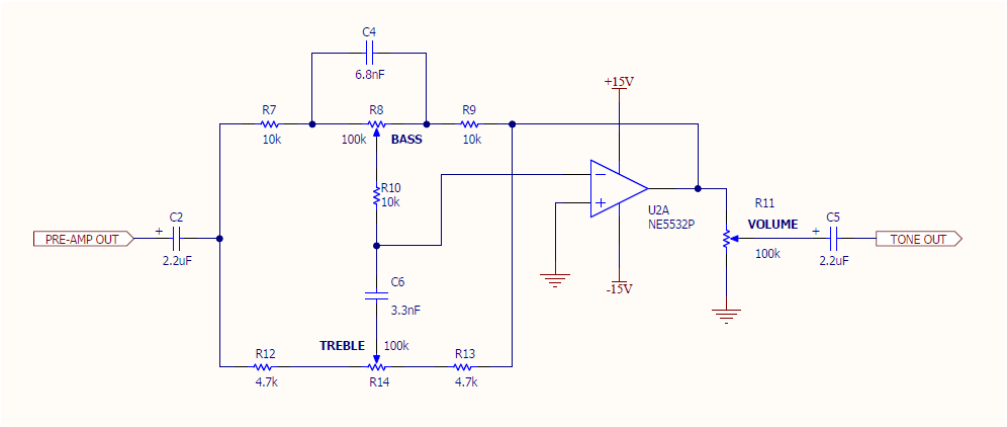


Figure 1.6: Schematic diagram of the tone controller

Task 11. *Connect a jumper clip to the J3 (see figure 1.2). Set both the bass potentiometer, the treble potentiometer and the volume potentiometer to their mid values.*

Task 12. *Configure a 100Hz 1Vpp Sinosoidal waveform on the signal generator, give that waveform as the input signal to the powered-up electronic module. Observe the output of the tone controller using the oscilloscope and adjust the volume potentiometer so that the tone controller output is not clipped.*

Task 13. *Change the value of the bass potentiometer and comment on its effect on the tone controller output.*

Task 14. *Set the bass potentiometer to back to its mid value.*

Task 15. *Change the value of the treble potentiometer and comment on its effect on the tone controller output.*

Task 16. *Set the treble potentiometer to back to its mid value.*

Task 17. *Configure a 10kHz 1Vpp Sinosoidal waveform on the signal generator, give that waveform as the input signal to the powered-up pre-amplifier. Observe the output of the tone controller using the oscilloscope and adjust the volume potentiometer so that the tone controller output is not clipped.*

Task 18. *Change the value of the bass potentiometer and comment on its effect on the tone controller output.*

Task 19. *Set the bass potentiometer to back to its mid value.*

Task 20. *Change the value of the treble potentiometer and comment on its effect on the tone controller output.*

Task 21. *Explain the purpose of the treble and bass potentiometers.*

1.4 Power Amplifier

The schematic diagram of the power amplifier is shown in figure 1.7.

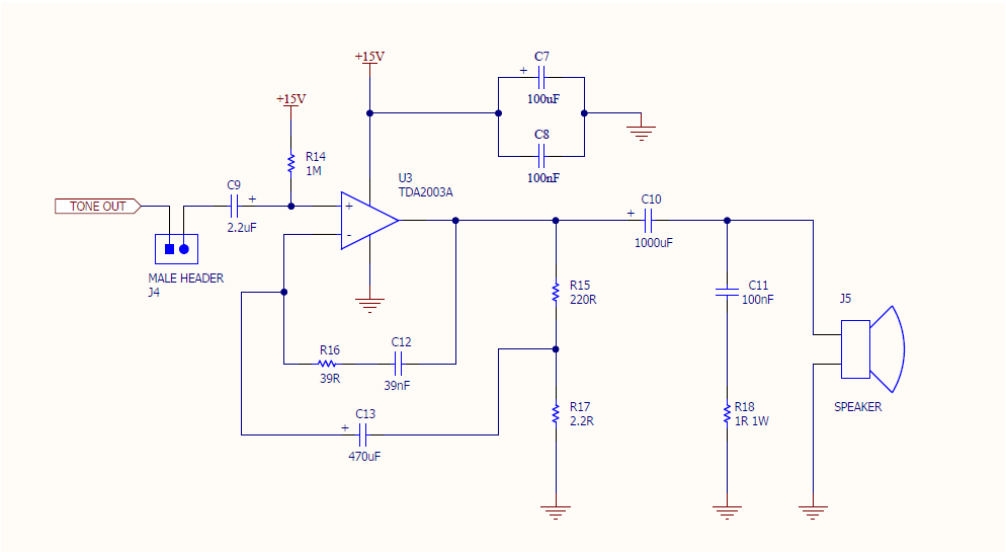


Figure 1.7: Schematic diagram of the power amplifier

Task 22. *Set the tone controller potentiometers to their mid values and connect a jumper clip to J4.*

Task 23. *Disconnect the signal generator from the pre-amplifier and connect the 3.5mm mono audio jack to the pre-amplifier input. Connect the speaker to J5 (figure 1.7)*

Task 24. *Connect the 3.5mm mono audio jack to your smartphone or PC and play a music track with noticeable bass and treble tones.*

Task 25. *Tune the volume control potentiometer so that the output from the speaker is minimally distorted. (A certain amount of distortion is unavoidable due to the wiring, heating of components etc.)*

Task 26. *Change the values of the tone controller potentiometers and verify the proper operation of the audio amplifier.*

♣ The End ♣

Workshop 2: Building and Taking Measurements of Operational Amplifier Circuits

Objective: To design and implement commonly used operational amplifier circuits

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

1. Design and implement the following operational amplifier application circuits:
   - Inverting voltage amplifier
   - Non-inverting voltage amplifier
   - Voltage comparator
   - Differentiator

Equipment Required:

1. Digital Oscilloscope
2. Signal Generator
3. DC power supply
4. Digital multi-meter
5. Breadboard and wires

Components Required:

1. NE5532 (1 No.)
2. Resistors - 1kΩ (3 Nos.), 10kΩ(3 Nos), 100kΩ (2 Nos.)
3. Capacitors - 1nF (1 No.)
4. 100K potentiometer (1 No.)

2.1 Introduction

Operational amplifiers, often referred to as *"op-amps"*, are high-gain voltage amplifiers with a differential input and , usually a signle-ended output. There are operational amplifiers with differential output as well. Such operational amplifiers are referred to as *"fully differential operational amplifiers"*. In this experiment, we will be focusing on the more commonly used general op-amps. The symbol and the pin-out for a general operational amplifier is shown in figure 2.1, where:

- V_+ is the non-inverting input
- V_- is the inverting input
- V_{s+} is the positive power supply
- V_{s-} is the negative power supply
- V_o is the output

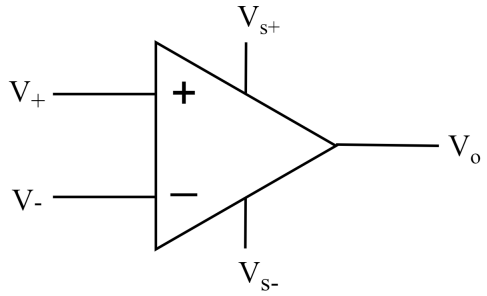


Figure 2.1: Symbol and pin-out of a general operational amplifier

The most noteworthy characteristics of operational amplifiers include very high open-loop voltage gain, very high input impedance and very low output impedance. Due to these special characteristics, the use of operational amplifiers in electronic circuits have become extremely popular nowadays. In this experiment, we will be building and analyzing four basic operational amplifier application circuits.

2.2 Pre-Lab

Task 1. *What is the typical range of values for the open-loop gain of practical operational amplifiers?*

Task 2. *What are the two golden rules which are used to explain the functionality of operational amplifiers? (These two rules explain the governing principles of ideal operational amplifiers. However, these rules are quite useful when analyzing practical operational amplifier circuits.)*

Task 3. *Unlike ideal operational amplifiers, practical operational amplifiers have a finite bandwidth. Explain what is meant by the bandwidth of an operational amplifier.*

Task 4. *What are the undesirable effects or limitations introduced by the finite bandwidth of practical operational amplifiers?*

Task 5. *In operational amplifier terminology, What is meant by "Saturation"?*

In this experiment, you will implement the following commonly used circuits which uses operational amplifiers.

- Inverting voltage amplifier
- Non-inverting voltage amplifier
- Voltage comparator
- Differentiator

2.3 Inverting Voltage Amplifier

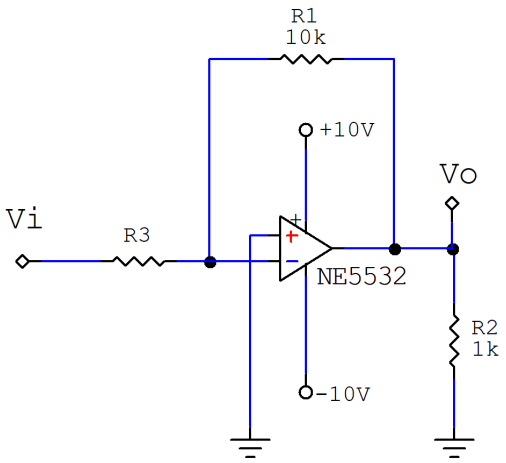


Figure 2.2: Schematic diagram of the inverting amplifier

Task 6. *Derive an equation for the voltage gain of the amplifier in terms of R1 and R3.*

Task 7. *Calculate the value suitable for R3 in order to achieve a gain of (-10).*

Task 8. *Construct the circuit show in figure 2.2. Use the value calculated in task 7 for R3.*

Task 9. *Configure a 10kHz 2Vpp Sinosoidal waveform on the signal generator, give that waveform as the input signal to the powered-up inverting amplifier. Observe both the input and the output of the amplifier using the oscilloscope and plot the observed waveforms on top of each other. (NOTE: Make sure you label each waveform)*

Task 10. *Comment on any differences in the observed output, with respect to the output observed if an ideal operational amplifier was used. (Hint: Calculate the expected output amplitude and measure the observed output amplitude.)*

Task 11. *Suggest any changes that can be made (in the components or the power supply) in order to make the output waveform similar to that of an ideal operational amplifier, without reducing the gain.*

2.4 Non-inverting Voltage Amplifier

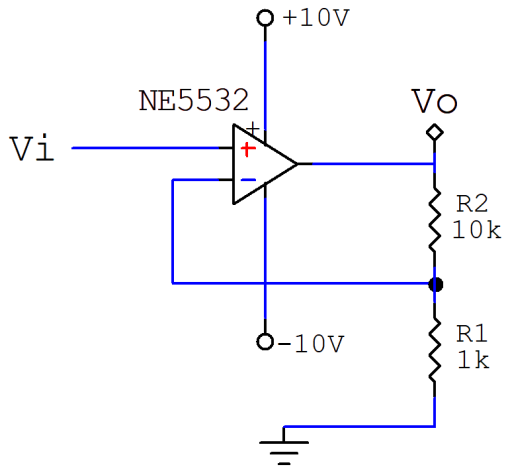


Figure 2.3: Schematic diagram of the non-inverting amplifier

Task 12. *Construct the circuit show in figure 2.3.*

Task 13. *Calculate the theoretical voltage gain.*

Task 14. *Configure a 1kHz 200mVpp Sinosoidal waveform on the signal generator, give that waveform as the input signal to the powered-up non-inverting amplifier. Observe both the input and the output of the amplifier using the oscilloscope and plot the observed waveforms on top of each other. (NOTE: Make sure you label each waveform)*

Task 15. *Calculate the practical voltage gain (using the observed input and output waveforms).*

2.5 Voltage Comparator

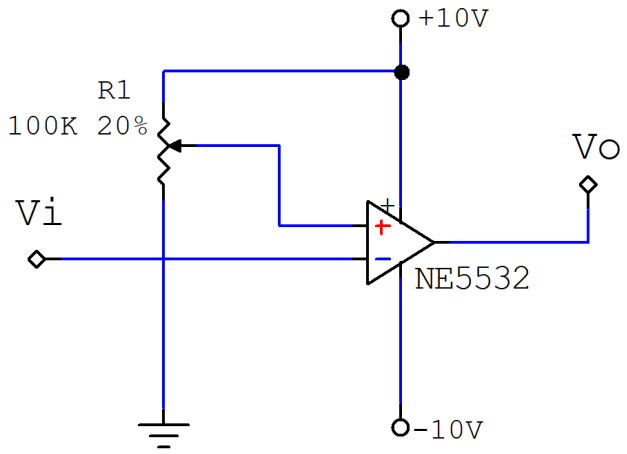


Figure 2.4: Schematic diagram of the comparator

Task 16. *Construct the circuit show in figure 2.4.*

Task 17. *Adjust the potentiometer and set the voltage at the non-inverting terminal to 2V.*

Task 18. *Configure a 1kHz 6Vpp Sinosoidal waveform on the signal generator, give that waveform as the input signal to the powered-up comparator. Observe both the input and the output of the amplifier using the oscilloscope.*

Task 19. *Adjust the vertical scales of the oscilloscope channels so that both channels have the **same** vertical scale.*

Task 20. *Enable voltage cursors of one channel and place a cursor at 2V.*

Task 21. *Plot the observed waveforms and the voltage cursor (set at 2V) on top of each other. Make sure you label each waveform and the cursor.*

2.6 Differentiator

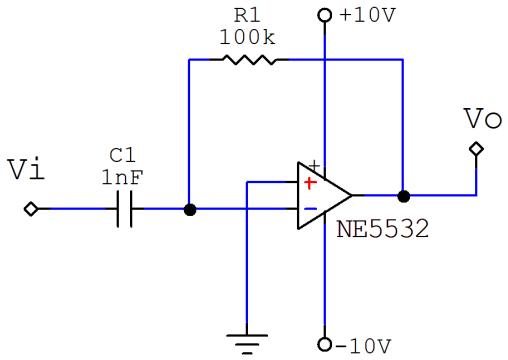


Figure 2.5: Schematic diagram of the differentiator

Task 22. *Construct the circuit show in figure 2.5.*

Task 23. *Configure a 1kHz 2Vpp Sinosoidal waveform on the signal generator, give that waveform as the input signal to the powered-up differentiator. Observe both the input and the output of the amplifier using the oscilloscope and plot the observed waveforms on top of each other. (NOTE: Make sure you label each waveform)*

Task 24. *Configure a 1kHz 2Vpp Square waveform on the signal generator, give that waveform as the input signal to the powered-up differentiator. Observe both the input and the output of the amplifier using the oscilloscope and plot the observed waveforms on top of each other. (NOTE: Make sure you label each waveform)*

Task 25. *Derive the relationship between V_i and V_o.*

♣ The End ♣

Workshop 3: Simple Zener Regulated DC Power Supply

Objective: To construct a simple Zener regulated DC power supply

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

1. Construct and test a Half Wave Rectifier (HWR)
2. Construct and test a Full Wave Rectifier (FWR)
3. Identify the effect of a capacitive filter on FWR output
4. Construct and test a Zener regulated DC power supply

Equipment Required:

1. 6 V/50 Hz AC source
2. Digital Oscilloscope
3. Analog multimeter
4. Breadboard and wires

Components Required:

1. Diodes: 1N4001 (4 nos.)
2. Zener Diode: 1N4732A
3. Bridge rectifier IC
4. Capacitors: 4.7μF, 220μF
5. Resistors: 10 kΩ, 1kΩ, 560Ω, 180Ω, 100Ω (2 nos.)

3.1 Half Wave Rectifier (HWR)

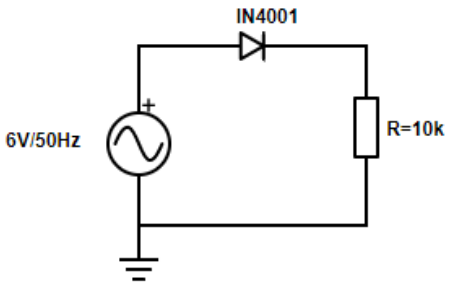


Figure 3.1: Schematic diagram of the HWR

Task 1. *Construct the circuit show in figure 3.1 on the breadboard with a 10 kΩ resistor as the load. Provide an AC input using the 6V/50Hz AC source.*

Task 2. *Observe the input AC voltage and the load voltage from the oscilloscope and draw the wave forms in the same diagram.*

Task 3. *Why do you think full-wave rectification is preferred instead of half-wave rectification?*

3.2 Bridge Rectifier (FWR)

The bridge rectifier in figure 3.2 is a full wave rectifier and converts AC to DC. The AC input is applied across A and B while the DC output is taken across P and Q.

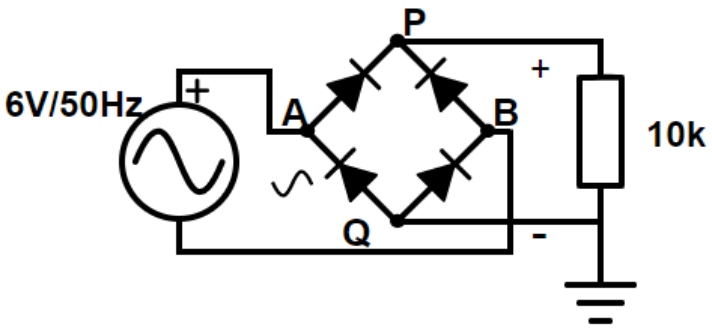


Figure 3.2: Schematic diagram of the FWR

The bridge rectifier is available in IC form as well. The AC inputs of the IC are usually marked with a ~ sign and the DC outputs are marked with their corresponding polarity signs (+ and −).

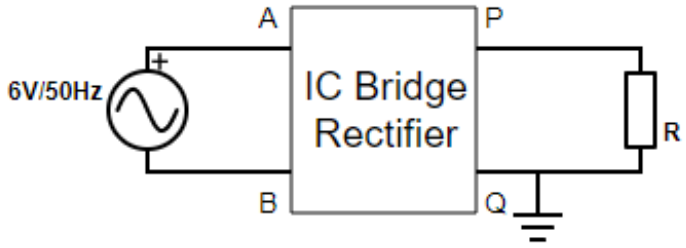


Figure 3.3: Bridge Rectifier IC

Task 4. *Fix the IC on the breadboard and apply a 6 V/50 Hz voltage (from the AC source) across the input terminals AB of the IC.*

Task 5. *Connect a 10 kΩ load resistor across the output of terminals PQ.*

Task 6. *Observe the AC input voltage across AB from the oscilloscope and draw the waveform.*

Task 7. *Observe the DC output voltage across PQ from the oscilloscope and draw the waveform.*

Task 8. *Using the multimeter, measure the DC voltage (V_{DC}) of the output.*

Task 9. *Measure the peak value of the output voltage (V_{max}) from the oscilloscope and estimate V_{DC} (Recall that, for a sinusoid, $V_{DC}=(2/\pi)V_{max}$).*

3.3 Capacitive Filter

The output of the rectifier is a varying DC voltage. It is almost always necessary to have steady DC voltages to get electronic circuits to function properly. Therefore, different smoothing circuits or filter circuits are used to produce a steady DC voltage at the rectifier output. The simplest filter is a capacitor.

Task 10. *Connect the 4.7 μF capacitor across the output of the rectifier as shown in figure 3.4.*

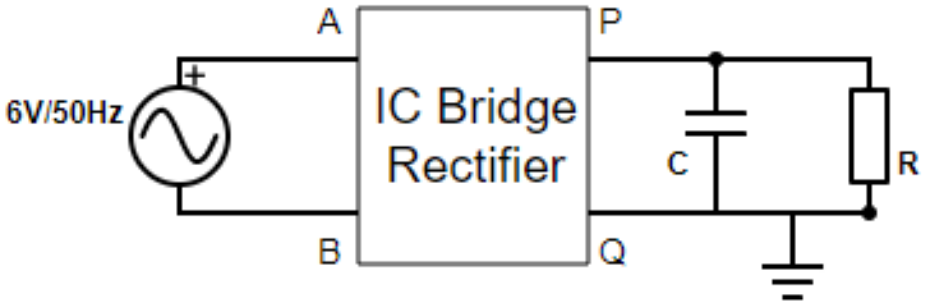


Figure 3.4: FWR with smoothing capacitor

Task 11. *Observe and draw the output voltage across the load resistor from the oscilloscope with a 4.7 μF capacitor fixed as C.*

Task 12. *Replace the 4.7 μF capacitor with C=220 μF capacitor and observe the output voltage waveform from the oscilloscope. Draw the waveform on the same sketch and notice the difference caused by increasing the value of the capacitor.*

Task 13. *Make a comment on your observations of the effect of the capacitive filter.*

3.4 Zener Regulator

The DC output voltage after the capacitor filter is a smooth DC signal. However, it would vary if the load resistor varies. Hence, the DC output voltage has to be stabilized. A simple Zener regulator can produce a reasonably stable DC output.

Connect the Zener diode and a series resistor as shown in figure, such that the output DC voltage is made stable or regulated.

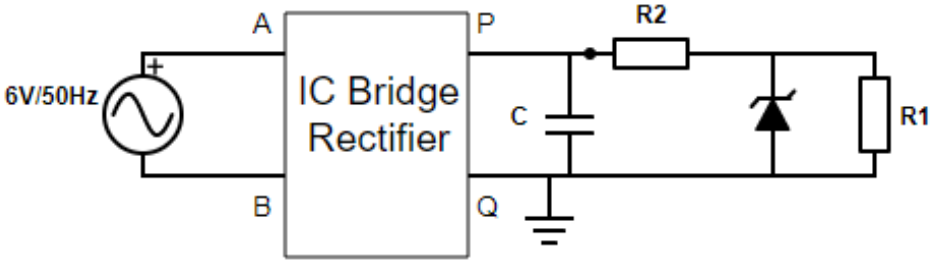


Figure 3.5: FWR with smoothing capacitor and Zener regulator. V_z=4.7 V, C = 220 μF, R_1=10 kΩ, R_2=100 Ω

Task 14. *Observe and record the DC output across the load resistor (V_{out}) from the oscilloscope.*

Task 15. *Change the load resistor R_L to the values mentioned in table 3.1 and record V_{out}.*

$R_L(\Omega)$	$V_{RL}(V)$
1,000	
560	
100	

Table 3.1: Effect of R_L on the regulated output voltage

Task 16. *Explain the behaviour of the DC output voltage as R_L varies.*

♣ The End ♣

Workshop 4: Bipolar Junction Transistor (BJT) Amplifier

Objective: To design a simple transistor amplifier using a BJT

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

1. Test a BJT
2. Build a simple BJT amplifier with a fixed bias current
3. Use the oscilloscope to observe and measure amplifier input and output voltage waveforms
4. Estimate the voltage gain of the amplifier
5. Identify the waveform distortion resulting from over-drive
6. Waveform distortion due to improper bias point

Equipment Required:

1. DC power supply
2. Digital Oscilloscope
3. Signal generator
4. Analog multimeter
5. Breadboard and wires

Components Required:

1. BJT: 2N2222
2. Capacitors: 4.7μF (2 nos.)
3. Resistors: 560 Ω, 10 kΩ, 470 kΩ

4.1 Testing a Bipolar Junction Transistor

Task 1. *Observe the 2N2222 BJT and identify its terminals using the data sheet. Draw the pin-out diagram.*

Task 2. *Is it a pnp or an npn transistor?*

A BJT has two p-n junctions. Base is the common layer for both the junctions. As each junction behaves somewhat similar to a diode, their proper functionality can be tested by following steps similar to checking a diode.

Task 3. *Mention the colour of the multimeter probe (in diode mode) that needs to be connected to the anode of a diode, in-order to check it under forward bias configuration.*

Task 4. *Perform the multimeter test to check whether the transistor is working or not. Use the digital multimeter in diode mode.*

4.2 Building a simple BJT amplifier with fixed bias

Task 5. *On the breadboard, construct the amplifier given in figure 4.1, using the 2N2222 transistor, which has already been tested and found to be working.*

Task 6. *Make the DC measurements to check if the transistor is properly biased for amplification of an AC signal.Take R_1=470 kΩ, R_2=560 Ω, C_1= C_2= 4.7 μF, V_{CC}= +13 V DC.*

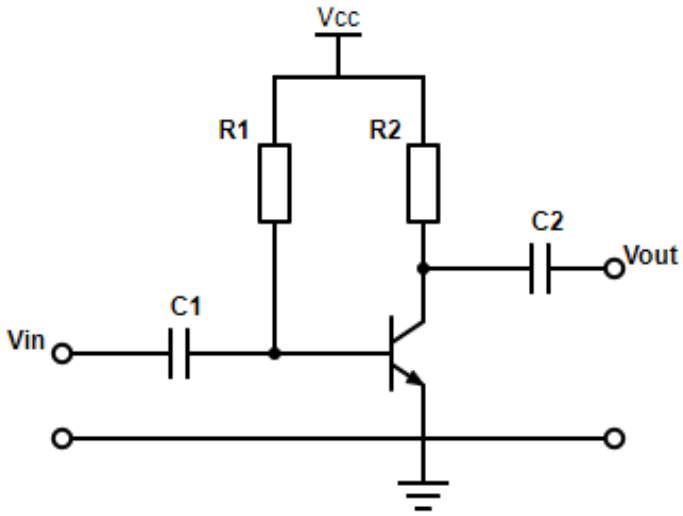


Figure 4.1: Schematic diagram of the BJT amplifier with fixed bias

We may utilize the Ohm's law to estimate the base and collector DC currents by measuring the DC voltages across each resistor using the multimeter.

Task 7. *Calculate the collector current, I_C, by measuring the voltage across R_2. Show your calculations.*

Task 8. *Calculate the base current, I_B, by measuring the voltage across R_1. Show your calculations.*

Task 9. *Measure and record the DC voltage across the collector and the emitter (V_{CE}).*

Generally, the quiescent point is selected such that $V_{CE} \approx \frac{1}{2} V_{CC}$ to allow a maximum swing of the amplified signal without causing a distortion due to saturation.

Task 10. *Considering the V_{CE} value obtained by measurement, comment on the suitability of that value for small signal amplification.*

Task 11. *What is the static or the DC current gain (h_{FE}) of this transistor? Show your calculations.*

4.3 Using the oscilloscope to observe amplifier waveforms

Task 12. *Connect a 10 kΩ resistor to the output of the amplifier circuit.*

Task 13. *Connect the signal generator to the amplifier circuit and apply the small sinusoidal AC signal as specified below. To observe the input and output waveforms, connect the channel 1 and 2 of the oscilloscope to the input and the output, respectively.*

Task 14. *In the signal generator, make adjustments to output a 30 mV (pk-pk), 100 kHz sinusoidal signal.*

Task 15. *Observe both input and output waveforms from the oscilloscope. Observe the amplifier gain and phase.*

Task 16. *What is the phase change introduced by the amplifier?*

Task 17. *Draw the input and the output voltage waveforms.*

4.4 Estimating the voltage gain of the Amplifier

Task 18. *What is the peak to peak input voltage, V_{in}?*

Task 19. *What is the peak to peak output voltage, V_{out}?*

Task 20. *What is the Voltage gain, A_V?*

4.5 Observing the waveform distortion resulting from over-drive

Task 21. *Increase the amplitude of the input signal generator and observe the distortion of the output voltage waveform due to increased drive signal or over-drive.*

Task 22. *Draw the input and output signals.*

4.6 Observing the waveform distortion due to improper bias point

Task 23. *Replace R_1 with a 1 MΩ potentiometer.*

Task 24. *Apply 30 mV peak-to-peak signal as the input and observe the output waveform.*

Task 25. *Reduce the value of R_1 until the output is distorted.*

Task 26. *Draw the output signal in the space provided.*

Task 27. *Increase the value of R_1 until the output is once again distorted.*

Task 28. *Draw the output signal in the space provided.*

Task 29. *Comment on the effect of R_1 on the output voltage waveform.*

♣ The End ♣

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).

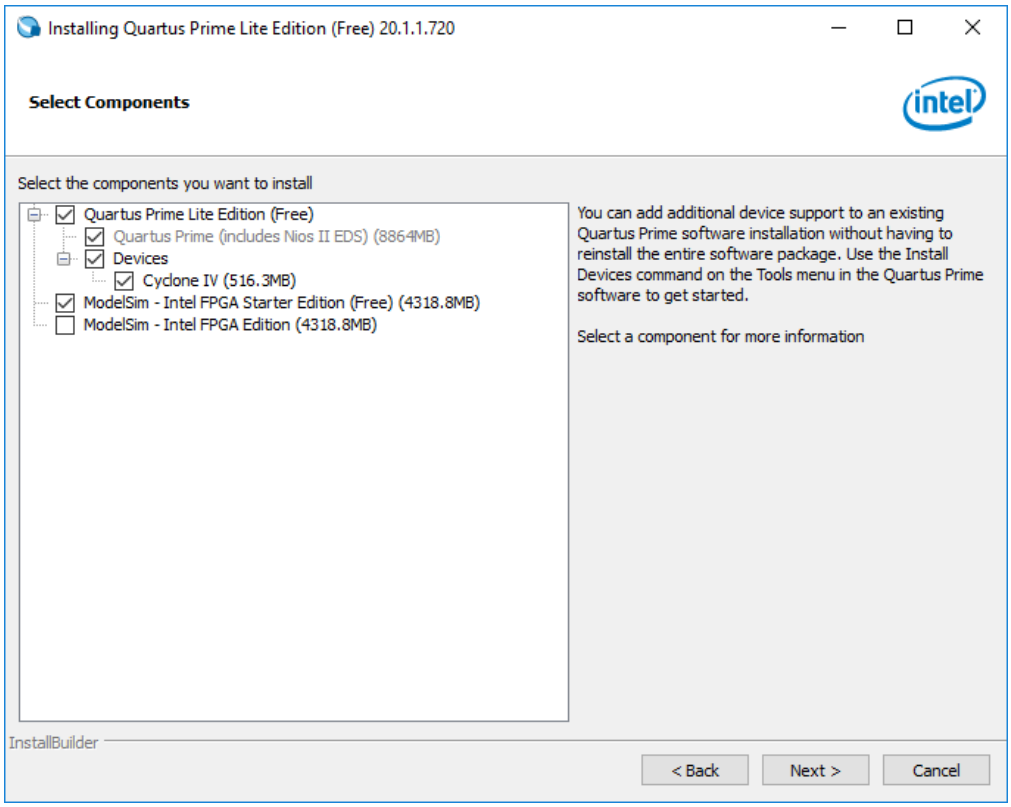


Figure 5.2: Quartus installer: selecting components to install

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.

