Software Based Signal Generator and Oscilloscope

Thesis Submitted to the EEE Department of BRAC University

By

Shaneela Zaheed (12321039)

Mahfuz Hossain (13121017)

Samiur Rahman Khan (12121048)

To

Dr. Mohammad Belal Hossain Bhuiyan

Associate Professor, Department of EEE, BRAC University

In Partial Fulfillment of the Requirements for the

Degree of Bachelor of Science in Electrical and Electronics Engineering

December, 2016

BRAC University

Mohakhali, Dhaka.

Declaration:

We hereby declare that the project titled “Software base signal generator and oscilloscope using MATLAB” Submitted to BRAC University, to the department of Electrical and Electronic Engineering is our thesis for the completion of Bachelors of Science in Electrical and Electronic Engineering. This Thesis has not been submitted previously for any degree. We conducted the research and work solely and the materials used for study are acknowledged as reference later in this paper.

Date of Submission: 14th December 2016

Shaneela Zaheed

Student ID: 12321039

Mahfuz Hossain

Student ID: 13121017

Samiur Rahman Khan

Student ID: 12121048

Thesis Supervisor

Dr. Mohammad Belal Hossain Bhuiyan

Associate Professor,

Department of EEE, BRAC University

Acknowledgement:

We are grateful to The Almighty Allah for making us capable to submit this work of our thesis to complete the Bachelors of Science Degree from Electrical and Electronic Engineering. Next we are thankful to Dr. Mohammad Belal Hossain Bhuiyan, our Supervisor for the Thesis project. Without his supervision, guidance and encouragement our project would not have been turned into reality, we wouldn't have been able to move forward with our vision without his valuable insight. We are also thankful to our Co-Supervisor, Atanu Kumar Saha, Lecturer of Electrical and Electronic Engineering, BRAC University. His constant support enabled us to proceed with our research and implementation. Finally we would like to thank our family and friends whose mental support has been tremendously encouraging for us.

Abstract:

This paper is written based on two major function, signal generator and signal detection. The functions are built in two parts form a structure of a Software base signal generator and oscilloscope which is being controlled by MATLAB 2013a. Data acquired here are directly from the different ports of the PC soundcard. Generation is done through the speaker port and the detection is done through the microphone port of the soundcard. The built in sound card in our laptop is used which allows convenience and allows us to use this application outside of our laboratories too.

Sound card uses four components to translate analog and digital information

- An **analog-to-digital** converter (ADC).
- A **digital-to-analog** converter (DAC).
- An **ISA or PCI interface** to connect the card to the motherboard.
- Input and output connections for a microphone and speakers.

Table of Contents

List of figures

Introduction:

In today's world, being able to afford an oscilloscope and a signal generator may not be a problem to all but for some it surely is not a cost effective environment but still it is required by most engineers to work in laboratories. It is mostly seen that many Universities use oscilloscopes and signal generators for practical class purposes and usually each oscilloscope and signal generator costs over Tk. 30-40 thousand. Repairing and replacement of these oscilloscopes and signal generators is also expensive and difficult to maintain; thus to make it cost effective and easier to maintain we have the SOFTWARE BASED SIGNAL GENERATOR AND OSCILLOSCOPE. This paper is based on how we utilized the highly functional software MATLAB 2013a and the soundcard of our PC or laptop to build a signal generator and oscilloscope. With the help of the PC's microphone port and audio port the oscilloscope's signal detection and generation task is performed. Using appropriate command on MATLAB, the required data are obtained and generated and each command has its own functionality where some of them rely on previous commands too. The system will use the built in PC sound card and perform the task. Using the microphone port signal detection is performed and from the audio port we perform the function of signal generation. For signal detection two channels of the 3.5 mm audio jack are being used among which one is grounded and another one does the generation task.

1.1 Introduction to MATLAB

MATLAB is a very highly well functioned tool for computation, computing and visualization in an integrated environment. MATLAB (matrix laboratory) is a multi paradigm numerical computing environment and a fourth generation programming language mostly used in today's world for matrix manipulation, data implementation and plotting of functions. Cleve Moler the chairman of computer science department in University of Mexico started building MATLAB in late 1970s which soon became very known among university students for its high efficiency in applied mathematics. MATLAB was first adopted by researchers and practitioners in control engineering and now it is being used worldwide by students for education purposes for linear algebra, numerical processing and image processing too.

Common purpose of MATLAB usage is:

- Math and Computation.
- Algorithm Development.
- Data acquisition.
- Modeling, simulation and prototyping.
- Scientific and engineering graphics.
- Application development including graphical user interface building.
- Data analysis, exploration and visualization.