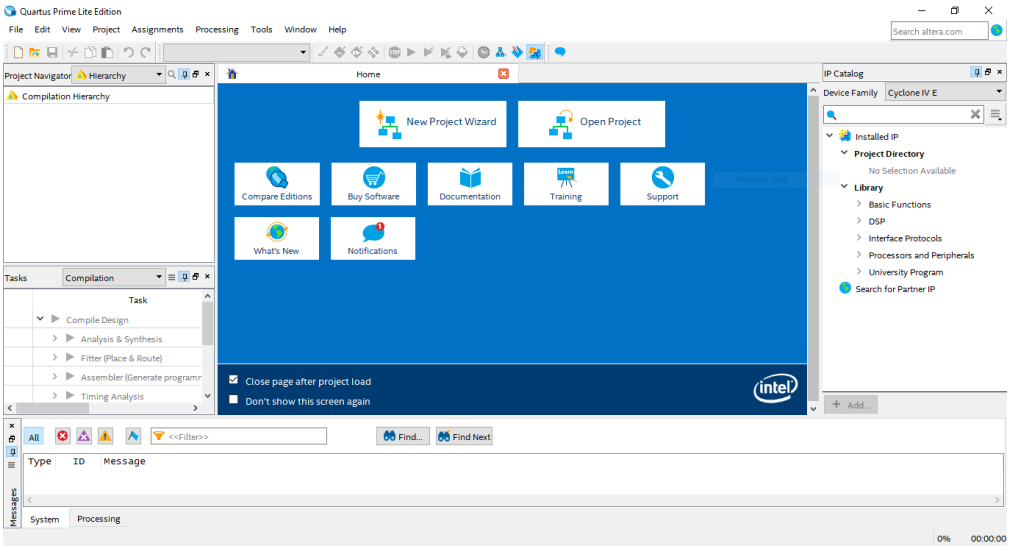


Figure 5.3: Quartus home screen

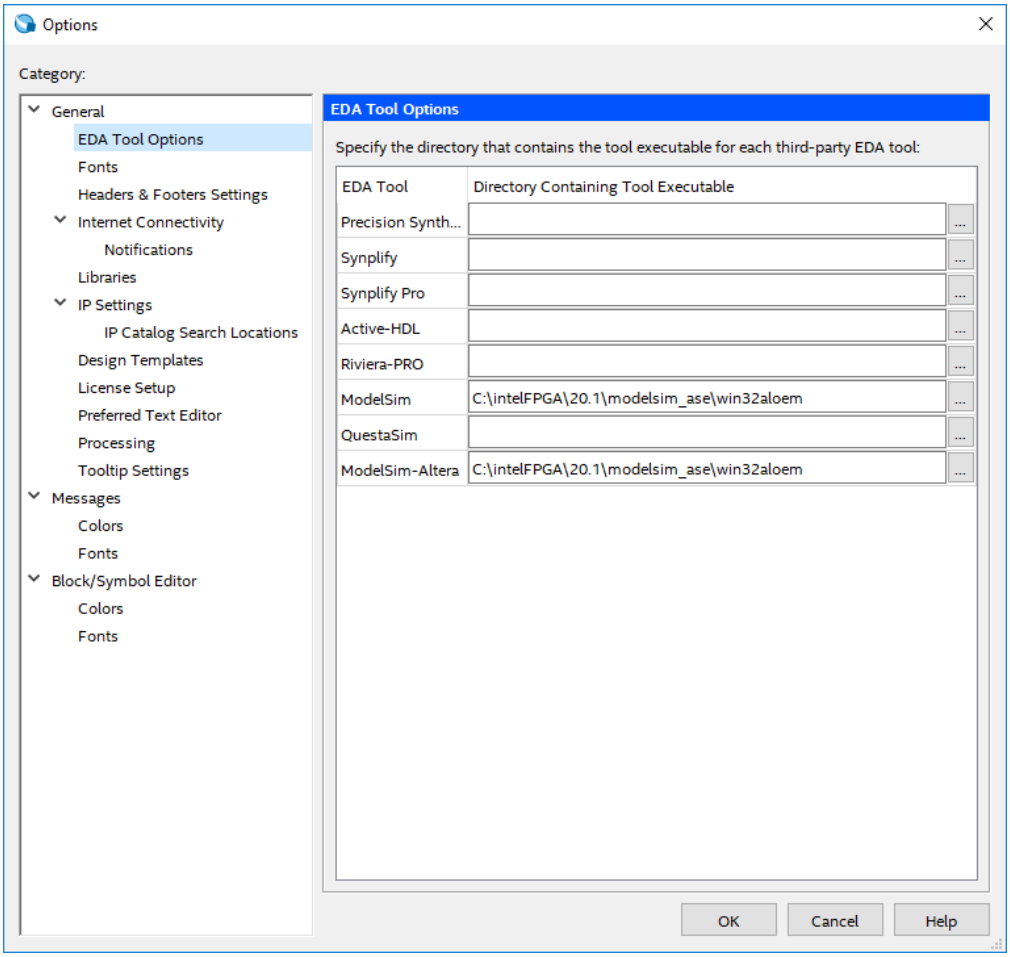


Figure 5.4: Setting ModelSim path

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.

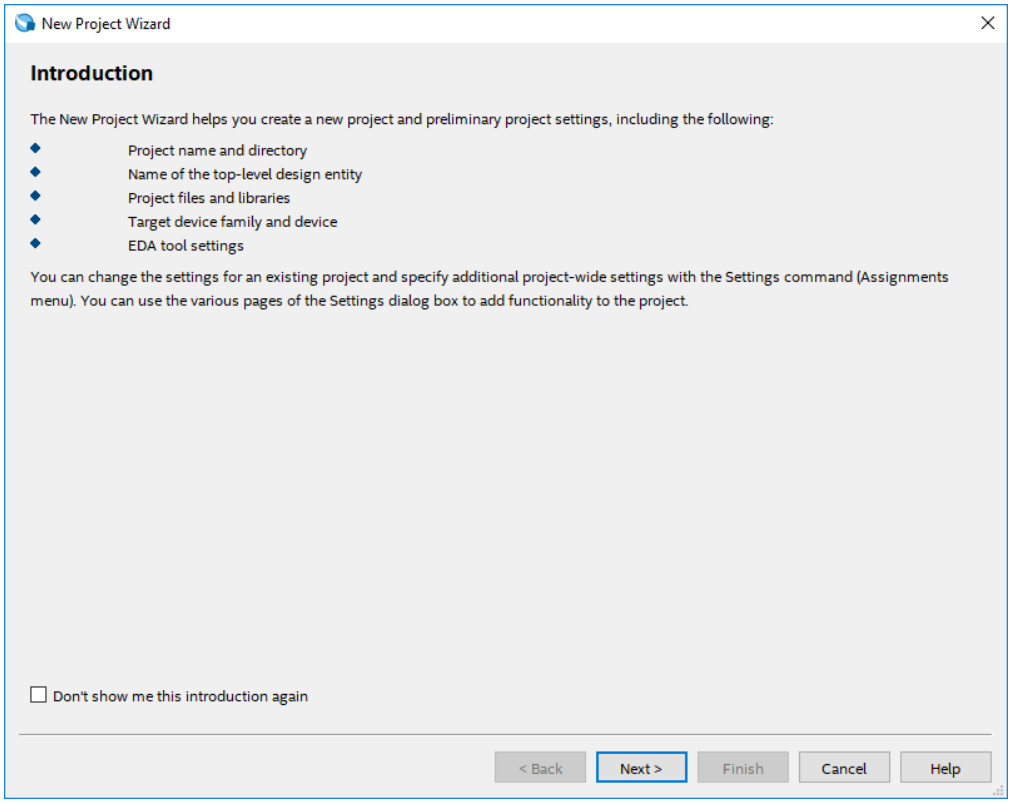


Figure 5.5: New Project Wizard

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.

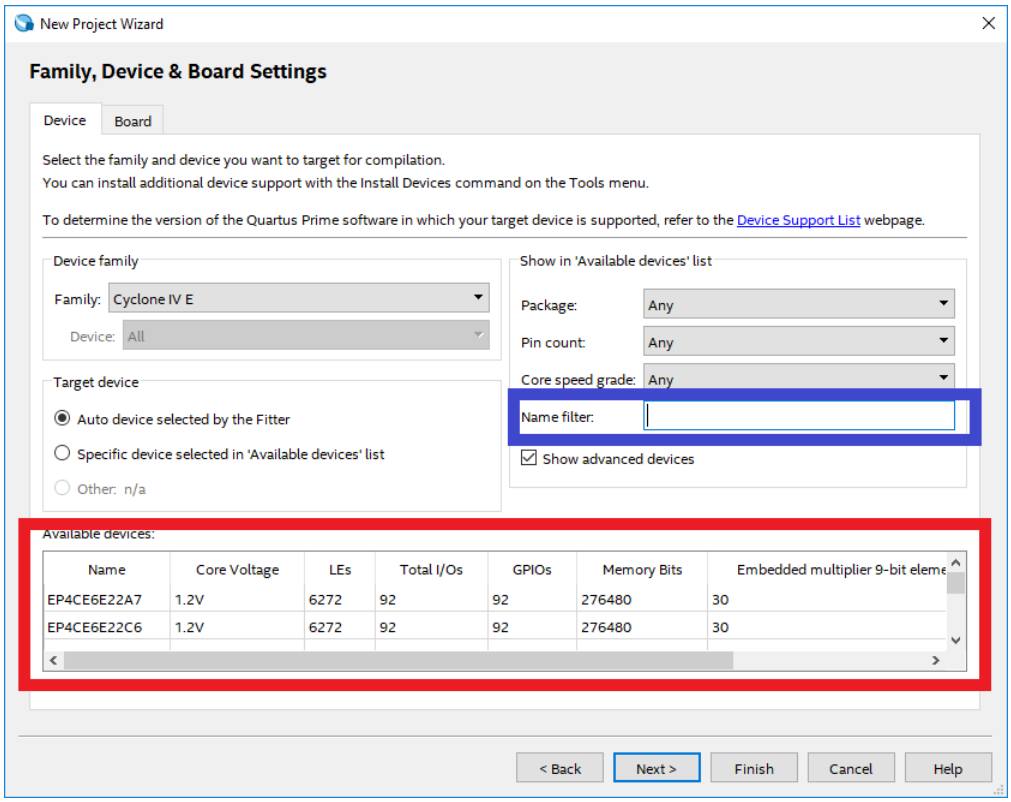


Figure 5.6: Family, device and board selection. Red square: select device type here. Blue square: type a part of the device name to filter the list

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).*

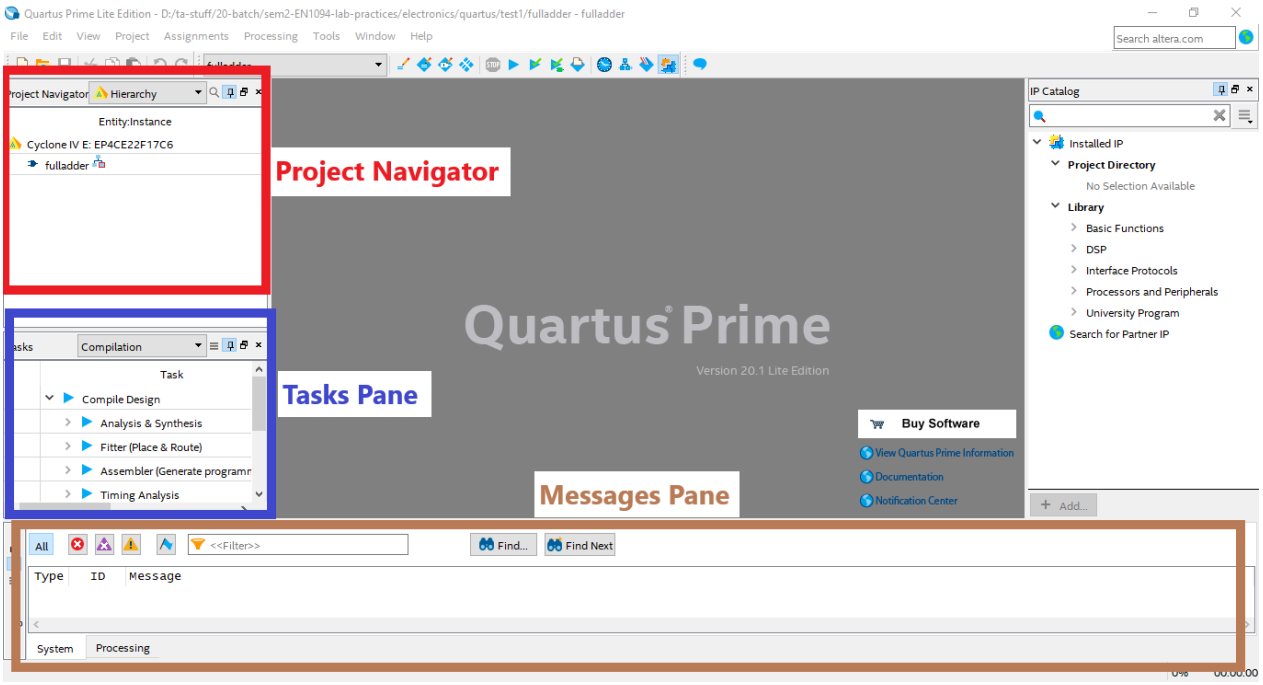


Figure 5.7: Project display. *Project Navigator* displays different files/modules of the project organized in different forms. *Tasks Pane* displays different tasks of the project flow and their status. *Messages Pane* displays messages corresponding to different tools and tasks.

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...**.

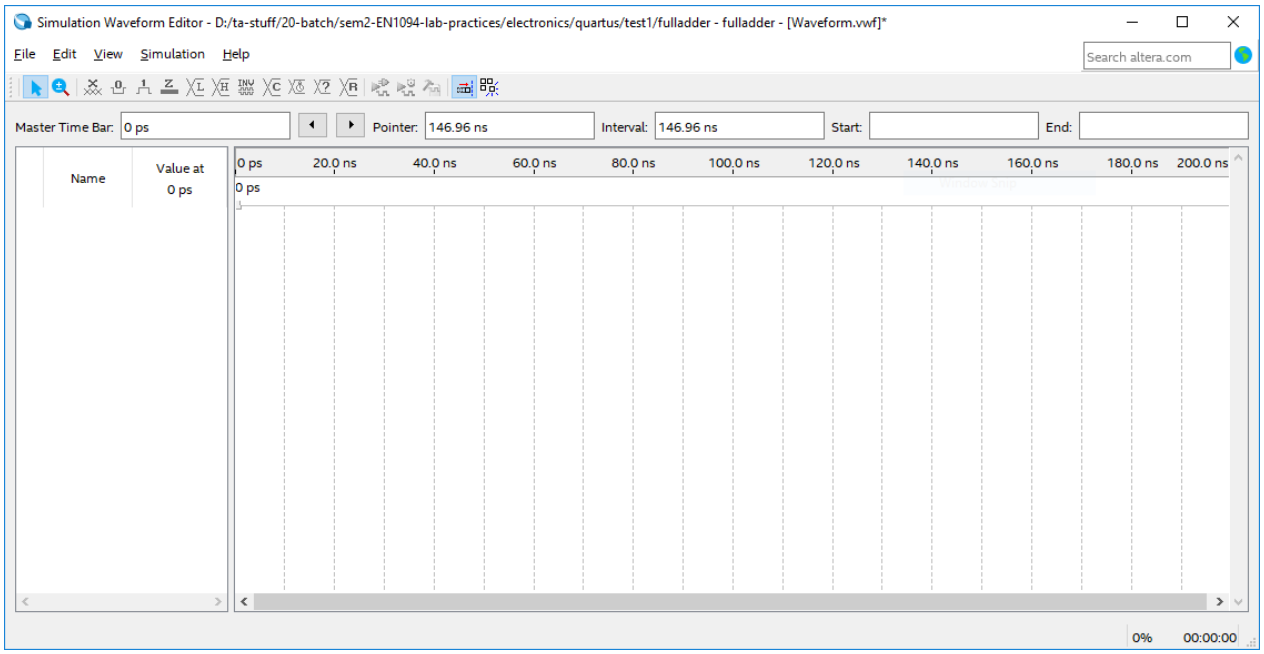


Figure 5.8: Simulation Waveform Editor

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.

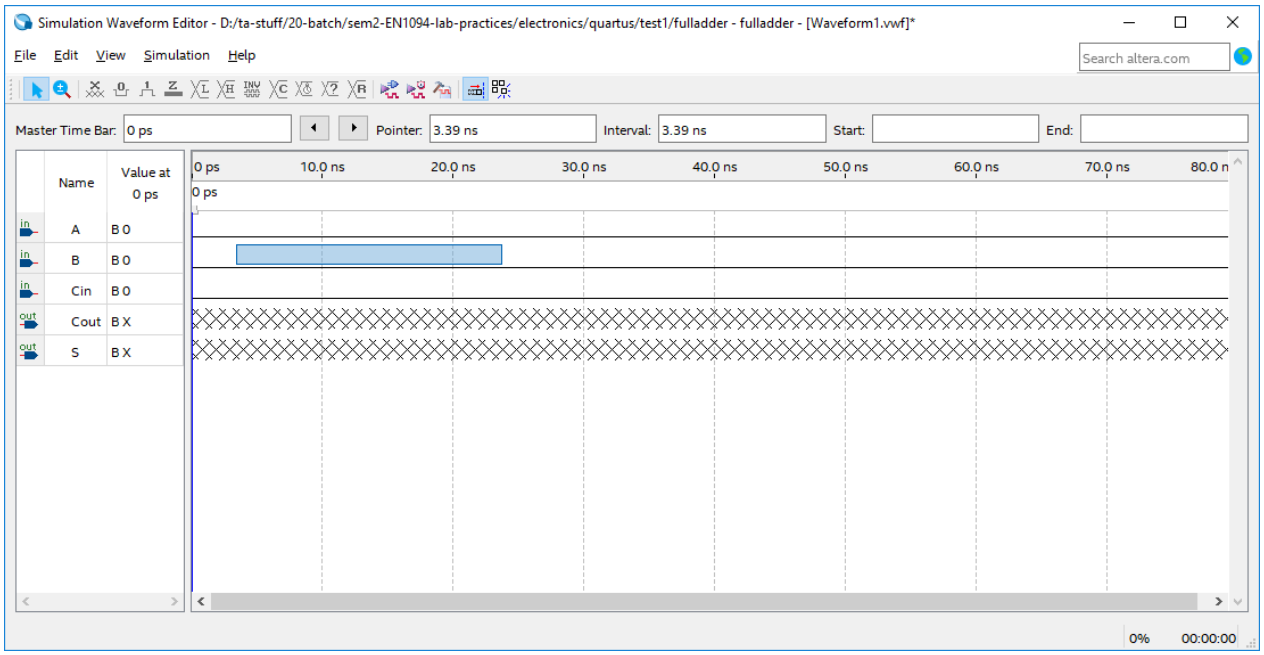
After all the input waveforms have been specified correctly, you can now run the simulation by following the instructions listed below;

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

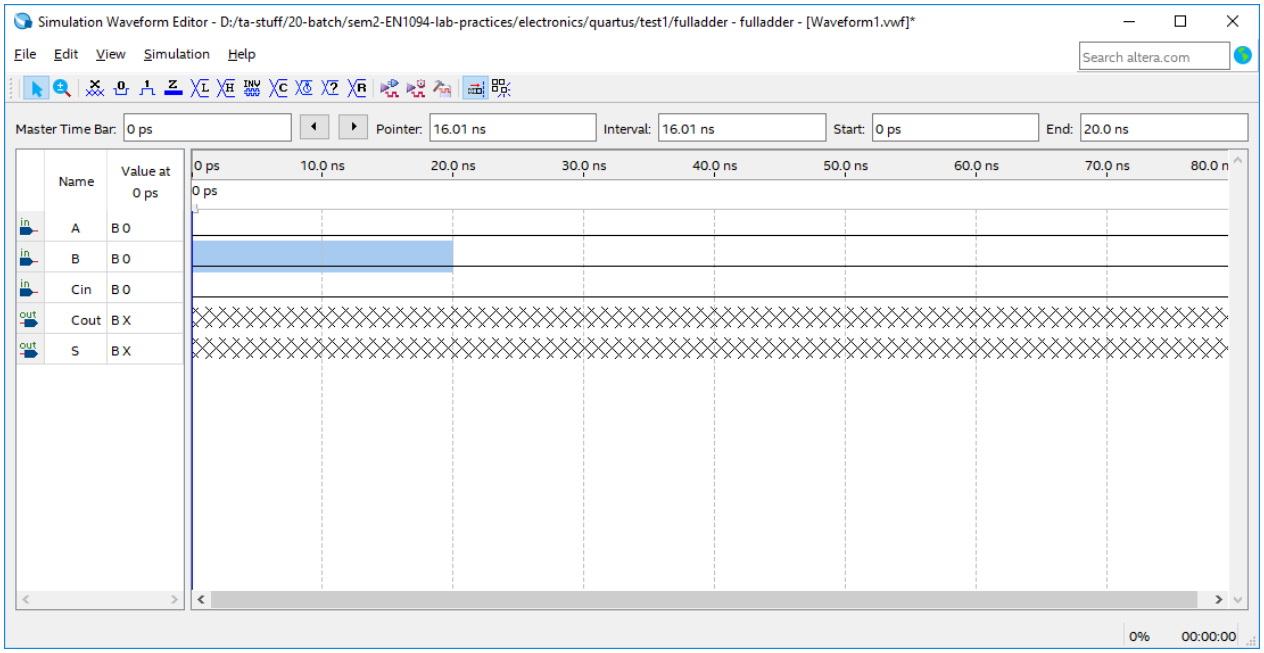
2. If not already selected, select **Verilog** as the **HDL Language**.

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.



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



(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.*

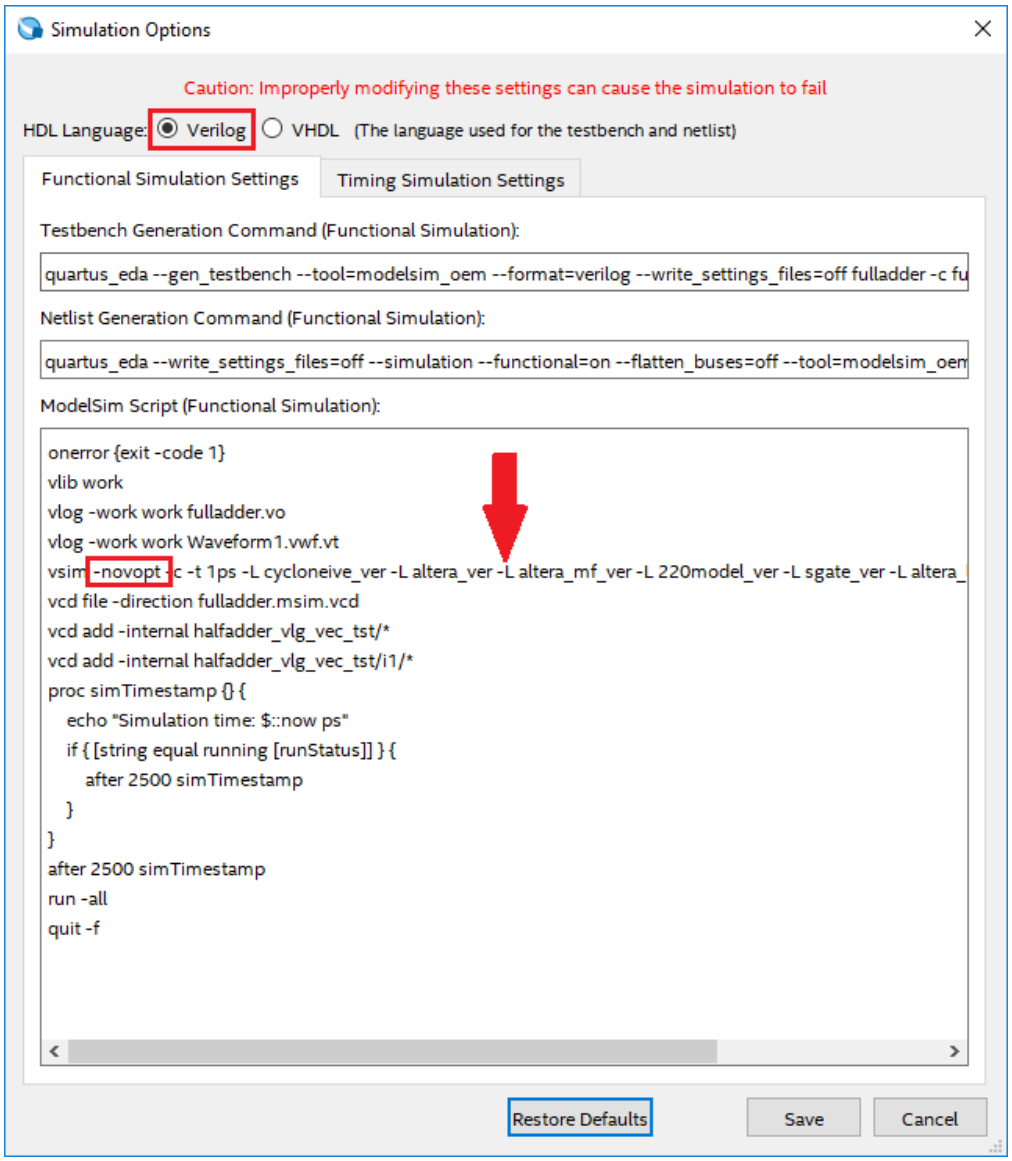


Figure 5.10: Simulation Settings

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)**.

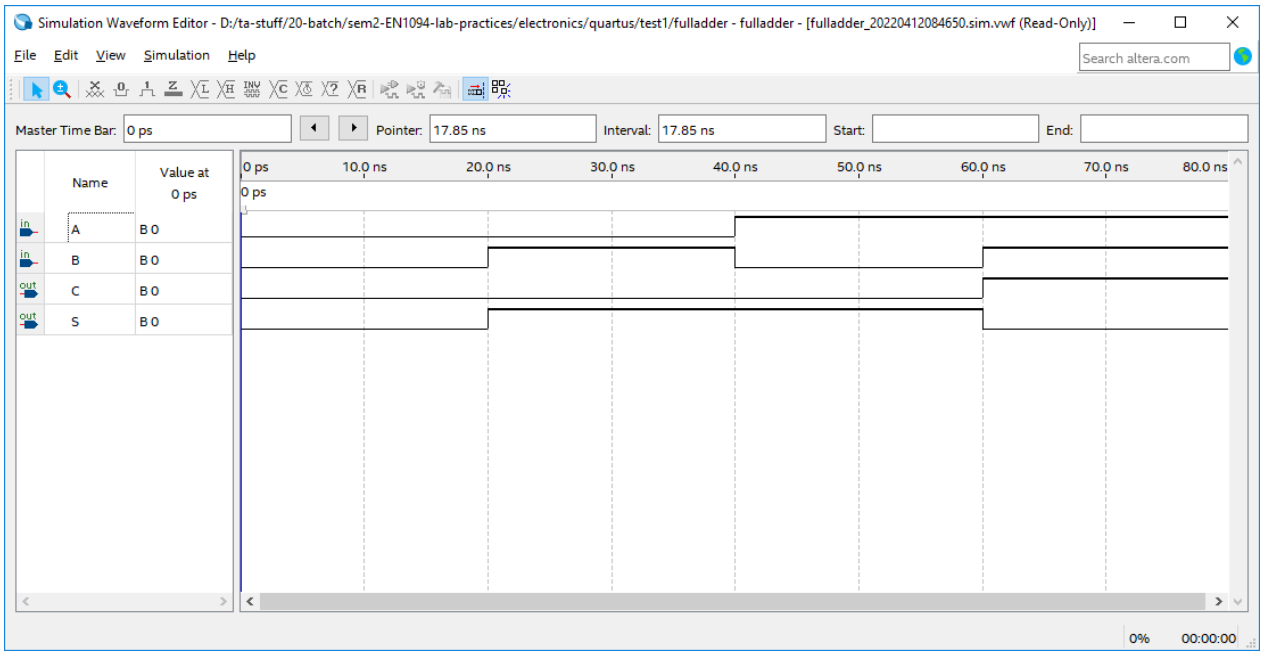


Figure 5.11: Simulation result

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.

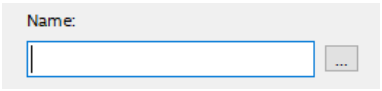


Figure 5.12: Browse button to locate new symbols

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.*

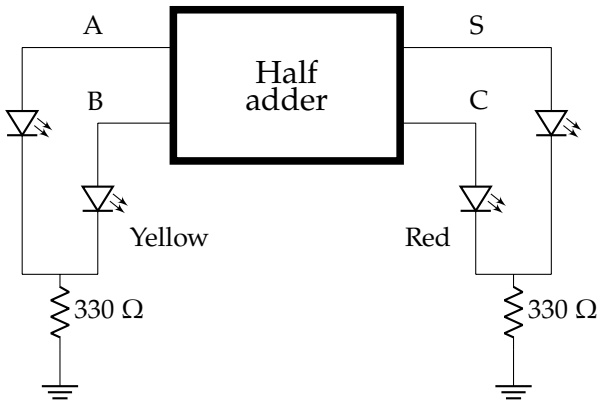


Figure 5.13: Inputs and outputs of half adder

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.*

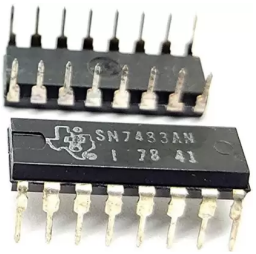


Figure 5.14: 7483 full adder IC

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.*

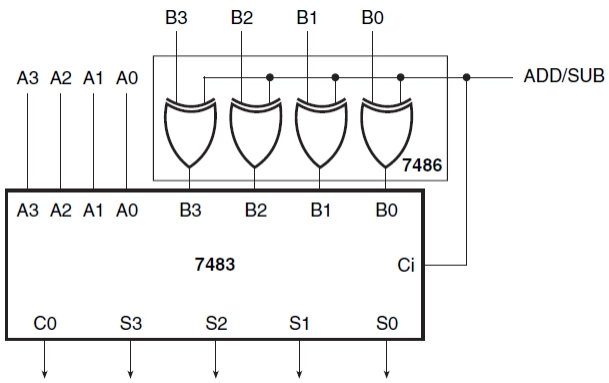


Figure 5.15: Adder-subtractor schematic

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

Operation	First input (A)				Second input (B)				Add/Sub	Output (darken lit LEDs)
	A_3	A_2	A_1	A_0	B_3	B_2	B_1	B_0	C_i	
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 ♣

Workshop 6: Design and Implementation of a Sequence Detector

Objective: To design, implement and verify a sequence detector as an example of a sequential logic circuit, using SystemVerilog and deploy on a Field Programmable Gate Array (FPGA)

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

1. Design sequential logic circuits using state diagrams
2. Use SystemVerilog to implement a sequential circuit
3. Develop a test bench and verify the hardware design
4. Perform pin planning and deploy the design on an FPGA

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. DE0-Nano Development Board
6. Oscilloscope
7. Jumper wires - male-to-female (3 Nos.)

Components Required: None

6.1 Introduction

In this experiment, we will design and implement a sequence detector on an FPGA. As the initial step, the required boolean expressions will be derived using a state diagram and truth tables. Then the detector, along with a sequence generator, will be implemented in SystemVerilog and simulated using ModelSim. Finally, the design will be programmed into a FPGA and the waveforms will be observed using the oscilloscope.

6.1.1 Finite State Machines

A Finite State Machine (FSM) is an abstract computational model which can be used to model sequential logic circuits. An FSM can take only one of the finite number of possible states at any given time. Transitions among the states are well defined, and depend on the current state and the current inputs. In addition to state transitions, FSMs also produce outputs. Depending on how the output is generated, FSMs can be categorized into two types namely;

- Mealy machines - the outputs depend on both the current state and the current inputs
- Moore machines - the outputs depend only on the current state.

The number of states of an FSM determines the number of memory elements that is required for implementing that FSM. FSMs can be illustrated graphically using state diagrams.

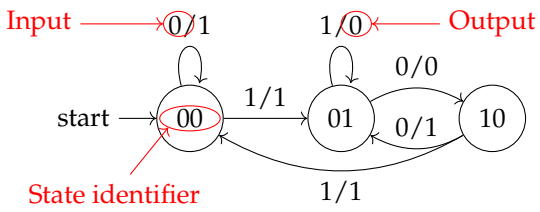


Figure 6.1: An example state diagram with three states

6.1.2 FPGAs and Development Boards

A Field Programmable Gate Array (FPGA) is an electronic chip (integrated circuit) inside which we can implement any digital circuit we want. Digital designers design their circuits, then express them in text using a Hardware Description Language (HDL) and use software tools to synthesize, place and route to implement it inside an FPGA.

When programming a microcontroller or a microprocessor, the software code we write is compiled into instructions which are to be executed by the microcontroller, which is a fixed digital circuit. On the other hand, when "programming" an FPGA, we describe our own circuit, which gets implemented. Therefore, we can design and implement our own custom processors inside an FPGA. Consequently, practical circuit design considerations such as placement of logic blocks and critical path delays must be taken into account. It is vital to understand this distinction.

FPGAs are field-programmable; that is, the circuit can be changed after deploying into a product. Therefore, they are widely used in products such as satellites, Mars rovers, scientific equipment such as particle detectors at LHC, high bandwidth network switches and video decoders. FPGA implementations of high-bandwidth algorithms are often faster and power-efficient than their generic CPU-based counterparts. For chips that can be manufactured in millions of units, like processors, it is economical to fabricate custom ICs instead, which are faster, consume less power and smaller than FPGAs.

For academic purposes and for prototyping needs, FPGA chips are available on development boards. A typical development board contains peripherals required for programming the FPGA along with switches and LEDs connected to the input/output pins of the FPGA. The development board that will be used in this experiment is the Terasic DE0-Nano board which consists of an Altera Cyclone IV FPGA.

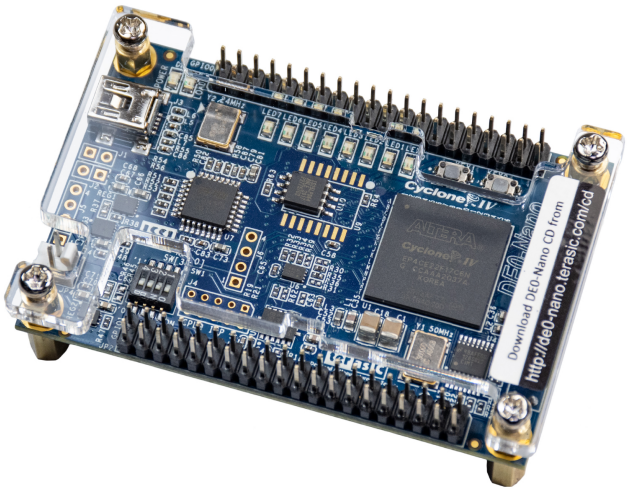


Figure 6.2: Terasic DE0-Nano development board with Altera Cyclone IV FPGA

6.1.3 HDLs and SystemVerilog

As the first step of programming an FPGA, we need to express our logic circuit in a text format. A Hardware Description Language (HDL) is a specialized computer language that can be used to describe the structure or the behaviour of logic circuits. Unlike a programming language, in an HDL, there is no control flow. That is, the statements are not executed one after the other. Instead, our hardware description is synthesized into a physical electronic circuit. However, some HDLs support non-synthesizable statements whose only purpose is to generate test benches, which are programs that run like software to drive inputs and check outputs of our hardware design. This is used to "verify" the hardware designs using simulators.

Verilog has become one of the most widely-used HDLs since its introduction in 1984. SystemVerilog, initially introduced as a Hardware Verification Language (HVL) has later been merged with Verilog to facilitate both hardware design and verification. While the synthesizable constructs of SystemVerilog are much similar to Verilog, testing and verification constructs support high-level programming language features such as Object-Oriented Programming.

6.1.4 Sequence Detectors

A *sequence detector* is used to detect a predefined binary sequence within a stream of incoming bits. Detecting flags (which usually indicate the beginning or the end of a data packet) in digital communication systems is a major application of sequence detectors. A sequence detector has a single-bit output which indicates whether the predefined sequence has arrived or not. Depending on the application, they are implemented in 2 different ways, namely, *overlapping* and *non-overlapping*. See figure 6.3 for a detailed illustration. A sequence detector can be considered as a Mealy machines since its output depends on both the current state and the current input.

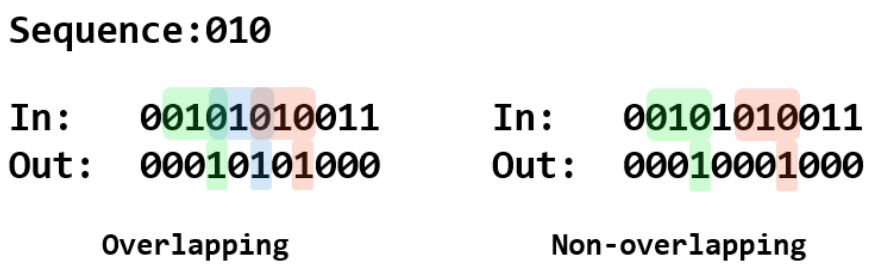


Figure 6.3: Types of sequence detectors

6.2 Pre-Lab

6.2.1 State Diagram and Truth Table

Task 1. *Consider the 3-bit binary sequence,* ***010****. Draw the state diagram for an* ***overlapping*** *sequence detector to detect this sequence. Name the states as follows;*

- $q_1 = 0, q_2 = 0$ - *none of the elements belonging to the sequence detected*
- $q_1 = 0, q_2 = 1$ - *'0' detected*
- $q_1 = 1, q_2 = 0$ - *'01' detected*
- $q_1 = 1, q_2 = 1$ - *'010' detected*

Task 2. *Complete the following truth table using the above state diagram. Assume that the states q_1 and q_2 will be stored in the first and the second JK flip-flops, respectively. Use 'X' for "don't care" terms.*

Current state		Input	Next state		Output	First flip-flop (q_1)		Second flip-flop (q_2)	
q_1	q_2	x	$\hat{q}_1$	$\hat{q}_2$	y	j_1	k_1	j_2	k_2

Table 6.1: Truth table for 3-bit sequence detector

Task 3. *Derive and simplify boolean expressions for y, j_1, k_1, j_2 and k_2, in terms of q_1, q_2 and x.*

Task 4. *Create a new project in Quartus and name it as **sequence_detector**. Create a new **SystemVerilog HDL** file (under **Design Files** category) named **jk_ff.sv**. Copy-paste into jk_ff.sv, and complete the following code snippet which implements a positive edge triggered JK flip-flop. Refer section 5.2.2 for more information.*
Hint: Refer the truth table of a positive edge triggered JK flip-flop.

```
module jk_ff(reset, clk, j, k, q);  // Module definition start

  input reset, clk, j, k; // Define inputs
                  // The 'reset'signal is used to reset the internal state of
                  // the flip-flop to logic zero.

  output q;  // Define outputs

  reg q;  // Allocate a register to hold the internal state.
      // Since both the output and the reg have the same name, they are
      // connected automatically.

  always @(posedge clk, posedge reset) begin  // Triggered at each positive edge
                              // of 'clk' or 'reset' signal.

    if (reset) begin
      q <= 0;    // If 'reset' is high, reset the internal state to zero.
    end
    else begin     // Otherwise, update q according to truth table.
      // ******************************
      // ****** your code goes here *****
      // ******************************
    end
  end

endmodule  // Module definition end
```

Listing 6.1: Positive edge triggered JK flip-flop

Task 5. *Set the **jk_ff** module as the top-level entity. Compile and simulate the module using the "Quartus Simulation Waveform Editor" and verify the functionality.*

6.3 Implementing the Sequence Detector

SystemVerilog facilitates *hierarchical* designing in order to allow design reused. Hence, we can use two instances of the previously constructed JK flip-flop to create the sequence detector. Example 3 demonstrates how to insert an instance of a previously implemented module.

Example 3. *Consider the schematic illustrated in figure 6.4. The module named* ***top*** *has two instances of the module named* ***bottom****. Both instances share the same* ***aa*** *input which is also connected to the* ***top*** *module's input* ***xx****. In such cases, a separate wire (****aa_to_aa****) has to be defined. However, the input* ***bb*** *of the first* ***bottom*** *instance and the output* ***cc*** *of the second* ***bottom*** *instance are directly connected to the* ***top*** *module's input* ***yy*** *and output* ***zz*** *respectively. Hence, they don't need additional wires to be defined. Code snippets below show how to implement this schematic in SystemVerilog.*

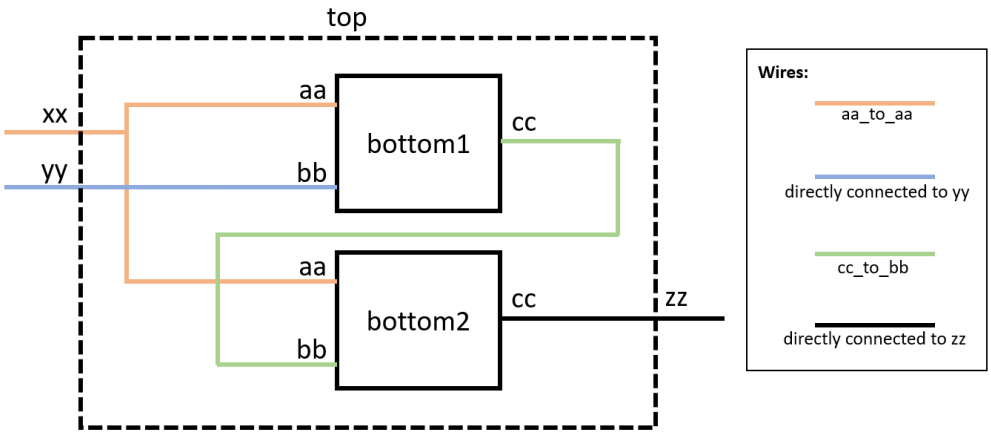


Figure 6.4: Example schematic of a hierarchical design. Wire names are also shown on the right-hand side

```
module bottom(aa, bb, cc); // Module definition start
  input aa, bb;  // Define inputs
  output cc;  // Define outputs
  reg cc;    // Allocate a register to hold output value.
  always @(posedge aa) begin     // Triggered at the positive edge of 'aa'.
    cc <= bb & cc;     // Some logic goes here.
  end
endmodule      //  Module definition end
```

Listing 6.2: Definition of **bottom** module

```
module top(xx, yy, zz); // Module definition start
  input xx, yy;  // Define inputs
  output zz;     // Define outputs
  wire aa_to_aa;  // Define a wire to connect 'aa's of the two instances.
```

```
  wire cc_to_bb;  // Define a wire to connect 'cc' and 'bb'.

  assign aa_to_aa = xx;   // Need to connect 'aa_to_aa' to 'xx' as well.

  // First instance of bottom module
  bottom bottom1(.aa(aa_to_aa),  // Connect 'aa' of the instance to 'aa_to_aa' wire
                 .bb(yy),        // Connect 'bb' of the instance directly to 'yy'
                 .cc(cc_to_bb)); // Connect 'cc' of the instance to 'cc_to_bb' wire

  // Second instance of bottom module
  bottom bottom2(.aa(aa_to_aa),   // Connect 'aa' of the instance to 'aa_to_aa' wire
                 .bb(cc_to_bb),   // Connect 'bb' of the instance to 'cc_to_bb' wire
                 .cc(zz));        // Connect 'cc' of the instance to 'zz'

endmodule          // Module definition end
```

Listing 6.3: Definition of **top** module

Task 6. *Create a new SystemVerilog file named **sequence_detector.sv** under the Quartus project that was created earlier (hereafter, referred to as "the project"). Copy and complete the code snippet given below in order to implement the sequence detector (to detect the sequence **'010'**) using the JK flip-flop module developed under Task 4. Refer Table 6.1, Task 3 and Example 3.*

```
module sequence_detector(reset, clk, x, y);

  input reset, clk, x; // Define inputs

  output y; // Define outputs

  wire j1, k1, q1, j2, k2, q2;    // Define wires

  jk_ff ff1(                  // First JK flip–flop to store state q1
        .reset(reset),
        .clk(clk),
        .j(j1),
        .k(k1),
        .q(q1));

  // Include the second JK flip–flop to store state q2
  // ********************************
  // ****** your code goes here *****
  // ********************************


  assign y = q1 & ~q2 & ~x;   // Boolean expression for y

  // Write 'assign' statements (boolean expressions) for flip–flop inputs
  // ********************************
  // ****** your code goes here *****
  // ********************************

endmodule
```

Listing 6.4: Sequence detector

Task 7. *Make **sequence_detector** the top-level entity and compile to see whether there are any syntax errors.*

In order to verify the implementation we can use the **RTL Viewer** to visualize the interconnections and the inferred logic blocks. It will display how each block of the design has been connected hierarchically. Figure 6.5 illustrates the **RTL Viewer** output of the **top** module in Example 3.

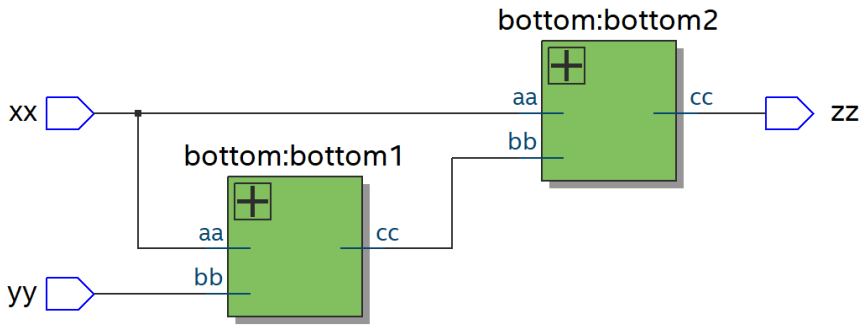


Figure 6.5: RTL Viewer output - **top** module in hierarchical design example

Task 8. *Goto **Tools > Netlist Viewers > RTL Viewer** from the menu bar to open the **RTL Viewer**. Interior of a block can be viewed by double-clicking on it. Verify the connections of **sequence_detector**.*

6.4 Using ModelSim for Simulation

In order to simulate the sequence detector, we will first create a sequence generator which generates a pseudo-random bit stream. Afterwards, we will create a test bench including both the sequence generator and the detector and simulate it using ModelSim.

6.4.1 Sequence Generator

The code snippet given below implements a sequence generator which is also known as a Linear Feedback Shift Register (LFSR). The schematic of the LFSR is illustrated in figure 6.6. The **init_seq** parameter in the snippet specifies the initial internal state of the LFSR.

```
module sequence_generator(reset, clk, seq); // Module definition start

  localparam init_seq = 4'b1000;  // Initial seed.
                        // Can be any 4-bit sequence except 4'b0000.
                        // 'localparam' specifies that this is just a
                        // constant which will be resolved at compile time.


  input reset, clk;   // Define inputs


  output seq;       // Define outputs


  reg[$size(init_seq)-1:0] seq_reg;   // Linear Feedback Shift Register (LFSR). Just
                          // an ordinary register which can store many
                          // bits :)

                          // '$size()' is a Verilog system function which
                          // will be resolved at compile time. It
                          // returns the length of its argument.


  assign seq = seq_reg[$size(init_seq)-1];    // Connect the output to the MSB
                                // of the LFSR.
```

```
  always @(posedge clk, posedge reset) begin  // Triggered at each positive edge
                                              // of 'clk' or 'reset'

    if (reset) begin    // If 'reset' is high, reset LFSR to initial sequence.
      seq_reg <= init_seq;
    end

    else begin        // Else, generate the next sequence
      seq_reg <= {seq_reg[$size(init_seq)-2:0], seq_reg[$size(init_seq)-1] ^ seq_reg[$size(init_seq)-2]};
    end
  end

endmodule
```

Listing 6.5: Sequence generator

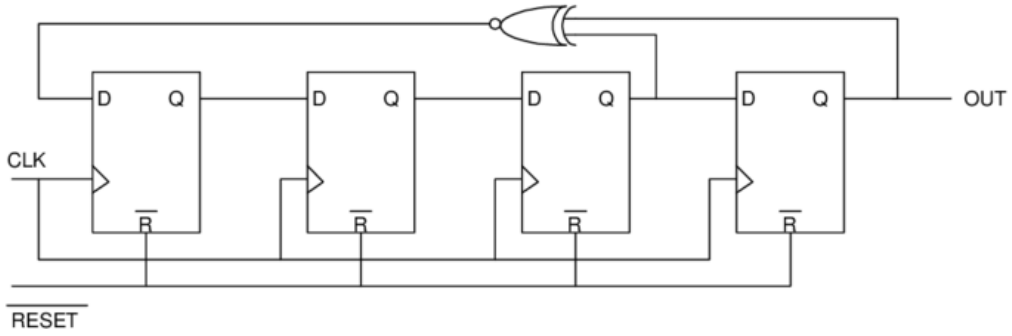


Figure 6.6: Schematic of a Linear Feedback Shift Register

Task 9. *Add a new SystemVerilog file named* ***sequence_generator.sv*** *to the project and copy-paste the above code snippet. Change the parameter* ***init_seq*** *to any 4-bit sequence of your preference other than* ***4'b0000*** *and save.*

6.4.2 Test Bench

A test bench is somewhat similar to an ordinary module, but has few additional properties such as;

- includes timing information for the simulator
- does not have any inputs or outputs
- includes un-synthesizable code (eg: delays - #1, utility functions - $display()) and hence, cannot be compiled

The test bench includes the synthesizable modules that are to be tested, and provides internally generated signals to the inputs of those modules. This way, we can observe the output of the synthesizable modules through a simulator such as ModelSim.

Implementation

Our test bench should;

- include one instance each of sequence generator and detector
- provide **clk** and **reset** signals to the above modules
- internally connect the output **seq** of the generator to the input **x** of the detector

- extract the output **y** of the detector

The **clk** and **reset** signals can be generated using the unsynthesizable delay operator (#*n* where # is the operator and *n* specifies the duration in number of timescales).

Task 10. *Add a new SystemVerilog file named **tb_sequence_detector.sv** under the project. Insert and complete the code given below, which implements a test bench for the sequence detector and the generator. Set the test bench as the top-level entity.*

```
`timescale 10ns/1ns // Timing information for the simulator.
                    // 10ns – means #1 corresponds to 10 nano seconds.
                    // 1ns – means the accuracy should be upto 1 nano second.

module tb_sequence_detector();  // Module definition start.
                                // Just another ordinary module, but without
                                // any inputs/outputs.

  reg reset, clk; // Registers to hold the values for 'clk' and 'reset'

  wire x, y;      // Wires to connect inputs and outputs of the modules being tested.


  // Include an instance of sequence_generator
  // Note that wire 'x' should be connected to the 'seq' output of the generator
  // ********************************
  // ****** your code goes here *****
  // ********************************


  // Include an instance of sequence_detector
  // ********************************
  // ****** your code goes here *****
  // ********************************


  initial begin       // Triggered at the very beginning. This is not synthesizeable.
    clk = 0;        // Initialize 'clk' to zero
    reset = 0;      // Initialize 'reset' to zero

    // All the modules need to be reset to start from a definite internal state.
    // Hence, we need to generate a 'reset' posedge in the beginnig.
    #1 reset = 1;   // After 10ns, make 'reset' high
    #1 reset = 0;   // After another 10ns, make 'reset' low
  end

  always #1 clk <= ~clk;  // Triggered at each 10ns period.
                          // Each time, the state of 'clk' will be inverted, causing
                          // it to act as a clock signal. Note that the
                          // delay operator '#' is not synthesizable.

endmodule // Module definition end
```

Listing 6.6: Test bench

Since the test bench includes un-synthesizable code, it cannot be compiled completely. However, we can perform the **Analysis** and **Elaboration** steps of the compilation workflow, which are sufficient for simulation.

Task 11. *After setting the test bench as the top-level entity, goto **Processing > Start > Start Analysis & Elaboration** from the menu bar. Monitor the **Messages Pane** for any error messages, and debug if any.*

Simulation

To start the simulation, goto **Tools > Run Simulation Tool > RTL Simulation** from menu bar. This will open up a ModelSim window. See figure 6.7 (note that the locations of the panes may differ depending on your initial settings).

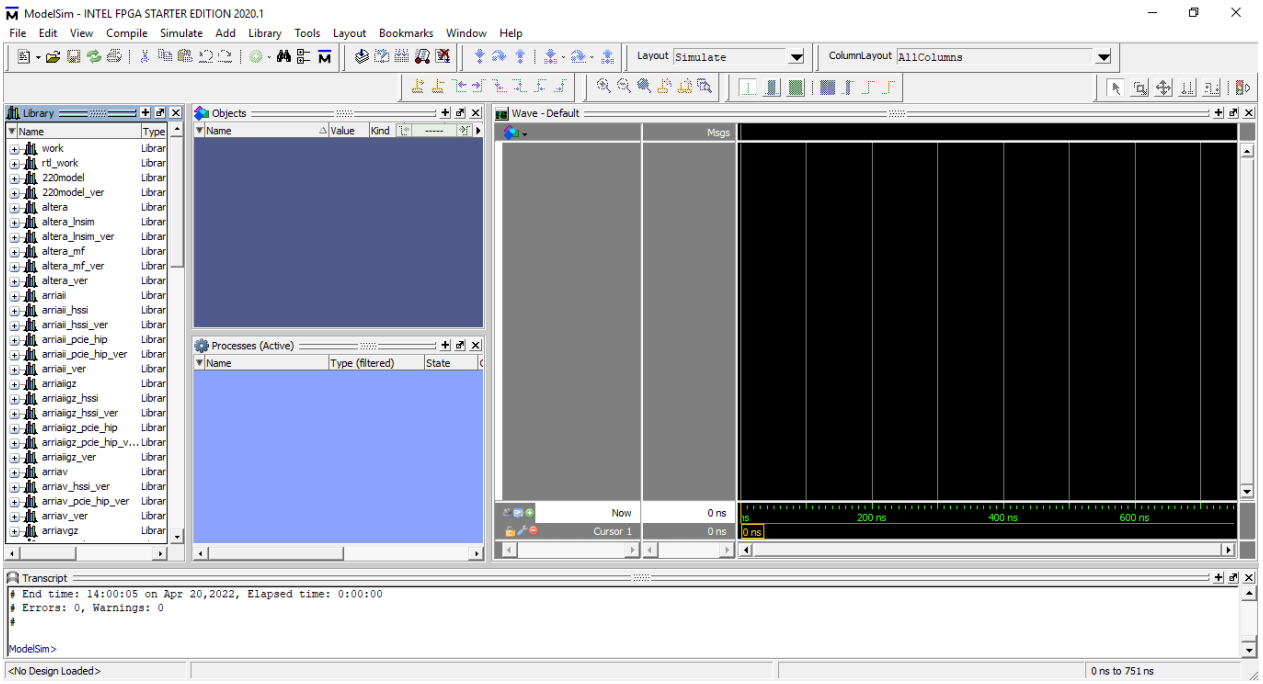
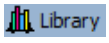


Figure 6.7: ModelSim main window

Task 12. *Go through the following instructions to set-up and run the simulation.*

1. *Locate the* ***Library***  *pane.*
2. *Expand the* ***work***  *node.*
3. *Locate* ***tb_sequence_detector****. Right click on it and select* ***Simulate****. This will open up* ***Sim***  *tab which lists all the module instances in the design, in a hierarchical structure. In addition, signals of the currently selected instance will be displayed under the* ***Objects***  *pane.*
4. *To view a signal on the* ***Wave***  *pane, right-click on the interested signal in* ***Objects*** *pane and select* ***Add Wave****. Do this for all the waves* ***clk, reset, x*** *and* ***y****.*
5. *Set the simulation duration at the* ***Run Length*** *text box*  *to* ***100 ns****.*
6. *Press* ***Run***  *button to run the simulation. See figure 6.8.*

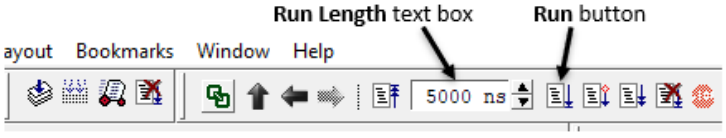


Figure 6.8: **Run Length** text box and **Run** button

6.5 Programming the FPGA

Under this section, we will program **DE0 Nano** development board with **Intel Altera Cyclone IV** FPGA and observe the waveforms through an oscilloscope.

6.5.1 Implementing the Top-level Module

As the first step, we will create a new top module which includes instances of the **sequence_generator** and the **sequence_detector**.

Task 13. *Add a new SystemVerilog file named **sequence_detector_top.sv** and insert the code given in the following snippet. Make this module the top-level entity of the project. Compile the project to see whether there are any syntax errors. Open the **RTL Viewer** and verify the connections.*

```
module sequence_detector_top(reset_inv, clk, x, y, blink);

  localparam clk_dev_coef = 10;  // This is a constant which is resolved at compile
                                 // time. Since the frequency of the FPGA clock
                                 // (runs at 50 MHz) is too high to observe clearly,
                                 // it is divided by 2^(clk_dev_coef+1).

  input reset_inv, clk;  // Define inputs. Note that pushbuttons are considered to
                         // be logic–high when not pressed. Hence, to use as the
                         // 'reset'signal for the rest of the modules, it should
                         // be inverted.

  output x, y, blink;    // Define outputs. 'blink' will be connected to a LED
                         // which blinks in an observable frequency.

  wire clk_dev, reset;   // Define wires for divided clock and inverted reset
                         // pushbutton signals.

  reg[24:0] clk_div_reg; // Register to divide the clock


  sequence_detector sd(              // Instance of sequence_detector
                .reset(reset),
                .clk(clk_dev),
                .x(x),
                .y(y));

  sequence_generator sg(             // Instance of sequence_generator
                .reset(reset),
                .clk(clk_dev),
                .seq(x));


  assign clk_dev = clk_div_reg[clk_dev_coef]; // Assign divided clock
  assign blink = clk_div_reg[24];             // Assign 'blink'
  assign reset = ~reset_inv;                  // Invert reset pushbutton signal


  always@(posedge clk, posedge reset) begin  // Incrementing the clock–div register

    if (reset) clk_div_reg <= 0;

    else clk_div_reg <= clk_div_reg + 1;

  end
```

```
endmodule
```

Listing 6.7: Top module for FPGA

6.5.2 Pin Assignment

Everything upto now did not have any dependency on the hardware to which the design is to be uploaded. However, the next step, which is called **Pin Assignment**, depends on the FPGA model that is being used. Under this step, we will assign physical pins of the FPGA to the I/O nodes of the module defined in SystemVerilog (i.e., **reset_inv, clk, x, y** and **blink**). There are two levels of pin mapping that we have to consider, which can be stated as follows;

- SystemVerilog design to the FPGA (e.g.: **clk** should be connected to FPGA pin **PIN_R8**)
- FPGA to the development board (e.g.: **KEY[0]** of the board which is a pushbutton is connected to the FPGA pin **PIN_J15**)

The first mapping is done through Quartus **Pin Planner** tool. The second mapping is specified in the user manual of the development board.

Task 14. *Complete the following pin assignment table by referring the DE0 Nano user manual.*

I/O node	DE0 Nano signal	FPGA pin
reset_inv	KEY[0]	PIN_J15
clk	CLOCK50	PIN_R8
x	GPIO_00	
y	GPIO_01	
blink	LED[1]	

Table 6.2: Pin assignment table

Task 15. *After compiling the top-level entity **sequence_detector_top**, click **Assignments > Pin Planner** from the menu bar to open the pin planner. Based on Table 6.2, assign the pins by specifying the pin name under the **Location** column of the table displayed at the bottom of the window (see Figure 6.9). Once the assignment is complete, close the pin planner and compile the module once again.*

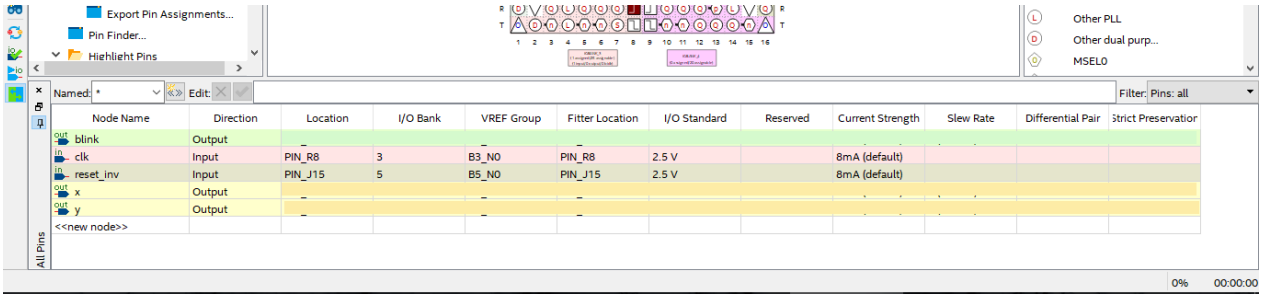


Figure 6.9: Pin Planner

6.5.3 Installing Drivers

To install the usb driver for the development board, follow the instructions given below.

1. Connect the DE0 Nano board to the PC using the provided cable.

2. Open **Device Manager**.
3. Locate **USB Blaster** listed under **Other devices**.
4. Right-click on to of **USB Blaster** and select **Update driver**.
5. From the opened dialog, select **Browse my computer for driver software**
6. Press the **Browse** button and navigate to **<Quartus installation folder>/quartus** and select **drivers** folder. E.g.: navigate to "C:/intelFPGA/textunderscore lite/20.1/quartus" and select "drivers"
7. If the installation completes successfully, you are good to go.

6.5.4 Uploading the Design

To program the FPGA with the design, complete the following steps.

Task 16. *Make sure that you compiled the design after all the changes have been made. While the development board is connected to the PC;*

1. *Goto **Tools > Programmer** from the menu bar. It will open the **Programmer** (see Figure 6.10).*

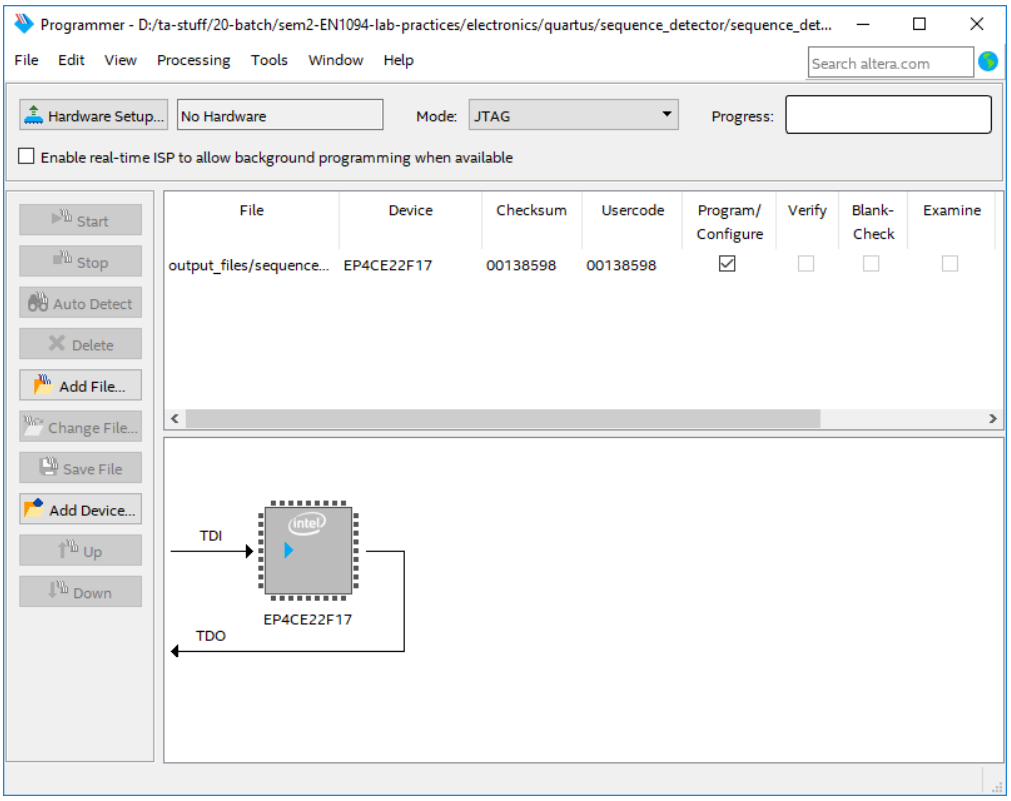


Figure 6.10: Programmer

2. *In the **Programmer**, select **JTAG** as the mode.*
3. *If the box next to **Hardware Setup...** buttons displays **No Hardware**, press **Hardware Setup...** button. It will open the **Hardware Setup** window.*
4. *In the **Hardware Settings** tab, set **USB-Blaster** as the **Currently selected hardware** and press **Close** (see Figure 6.11).*

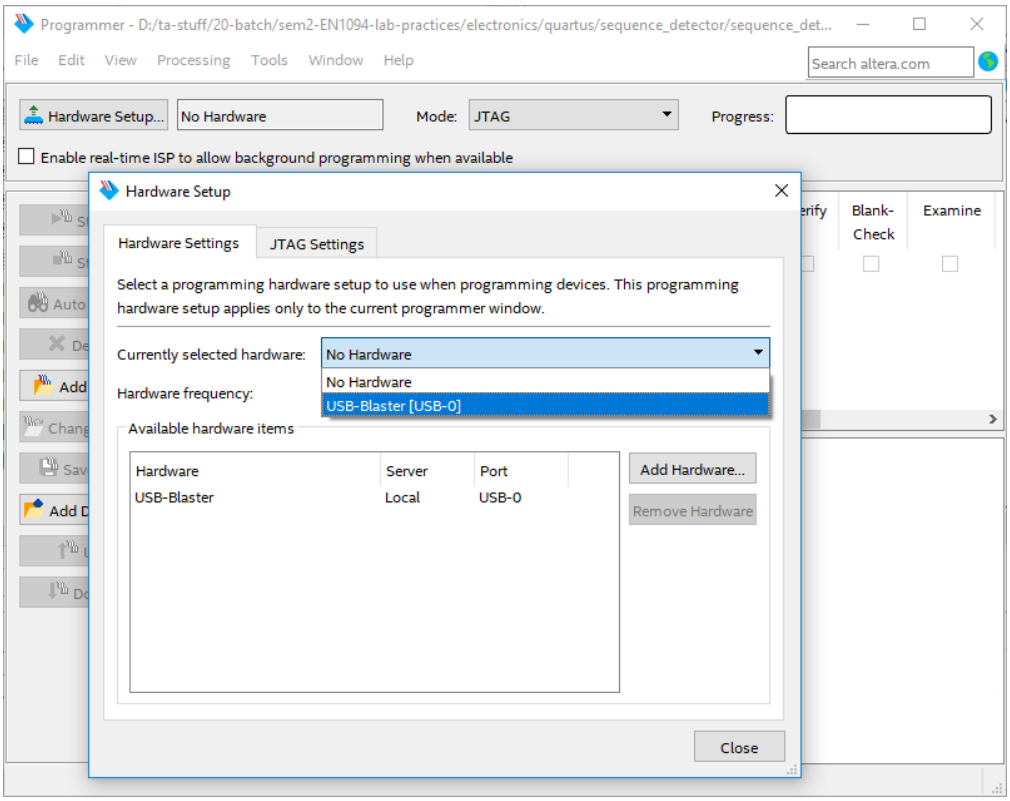


Figure 6.11: Hardware Setup

5. *Press the **Start** button on the **Programmer** window to upload the design to the FPGA.*
6. *If the design has been uploaded successfully, LED[1] should start blinking.*

6.6 Observing the Waveforms

Due to the very high operational frequency of the sequence detector, we cannot verify the functionality with the naked eye (say for instance, by using a logic probe). Hence, we will use the oscilloscope to observe the waveforms.