This is how the MATLAB looks like from inside

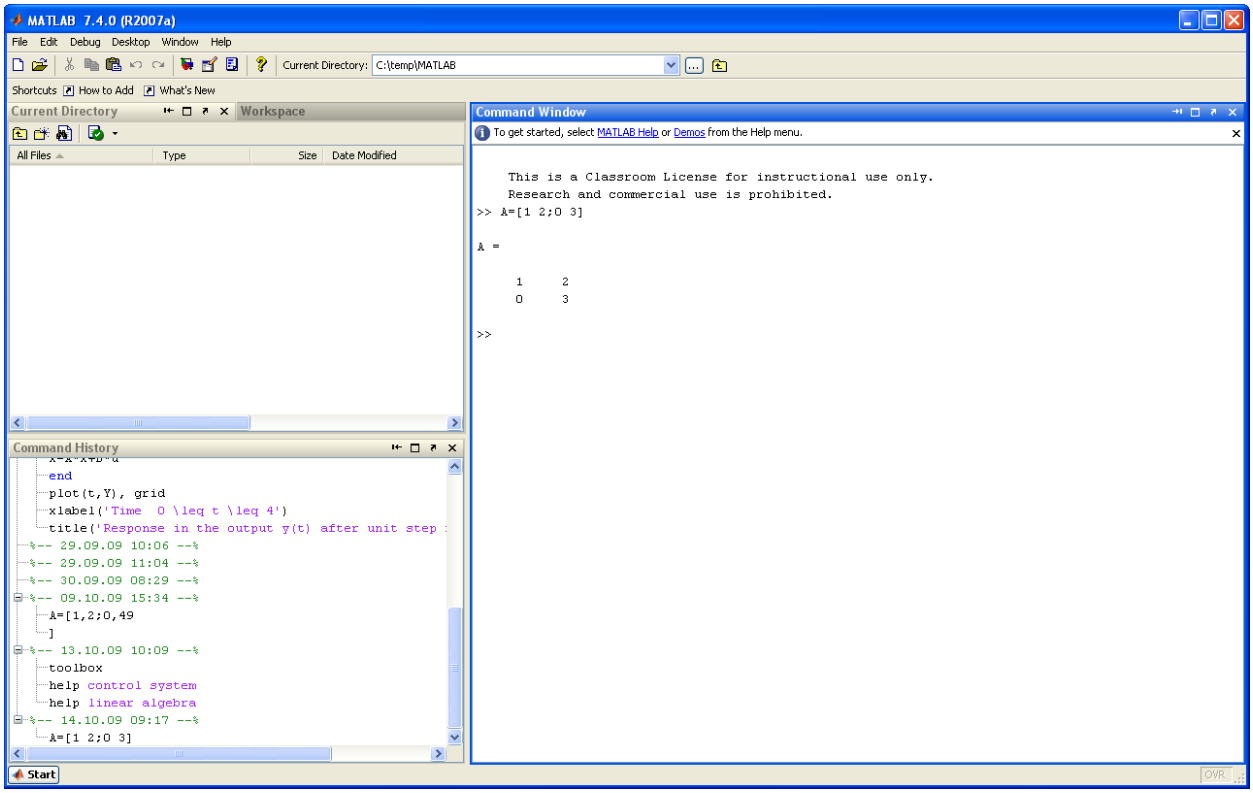


Figure 1.1

MATLAB consists of four separate windows:

- Command Window.
- Command History.
- Workspace.
- Current Directory

1.2 Introduction to Software base signal generator and oscilloscope

This is solely an attempt to familiarize today's education system to a new type of oscilloscope and signal generator which is completely based on software and run by the computer or laptop to reduce the cost of oscilloscope. Here the PC provides the display interface, the controlling of frequency and amplitude is also performed through the PC and also provides the electrical power of the acquisition hardware. The entire process is briefly elaborated in the flowchart below.

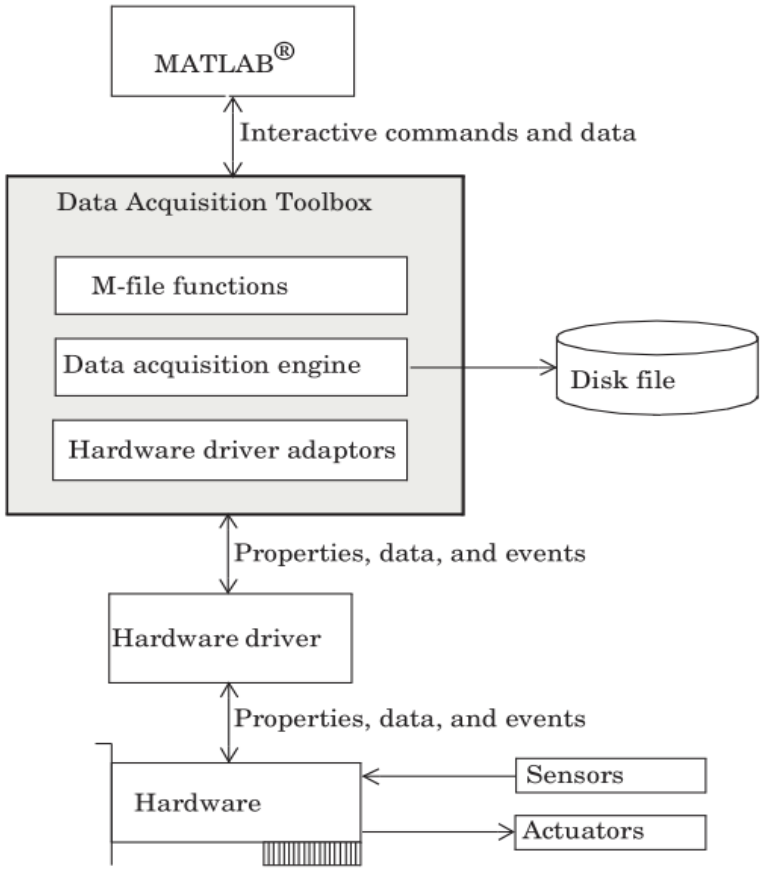


Figure 1.2

As mentioned earlier the reasons behind to choose this project was to make it easier for the students of engineering department these days to get an easy access to the oscilloscope and signal generator anytime they require. It is to be seen that mostly due to the cost of oscilloscope and signal generator it gets difficult to laboratories to afford more oscilloscope and signal generator but the completion of this project will enable you to generate and detect signals in a very user friendly and cost effective way. The oscilloscope detects input signals passed through soundcard microphone port and generates a wave on the monitor screen for display and similarly through the audio port a signal of minimal voltage is generated which is then amplified to the desired voltage and the generation portion is done.

Data acquired from the microphone port through soundcard is directly gathered in MATLAB through programming and here it is processed for further processing.

Generating m.file with desired programming commands and then merging it for different type of waveforms in a single GUI (Graphical User Interface) enables us to see the signal being detected more precisely. The control of the display of signal being generated is performed manually through the GUI.

2. Literature Review

Oscilloscopes were previously known as oscillograph and are mainly useful to give an overview of constantly varying signal voltages usually as a two dimensional plot of one or more signals other than that signals can be converted to voltages and displayed as well. The oscillograph previously was hand drawn chart which later was automated. Later the cathode ray tube came along and displaced the oscillograph taking over the majority of the market when advancements such as triggers were added to them. They could be very useful to observe the changes in signals over time on a calibrated scale. The properties of the signal that is being observed can be over several factors and observed more in detail by varying amplitude, frequency, rise time, time intervals etc.

Oscilloscopes are being widely used in science, engineering, medicine, automotive and mostly in telecommunication industry. The basic oscilloscope looks like the one below.