Task 17. *Connect 3 male-to-female jumper wires to **GPIO_00**, **GPIO_01** and the **Ground** pin. Refer the DE0 Nano user manual to identify the pins. Connect oscilloscope ground probe to the FPGA **Ground**. Observe and draw the waveforms at **GPIO_00** and **GPIO_01**. Measure the frequency of the divided clock.*

♣ The End ♣

Part IV

Telecommunication

Workshop 1: Communication Channel Characteristics & Effects of Noise

Objective: To Observe Communication Channel Characteristics & Effects of Noise

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

1. Identify the cut-off frequencies and bandwidth associated with channel characteristics
2. Estimate the cut-off frequencies and bandwidth by measurement
3. Identify the effects of bandwidth limitation on signals carried by a channel
4. Examine the effect S/N ratio in analog signal transmission
5. Estimate the Bit Error Rate (BER) in digital signal transmission as a function of S/N

Equipment Required:

1. Oscilloscope
2. Signal Generator
3. Proto board
4. A Personal Computer
5. MATLAB software or MATLAB online

Components Required:

1. Resistors 270Ω (1), 330Ω, (1), 1 kΩ (1)
2. Capacitors 1µF (1), 0.33µF (1)

1.1 Communication Channel

Figure 1.1 shows the block diagram of a basic communication system (analog/digital).

The **source** is the device or the person which generates the data to be transmitted. The **transmitter** processes and transmits the data. **Channel** is the medium through which the data is transmitted. **Receiver** receives and processes the transmitted data. The **destination** is the final device or person to which the data is intended.

In this practical we will be focusing on the effects induced by the channel. The channels typically induce three major effects to the transmitted data.

- Attenuation
- Noise Addition
- Bandwidth limitation

Let the transmitted signal be $X(t)$. Attenuation is simply the reduction of the amplitude of the signal. The signal after attenuation is $AX(t)$, where $0 < A < 1$ is the attenuation factor.

Noise is the random fluctuation of the signal around it's actual value. The signal after attenuation and noise can be written as $AX(t) + N(t)$, where $N(t)$ represents a random signal.

The channels typically limit the bandwidth of the transmitted signal. In other words certain frequencies of the transmitted signal are filtered. Hence, the signal after all three effects can be written as bandlimit$(AX(t) + N(t))$ where, the bandlimit function represents the bandwidth limitation.

In this practical we will be focusing on understanding these effects in detail.

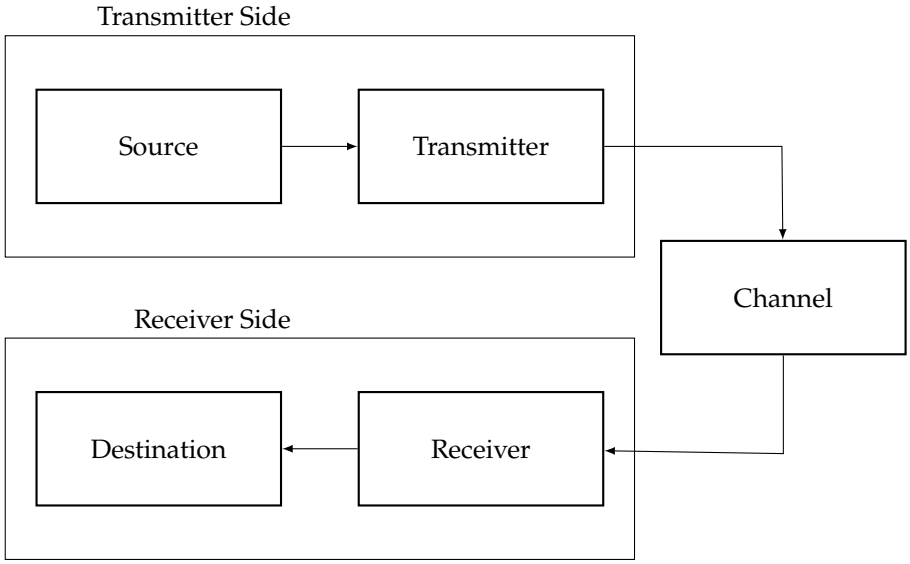


Figure 1.1: The block diagram of a communication system

1.2 Pre-Lab

Prior to the lab you will model a basic communication channel using Simulink. Channels for both analog and digital transmission can be modelled using the above effects.

1.2.1 Analog Channel

We will first focus on an analog channel. In an analog channel an analog signal is transmitted through the channel. In this case we model the noise $N(t)$ as a Gaussian noise. The bandlimiting filter is a low-pass filter. Build the following simulink model (Figure 1.2) in bring it to the lab.

Use the following parameters.

- Transmitted sinusoidal signal [**Sine Wave** block in **Simulink** > **Sources**]
  - Amplitude: 10
  - Frequency: 200 rad/s
  - Sample time: 0.001 s
- Channel attenuation [**Gain** block in **Simulink** > **Commonly used blocks**]
  - Gain: 0.8
- Attenuated signal, Signal corrupted by noise, Signal output from channel, Information signal, Noise, SNR grpah, SNR average [**Scope** block in **Simulink** > **Commonly used blocks**]
- Add [**Add** block in **Simulink** > **Math Operations**]
- Channel noise [**Random Source** block in **DSP System Toolbox** > **Sources**]
  - Source type: Gaussian
  - Mean value: 0
  - Sample mode: Discrete
  - Variance: 1
  - Sample time: 0.001 s

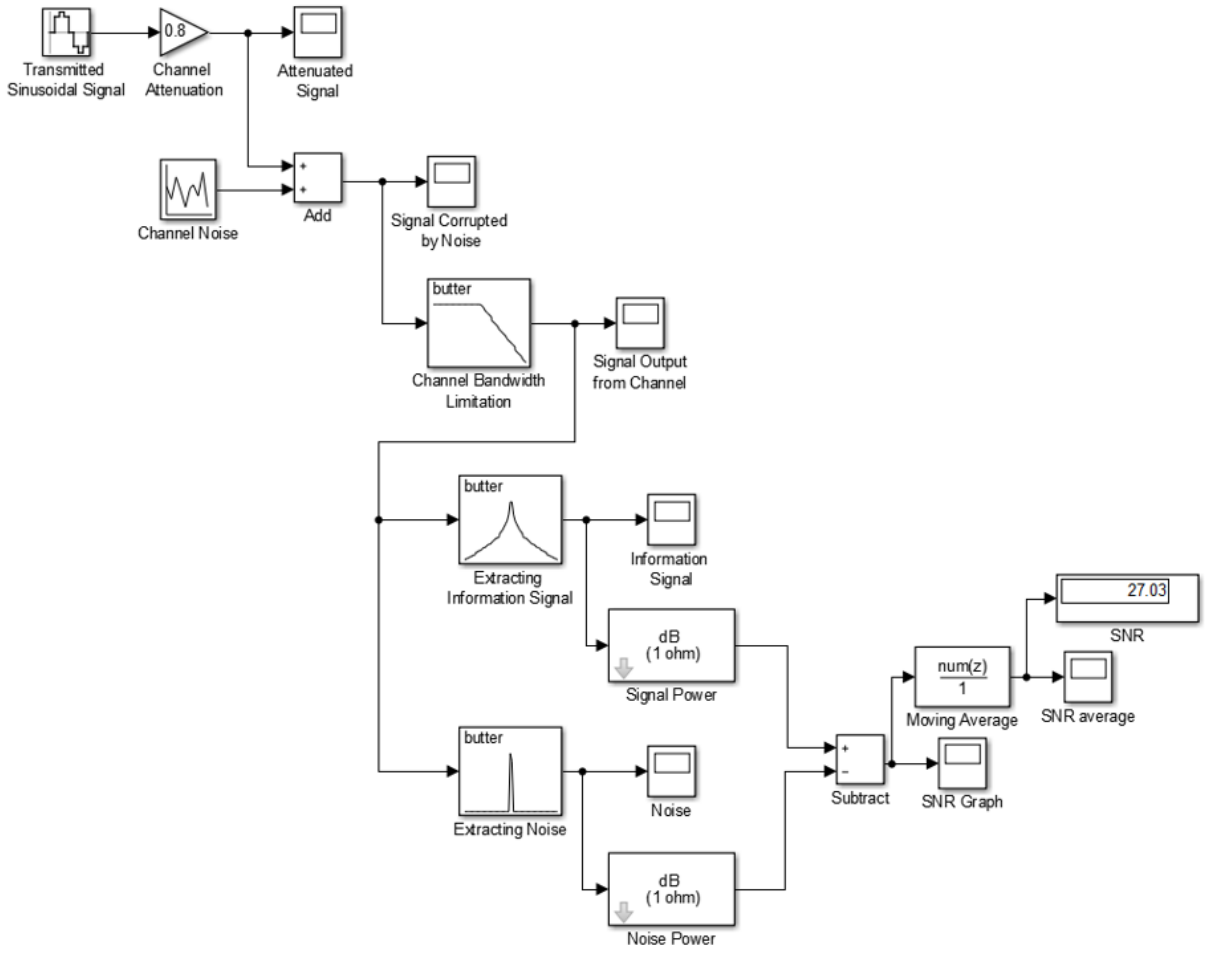


Figure 1.2: Simulink Model for Analog Channel.

- Channel bandwidth limitation [**Analog Filter Design** block in **DSP System Toolbox** > **Filtering** > **Filter Implementations**]
  - Design method: Butterworth
  - Filter type: Lowpass
  - Filter order: 10
  - Passband edge frequency: 1000 rad/s
- Extracting information signal [**Analog Filter Design** block in **DSP System Toolbox** > **Filtering** > **Filter Implementations**]
  - Design method: Butterworth
  - Filter type: Bandpass
  - Filter order: 10
  - Lower passband edge frequency: 195 rad/s
  - Upper passband edge frequency: 205 rad/s
- Extracting noise [**Analog Filter Design** block in **DSP System Toolbox** > **Filtering** > **Filter Implementations**]

  - Design method: Butterworth
  - Filter type: Bandstop
  - Filter order: 10
  - Lower passband edge frequency: 195 rad/s
  - Upper passband edge frequency: 205 rad/s
- Signal power, Noise power [**dB Conversion** block in **Communications System Toolbox** > **Utility Blocks**]
  - Convert to: dB
  - Input signal: Amplitude
  - Load resistance: 1 ohm
- Subtract [**Subtract** block in **Simulink** > **Math Operations**]
- SNR [**Display** block in **Simulink** > **Sinks**]
- Moving average [**Discrete FIR Filter** block in **Simulink** > **Discrete**]
  - Filter structure: Direct form
  - Sample time: 0.001
  - Coefficients: ones(1,1000)/1000

Task 1. *Set the simulation time to 20s and run the simulation. Observe the effects of attenuation, channel noise and channel bandwidth limitation. Explain how the SNR is calculated.*

1.2.2 Digital Channel

In a digital channel a bit stream is transmitted. The bit 1 is transmitted with a pulse of amplitude 4, and the bit 0 is transmitted with a pulse of amplitude -4. We model the noise as Gaussian noise similar to the analog channel. Build the following simulink model (Figure 1.3) in bring it to the lab.

Use the following parameters.

- Binary Generator [**Bernoulli Binary Generator** block in **Communication System Toolbox** > **Comm sources** > **Random Data Sources**]
  - Probability of a zero: 0.5
  - Source of initial seed: Parameter
  - Initial seed: 61
  - Sample time: 0.1 s
- Gain [**Gain** block in **Simulink** > **Math Operations**]
  - Gain: 8
- Constant [**Constant** block in **Simulink** > **Sources**]
  - Constant value: -4
- Add [**Add** block in **Simulink** > **Math Operations**]
- Digital information signal, Signal corrupted by noise, Channel output, Output from sampler, Output from thresholder, Output from detector [**Scope** block in **Simulink** > **Commonly used blocks**]
- Channel attenuation [**Gain** block in **Simulink** > **Math Operations**]
  - Gain: 0.5

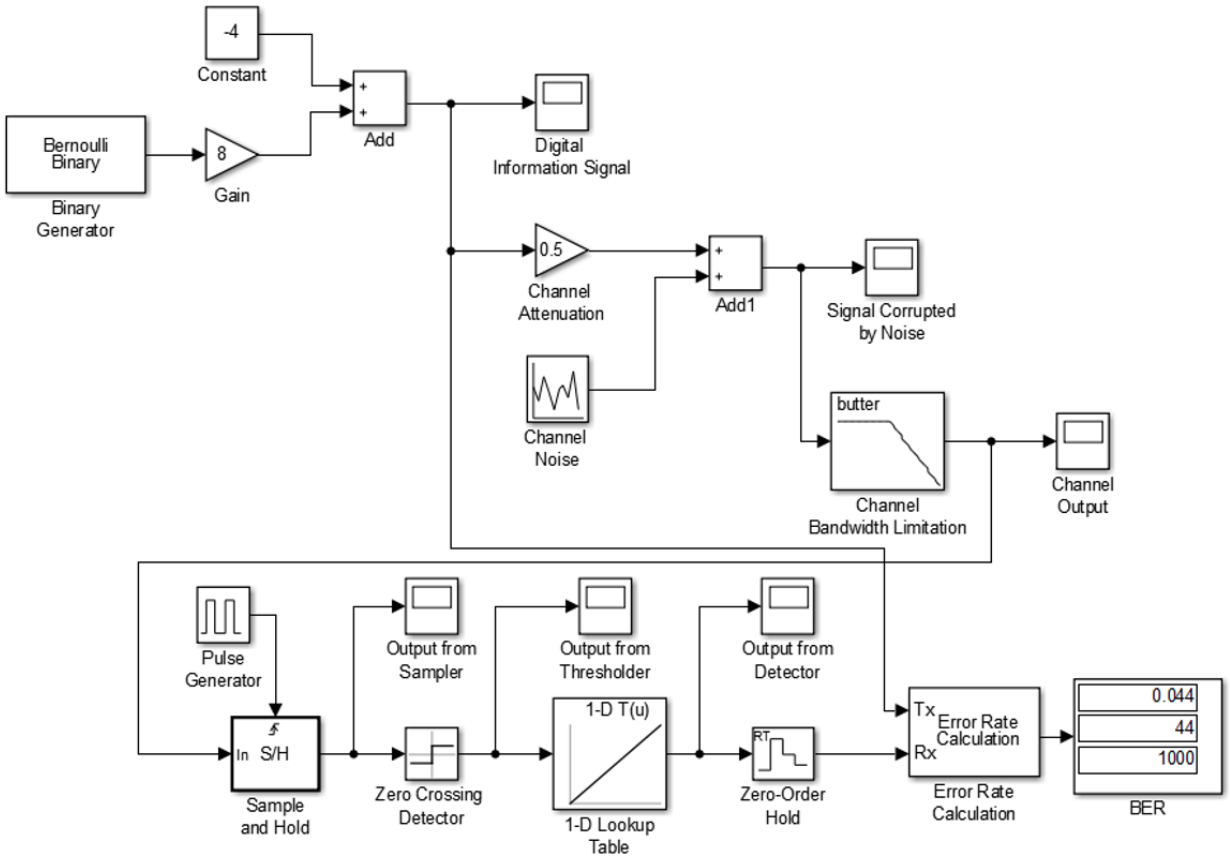


Figure 1.3: Simulink Model for Digital Channel.

- Channel noise [**Random Source** block in **DSP System Toolbox** > **Sources**]
  - Source type: Gaussian
  - Mean value: 0
  - Variance: 1
  - Sample time: 0.01 s
- Channel bandwidth limitation [**Analog Filter Design** block in **DSP System Toolbox** > **Filtering** > **Filter Implementations**]
  - Design method: Butterworth
  - Filter type: Lowpass
  - Filter order: 10
  - Passband edge frequency: 1000 rad/s
- Pulse Generator [**Pulse Generator** block in **Simulink** > **Sources**]
  - Pulse type: Time based
  - Time: Use simulation time
  - Amplitude: 1
  - Period: 0.1 s
  - Pulse width: 50
  - Phase delay: 0 s

- Sample and Hold [**Sample and Hold** block in **DSP System Toolbox** > **Signal Operations**]
  - Trigger type: Rising edge
  - Initial condition: 0
- Zero crossing detector [**Sign** block in **Simulink** > **Math Operations**]
  - Enable zero crossing detection
  - Sampling time: -1
- Lookup Table [**1-D Lookup Table** block in **Simulink** -> **Lookup Tables**]
  - Table Data: [-4,4]
  - Break Points: [-1,1]
  - Sample time: -1
- Zero-Order Hold [**Zero-Order Hold** block in **Simulink** > **Discrete**]
  - Sample time: 0.1 s
- Bit Error Rate Calculator [**Error Rate Calculation** block in **Communication Systems Toolbox** > **Comm Sinks**]
  - Receive delay: 1 s
  - Computation delay: 0 s
  - Output Data: Port
- BER [**Display** block in **Simulink** > **Sinks**]

Task 2. *Set the simulation time to 100s and run the simulation. Observe the signal at each scope output and identify the operation at each stage. Explain how the BER is calculated.*

1.3 Analyzing the Analog Channel

Answer the following questions using the simulink model developed in section 1.2.1.

Task 3. *Complete the Table in the Task Sheet by changing the channel noise variance as given.*

Task 4. *Complete the Table in the Task Sheet by changing the channel bandwidth as given. Keep the noise variance as 1 for all cases.*

Task 5. *Discuss the reasons for the observed variation of SNR with the channel bandwidth.*

Task 6. *Discuss the observed variation of SNR with the information signal amplitude and its relevance in signal transmission over communication channels in practice.*

Task 7. *It is often required to transmit composite analog signals through transmission channels. To evaluate the impact of noise and other factors on such applications, replace the previously transmitted analog signal to the following signal by using suitable blocks available in Simulink. Show your work to an instructor.*

$$x(t) = \sum_{k=0}^{4} (10 - 2k)\sin\left((200 + 20k)t\right). \tag{1.1}$$

Task 8. *Include the time domain behavior of the new composite signal.*

Task 9. *Adjust the passband of the Butterworth bandpass filter for information extraction and stopband of the Butterworth bandstop filter for noise extraction. What are the new cutoff frequencies for passband and stop band of the two filters for information extraction and noise extraction?*

Task 10. *How do you adjust the cutoff frequencies for passband and stop band of the two filters for information extraction and noise extraction? Elaborate your answer from the results obtainable from the simulations.*

1.4 Analyzing the Digital Channel

Answer the following questions using the simulink model developed in section 1.2.2.

Task 11. *Complete the Table in the Task Sheet by changing the channel noise variance as given.*

Task 12. *Discuss the reasons for the observed variation of BER with noise variance.*

Task 13. *Plot the BER with respect to signal to noise ratio (Eb/No).*

1.5 Modelling the Bandlimiting Effect of a Communication Channel Using Hardware

In this section we will be modelling the bandlimiting effect of a communication channel using hardware.

- Low-pass channel - filters the frequencies above a particular cutoff
- High-pass channel - filters the frequencies below a particular cutoff
- Band-pass channel - allows frequencies in a particular range to pass through while filtering the other frequencies
- Band-stop channel - filters frequencies in a particular range

1.5.1 Signal Transmission Through a Low-Pass Channel

We will first implement a low pass channel using hardware. Implement the circuit shown in Figure 1.4.

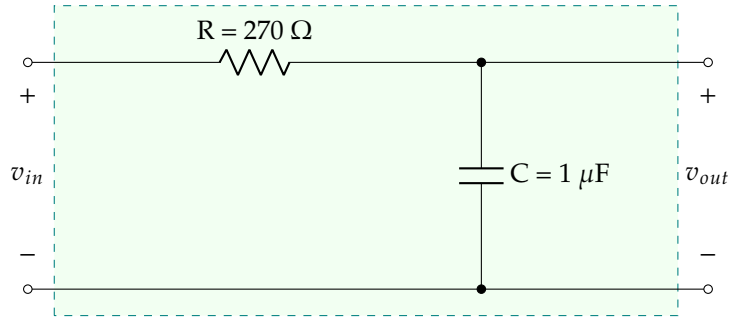


Figure 1.4: Low-Pass Channel

Connect the signal generator to v_{in} and the two channels of the oscilloscope to v_{in} and v_{out}.

Task 14. *Select the sinusoidal mode of the signal generator and apply 1V peak-to-peak signals having frequencies as shown in the Table in the Task Sheet. Complete the Table by taking measurements of v_{out}. Please note that you need to keep v_{in} at 1V peak-to-peak in each measurement.*

Task 15. *Plot the output voltage vs. Log (frequency) in the Task Sheet, complete the curve and write down the cut-off frequency obtained from graph.*

Task 16. *What is the obtained cut-off frequency?*

Task 17. *Find the theoretical cut-off frequency of this channel and compare with the value obtained by the above graph.*

1.5.2 Distortion in Signals When Transmitted Through Low-Pass Channels

The signals are subjected to distortion (change from the original signal) as a result of passing through channels. This is due to the partial or full loss of signal energy in certain frequencies due to bandwidth limitation. In this section we will try to observe the distortion.

Select the square wave mode of the signal generator. Set the peak-to-peak input voltage to 2V, and increase the frequency from 0 while looking at the output on the oscilloscope.

Task 18. *Complete the Table in the Task Sheet with sketches showing relative amplitudes correctly.*

Task 19. *What is the frequency at which you begin to see distortion in the shape of the output signal.*

1.5.3 Signal Transmission Through a Band-Pass Channel

In this section we will implement a band pass channel using hardware. Implement the circuit shown in Figure 1.5.

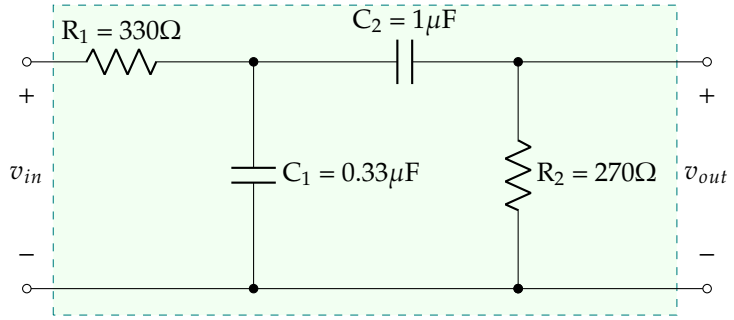


Figure 1.5: Band-Pass Channel

Connect the signal generator to the input and select the sinusoidal mode and apply 1V peak-to-peak signal.

Task 20. *Measure the output voltages v_{out} at the frequencies shown in the Table in the Task Sheet. Please note that you need to keep v_{in} at 1V peak-to-peak in each measurement.*

Task 21. *Plot the output voltage vs. Log (frequency) in the Task Sheet, complete the curve and write down the lower cut-off frequency and upper cut-off frequency obtained from the graph.*

Task 22. *What is the obtained lower cut-off frequency?*

Task 23. *What is the obtained upper cut-off frequency?*

Task 24. *Find the theoretical Lower cut-off and Upper cut-off frequencies and bandwidth of the channel.*

Task 25. *What is the frequency of resonance?*

1.5.4 Distortion in Signals When Transmitted Through Band-Pass Channels

Select the square wave mode of the signal generator, set the input signal amplitude to 1V peak-to-peak, and vary the frequency in both directions from the frequency at which you got the resonance and observe the output.

Task 26. *Complete the Table in the Task Sheet with sketches showing relative amplitudes correctly.*

♣ The End ♣

Workshop 2: Baseband Communication

Objective: To simulate and analyze baseband transmission.

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

1. Identify the basic elements of a baseband communication system
2. Implement a simple baseband communication system using MATLAB
3. Simulate and analyze the bit-error rate of a baseband communication system under different settings
4. Implement a simple error-correction mechanism

Equipment Required:

1. A Personal Computer
2. MATLAB software or MATLAB online
3. A headphone set (to be brought by the student)

Components Required: None.

2.1 Baseband Communication

In telecommunications the signals are usually transmitted after converting them to higher frequencies. This process is called modulation and the transmitted signals are known as broadband signals. But the properties of the transmission can be analyzed using a communication system without modulation. Such a system is known as a baseband communication system. Figure 2.1 shows a baseband and a broadband signal in frequency domain side-by-side.

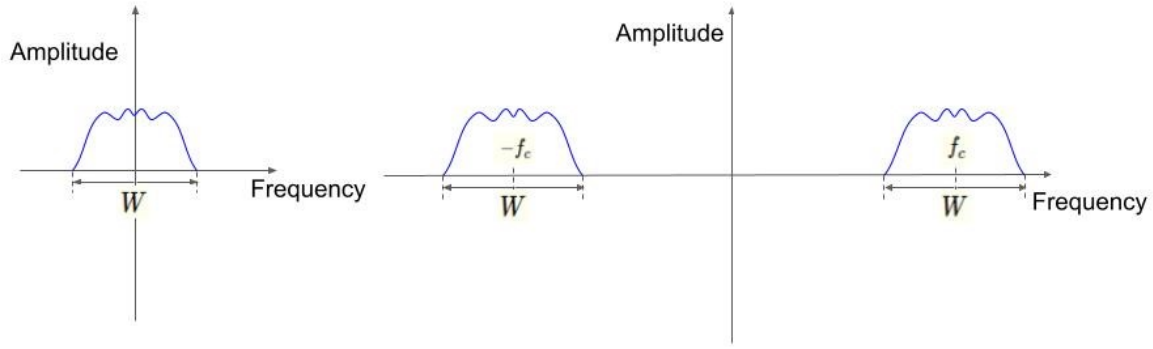


Figure 2.1: Frequency domain representation of a signal in baseband (left) vs broadband (right). Bandwidth of the signal is W and the center frequency of the broadband signal is f_c.

In this practical we will be simulating a baseband communication system using MATLAB and we will be analyzing the bit error rate of the system under different settings. Finally we will study a simple error-correction mechanism.

2.2 Pre-Lab

Prior to the lab we will implement the baseband communication system using MATLAB. You will have to record audio from your headphones. The recorded audio will be tramsitted through the communication system and will be reconstructed at the receiver.

2.2.1 Implementing a Baseband Communication System

Figure 2.2 illustrates the block diagram of a simple baseband communication system.

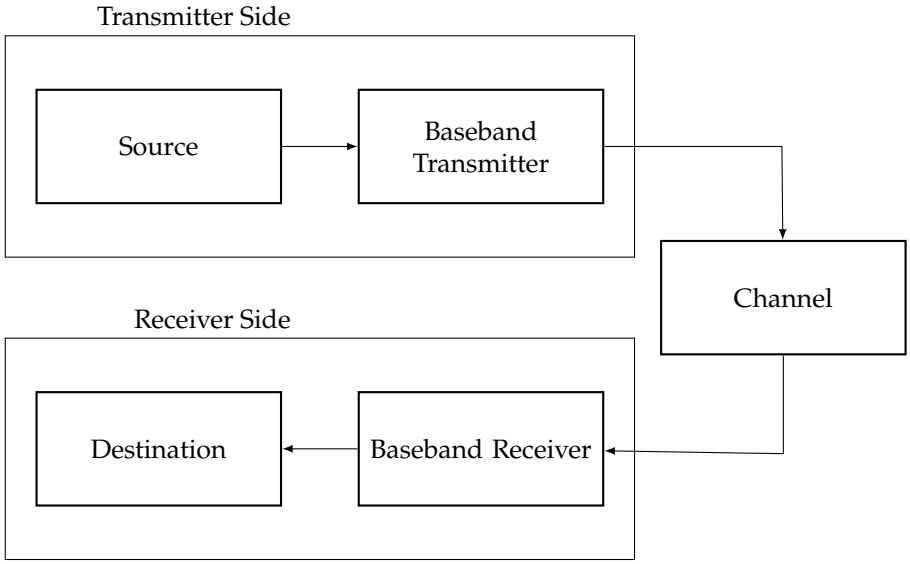


Figure 2.2: The block diagram of a baseband communication system

Next we will understand each block along with the implementation details.

Source

Source is the device/person which generates the signal to be transmitted. In this case you will be the source since you are generating the audio waveform. The analog signal from the source is sent to the transmitter.

Transmitter

Transmitter performs several modifications to the signal received from the source. First, the analog signal is sampled at discrete time intervals (The analog signal is represented as a set of samples). This process is known as **sampling**. Your headphone will perform the sampling.

```
Fs = 6000;  %The sampling frequency used to sample the audio
qbits = 8;   %The number of bits used to encode a single sample of the audio
recObj = audiorecorder(Fs,qbits,1);  %Starting the audio recorder object
disp('Start speaking.')      %Recording audio fro 5 seconds
recordblocking(recObj, 5);
disp('End of Recording.');
audio_samples = getaudiodata(recObj);  %array containing samples of the recorded audio
```

If you have trouble accessing the audio recorders (this can happen in MATLAB online), you can use recorded audio. An audio clip named "Recording.wav" will be uploaded to the Moodle. You can directly

use this audio clip. In case of MATLAB online, upload the file to MATLAB online. Then use the following code instead of the above code.

```
Fs = 6000;  %The sampling frequency used to sample the audio
qbits = 8;   %The number of bits used to encode a single sample of the audio
[audio_samples,Fs] = audioread('Recording.wav'); %Replace filename with the location of your file.
```

If you prefer to record your own audio, first you will have to record a 5 second audio clip from your PC. Then convert it to .wav format using an online converter. You can use https://www.aconvert.com/audio/. Make sure you set the sampling rate to 6000. Rename the file as "Recording.wav" and use it for the workshop.

Next, we have to represent the set of samples using a bit stream in order to transmit as a digital signal. For that, we **quantize** each sample, where we approximate each sample with a discrete level. In this case we represent each sample with one of 256 discrete levels.

```
siz = size(audio_samples);   %Quantizing the audio samples (Each sample is quantized with 8 bits)
size_audio_samples = siz(1);
number_of_bits = qbits*size_audio_samples
bit_stream = zeros([1,number_of_bits]); %array which is used to store the binary representation
                            %of the recorded audio

for i = 1:size_audio_samples
 num = audio_samples(i,1);
 if(num<0)
   bit_stream(1,(i-1)*qbits+1) = 1;
 else
   bit_stream(1,(i-1)*qbits+1) = 0;
 end
 num = abs(num);
 for j = 2:qbits
   num = 2*num;
   if(num>=1)
    bit_stream(1,(i-1)*qbits+j) =  1;
    num = num - 1;
   else
    bit_stream(1,(i-1)*qbits+j) =  0;
   end
 end
end
```

Now we have the binary stream of 1's and 0's which we aim to transmit. Although we transmit bits we still have to represent (**encode**) them using analog signals. We will have to represent bit zero and bit one using two distinct finite duration signals. For simplicity, we use two sinusoidal signals with different phases (phase_0 and phase_1). For this part we will use $\text{phase}_1 = 0$ and $\text{phase}_1 = \pi$ (This formation is called as antipodal signalling since a bit 0 is represented by the negative of the signal used to represent bit1).

$$\begin{aligned} \text{bit0} &\longrightarrow \text{Amplitude} \times \sin(W_c t + \text{phase}_0) \quad 0 \leq t \leq T, \\ \text{bit1} &\longrightarrow \text{Amplitude} \times \sin(W_c t + \text{phase}_1) \quad 0 \leq t \leq T, \end{aligned} \tag{2.1}$$

where T is the bit period.

Although in a practical implementation these signals are implemented in analog domain, in MATLAB we approximate the actual analog signal with a set of samples (We cannot create analog signals in MATLAB).

```
samples_per_bit = 100;  %Number of samples used to approximate the
                %sinosoid for a single bit (We approximate
                % sin(Wc*t+phase0) or sin(Wc*t+phase1) with 100
                % bits in this case
```

```
Amplitude = -5; %Amplitude in decibels
sampling_rate = Fs*qbits*samples_per_bit; %The sampling rate used to represent the
                                %anlog singal in MATLAB
fc = sampling_rate/100; % frequency of the sinosoid
Wc = 2*pi*fc;
delt = 1/sampling_rate; %Sample time period
phase0 = pi;
phase1 = 0;


number_of_samples = number_of_bits*samples_per_bit; %Total number of samples of the transmitted signal
X = zeros([1,number_of_samples]); %X(t) - This array is used to store the transmitted analog signal


%This section does the encoding which is described previously

power = 10^(Amplitude/20); %Signal power in watts.
t_bit = delt:delt:delt*samples_per_bit;
bit1  = sqrt(power)*sin(Wc*t_bit+phase1); %Representation of bit 1
bit0  = sqrt(power)*sin(Wc*t_bit+phase0); %Representation of bit 0
for i = 1:number_of_bits
  if(bit_stream(i)==1)
    X(samples_per_bit*(i-1)+1:samples_per_bit*i) = bit1;
  else
    X(samples_per_bit*(i-1)+1:samples_per_bit*i) = bit0;
  end
end
```

At this point the transmitted signal is stored in the array X and is ready to be transmitted (X is a discrete time approximation of the actual transmitted signal $X(t)$).

Channel

The signals are transmitted through channels (cables for wired transmissions and free space for wireless transmissions). The channel induces several effects to the transmitted signal. Below we model the induced effects.

First, the power of the signal is reduced when it is propagating through the channel. This is known as **attenuation**. Next, channels add **noise** to the transmitted signal. In addition, channels also filter out certain frequencies of the signal (This effect is known as channel **bandwidth limitation**).

The signal after attenuation and noise is

$$\tilde{X}(t) = AX(t) + N(t). \tag{2.2}$$

Here A is the attenuation factor (The proportion of the power of the transmitted signal received at the receiver. $N(t)$ is the noise which is a random signal containing random values. The signal $\tilde{X}(t)$ is filtered (bandlimited) to produce the signal $Y(t)$. We use a low-pass butterworth filter of order 10 to model the filtering effect.

```
att = 0.8;%attenuation factor
mu = 0; %Paramenters of the noise signal (mu, and sigma). We
     % model noise as a Gaussian random variable
sigma = 1;
N = normrnd(mu,sigma,1,samples_per_bit*number_of_bits); %N(t) - noise signal
X_hat = att*X + N; %signal after noise

%Filtering Effect
fc_butter = fc*25;
```

```
[fil_b,fil_a] = butter(10,fc_butter/(sampling_rate/2));
Y = filter(fil_b,fil_a,X_hat);%recived signal after bandwidth limitation (filtering)
```

Receiver

The receiver receives $Y(t)$. Since $Y(t)$ and $X(t)$ are not the same, we have to figure out a method to obtain (**decode**) the transmitted bit stream.

In this case we use phase estimation to perform decoding. Observe that if the transmitted bit was a zero, the phase of the encoding sinosoid was phase_0, and if the transmitted bit was a one, the phase of the encoding sinosoid was phase_1. But due to noise addition and bandwidth limitation the phase may get changed. If the estimated phase is close to phase_0 than phase_1, we interpret the bit as a zero and vice versa. Observe that depending on the strength of the noise signal, bit errors may occur at the receiver.

```
decoded_bit_stream = zeros([1,number_of_bits]); %The array to store the decoded bit stream
H = [cos(Wc*delt*[1:samples_per_bit]'),sin(Wc*delt*[1:samples_per_bit]')]; %Phase estimation matrox

for i = 1:number_of_bits %This loop deals with the phase estimation
  f = inv(H'*H)*H'*Y((i-1)*samples_per_bit+1:i*samples_per_bit)';
  decoded_phase = 0;
  if(f(2) == 0)
    if(f(1) >= 0)
      decoded_phase = pi/2;
    else
      decoded_phase = -pi/2;
    end
  else
    decoded_phase = atan(f(1)/f(2));
    if(f(1)<=0 & f(2)<0)
      decoded_phase = decoded_phase-pi;
    elseif(f(1)>0 & f(2)<0)
      decoded_phase = decoded_phase+pi;
    end
  end
  p0 = min(abs(decoded_phase - phase0),2*pi-abs(decoded_phase - phase0));
  p1 = min(abs(decoded_phase - phase1),2*pi-abs(decoded_phase - phase1));

  if(p0<p1)
    decoded_bit_stream(i) = 0;
  else
    decoded_bit_stream(i) = 1;
  end
end
```

Next, we **reconstruct** the audio samples from the bit-stream.

```
decoded_audio_samples = zeros([size_audio_samples,1]);
for i=1:size_audio_samples
   st = 0.5;
   for j=2:qbits
     decoded_audio_samples(i) = decoded_audio_samples(i)+st*decoded_bit_stream((i-1)*qbits+j);
     st = st/2;
   end
   if(decoded_bit_stream((i-1)*qbits+1) == 1)
      decoded_audio_samples(i) = -decoded_audio_samples(i);
   end
```

```
end
```

Destination

Destination is the device/person to which the signal transmission is intended. In this case you are the destination. Now we will listen to the reconstructed signal.

```
pause(10);
sound(5*decoded_audio_samples, Fs);
```

Change the amplitude of the transmitted signal and observe the impact of amplitude on the quality of the reconstructed signal.

2.2.2 Comparing the Original and the Reconstructed Signals

Now we will plot the original recorded analog audio signal and the signal reconstructed at the receiver, side by side.

```
figure(1)
subplot(1,2,1)
plot(audio_samples)
title('Original Audio Signal')
xlabel('t (s)')
ylabel('Amplitude')
subplot(1,2,2)
plot(decoded_audio_samples);
title('Decoded Audio Signal')
xlabel('t (s)')
ylabel('Amplitude')
```

Task 1. *Compare the two signals and comment on the observed differences.*

2.2.3 Comparing the Original and the Transmitted Signals in Frequency Domain

Use the following code to plot the frequency domain representations of the original recorded analog audio signal and the transmitted signal, side by side.

```
L_1 = size_audio_samples;
Audio_freq = fft(audio_samples);
Audio_freq_norm = abs(Audio_freq/L_1);
Audio_freq_norm_one = Audio_freq_norm(1:L_1/2+1);
Audio_freq_norm_one(2:end-1) = 2*Audio_freq_norm_one(2:end-1);

L_2 = samples_per_bit*number_of_bits;
X_freq = fft(X);
X_freq_norm = abs(X_freq/L_2);
X_freq_norm_one = X_freq_norm(1:L_2/2+1);
X_freq_norm_one(2:end-1) = 2*X_freq_norm_one(2:end-1);

f_1 = Fs*(0:(L_1/2))/L_1;
f_2 = sampling_rate*(0:(L_2/2))/L_2;

figure(2)
```

```
subplot(1,2,1)
plot(f_1,Audio_freq_norm_one);
title('Single–Sided Amplitude Spectrum of the Original Audio Stream');
xlabel('f (Hz)')
ylabel('|P1(f)|')

subplot(1,2,2)
plot(f_2,X_freq_norm_one);
title('Single–Sided Amplitude Spectrum of X(t)')
xlabel('f (Hz)')
ylabel('|P1(f)|')
```

Task 2. *Compare the two frequency spectra and comment on the observed differences*

2.3 Analyzing the Bit-Error Rate (BER) Using a Baseband Communication System

Next we will calculate the bit-error rate of the system.

Task 3. *Write a code snippet to calculate the bit-error rate (BER) for the transmission discussed in section 2.2.1. Include your code at the bottom of the "Pre_Lab.m" file given in the Moodle, and submit the updated MATLAB file with the BER calculation to Moodle. In your code, assign the calculated BER to a variable named "BER". (Hint: The original bit stream is the array "bit_stream" and the decoded bit stream is the array decoded_bit_stream". Your code should calculate the error rate using the difference between these two arrays.)*

Task 4. *What is the calculated bit-error rate?*

Now we will analyze the bit-error rate of a baseband communication system. You will have to vary the signal amplitude and observe it's effect on the bit-error rate. Note that since we keep the noise variance constant, the Signal-to-Noise ratio is proportional to the signal amplitude.

Task 5. *Repeat section 2.2.1 with integer amplitudes ranging from -10 to 10 to 10 dB (Hint: You can use a for loop on the Amplitude variable to achieve this). Plot the Amplitude (dB) vs BER graph in the Task Sheet.*

Task 6. *Repeat the same procedure of task 5 with* $phase_1 = 0$ *and* $phase_1 = \pi/2$ *and plot the graph in the Task Sheet.*

Task 7. *Comment on the differences of the two graphs and the reasons for the difference.*

2.4 Implementing a Simple Error Correction Mechanism

We observed that bit-errors occur due to the channel noise and bandwidth limitation. We can perform error correction at the receiver in order to correct some of the errors occurred. In this section we will be studying a simple error-correction mechanism (Known as an error correcting code).

In this mechanism we will transmit a single bit three times. At the receiver we will observe the three bits and consider the most frequent bit out of the three to be the correct bit.

Replace the code we used for 4.2 with the following code.

```
Fs = 6000;  %The sampling frequency used to sample the audio
qbits = 8;   %The number of bits used to encode a single sample of the audio
recObj = audiorecorder(Fs,qbits,1); %Starting the audio recorder object
disp('Start speaking.')      %Recording audio fro 5 seconds
recordblocking(recObj, 5);
disp('End of Recording.');
```

```
audio_samples = getaudiodata(recObj);  %array containing samples of the recorded audio
siz = size(audio_samples);   %Quantizing the audio samples (Each sample is quantized with 8 bits)
size_audio_samples = siz(1);
number_of_bits = qbits*size_audio_samples
bit_stream = zeros([1,number_of_bits]); %array which is used to store the binary representation
                                %of the recorded audio
for i = 1:size_audio_samples
 num = audio_samples(i,1);
 if(num<0)
   bit_stream(1,(i-1)*qbits+1) = 1;
 else
   bit_stream(1,(i-1)*qbits+1) = 0;
 end
 num = abs(num);
 for j = 2:qbits
    num = 2*num;
    if(num>=1)
      bit_stream(1,(i-1)*qbits+j) =  1;
      num = num - 1;
    else
      bit_stream(1,(i-1)*qbits+j) =  0;
    end
 end
end
samples_per_bit = 100;
Amplitude = -5; %Amplitude in decibels
sampling_rate = Fs*qbits*samples_per_bit; %The sampling rate used to represent the
                                  %anlog singal in MATLAB
fc = sampling_rate/100; % frequency of the sinosoid
Wc = 2*pi*fc;
delt = 1/sampling_rate; %Sample time period
phase0 = pi;
phase1 = 0;


number_of_samples = number_of_bits*samples_per_bit; %Total number of samples of the transmitted signal

X = zeros([1,3*number_of_samples]); %X(t) - This array is used to store the transmitted analog signal
power = 10^(Amplitude/20); %Signal power in watts.
t_bit = delt:delt:delt*samples_per_bit;
bit1  = sqrt(power)*sin(Wc*t_bit+phase1); %Representation of bit 1
bit0  = sqrt(power)*sin(Wc*t_bit+phase0); %Representation of bit 0
for i = 1:number_of_bits
  if(bit_stream(i)==1)
    X(3*samples_per_bit*(i-1)+1:3*samples_per_bit*(i-1)+samples_per_bit) = bit1;
    X(3*samples_per_bit*(i-1)+samples_per_bit+1:3*samples_per_bit*(i-1)+2*samples_per_bit) = bit1;
    X(3*samples_per_bit*(i-1)+2*samples_per_bit+1:3*samples_per_bit*i) = bit1;
  else
    X(3*samples_per_bit*(i-1)+1:3*samples_per_bit*(i-1)+samples_per_bit) = bit0;
    X(3*samples_per_bit*(i-1)+samples_per_bit+1:3*samples_per_bit*(i-1)+2*samples_per_bit) = bit0;
    X(3*samples_per_bit*(i-1)+2*samples_per_bit+1:3*samples_per_bit*i) = bit0;
  end
end

att = 0.8;%attenuation factor
mu = 0; %Paramenters of the noise signal (mu, and sigma). We
      % model noise as a Gaussian random variable
sigma = 1;
N = normrnd(mu,sigma,1,3*number_of_samples); %N(t) - noise signal
```

```
X_hat = att*X + N; %signal after noise
fc_butter = fc*25;
[fil_b,fil_a] = butter(10,fc_butter/(sampling_rate/2));
Y = filter(fil_b,fil_a,X_hat);%recived signal after bandwidth limitation (filtering)
decoded_bit_stream = zeros([1,3*number_of_bits]); %The array to store the decoded bit stream
H = [cos(Wc*delt*[1:samples_per_bit]'),sin(Wc*delt*[1:samples_per_bit]')]; %Phase estimation matrox

for i = 1:3*number_of_bits %This loop deals with the phase estimation
  f = inv(H'*H)*H'*Y((i-1)*samples_per_bit+1:i*samples_per_bit)';
  decoded_phase = 0;
  if(f(2) == 0)
   if(f(1) >= 0)
     decoded_phase = pi/2;
   else
     decoded_phase = -pi/2;
   end
  else
    decoded_phase = atan(f(1)/f(2));
    if(f(1)<=0 & f(2)<0)
      decoded_phase = decoded_phase-pi;
    elseif(f(1)>0 & f(2)<0)
      decoded_phase = decoded_phase+pi;
    end
  end
  p0 = min(abs(decoded_phase - phase0),2*pi-abs(decoded_phase - phase0));
  p1 = min(abs(decoded_phase - phase1),2*pi-abs(decoded_phase - phase1));

  if(p0<p1)
    decoded_bit_stream(i) = 0;
  else
    decoded_bit_stream(i) = 1;
  end
end
decoded_bit_stream_error_corrected = zeros([1,number_of_bits]); %The array to store the decoded bit
                                        %stream after error-correction
for i = 1:number_of_bits %This loop does the task of correcting erros by looking at the most frequent
                  %bit from the three bits
  for j = 1:3
    decoded_bit_stream_error_corrected(i) = decoded_bit_stream_error_corrected(i)+decoded_bit_stream(3*(i-1)+j);
  end
  decoded_bit_stream_error_corrected(i)= round(decoded_bit_stream_error_corrected(i)/3);
end
decoded_audio_samples = zeros([size_audio_samples,1]);
for i=1:size_audio_samples
  st = 0.5;
  for j=2:qbits
    decoded_audio_samples(i) = decoded_audio_samples(i)+st*decoded_bit_stream_error_corrected((i-1)*qbits+j);
    st = st/2;
  end
  if(decoded_bit_stream_error_corrected((i-1)*qbits+1) == 1)
    decoded_audio_samples(i) = -decoded_audio_samples(i);
  end
end
```

Go through the above code and understand how the code for section 2.2.1 is changed to perform error-correction.

Task 8. *Similar to task 3 implement the code to calculate the bit-error rate (Hint: the arrays "bit_stream" and*

"decoded_bit_stream_error_corrected" contain the transmit and received bit streams after error correction). What is the calculated BER?

Task 9. *Repeat task 5 and task 6 with error correction and include the graphs in the Task Sheet.*

Task 10. *Comment on the observations and the effect of using the error-correction mechanism on the bit-error rate.*

Task 11. *What are the disadvantages of using this error-correction mechanism*

♣ The End ♣

Workshop 3: Digital Modulation Schemes

Objective: To identify digital modulation schemes using Matlab

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

1. Demonstrate an understanding of amplitude shift keying (ASK)
2. Demonstrate an understanding of frequency shift keying (FSK)
3. Demonstrate an understanding of phase shift keying (PSK)
4. Develop skills in using Matlab as a tool for communication systems study

Equipment Required:

1. A Personal Computer
2. MATLAB software or MATLAB online

Components Required:

1. None

3.1 Digital Modulation

At this point we understand the basics of digital data transmission. In the previous practical we analyzed a baseband communication system, where one's and zero's are simply represented using two finite duration signals.

In this practical we will be analyzing digital modulation schemes. Digital modulation is the process of encoding a digital information signal into the amplitude, phase, or frequency of the transmitted signal. Hence, there are three major digital modulation techniques.

- Amplitude-Shift Keying (ASK)
- Frequency-Shift Keying (FSK)
- Phase-Shift Keying (PSK)

In simple terms, we will use sinusoids to encode bits or a group of bits. Consider the sinusoid $x(t) = A\sin(2\pi ft + \phi)$, where A is the amplitude, f is the frequency and ϕ is the phase. We can change one of these three properties to obtain distinct finite duration signals, which we use to encode bits or groups of bits.

For example we will consider ASK. Here, we first divide the bit stream into blocks of length n, where n is referred to as the order of the modulation scheme. Observe that each block may have 2^n possible strings. Hence, we will create 2^n sinusoids with distinct amplitudes each of which is mapped to a single string of length n. Then we encode each block with the respective sinusoid. This is known as 2^n-level ASK modulation. In case of FSK and PSK, we change the frequency and the phase of the sinusoids instead of the amplitude to generate the 2^n sinusoids.

3.2 Pre-Lab

We use constellation diagrams to represent digital modulation schemes. Read about constellation diagrams and understand how they are drawn (https://en.wikipedia.org/wiki/Constellation_diagram).

Task 1. *Draw the constellation diagrams of a 2-level ASK schemes where the amplitudes are (1) bit 0 $\longrightarrow$ -1 and bit 1 $\longrightarrow$ 1 (2) bit 0 $\longrightarrow$ 0 and bit 1 $\longrightarrow$ 1 side by side.*

Task 2. *Draw the constellation diagrams of a 2-level PSK schemes where the phases are (1) bit 0 $\longrightarrow$ 0 and bit 1 $\longrightarrow \pi$ (2) bit 0 $\longrightarrow$ 0 and bit 1 $\longrightarrow \pi/2$ side by side.*

3.3 Amplitude-Shift Keying

We will first generate a 2-level ASK. Save the following function as ask.m and change the current directory path in Matlab to the location where you saved this M-File.

```
function []=ask(bit_pattern,n)
Cf = 1.2E6; % Carrier frequency 1.2 MHz;
% We will represent the signal using samples taken at intervals of le−8 S
% i.e., a sampling frequency of 100 MHz
% and a bit rate of 400 kbps i.e. 250 samples per bit
delt = 1E−8;
fs = 1/delt;
samples_per_bit=250;
tmax = (samples_per_bit∗length(bit_pattern)−1)∗delt;
t= 0:delt:tmax; % Time window we are interested in
% Generation of the binary info signal
bits=zeros(1,length(t));
for bit_no=1:1:length(bit_pattern)
 for sample=1:1:samples_per_bit
 bits((bit_no−1)∗samples_per_bit+sample)=bit_pattern(bit_no);
 end
end

% See what it looks like
figure;
subplot(2,1,1);plot(t,bits);
ylabel('Amplitude');
title('Info signal');
axis([0 tmax −2 2]);
% ASK modulation
ASK=[];
if n==2
 for bit_no=1:1:length(bit_pattern)
 if bit_pattern(bit_no)==1
 t_bit = (bit_no1)∗samples_per_bit∗delt:delt:(bit_no∗samples_per_bit−1)∗delt;
 Wc = Cf∗2∗pi∗t_bit;
 mod = (1)∗sin(Wc);
 elseif bit_pattern(bit_no)==0
 t_bit = (bit_no1)∗samples_per_bit∗delt:delt:(bit_no∗samples_per_bit−1)∗delt;
 Wc = Cf∗2∗pi∗t_bit;
 mod = (0)∗sin(Wc);
 end
 ASK=[ASK mod];
 end
 subplot(2,1,2); plot(t,ASK);
 ylabel('Amplitude');
 title('ASK Modulated Signal');
 axis([0 tmax −2 2]);
end
end
```

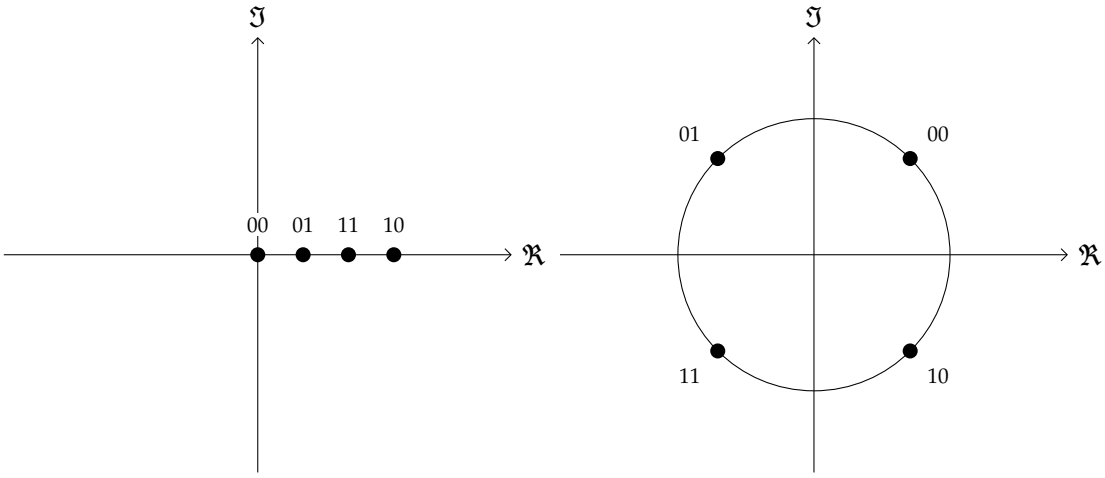
Generate the binary ASK modulation using the following.

```
close all;
clear all;
bit_pattern= [ 1 0 0 1 1 0 1 1 1 0 1 1 0 0 0 1 1 1 1 0 0 0 1 1 ] ;
ask(bit_pattern,2);
```

Observe the binary ASK modulation and show the output to a Laboratory Instructor.

Task 3. *Fill the Table in the Task Sheet based on you observations.*

Task 4. *Extend the function to generate 4-level ASK modulation, as specified by the signal constellation in Figure 3.1a, when the second parameter of the function is set as 4. Show the code and the output to a Laboratory Instructor. Fill the Table in the Task Sheet.*



(a) 4-level ASK (b) QPSK

Figure 3.1: Constellation diagrams

3.4 Frequency-Shift Keying

We will now move onto FSK. Generate the FSK modulation using the following script.

```
%fsk.m
close all;
clear all;
bit_pattern=[ 1 0 0 1 1 0 1 1 1 0 1 1 0 0 0 1 1 1 1 0 0 0 1 1 ];
Cf0 = 0.8E6; % Carrier frequency to encode binary 0, 0.8 MHz;
Cf1 = 2.4E6; % Carrier frequency to encode binary 1, 2.4 MHz;
% We will represent the signal using samples taken at intervals of le−8 S
% i.e., a sampling frequency of 100 MHz
% and a bit rate of 400 kbps i.e. 250 samples per bit
delt=1E−8;
fs=1/delt;
samples_per_bit=250;
tmax = (samples_per_bit*length(bit_pattern)−1)*delt;
t = 0:delt:tmax; %time window we are interested in
% Generation of the binary info signal
bits=zeros(1,length(t));
for bit_no=1:1:length(bit_pattern)
 for sample=1:1:samples_per_bit
 bits((bit_no−1)*samples_per_bit+sample)=bit_pattern(bit_no);
 end
end
% See what it looks like
figure;
subplot(2,1,1); plot(t,bits);
```

```
ylabel ('Amplitude');
title ('Info signal');
axis([0 tmax –2 2]);
% FSK Modulation
FSK=[];
for bit_no=1:1:length(bit_pattern)
 if bit_pattern(bit_no)==1
 t_bit =
(bit_no–1)*samples_per_bit*delt:delt:(bit_no*samples_per_bit–1)*delt;
 Wc = Cf1*2*pi*t_bit;
 mod = (1)*sin(Wc);
 elseif bit_pattern(bit_no)==0
 t_bit =
(bit_no–1)*samples_per_bit*delt:delt:(bit_no*samples_per_bit–1)*delt;
 Wc = Cf0*2*pi*t_bit;
 mod = (1)*sin(Wc);
 end
 FSK=[FSK mod];
end
subplot(2,1,2); plot(t,FSK);
ylabel ('Amplitude');
title ('FSK Modulated Signal');
axis([0 tmax –2 2]);
```

Observe FSK modulation scheme and show the output to a Laboratory Instructor.

Task 5. *Fill the Table in the Task Sheet.*

3.5 Phase-Shift Keying

Finally, we will analyze phase-shift keying. Save the Matlab function given below as npsk.m and change the current directory path in Matlab to the location where you saved this M-File.

```
function []=npsk(bit_pattern,n)
Cf = 4E5; %Carrier frequency 0.4 MHz;
% We will represent the signal as samples taken at intervals of le–8 S
% i.e., a sampling frequency of 100 MHz
delt=1E–8;
fs=1/delt;
samples_per_bit=250;
tmax = (samples_per_bit*length(bit_pattern)–1)*delt;
t = 0:delt:tmax; %time window we are interested in
% Generation of the binary info signal
bits=zeros(1,length(t));
for bit_no=1:1:length(bit_pattern)
 for sample=1:1:samples_per_bit
 bits((bit_no–1)*samples_per_bit+sample)=bit_pattern(bit_no);
 end
end
% See what it looks like
figure;
subplot(2,1,1); plot(t,bits);
ylabel ('Amplitude');
title ('Info signal');
axis([0 tmax –2 2]);
if n==2
 % BPSK Modulation
```

```
BPSK=[];
for bit_no=1:1:length(bit_pattern)
if bit_pattern(bit_no)==1
t_bit = (bit_no1)*samples_per_bit*delt:delt:(bit_no*samples_per_bit-1)*delt;
Wc = Cf*2*pi*t_bit;
mod = (1)*sin(Wc);
elseif bit_pattern(bit_no)==0
t_bit = (bit_no1)*samples_per_bit*delt:delt:(bit_no*samples_per_bit-1)*delt;
Wc = Cf*2*pi*t_bit;
mod = (1)*sin(Wc+pi);
end
BPSK=[BPSK mod];
end
subplot(2,1,2); plot(t,BPSK);
ylabel ('Amplitude');
title ('BPSK Modulated Signal');
axis([0 tmax -2 2]);
end
end
```

Generate the binary PSK modulation (BPSK) using the following code.

```
close all;
clear all;
bit_pattern= [ 1 0 0 1 1 0 1 1 1 0 1 1 0 0 0 1 1 1 1 0 0 0 1 1 ] ;
npsk(bit_pattern,2);
```

Observe BPSK modulation and show the output to a Laboratory Instructor.

Task 6. *Fill the Table in the Task Sheet.*

Task 7. *Extend the function to generate the Quaternary Phase Shift Keying (QPSK) modulation as specified by the signal constellation in Figure 3.1b, when the second parameter of the function is set as 4. Show the code and the output to a Laboratory Instructor. Fill the Table in the Task Sheet.*

Task 8. *Sketch constellation diagrams for 8-level and 16-level PSK, side by side.*

Task 9. *Comment on the advantages and disadvantages of higher order modulation schemes.*

♣ The End ♣

Workshop 4: Communication Networks and Protocols

Objective: To observe and analyze the basic operation of a communications network using packet sniffing.

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

1. Become familiar with the Wireshark packet sniffing tool
2. Be able to appreciate how the layered protocol stack facilitates the functioning of a communications network.

Equipment Required:

1. A Personal Computer with a network interface
2. Internet connectivity
3. Wireshark (the most recent version)
4.

Components Required:

None

This lab is adapted from a series of `Wireshark` Labs provided as a supplement to *Computer Networking: A Top-Down Approach*, 7th ed., J.F. Kurose and K.W. Ross.

4.1 Introduction

Our understanding of communication networks and protocols can often be greatly deepened by "seeing protocols in action" and by "playing around with protocols". In this lab, we will be working with a "real" network environment, and use `Wireshark` to capture data packets that are passing through. You will run various network applications in different scenarios using your own computer and observe the network protocols "in action," interacting and exchanging messages with entities elsewhere in the Internet. Thus, you and your computer will be an integral part of these "live" labs. You'll observe, and you'll learn, by doing.

In this lab, you'll get acquainted with `Wireshark`, make some simple packet captures with it, and gain some insight into how communications networks operate.

4.2 Pre-Lab

Step 1: Please read Annex 4.5: "An Introduction to `Wireshark`" and install the software on your computer using the given instructions.
Step 2: Take `Wireshark` for a Test Run following the instructions given in Annex 4.5: "Wireshark Test Run".
Step 3:

Task 1. *Copy the protocol listing obtained after the last step in the Test Run in the space given in the Task Sheet.*

4.3 Observing the TCP/IP protocol stack

In this lab, you will observe the TCP/IP protocol in action using the common data communication activity of browsing the web. You will also learn some more features of `Wireshark` during this activity. Follow the steps given below.
Step 1: Erase your recent browsing history, execute the Test Run again and answer the following questions. Explain how you obtained your answer in each case.

Task 2. *How long did it take from when each HTTP GET message was sent until the corresponding HTTP OK reply was received? (By default, the value of the Time column in the packet-listing window is the amount of time, in seconds, since* `Wireshark` *tracing began. To display the Time field in time-of-day format, select the* `Wireshark` *View pull down menu, then select Time Display Format, then select Time-of-day.).*

Task 3. *What is the IP address of the gaia.cs.umass.edu (also known as www-net.cs.umass.edu)? What is the IP of your computer?*

Step 2: Select the first HTTP GET message and from the top menu select

Analyze -> Conversation Filter -> TCP.

Task 4. *Sketch the sequence of activities that you see between your computer and the web server at gaia.cs.umass.edu. The Conversation Filter extracts the entire sequence of messages (involving TCP in this case) between the endpoints in the selected message.*

Step 3: Remove the packet filtering.

Task 5. *List 3 different protocols that appear in the protocol column in the packet-listing window.*

Step 4: Using the filter, select only the messages involving your computer (either as source or destination).

Task 6. *What new destinations do you observe? What new protocols do you see in action? What are their purposes?*

4.4 Discussion

Task 7. *Describe how this lab has helped sharpen your interest in communication networks and protocols. Also state how this lab can be improved in future. Use 150 – 300 words.*

4.5 Annex: An introduction to Wireshark

4.5.1 What is a Packet Sniffer ?

The basic tool for observing the messages exchanged between executing protocol entities is called a **packet sniffer**. As the name suggests, a packet sniffer captures ("sniffs") messages being sent/received from/by your computer; it will also typically store and/or display the contents of the various protocol fields in these captured messages. A packet sniffer itself is passive. It observes messages being sent and received by applications and protocols running on your computer, but never sends packets itself. Similarly, received packets are never explicitly addressed to the packet sniffer. Instead, a packet sniffer receives a *copy* of packets that are sent/received from/by application and protocols executing on your machine.

Figure 4.1 shows the structure of a packet sniffer. At the right of Figure 4.1 are the protocols (in this case, Internet protocols) and applications (such as a web browser or ftp client) that normally run on your computer. The packet sniffer, shown within the dashed rectangle in Figure 4.1 is an addition to the usual software in your computer, and consists of two parts. The **packet capture library** receives a copy of every link-layer frame that is sent from or received by your computer. Messages exchanged by higher layer protocols such as HTTP (that we use to browse the web) all are eventually encapsulated in link-layer frames that are transmitted over physical media such as an Ethernet cable or a wireless medium. In Figure 4.1, the assumed physical media is an Ethernet, and so all upper-layer protocols are eventually encapsulated within an Ethernet frame. Capturing all link-layer frames thus gives you all messages sent/received from/by all protocols and applications executing in your computer.