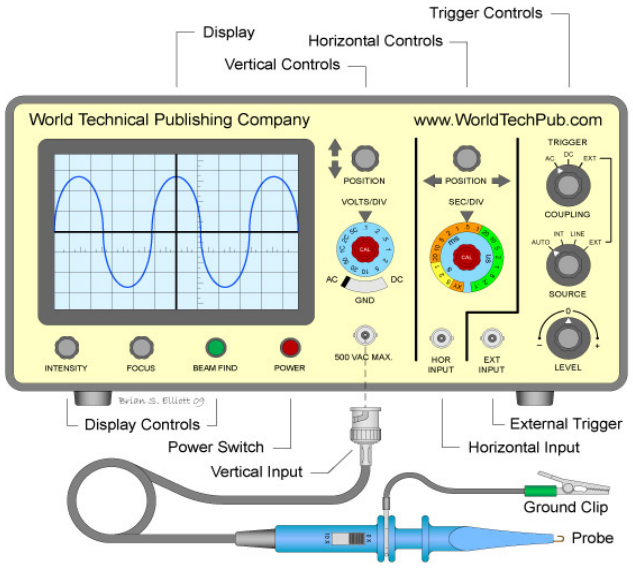


Figure 2.1

It consists of four sections and they are

- Display
- Vertical control
- Horizontal control
- Trigger control

Displays are usually of LCD or CRT with horizontal and vertical grid lines referred to as the gratitude it also has a focus knob, intensity knob and finder button which is rarely used. Vertical section controls the amplitude of the signal. It has a Volts-per-Division (Volts/Div) selector knob, an AC/DC/Ground selector switch and the vertical (primary) input for the instrument. The horizontal section controls the time base or "sweep" of the instrument where the primary control is the Seconds-per-Division (Sec/Div) selector switch. Also included is a horizontal input for plotting dual X-Y axis signals. The horizontal beam position knob is generally located in this section. And lastly the trigger section controls the start event of the sweep. The trigger can be set to automatically restart after each sweep or it can be configured to respond to an internal or external event. The principal controls of this section will be the source and coupling selector switches. An external trigger input (EXT Input) and level adjustment will also be included.

In addition most oscilloscopes are provided with scopes to connect to any instrument usually with a resistor with ten times the oscilloscope input impedance. We have tried

to come up with a concept to make this entire process of signal detection and modulation compiled in software based module through MATLAB and present a Software base signal generator and oscilloscope. The purpose as mentioned above was to make the oscilloscope and signal generator cost effective and also to make is user friendly. As we have faced challenges using a normal oscilloscope and signal generator for first few days and it is also to be seen then often if one of the knobs of the oscilloscope does not function properly then we have to shift to another one for our work. A Software base signal generator and oscilloscope would eliminate that issue of technical errors, the only technical error using a Software base signal generator and oscilloscope we might face is the one related to the power of the PC. Any other bugs or difficulty could be fixed mostly by editing the code of programming through a thorough check. Keeping in mind that the ultimate result of the oscilloscope and signal generator should be such that could be used by students and engineers easily we tried building it using MATLAB GUI and with several codes compiled into one and a proper user interface and knobs the final m.file was generated. Using MATLAB enabled us to view instant changes when signal was varied without the codes for various callbacks to alter the change.

Writing “guide” in the command window the graphical user interface (GUI) appears.
Here the users can drag-and-drop buttons, sliders and other windows style controls and indicators needed.

2.1 Signal Generation

The signal generator is a device used to generate an electronic signal with specific known characteristics, thereby enabling an engineer or technician to test and examine a circuit. The goal was clear, that is to code for the implementation of signal generation through the audio port of a laptop or PC. First you need to make sure that you have all the software installed and that the hardware is working properly. You need MATLAB installed with the Data Acquisition Toolbox (DAT) and to ensure that the Data Acquisition Toolbox is installed you can go to the MATLAB prompt and type the command,

>>ver

This command will return which components and version of MATLAB that you installed, hopefully the Data Acquisition Toolbox is installed. Software used for the programming is MATLAB as mentioned for its simplicity and various functions. The graphical user interface of MATLAB is being used here mainly for oscilloscope and signal generator interface creation. Other software likes LABVIEW could be used too but the advantage of MATLAB is the instant change when any variable is varied is shown instantly without the needs of callback which makes it easier to operate with while running complicated codes.