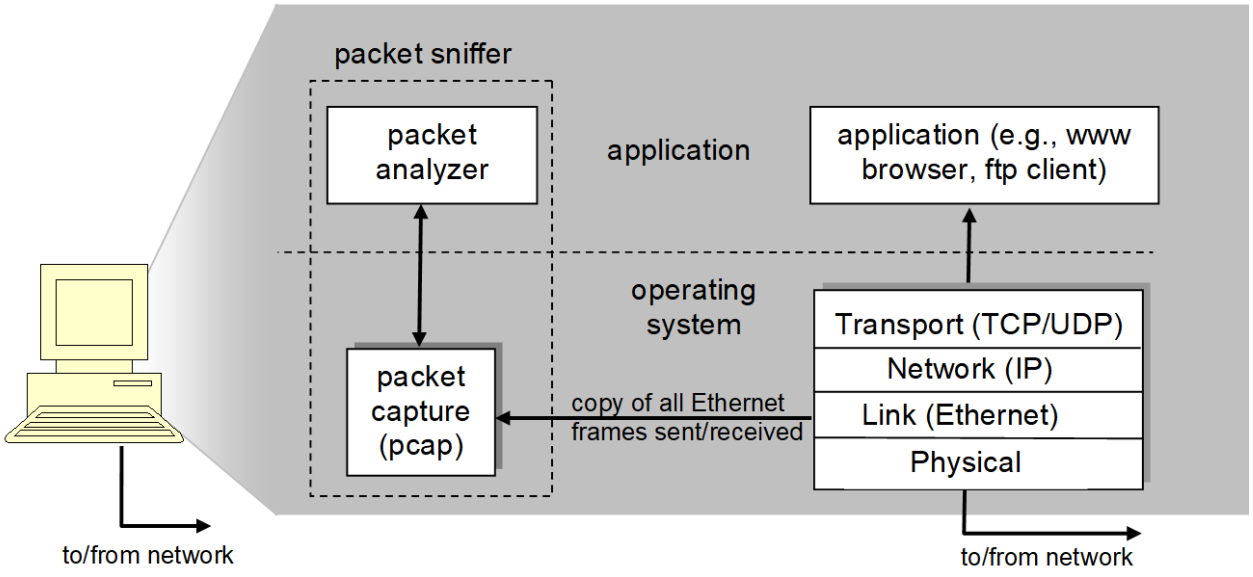


Figure 4.1: Packet Sniffer Structure

The second component of a packet sniffer is the **packet analyzer**, which displays the contents of all fields within a protocol message. In order to do so, the packet analyzer must "understand" the structure of all messages exchanged by protocols. For example, suppose we are interested in displaying the various fields in messages exchanged by the HTTP protocol in Figure 4.1. The packet analyzer understands the format of Ethernet frames, and so can identify the IP datagram within an Ethernet frame. It also understands the IP datagram format, so that it can extract the TCP segment within the IP datagram. Finally, it understands the TCP segment structure, so it can extract the HTTP message contained in the TCP segment. Finally, it understands the HTTP protocol and so, is able to identify the contents of HTTP message.

4.5.2 Wireshark

`Wireshark` is a packet sniffer http://www.wireshark.org/ which allows us to display the contents of messages being sent/received from/by protocols at different levels of the protocol stack. (Technically speaking, `Wireshark` is a packet analyzer that uses a packet capture library in your computer). `Wireshark` is a free network protocol analyzer that runs on Windows, Mac, and Linux/Unix computer. It's an ideal packet analyzer for our labs – it is stable, has a large user base and well-documented support that includes a user-guide (http://www.wireshark.org/docs/wsug_html_chunked/), man pages (http://www.wireshark.org/docs/man-pages/), and a detailed FAQ (http://www.wireshark.org/faq.html), rich functionality that includes the capability to analyze hundreds of protocols, and a well-designed user interface.

4.5.3 Installing Wireshark

In order to run `Wireshark`, you will need to have access to a computer that supports both `Wireshark` and the *libpcap* or *WinPCap* packet capture library. The *libpcap* software will be installed for you, if it is not installed within your operating system, when you install `Wireshark`. See http://www.wireshark.org/download.html for a list of supported operating systems and download sites

Download and install the `Wireshark` software:

- Go to http://www.wireshark.org/download.html and download and install the `Wireshark` binary for your computer.

4.5.4 Running Wireshark

When you run the `Wireshark` program, you'll get a startup screen that looks similar to Figure 4.2. (Note: Different versions of `Wireshark` will have different startup screens.) In the Capture section of the screen, there is a list of communication interfaces available in the computer. To capture packets, you need to first select an interface from this list.

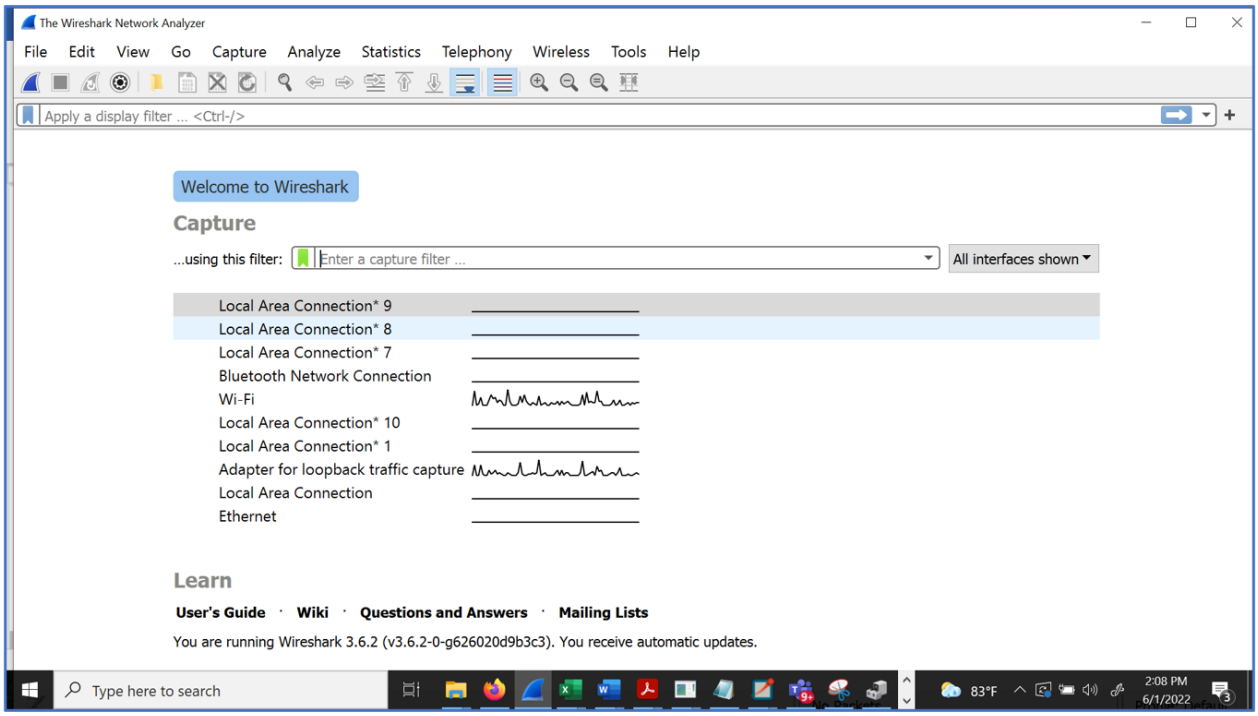


Figure 4.2: The Wireshark start-up screen

When you click on one of these interfaces to start packet capture (i.e., for `Wireshark` to begin capturing all packets being sent to/from that interface), a screen like the one in Fig. A16.1.3 will be displayed, showing information about the packets being captured. Once you start packet capture, you can stop it by using the *Capture* pull down menu and selecting *Stop*.

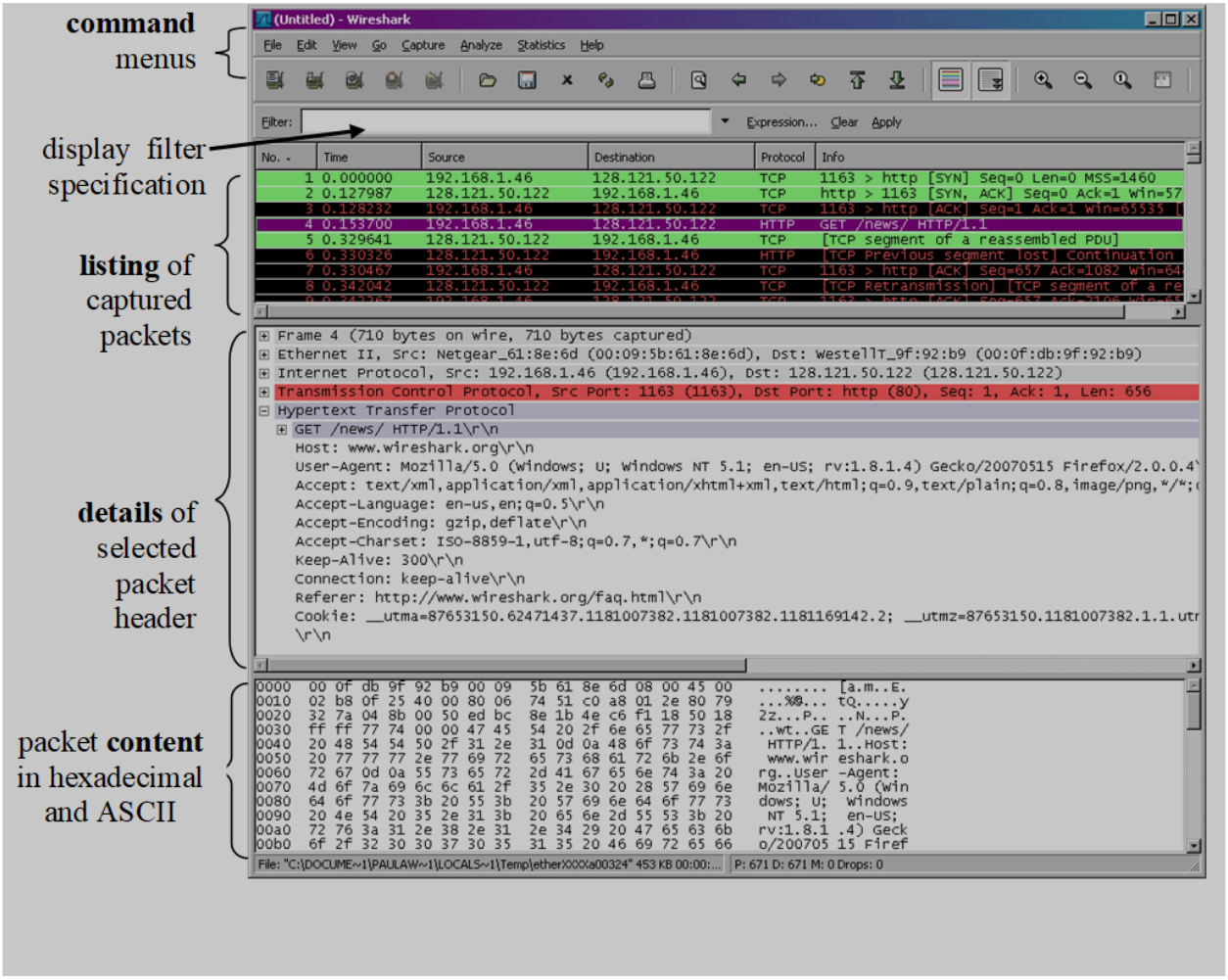


Figure 4.3: Wireshark Graphical User Interface during packet capture and analysis

The `Wireshark` interface has five major components as shown in Figure 4.3. These are described below:

- The **command menus** are standard pulldown menus located at the top of the window. Of interest to us now are the File and Capture menus. The File menu allows you to save captured packet data or open a file containing previously captured packet data, and exit the `Wireshark` application. The Capture menu allows you to begin packet capture.
- The **packet-listing window** displays a one-line summary for each packet captured, including the packet number (assigned by `Wireshark`; this is *not* a packet number contained in any protocol's header), the time at which the packet was captured, the packet's source and destination addresses, the protocol type, and protocol-specific information contained in the packet. The packet listing can be sorted according to any of these categories by clicking on a column name. The protocol type field lists the highest-level protocol that sent or received this packet, i.e., the protocol that is the source or ultimate sink for this packet.
- The **packet-header details window** provides details about the packet selected (highlighted) in the packet-listing window. (To select a packet in the packet-listing window, place the cursor over the

packet's one-line summary in the packet-listing window and click with the left mouse button.) These details include information about all the protocols over which the packet has been carried, up to the highest level.

- The **packet-contents** window displays the entire contents of the captured frame, in both ASCII and hexadecimal format.
- Towards the top of the `Wireshark` graphical user interface, is the **packet display filter field**, into which a protocol name or other information can be entered in order to filter the information displayed in the packet-listing window (e.g. to select and display only packets using a particular protocol).

4.6 Annex: A Wireshark Test Run

Follow the steps given below to complete your first run with `Wireshark`. You will access a web page, and examine the protocols that enable you to do so.

1. Start up your web browser, which will display your selected homepage.
2. Start `Wireshark`.
3. To begin packet capture, select the Capture pull down menu and select *Options*. This will cause the "Wireshark: Capture Options" window to be displayed with the available interfaces as shown in Figure 4.1 earlier.
4. Double-click on the interface that you wish to capture packets on (i.e., the network interface you use for Internet access). Packet capture will now begin - `Wireshark` is now capturing all packets being sent/received from/by your computer! Once you begin packet capture, a window similar to that shown in Figure 4.3 will appear. This window shows the packets being captured. By selecting `Capture` pulldown menu and selecting *Stop*, you can stop packet capture. But don't stop packet capture yet.
5. Let's capture some interesting packets now. To do so, we'll need to generate some network traffic. We will look at the HTTP protocol that is used to download content from a website. While `Wireshark` is running, enter the URL: `http://gaia.cs.umass.edu/wireshark-labs/INTRO-wireshark-file1.html` and have that page displayed in your browser.

   In order to display this page, your browser will contact the HTTP server at gaia.cs.umass.edu and exchange HTTP messages with the server in order to download this page. The Ethernet frames containing these HTTP messages (as well as all other frames passing through your Ethernet adapter) will be captured by `Wireshark`.
6. After your browser has displayed the INTRO-wireshark-file1.html page (it is a simple one line of congratulations), stop `Wireshark` packet capture. The main window should now look similar to Figure 4.3.
7. You now have live packet data that contains all protocol messages exchanged between your computer and other network entities! The HTTP message exchanges with the gaia.cs.umass.edu web server should appear somewhere in the listing of packets captured. But there will be many other types of packets displayed as well (see, e.g., the many different protocol types shown in the *Protocol* column in Figure 4.3). Even though the only action you took was to download a web page, there were evidently many other protocols running on your computer that are unseen by the user. We'll learn much more about these protocols as we progress through the text! For now, you should just be aware that there is often much more going on than "meet's the eye"!
8. Type in "http" (without the quotes, and in lower case – all protocol names are in lower case in `Wireshark`) into the display filter specification window at the top of the main `Wireshark` window. Then select *Apply* (to the right of where you entered "http"). This will cause only HTTP message to be displayed in the packet-listing window.

♣ The End ♣

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

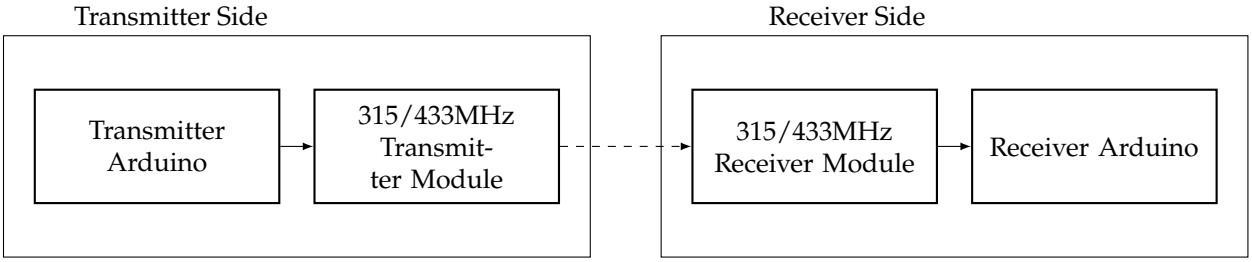


Figure 5.1: The block diagram of the point to point communication system we will be implementing

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.

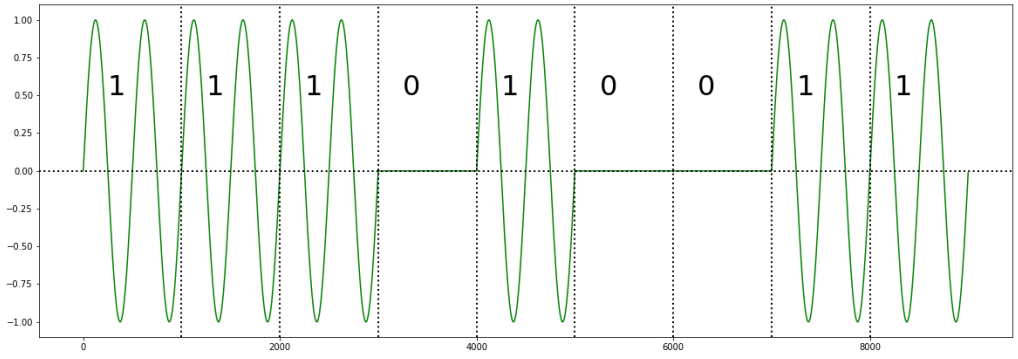


Figure 5.2: OOK waveform for the bit stream "111010011

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.

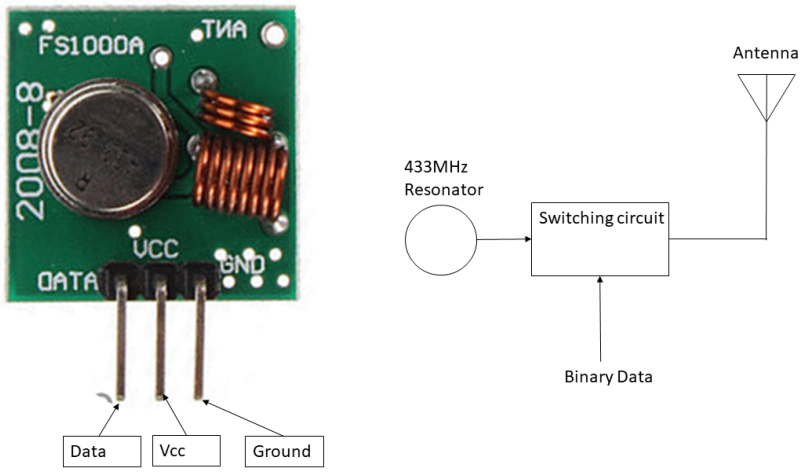


Figure 5.3: The transmitter module

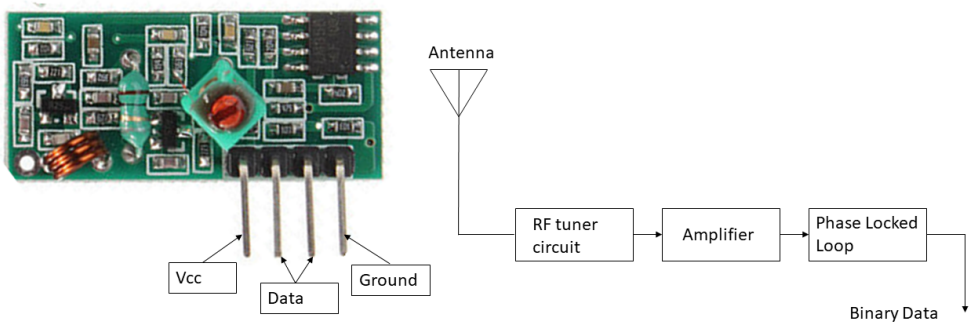


Figure 5.4: The receiver module

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.

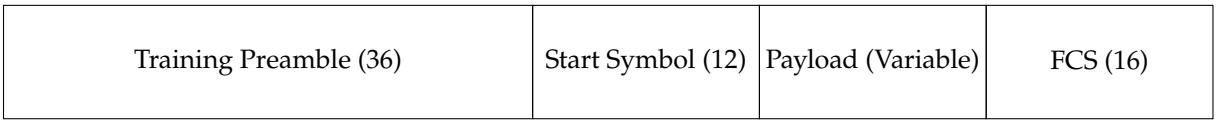


Figure 5.5: The RadioHead Packet

‘

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.

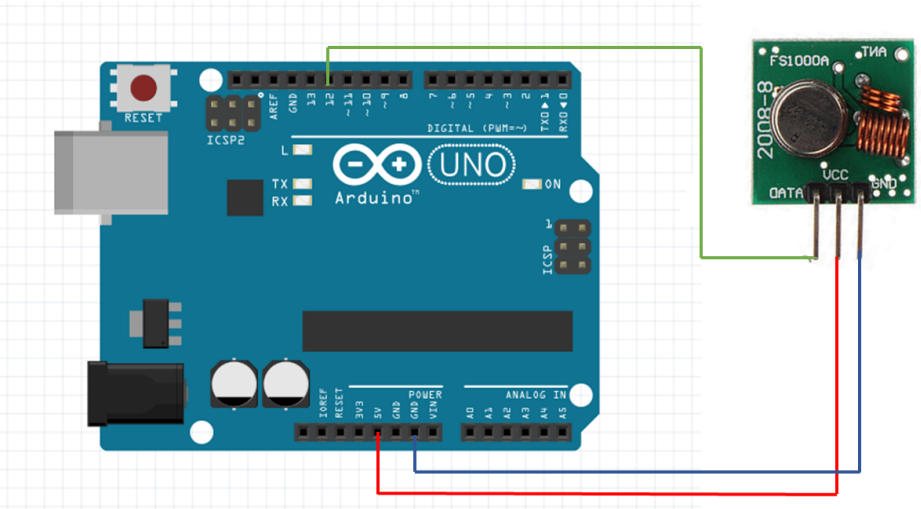


Figure 5.6: Configuration of the transmitter side

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).

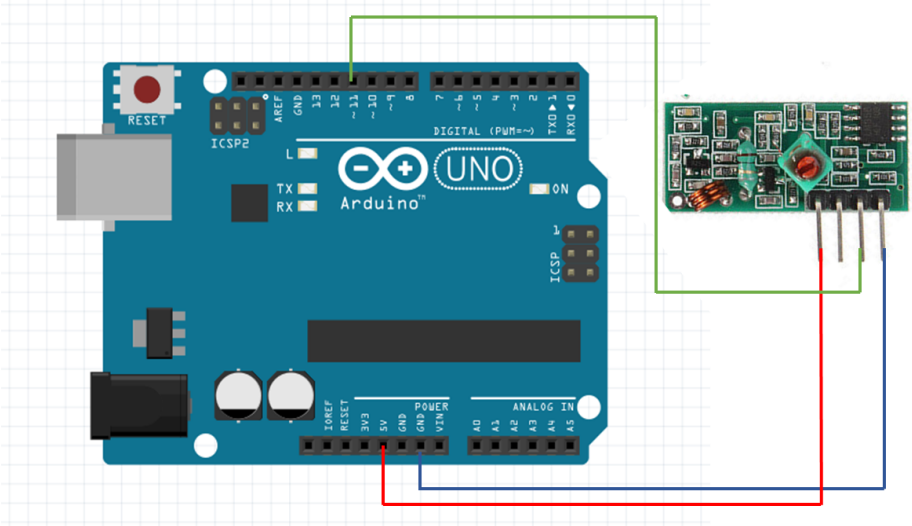


Figure 5.7: Configuration of the receiver side

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 ♣

Part V

Task Sheets

EN1094 Laboratory Practice
Orientation - Workshop 1 Task Sheet

Index No.: **Group No.:**

Date:

Task 1.

Label	1st band	2nd band	3rd band	4th band	Value	Tolerance
R_1						
R_2						
R_3						
R_4						

Task 2.

Label	Printed value/identifier	Capacitance	Type
C_1			
C_2			
C_3			
C_4			

Task 3.

Label	Analog MM reading	Analog MM scale	Digital MM reading	Digital MM scale
R_1				
R_2				
R_3				
R_4				

Task 4.

R_f =_____ R_r =_____ Is the diode working properly? _____

Task 5.

Task 6.

Task 7.

Forward Biased		Reversed Bias	
R_{BE}	R_{BC}	R_{BE}	R_{BC}

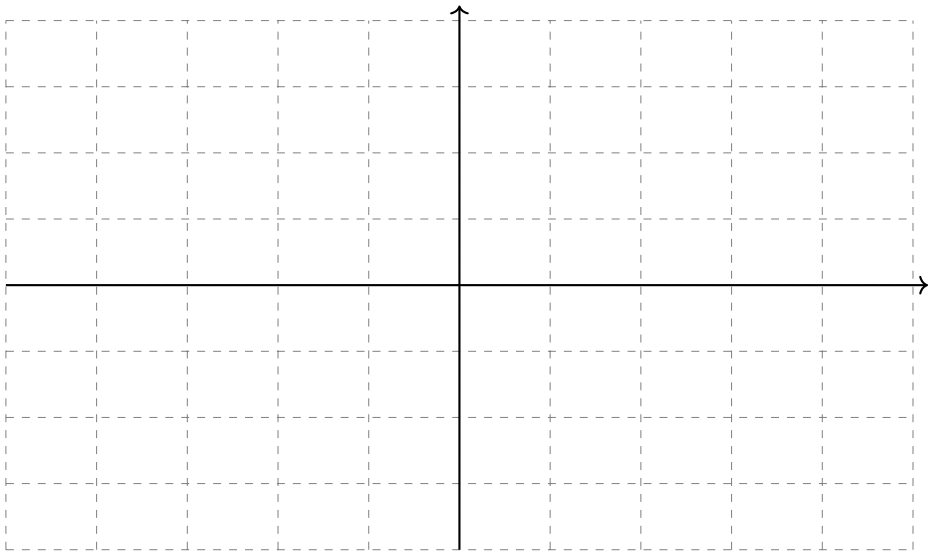
Task 8.

a) V_{CE} =_____ Is the LED on?_____

b) V_{CE} =_____ Is the LED on?_____

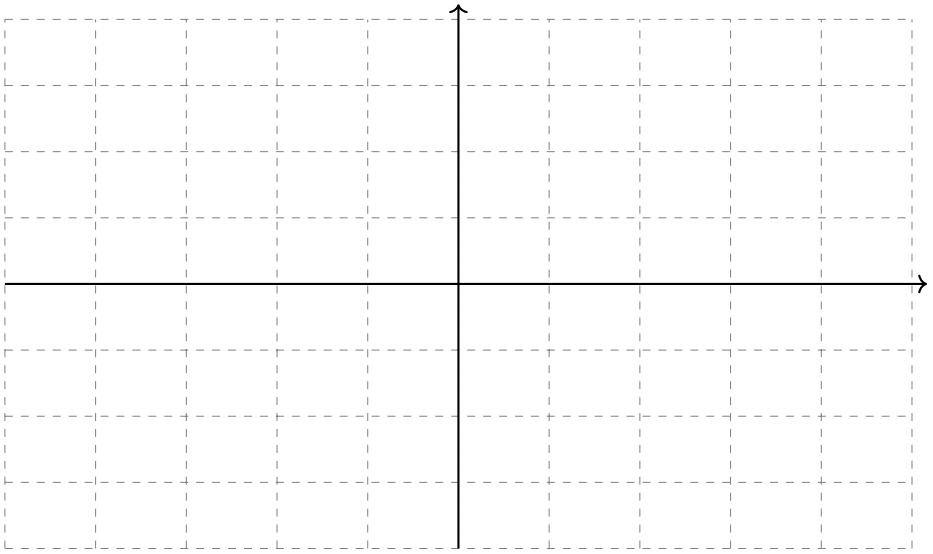
c) I_B =_____ I_C =_____

Task 9.

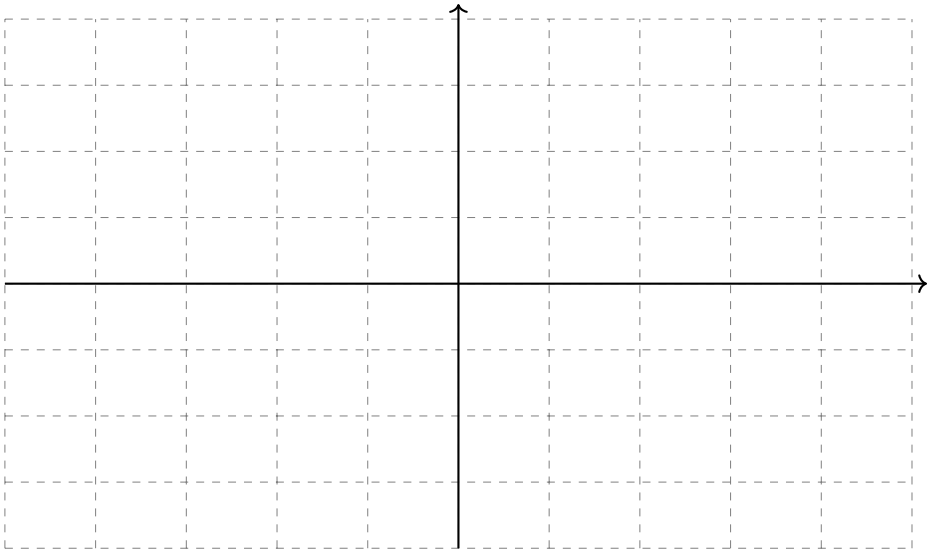


Task 10.
V_{pp} =_____

Task 11.
Period = _____



Task 12.



Task 13.

V_{RMS} =_____

Task 14.

EN1094 Laboratory Practice
Signals Circuits and Systems - Workshop 4 Task Sheet

Index No.: **Group No.:**

Date:

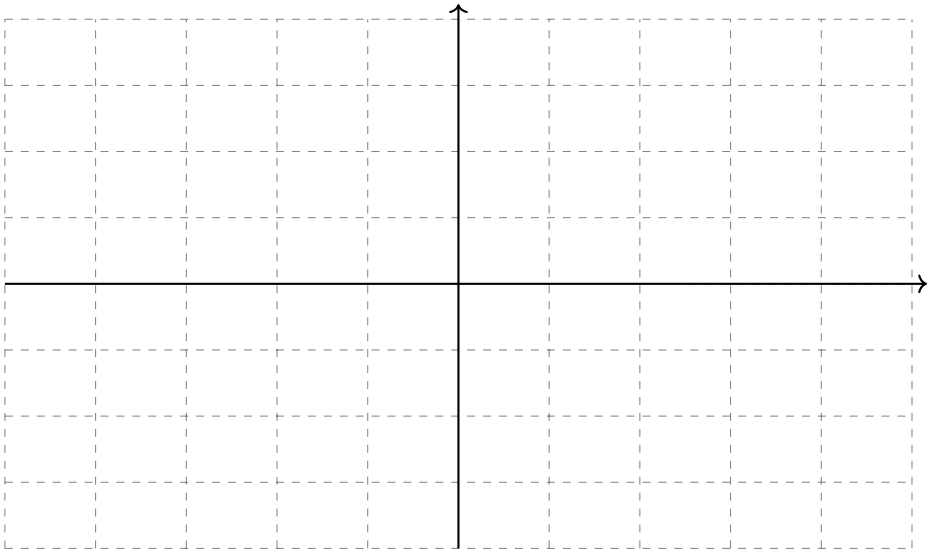
Task 1.

Task 2.

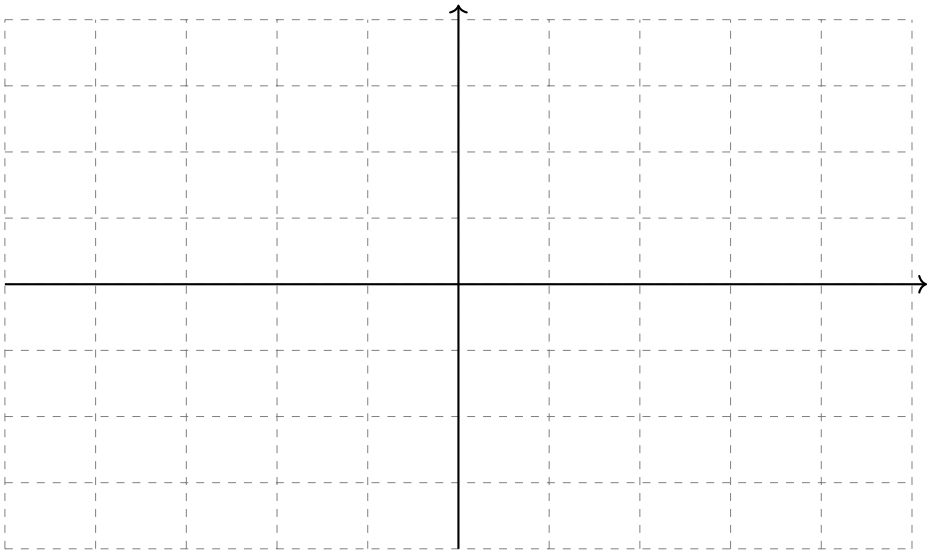
Task 3.

Task 4.

Task 6.



Task 8.



Task 10.

Frequency	Input amplitude	Output amplitude	Voltage gain
1 kHz			
10 kHz			
100 kHz			

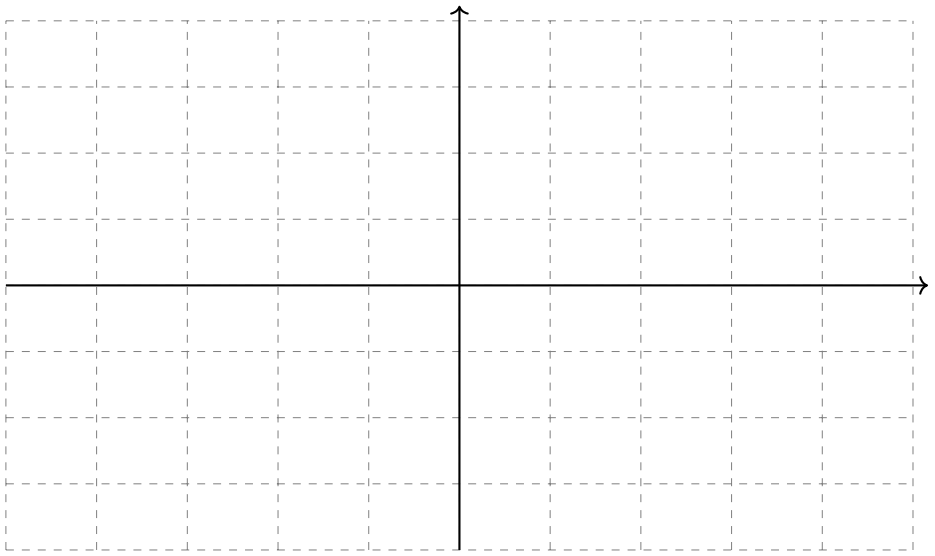
Task 11.

Task 12.

Task 14.

Frequency	Input amplitude	Output amplitude	Voltage gain
1 kHz			
10 kHz			
100 kHz			

Task 15.



Task 16.

Task 18.

Frequency	Input amplitude	Output amplitude	Voltage gain
1 kHz			
10 kHz			
100 kHz			

Task 19.

Task 21.

Frequency	Input amplitude	Output amplitude	Voltage gain
1 kHz			
10 kHz			
100 kHz			

Task 22.

Task 24.

Frequency	Input amplitude	Output amplitude	Voltage gain
1 kHz			
10 kHz			
100 kHz			

Task 25.

Task 26.

$\hat{f}$ =_____

EN1094 Laboratory Practice
Signals Circuits and Systems - Workshop 5 Task Sheet

Index No.: **Group No.:**

Date:

Task 1.

Task 2.

$f = 1\ kHz$

$f = 10\ kHz$

Task 3.

Task 4.

$f = 10\ kHz$

$f = 100\ kHz$

Task 5.

Task 6.

$f = 1\ kHz$

$f = 10\ kHz$

Task 7.

$f = 1\ kHz$

$f = 10\ kHz$

Task 8.

Task 9.

Task 10.

$f = 10\ kHz$

$f = 100\ kHz$

Task 11.

Task 12.

$f = 1\ kHz$

$f = 10\ kHz$

Task 13.

Task 14.

EN1094 Laboratory Practice
Signals Circuits and Systems - Workshop 6 Task Sheet

Index No.: **Group No.:**

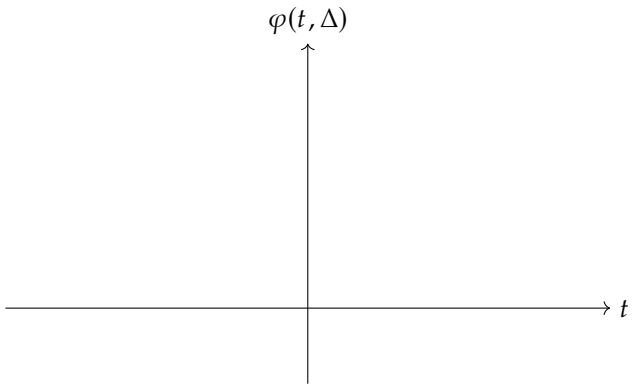
Date:

Task 1.

Task 2.

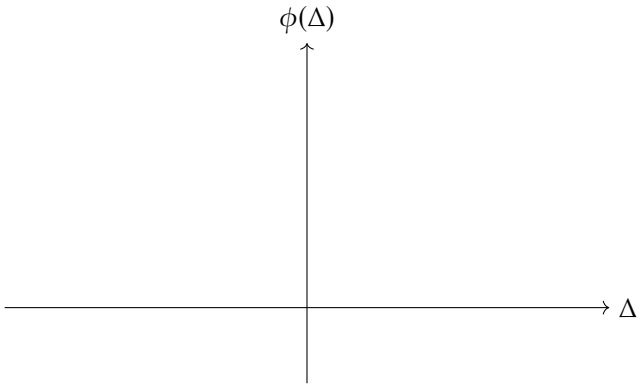
Task 3.

Task 4.



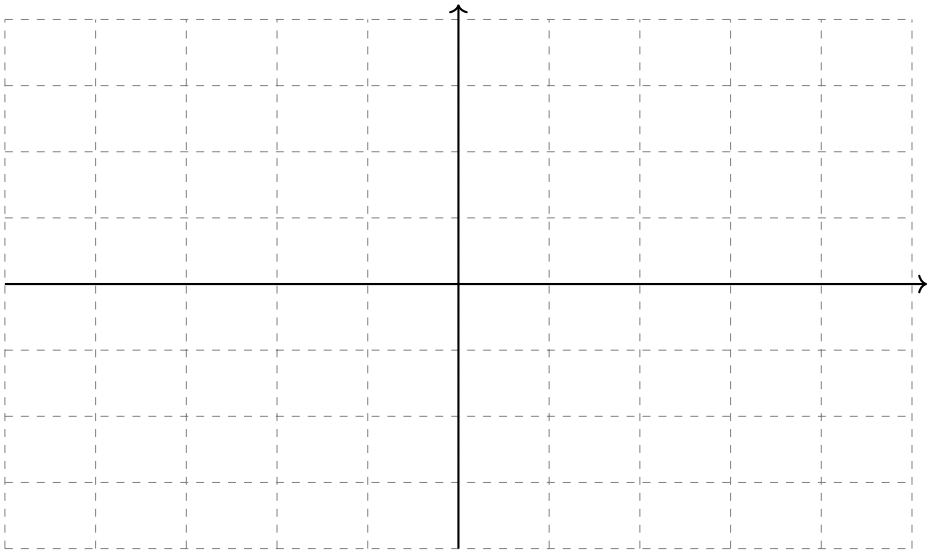
Task 5.

Task 6.



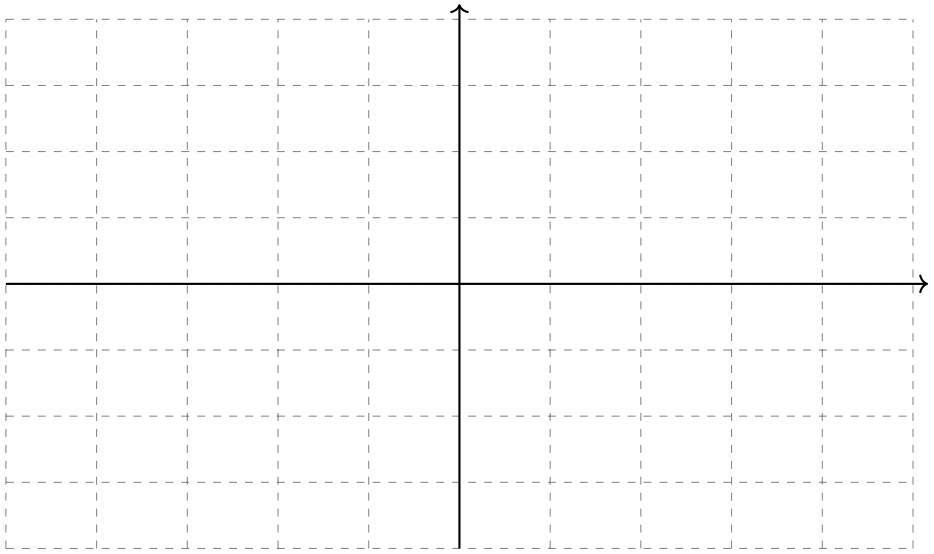
Task 7.

Task 9.

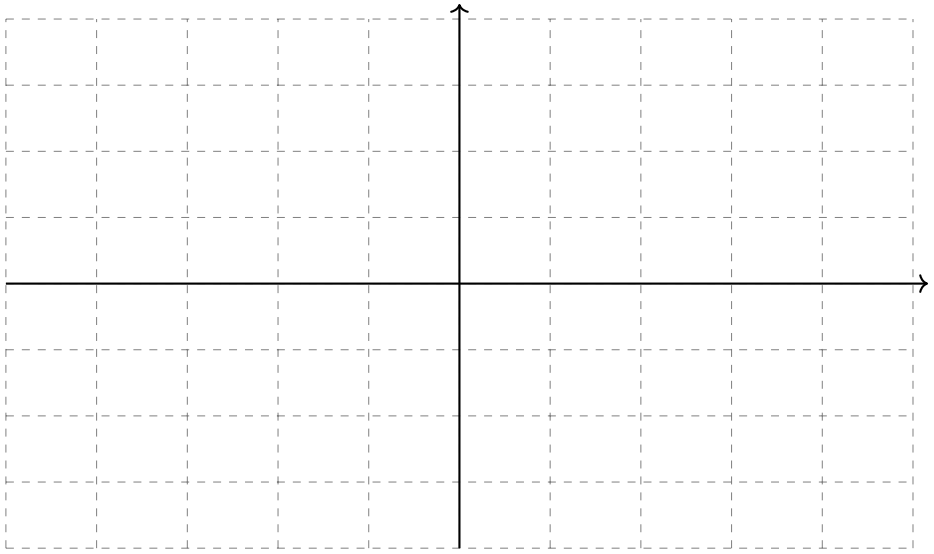


Task 10.

Task 11.

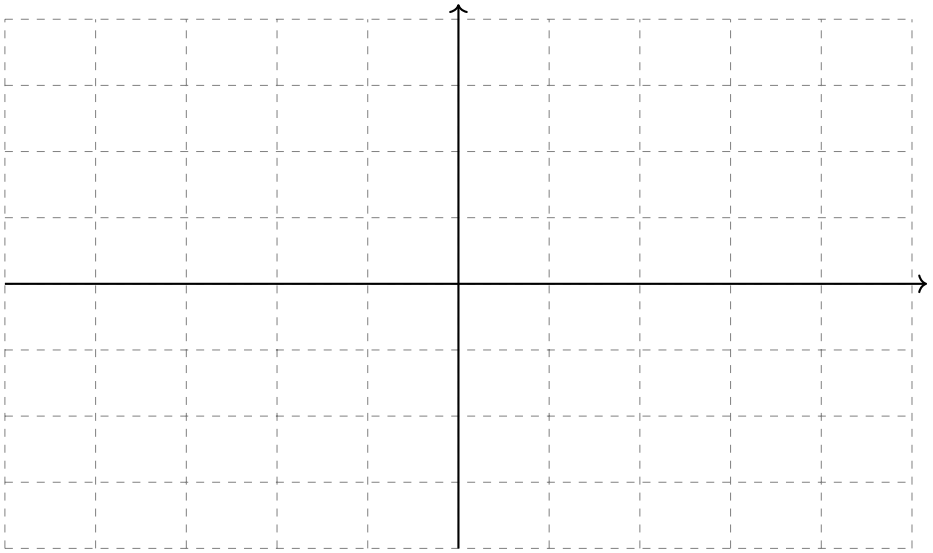


Task 13.



Task 14.

Task 16.

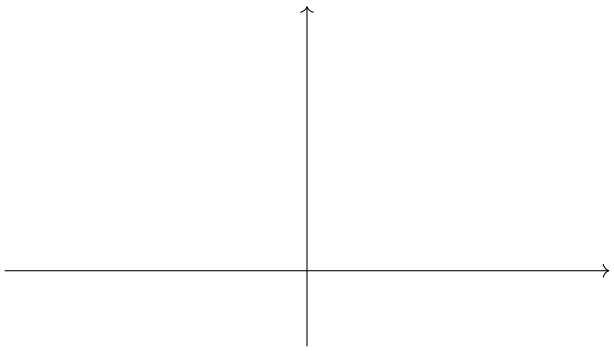


Task 17.

Task 18.

Pulse width	Amplitude	Offset	Pulse duration	Peak output voltage
10%	0.5 V	0.25 V		
5%	1 V	0.5 V		
2%	2.5 V	1.25 V		
1%	5 V	2.5 V		

Task 19.



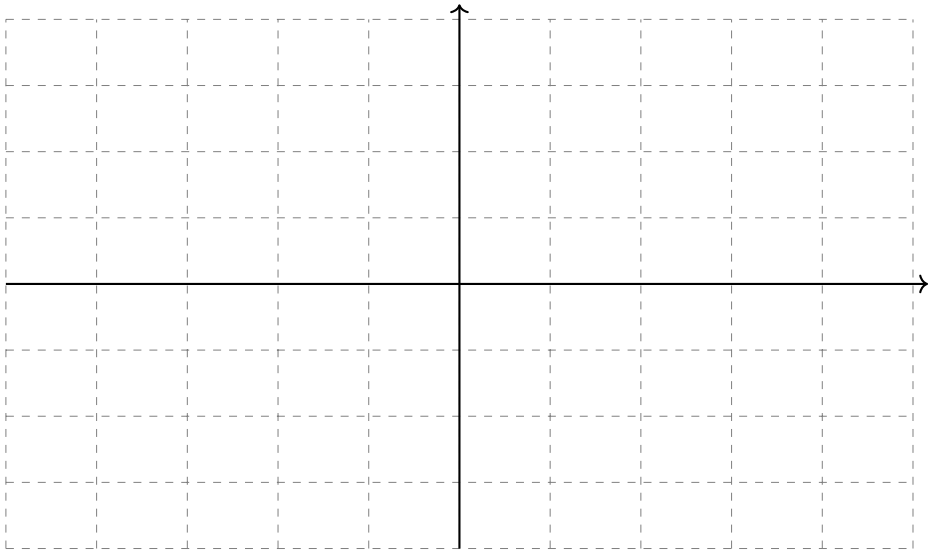
EN1094 Laboratory Practice
Electronics - Workshop 1 Task Sheet

Index No.: **Group No.:**

Date:

Task 1.

Task 3.

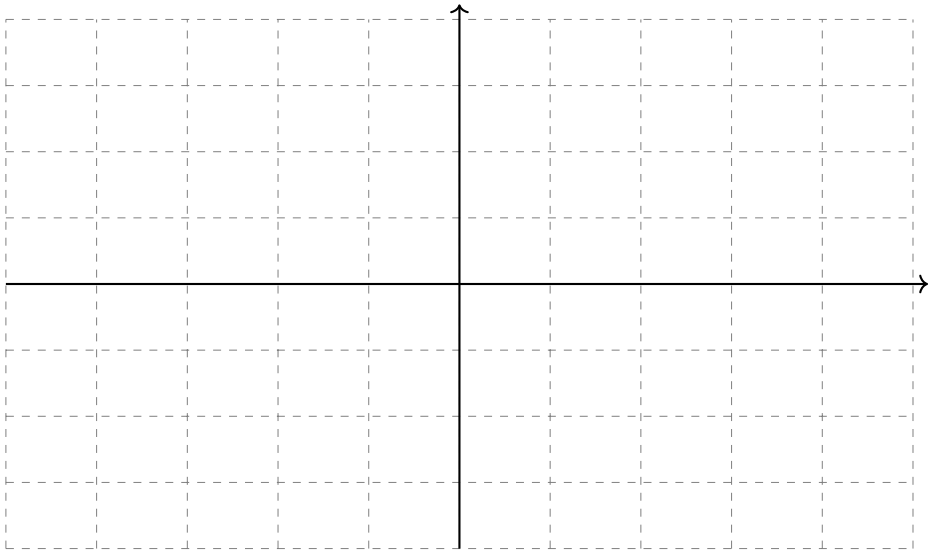


Task 4.

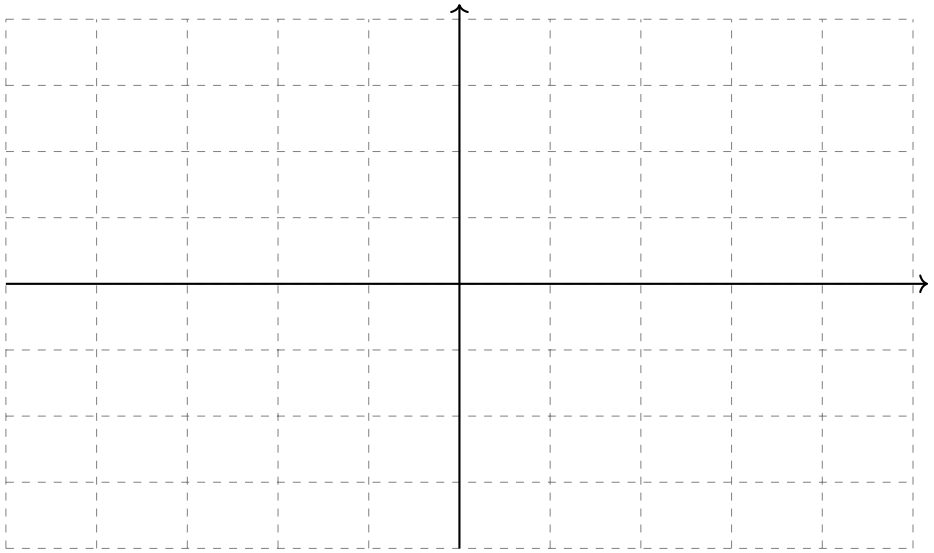
Task 5.

Task 6.

Task 7.



Task 8.



Task 9.

Task 10.

Task 13.

Task 15.

Task 18.

Task 20.

Task 21.

EN1094 Laboratory Practice
Electronics - Workshop 2 Task Sheet

Index No.: **Group No.:**

Date:

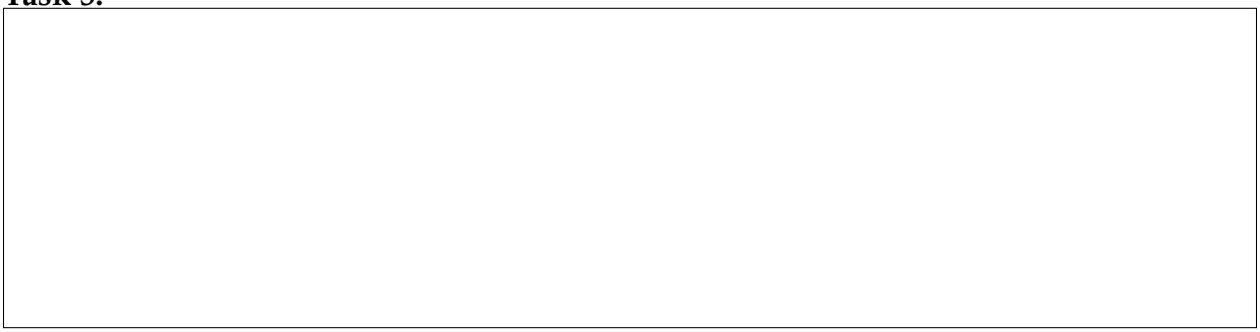
Task 1.

Task 2.

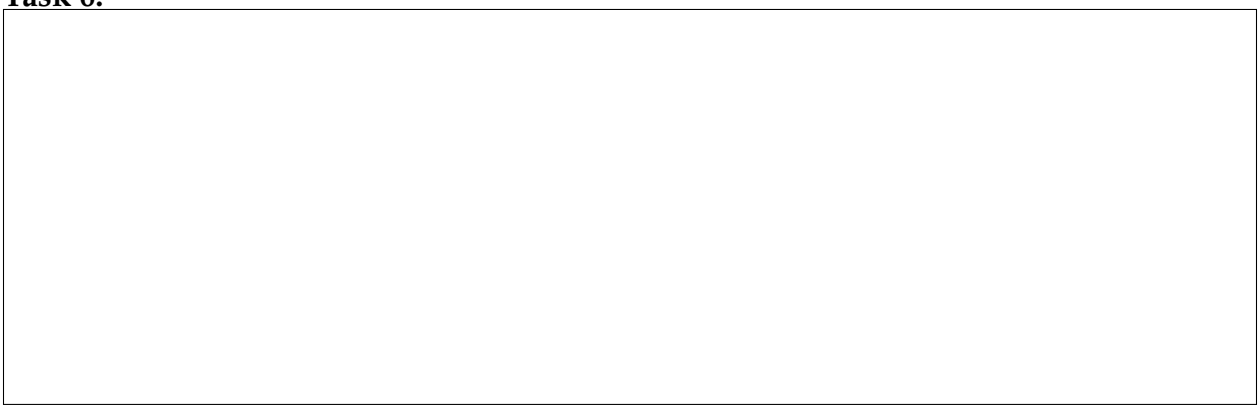
Task 3.

Task 4.

Task 5.

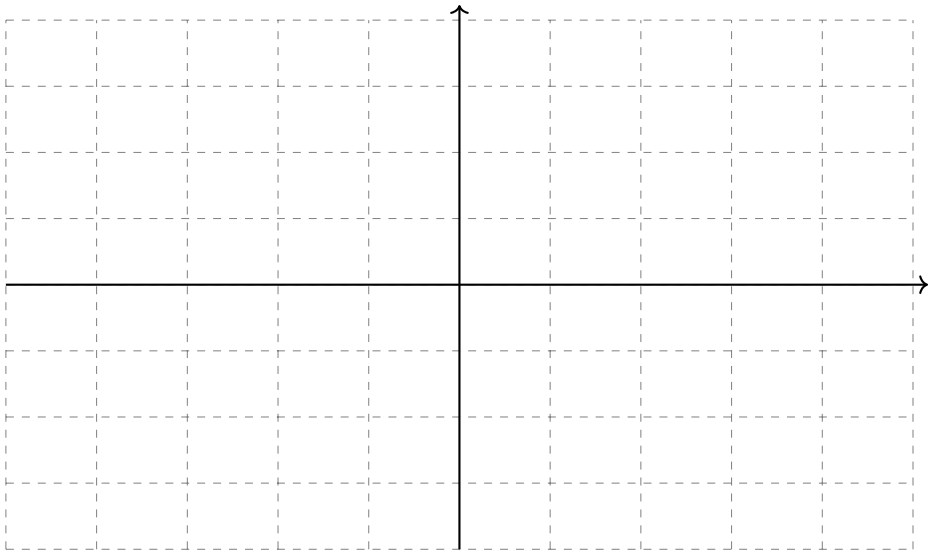


Task 6.



Task 7.

Task 9.

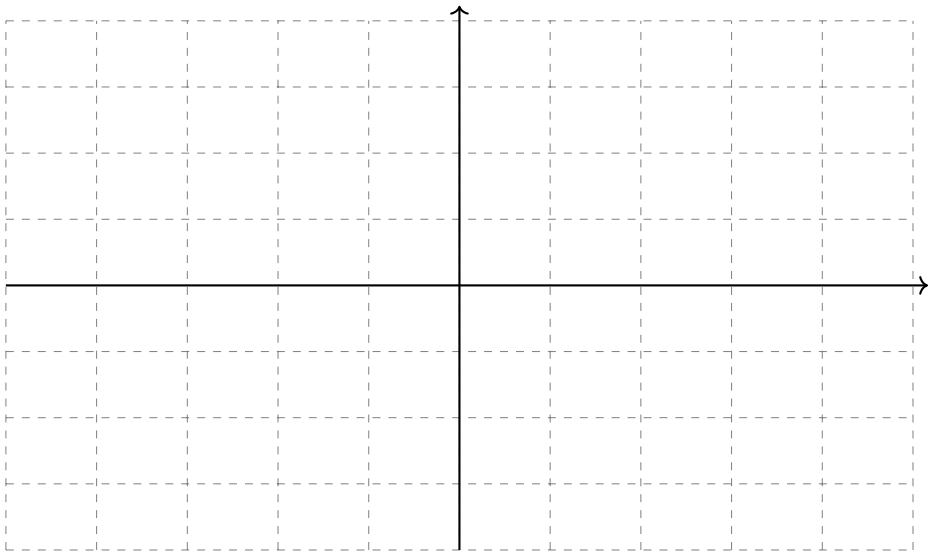


Task 10.

Task 11.

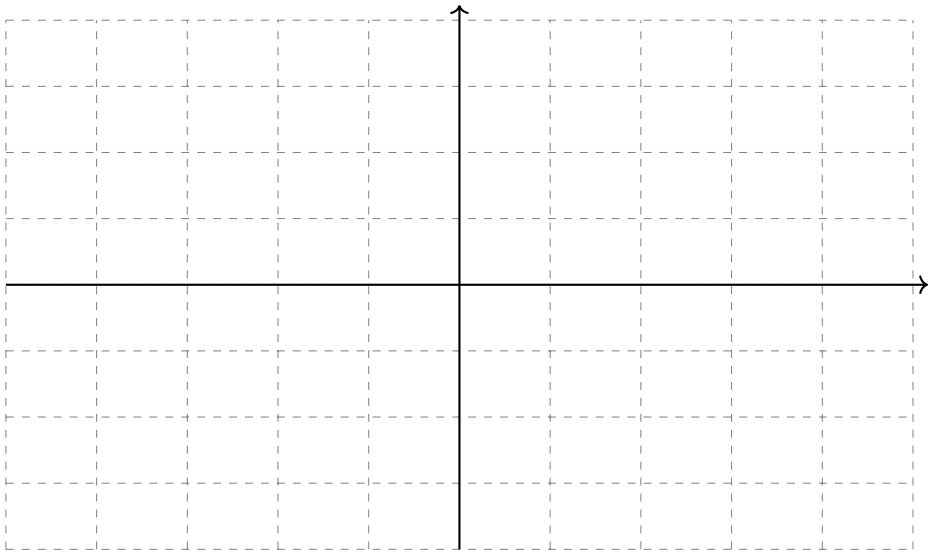
Task 13.

Task 14.

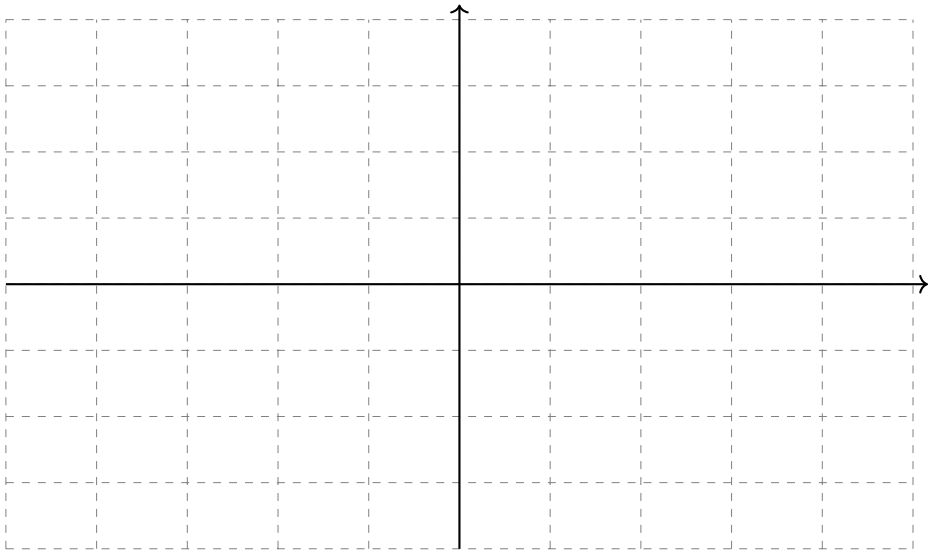


Task 15.

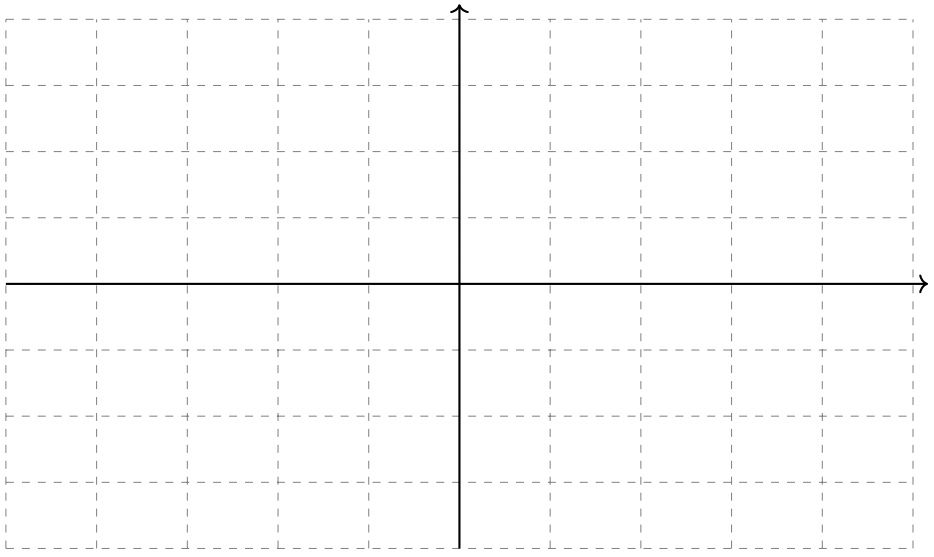
Task 21.



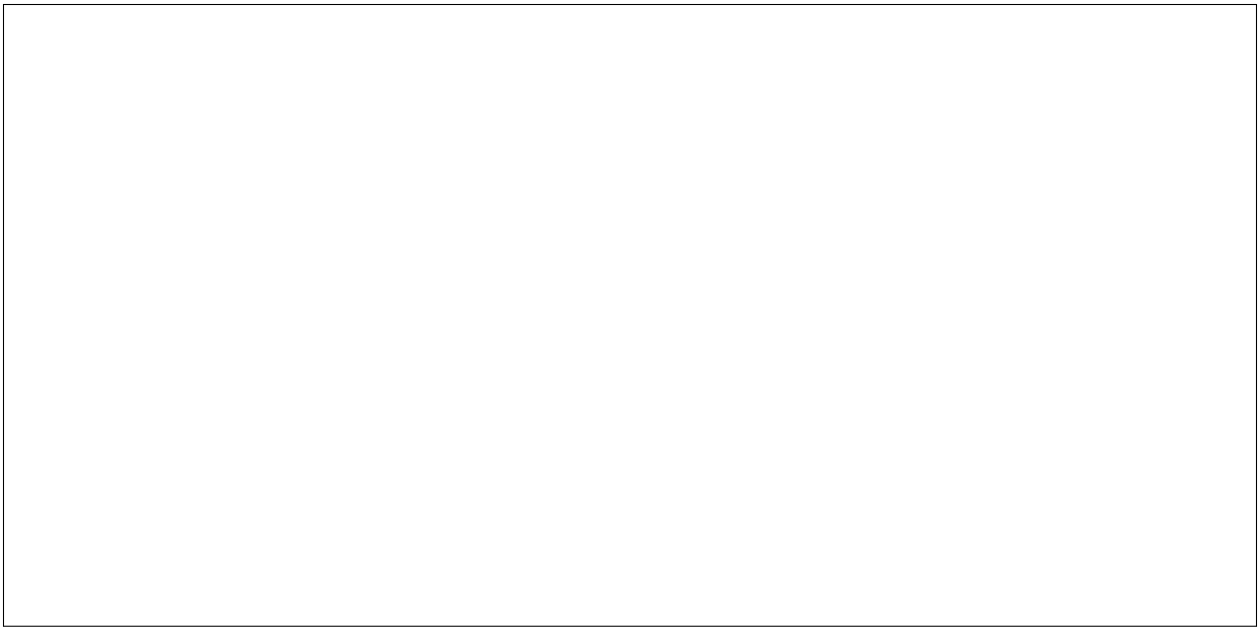
Task 23.



Task 24.



Task 25.