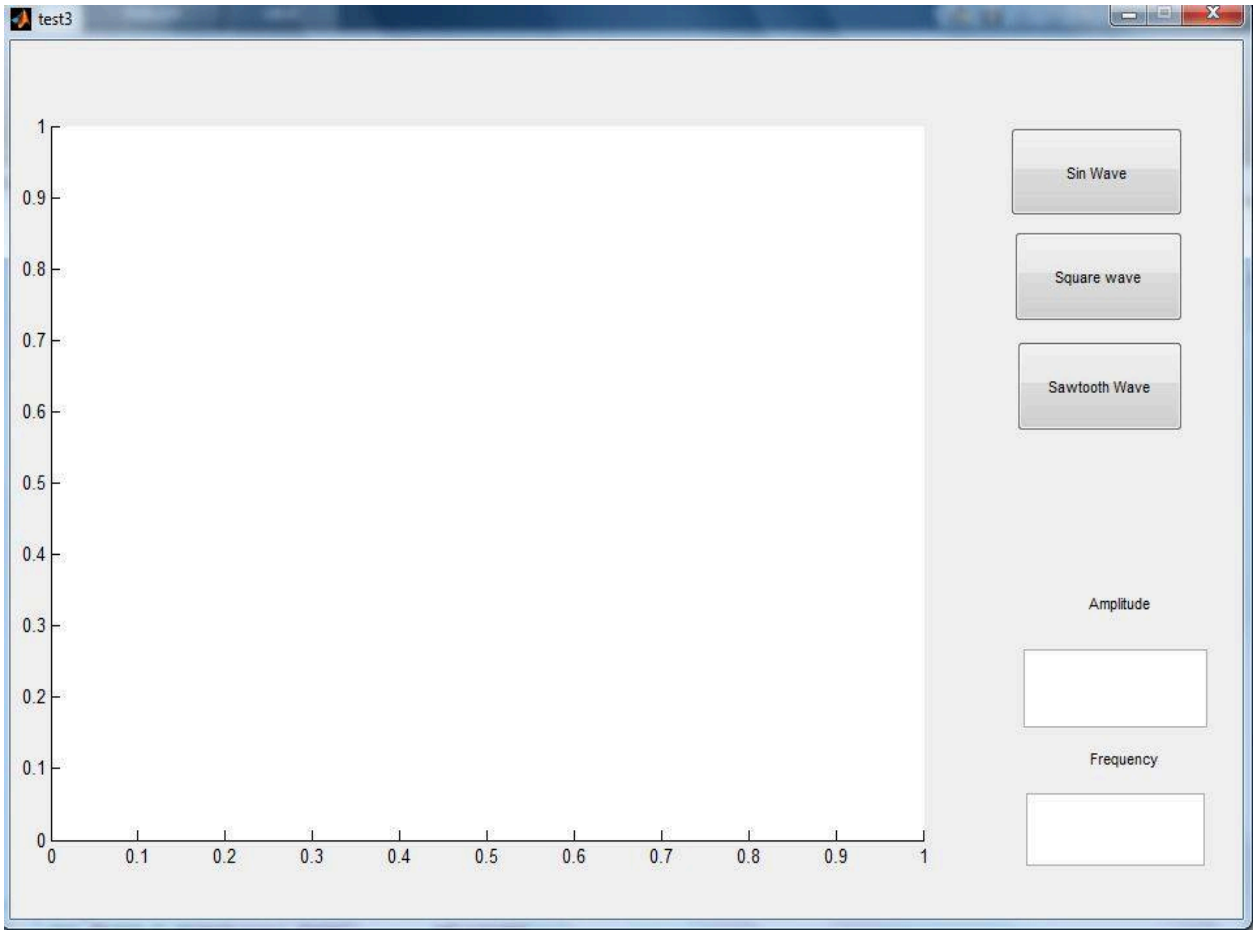
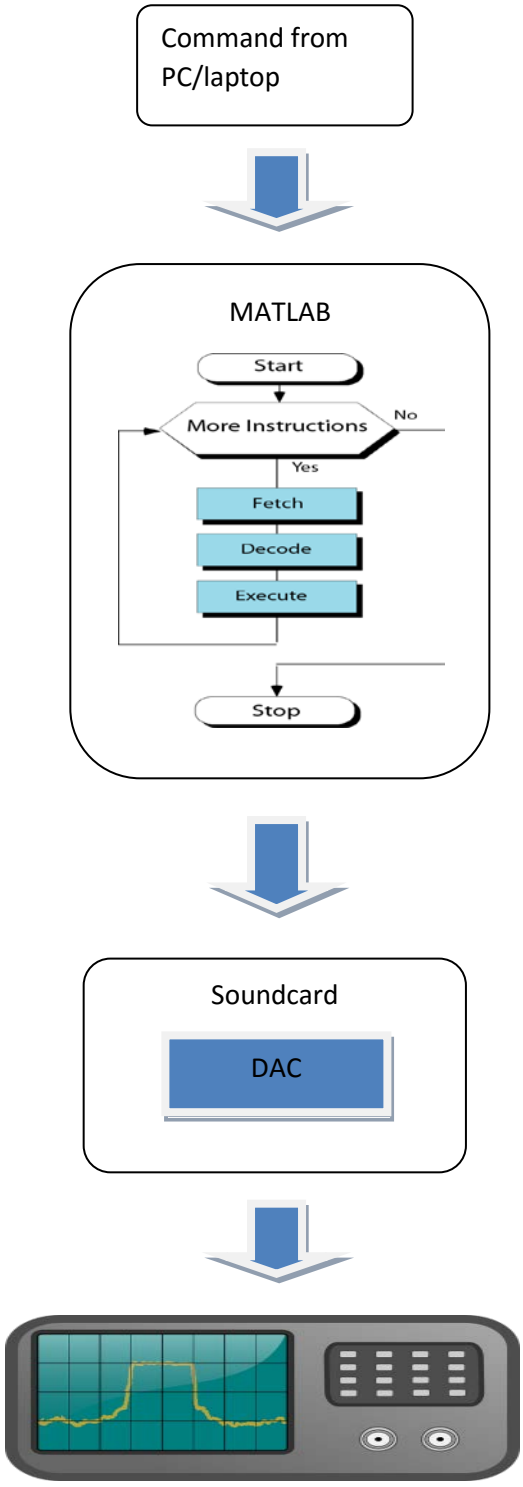


Figure 2.1.1

This was the final GUI interface for signal generation. We tried keeping the overall display very user-friendly and easy to access so that anyone can easily operate with this. The generation window consists of all kind of wave including sine, square and saw tooth.

At first we tried the entire process with simulink blocks as it was easier to access by new programmers but there were several problems and limitations we faced due to which we could not proceed with it. The Simulink blocks we used was a Sine Wave block generating a multichannel real or complex sinusoidal signal, with independent amplitude, frequency, and phase in each output channel a real sinusoidal signal was being generated when the output complexity parameter was set to Real, and was defined by an expression of the type. The limitations in this procedure was every time we called the simulink module through programming a wave was only being formed on the display of simulink grid axes but no external signal through soundcard audio port was being generated. Later we decided to go directly for programming to generate an external signal this way the codes will provide us with a signal and work like a signal generator.

Figure:2.1.2


Command from PC/laptop
MATLAB
Start
More Instructions
No
Yes
Fetch
Decode
Execute
Stop
Soundcard
DAC

At first we open the MATLAB software on our PC/laptop then on MATLAB m.script file; Required MATLAB programming codes are written and compiled, each code instructs the software to process the signal being generated in a particular way after the execution of each line on the script it automatically fetches for more instruction on the next line for further processing. After fetching for more instruction the generated function than decodes the instruction for proper processing and finally executes for generating the desired signal. The final signal being generated passes through the soundcard audio port and this is also done through proper coding instruction and execution of that code. The signal can be tested on the oscilloscope through which we can also thoroughly observe the signal and ensure that generation has been done properly. Apart from oscilloscope another way to ensure that the signal is being generated is, as we are utilizing the soundcard audio port for generation therefore generating a signal of certain amplitude and frequency will also cause the system to generate a sound which we can hear from the speakers/headphones connected to the PC/laptop. If the sound is produced it will then confirm us that the signal is being generated and the pitch of the sound being generated will give us an idea of whether the sound is being generated at given frequency or not.

2.1.1.Sine Wave:

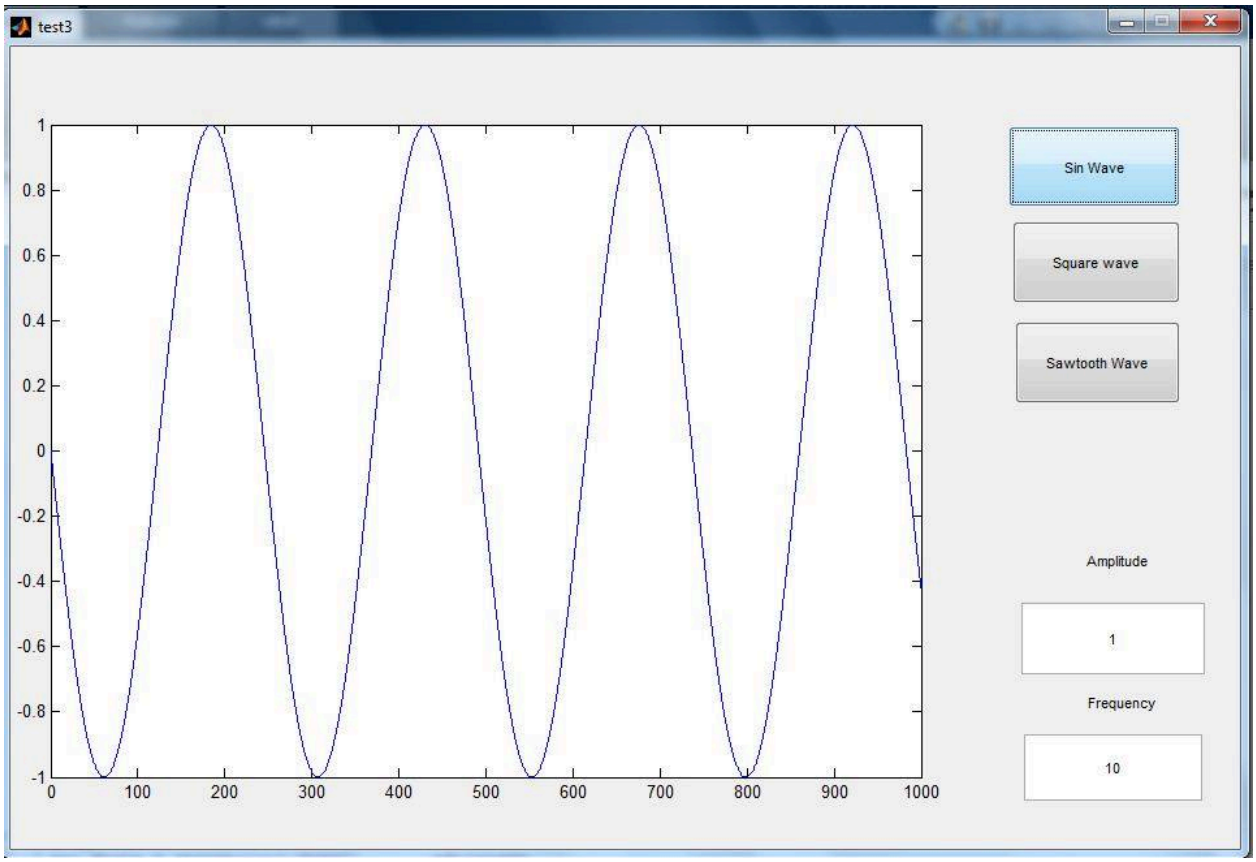


Figure 2.1.3

Sine wave usually represents a smooth recitative oscillation named under the trigonometric function sine, its most basic form as a function of time (*t*) is**:**

$$y=A\sin(2\pi ft+\varphi)$$

- *A* = the *amplitude*, the peak deviation of the function from zero.

- f = the *ordinary frequency*, the *number* of oscillations (cycles) that occur each second of time.
- $\omega = 2\pi f$, the *angular frequency*, the rate of change of the function argument in units of radians per second.
- *Φ= the phase, specifies (in radians) where in its cycle the oscillation is at t = 0.*