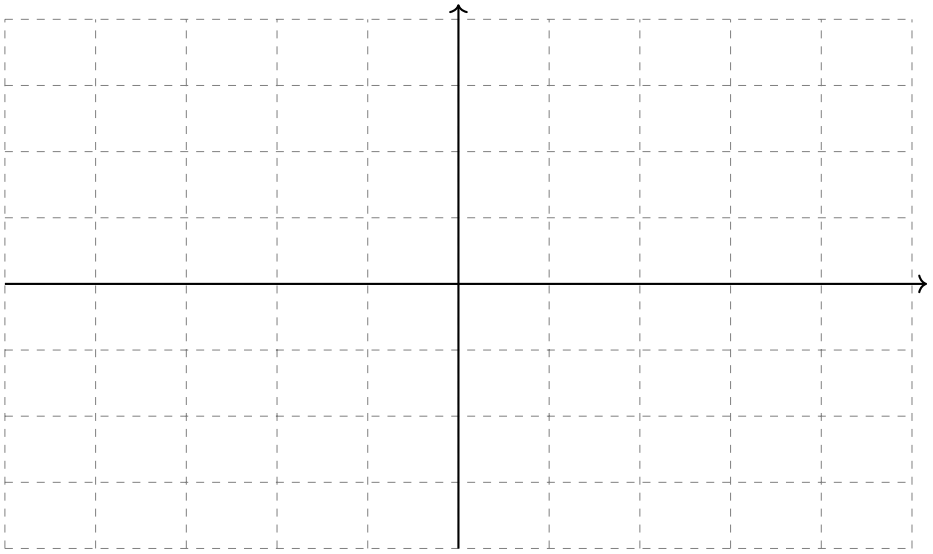


EN1094 Laboratory Practice
Electronics - Workshop 3 Task Sheet

Index No.: **Group No.:**

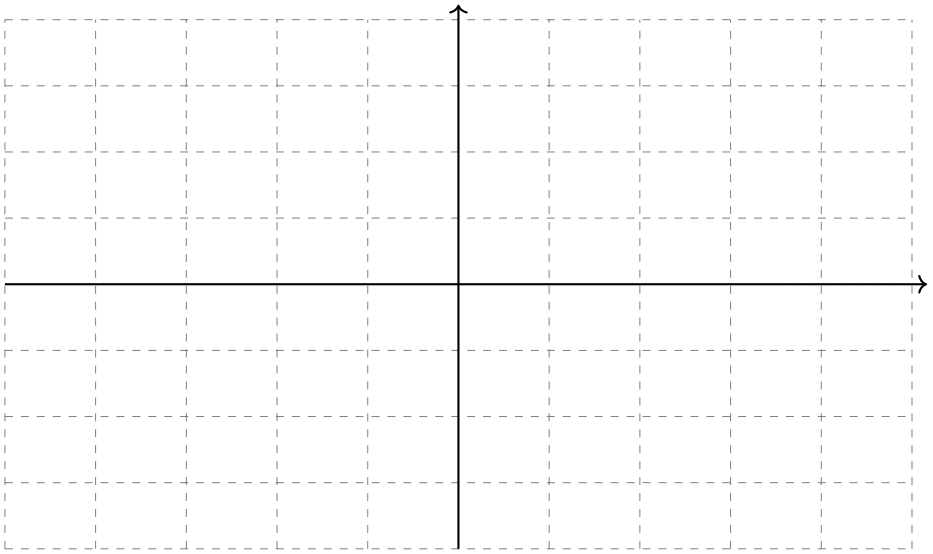
Date:

Task 1.

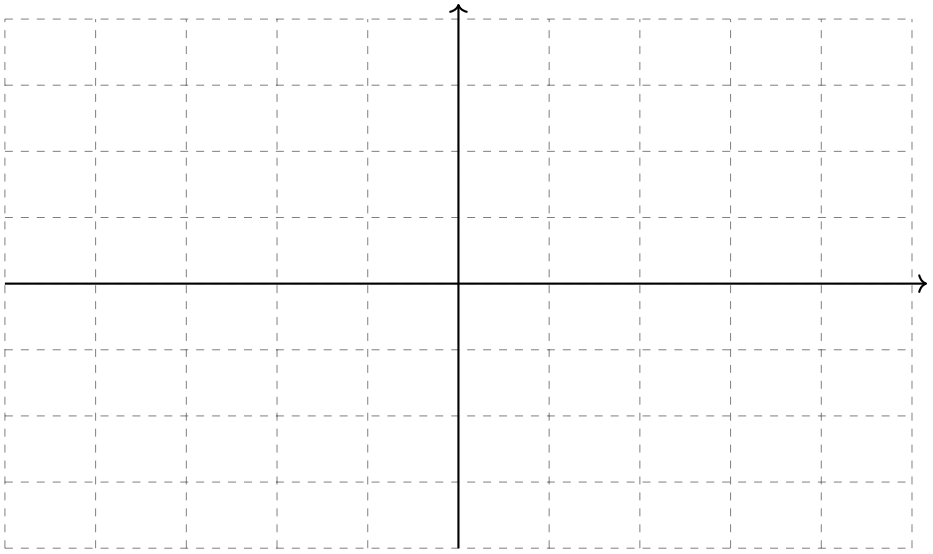


Task 2.

Task 6.



Task 7.

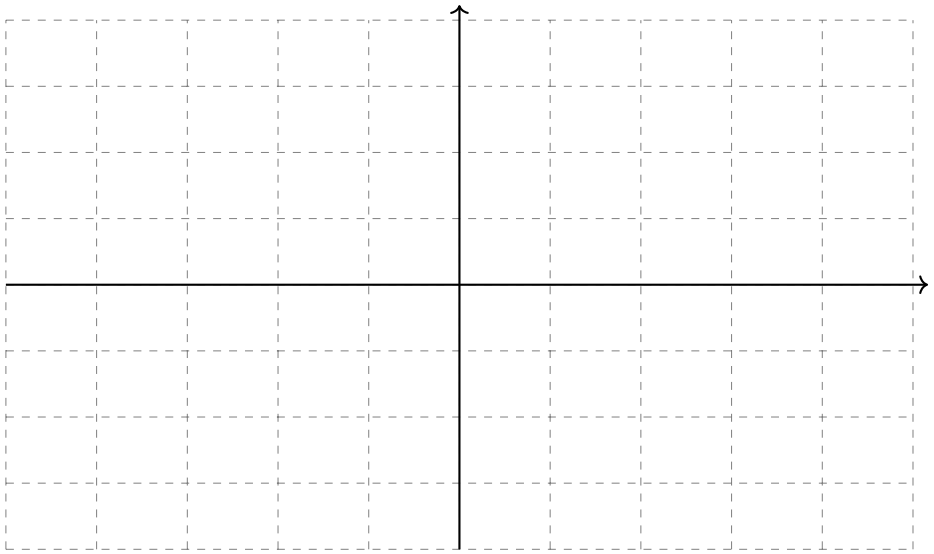


Task 8.

V_{DC} = ...

Task 9.

Task 11.

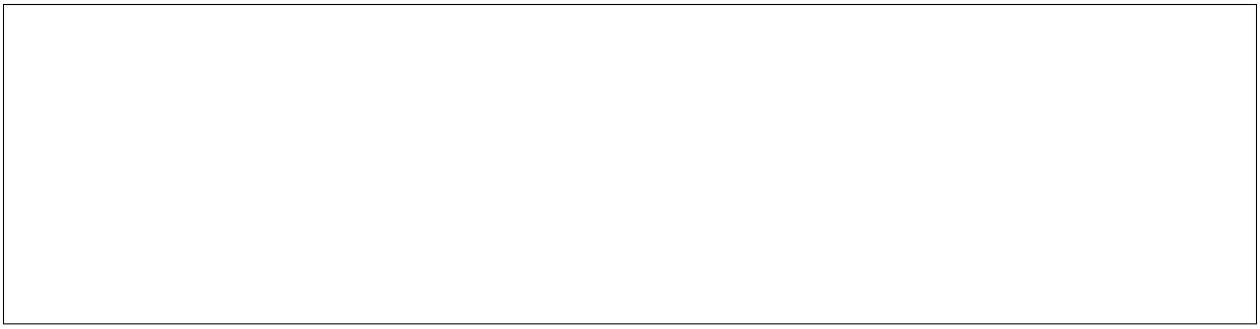


Task 13.

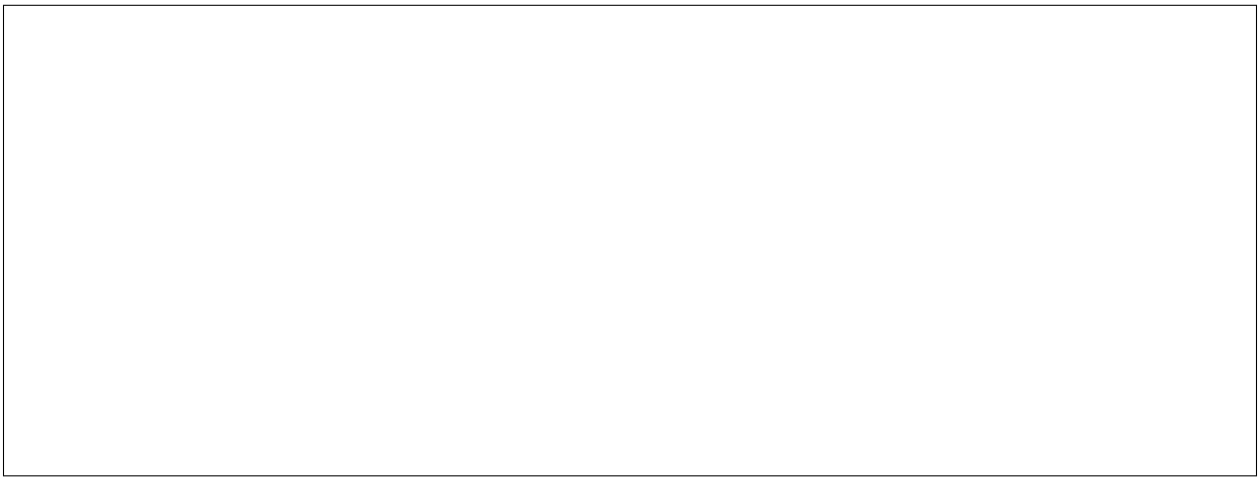
Task 14.

Task 15.

$R_L(\Omega)$	$V_{RL}(V)$
1,000	
560	
100	



Task 16.



EN1094 Laboratory Practice
Electronics - Workshop 4 Task Sheet

Index No.: **Group No.:**

Date:

Task 1.

Task 2.

Task 3.

Task 7.

Task 8.

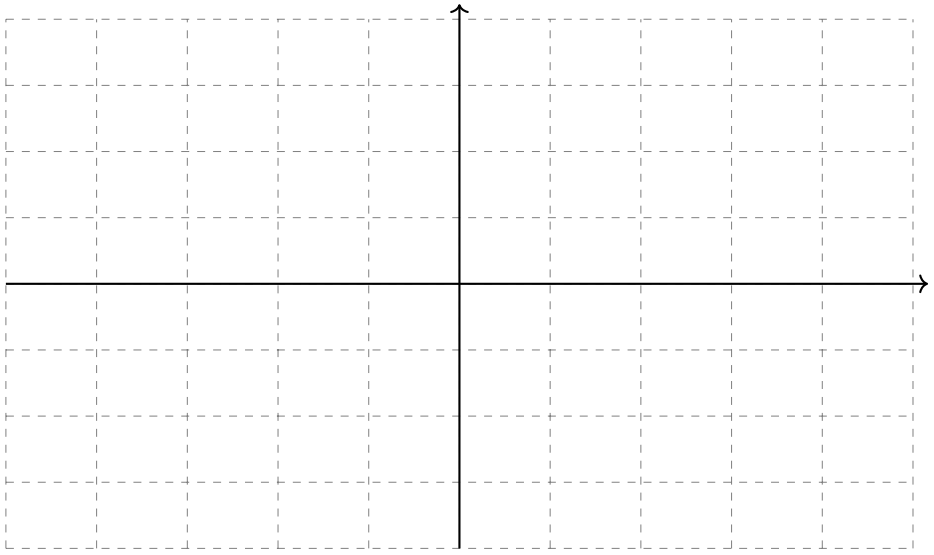
Task 9.

Task 10.

Task 11.

Task 16.

Task 17.

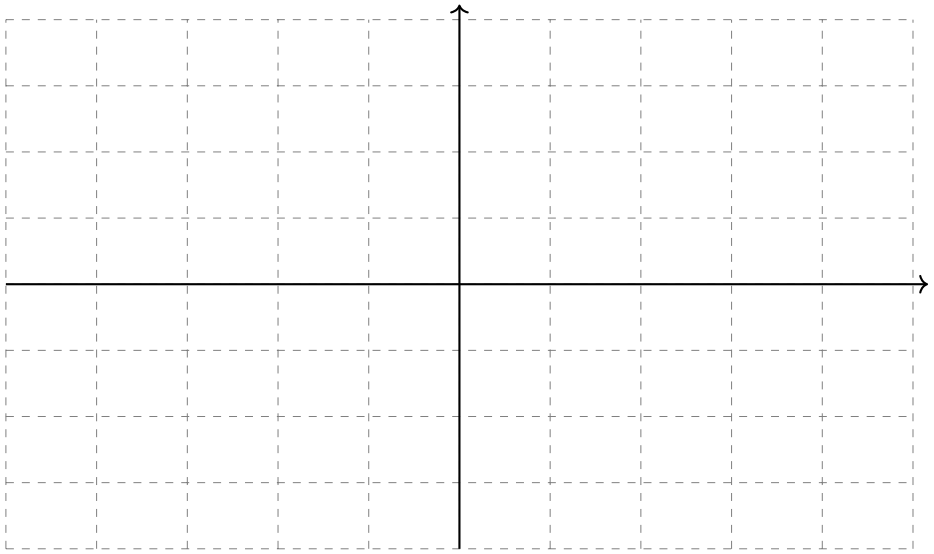


Task 18.

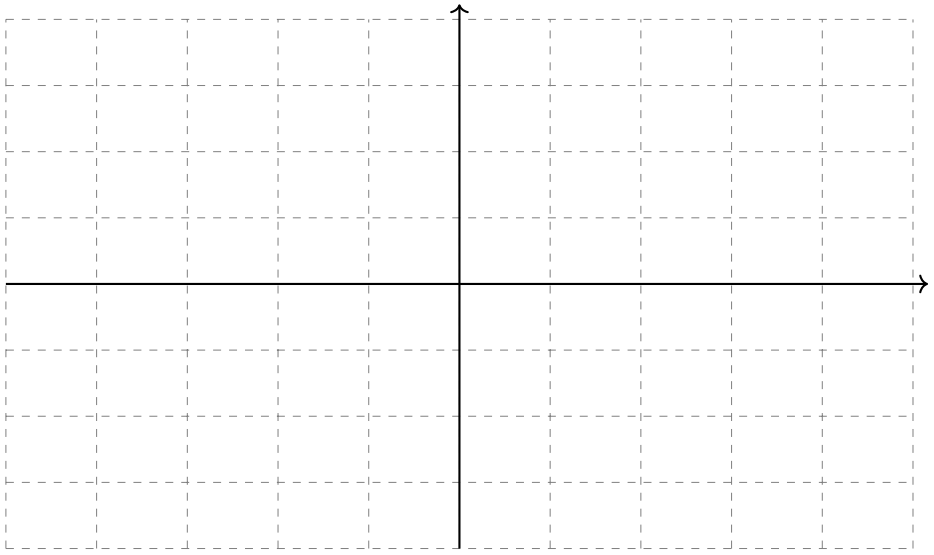
Task 19.

Task 20.

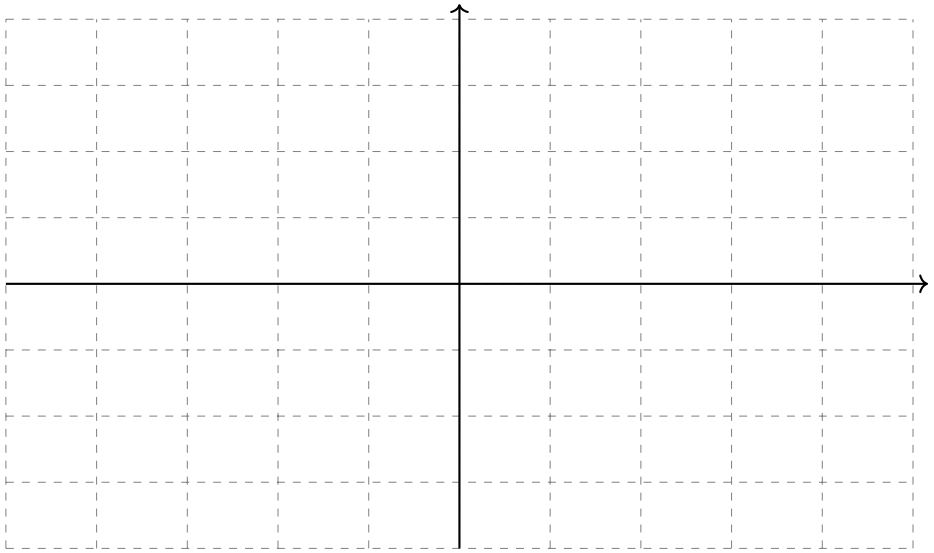
Task 22.



Task 26.



Task 28.



Task 29.

EN1094 Laboratory Practice
Electronics - Workshop 5 Task Sheet

Index No.: **Group No.:**

Date:

Task 1.

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

Task 2.

S:

C:

Task 3.

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		

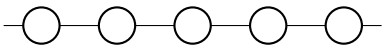
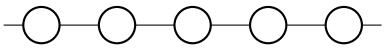
Task 4.

S:

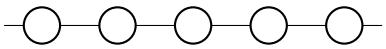
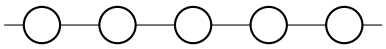
C_out:

Task 5.

Task 13.

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

Task 15.

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

EN1094 Laboratory Practice
Electronics - Workshop 6 Task Sheet

Index No.: **Group No.:**

Date:

Task 1.

Task 2.

Current state		Input	Next state		Output	First flip-flop (q_1)		Second flip-flop (q_2)	
q_1	q_2	x	$\hat{q}_1$	$\hat{q}_2$	y	j_1	k_1	j_2	k_2

Task 3.

y:

j_1:

k_1:

j_2:

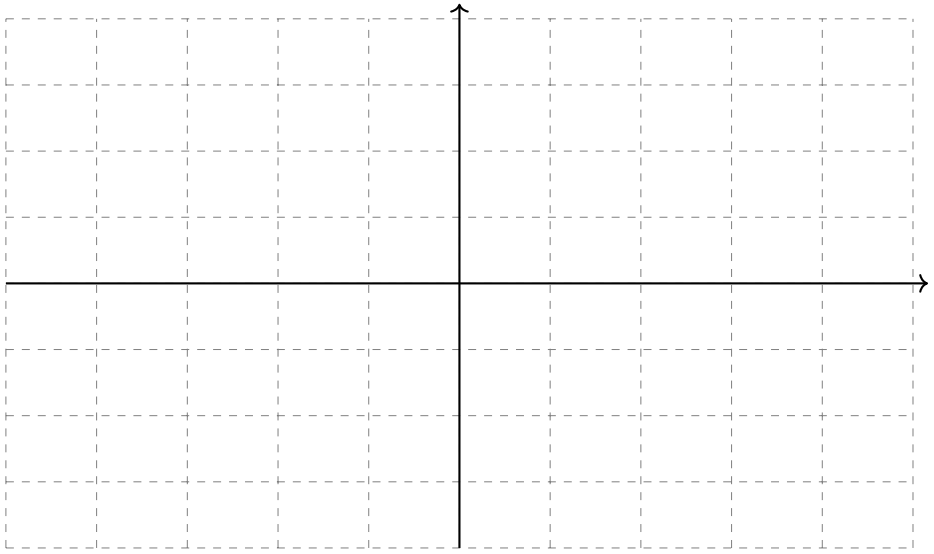
k_2:

Task 14.

I/O node	DE0 Nano signal	FPGA pin
reset_inv	KEY[0]	PIN_J15
clk	CLOCK50	PIN_R8
x	GPIO_00	
y	GPIO_01	
blink	LED[1]	

Task 17.

Clock frequency:



EN1094 Laboratory Practice
Telecommunication - Workshop 1 Task Sheet

Index No.: **Group No.:**

Date:

Task 1.

Task 2.

Task 3.

Table 1: Noise Variance vs SNR

Noise Variance	SNR (dB)	Noise Variance	SNR (dB)
0.1		1.5	
0.5		2	
1		2.5	

Task 4.

Table 2: Channel Bandwidth vs SNR

Bandwidth	SNR (dB)	Bandwidth	SNR (dB)
500		1000	
750			

Task 5.

Task 6.

Task 8.

Task 9.

Task 10.

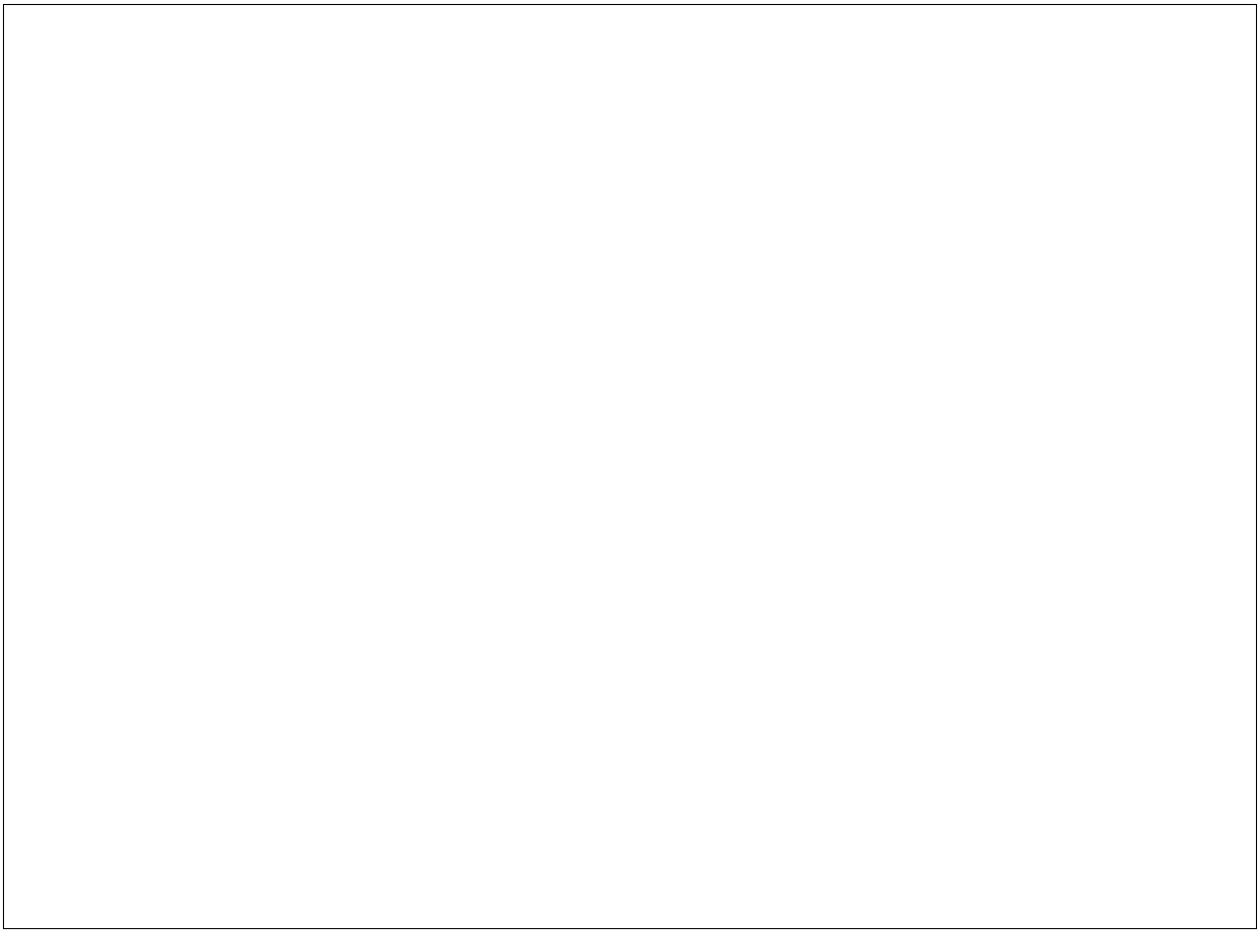
Task 11.

Table 3: Noise Variance vs BER

Noise Variance	BER	Noise Variance	BER
0.1		1.5	
0.5		2	
1		2.5	

Task 12.

Task 13.



Task 14.

Table 4: Frequency vs v_{out}

Frequency	v_{out}	Frequency	v_{out}
10 Hz		500Hz	
100 Hz		600 Hz	
200 Hz		700 Hz	
300 Hz		1 kHz	
400 Hz		10 kHz	

Task 15.

Task 16.

Task 17.

Task 18.

Table 5: Signal Distortion Due to Low-Pass Channels

Frequency	Input Waveform	Output Waveform
20 Hz		
550 Hz		
650 Hz		
1 kHz		
5 kHz		

Task 19.

Task 20.

Table 6: Frequency vs v_{out}

Frequency	v_{out}	Frequency	v_{out}
10 Hz		500Hz	
100 Hz		600 Hz	
200 Hz		700 Hz	
300 Hz		1 kHz	
400 Hz		10 kHz	

Task 21.

Task 22.

Task 23.

Task 24.

Task 25.

Task 26.

Table 7: Signal Distortion Due to Band-Pass Channels

Frequency	Input Waveform	Output Waveform
50 Hz		
300 Hz		
500 Hz		
575 Hz		
650 Hz		
850 Hz		
1 kHz		

EN1094 Laboratory Practice
Telecommunication - Workshop 2 Task Sheet

Index No.: **Group No.:**

Date:

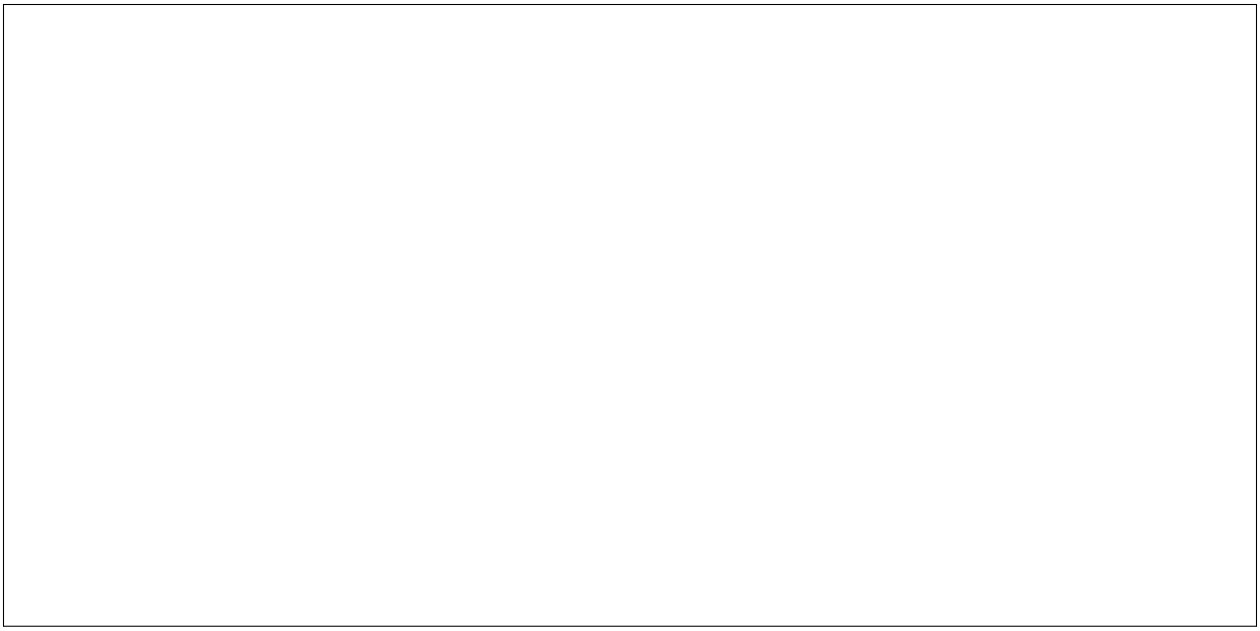
Task 1.

Task 2.

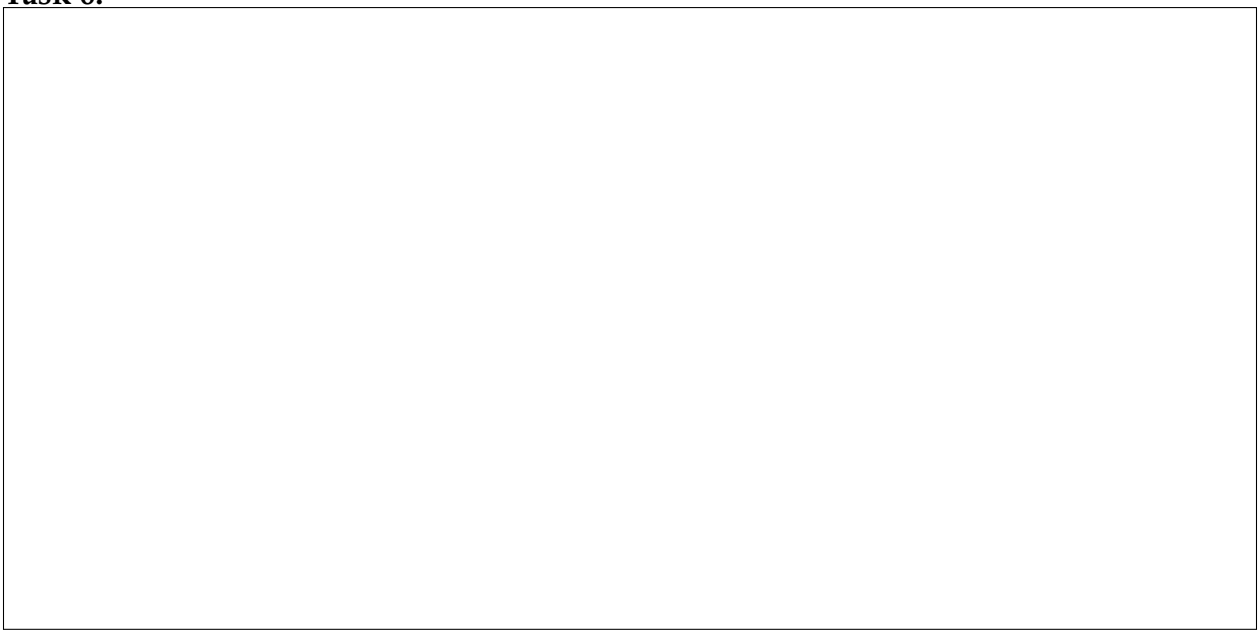
Task 3.

Task 4.

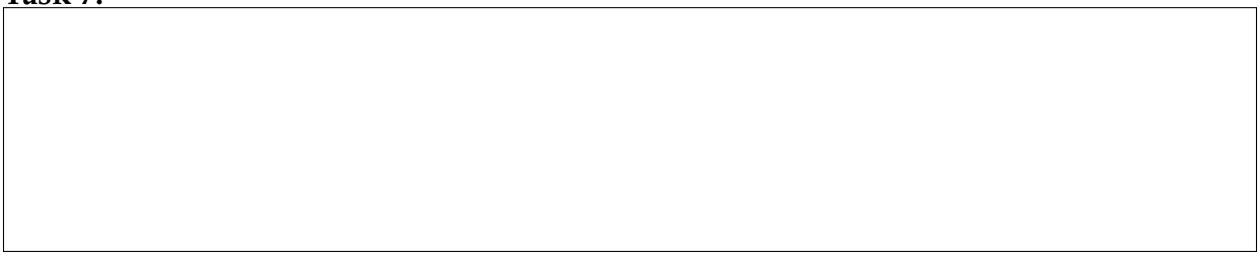
Task 5.



Task 6.



Task 7.



Task 8.

Task 9.

Task 10.

Task 11.

EN1094 Laboratory Practice
Telecommunication - Workshop 2 Task Sheet

Index No.: **Group No.:**

Date:

Task 1.

Task 2.

Task 3.

Table 8: Properties of Binary ASK

Bit Pattern	Maximum Amplitude	Initial Phase

Task 4.

Table 9: Properties of 4-level ASK

Bit Pattern	Maximum Amplitude	Initial Phase

Task 5.

Table 10: Properties of 2-level FSK

Bit Pattern	Maximum Amplitude

Task 6.

Table 11: Properties of BPSK

Bit Pattern	Maximum Amplitude	Initial Phase

Task 7.

Table 12: Properties of QPSK

Bit Pattern	Maximum Amplitude	Initial Phase

Task 8.

Task 9.

EN1094 Laboratory Practice
Telecommunication - Workshop 4 Task Sheet

Index No.: **Group No.:**

Date:

Task 1.

Task 3.

Task 4.

Task 5.

Task 6.

Task 7.

EN1094 Laboratory Practice
Telecommunication - Workshop 5 Task Sheet

Index No.: **Group No.:**

Date:

Task 1.

Task 2.

Distance					
PER					

Task 3.

Task 4.

Distance					
PER					

Task 5.