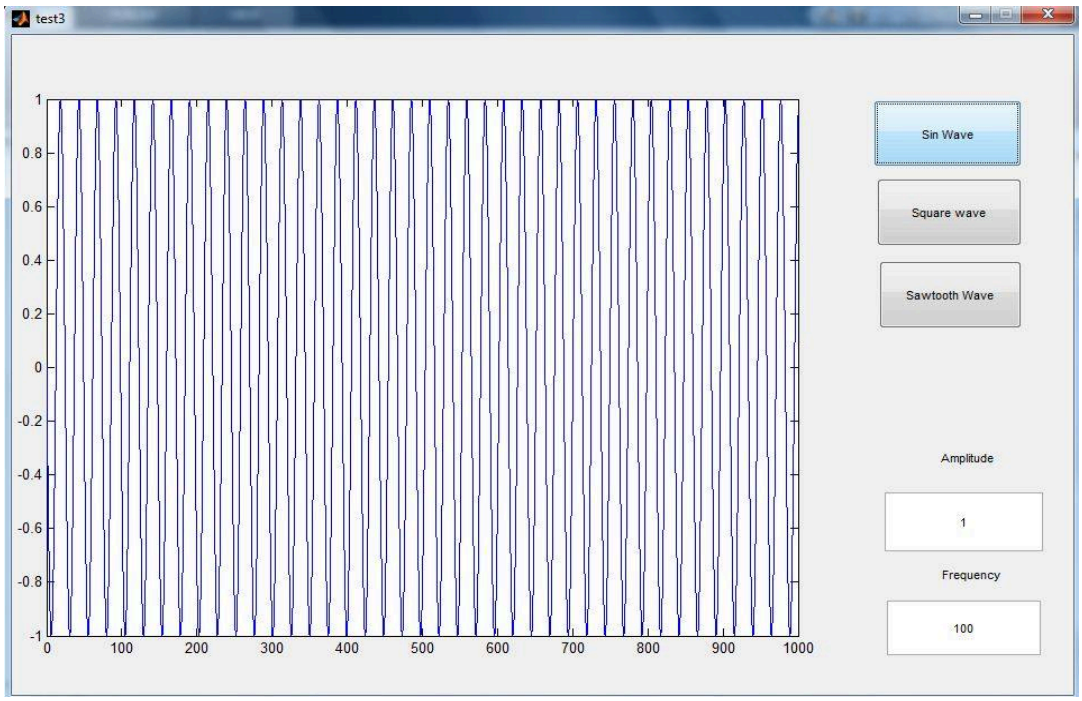


Figure 2.1.4

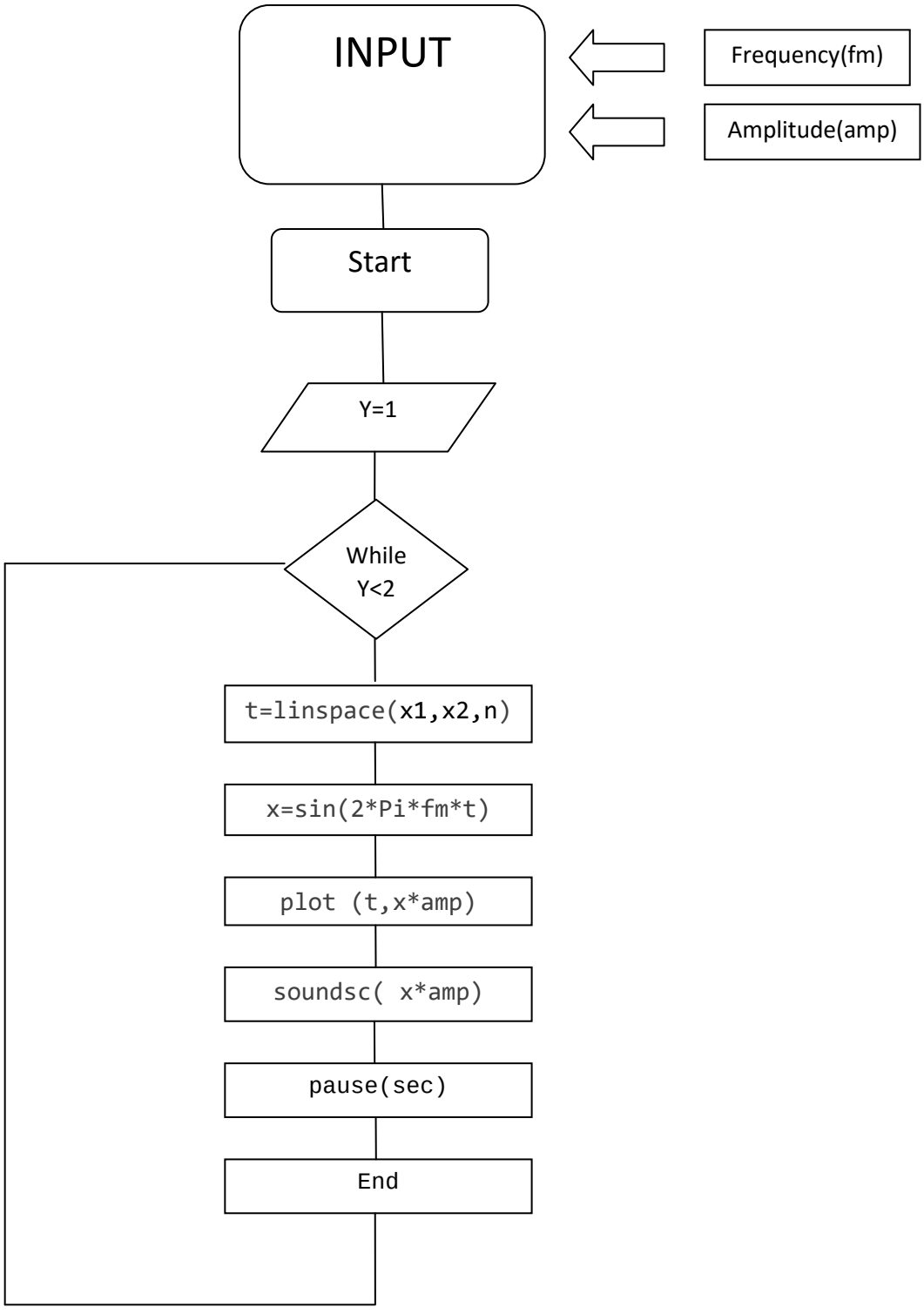

INPUT
Frequency(fm)
Amplitude(amp)
Start
Y=1
While
Y<2
t=linspace(x1,x2,n)
x=sin(2*Pi*fm*t)
plot (t,x*amp)
soundsc(x*amp)
pause(sec)
End

Figure 2.1.5

This is the algorithm used to generate a sine wave, firstly through MATLAB coding the signal is generated then after the given amplitude and frequency is entered as input on the GUI window shown above, the signal with desired amplitude and frequency is processed and goes under a while loop which has at first generates a linearly spaced vector and then an audio signal is sent to the speaker with a sampling frequency rate Fs if a rate is not specified then the signal generates a frequency at 8192 Hz. The pausing of the signal we take it through the instruction pause. The final sine wave being generated is then passed through the soundcard of the PC/laptop through proper instructions and finally the signal generated is obtained and tested on an external oscilloscope to confirm that we got our desired signal.

2.1.2.Square Wave:

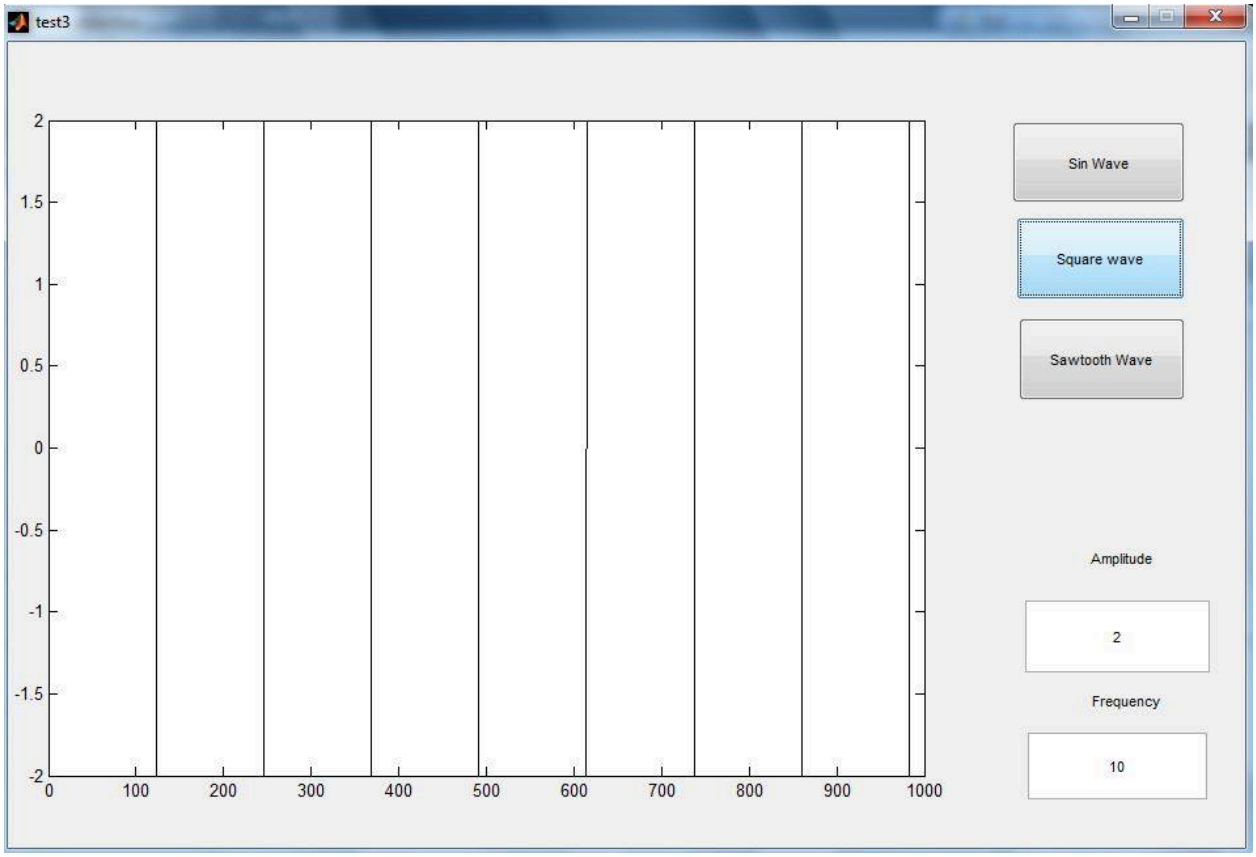


Figure 2.1.6

Square wave is a non-sinusoidal periodic waveform where the amplitude usually varies at a steady frequency between the fixed minimum and maximum values, these waves are usually encountered in electronics and signal processing. The transition is instantaneous between maximum and minimum in an ideal square wave. Function of a square wave is to generate a square wave with period 2π for the elements of time vector t. The command square (t) is similar to sin (t), but creates a square wave with peaks of ±1 instead of a sine wave. The command x=square (t,duty) generates a square

wave with specified duty cycle, duty, which is a number between 0 and 100. The **duty cycle** is the percent of the period in which the signal is positive. A true square wave would have a duty cycle of 50%, equal highs and equal low periods.

For instance if we want to generate a square wave of vector of 100 equally spaced numbers from 0 to 3π then the following command needs to written on a m file on MATLAB and run:

t = linspace(0,3*pi)';

x = square (t);

"*The generated square wave has a value of* 3π *at even multiples of* 3π *and a value of* 3π *at odd multiples of* 3π*. It is never* 3π*."* Again writing the commands below will help us differentiate the differences between the sine wave and square wave:

plot(t/pi,x,'.-',t/pi,sin(t))

xlabel('t / \pi')

grid on

For the generation of a square wave with amplitude 1 volts peak to peak we have used the commands written below along with the codes mentioned for sine wave. The same algorithm used for sine is used to generate a square wave of certain frequency and amplitude.

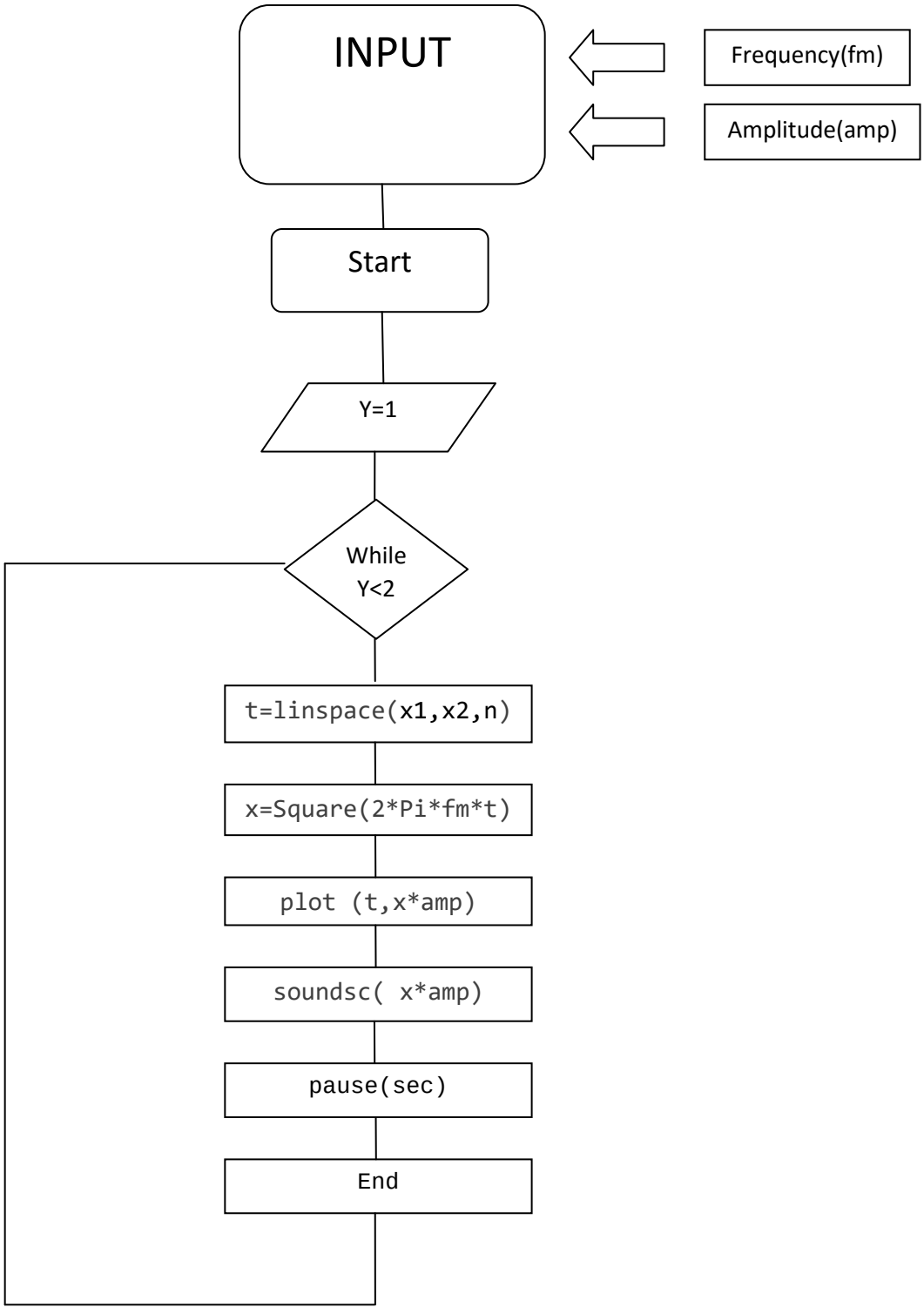

INPUT
Frequency(fm)
Amplitude(amp)
Start
Y=1
While
Y<2
t=linspace(x1,x2,n)
x=Square(2*Pi*fm*t)
plot (t,x*amp)
soundsc(x*amp)
pause(sec)
End

Figure 2.1.7

2.1.3.Sawtooth wave:

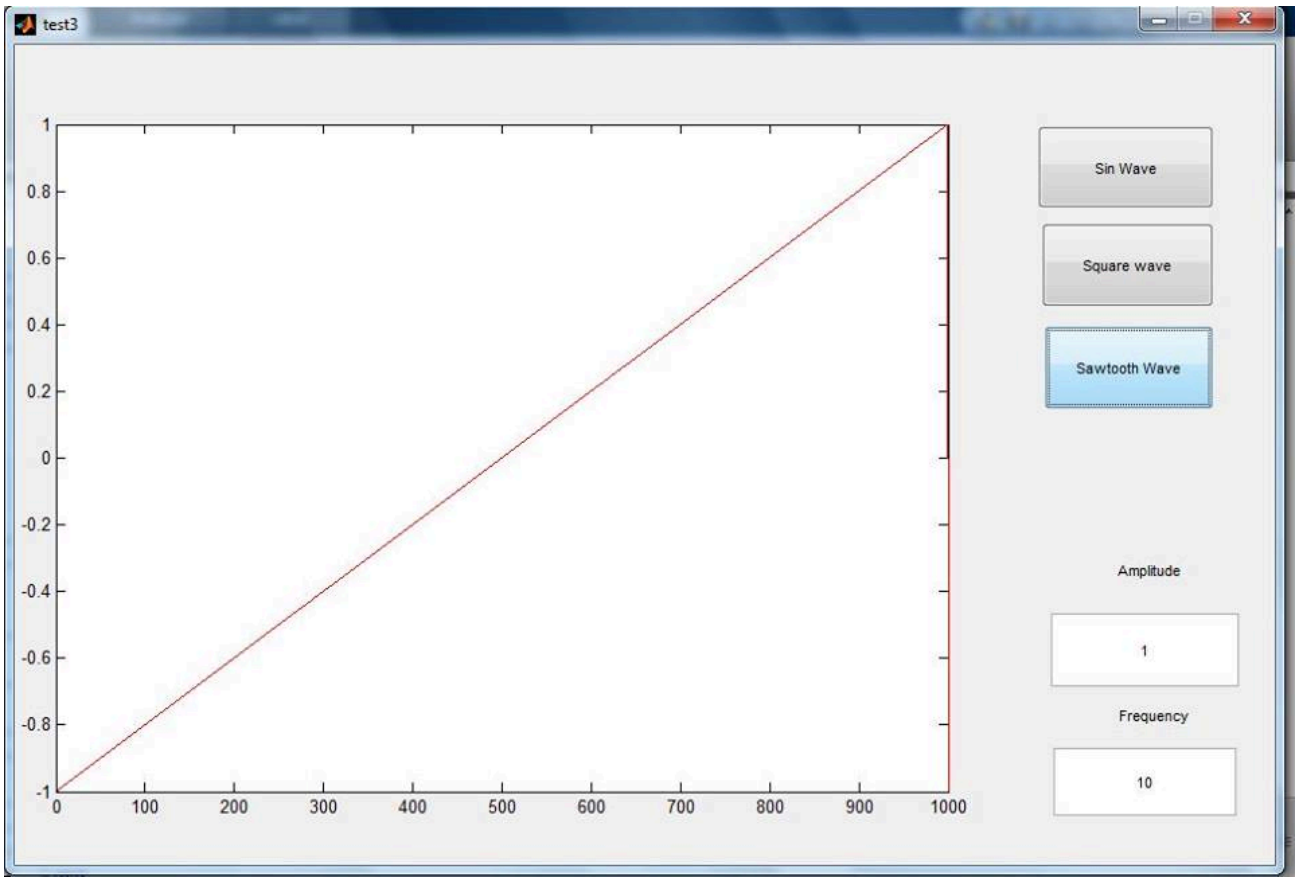


Figure 2.1.8

This is how a sawtooth or triangular wave look like and they usually work when in the codes these commands are mentioned:

sawtooth(t)

sawtooth(t,width)

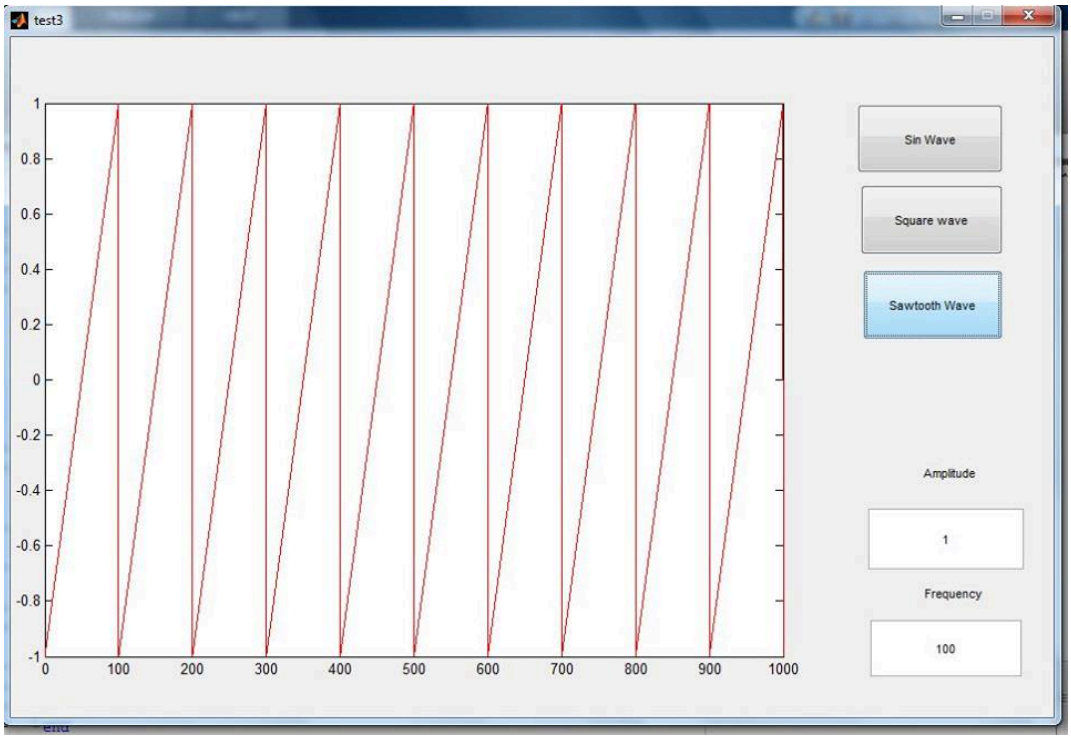


Figure 2.1.9

Sawtooth is similar to sin (t) just the difference is it creates a sawtooth signal wave of +1 to -1 volt peak to peak with a period of **2π** for the elements of time vector t.

Sawtooth are assumed to be -1 volt at intervals of **2π** and it linearly increases with time with a slope of 1/π at all other times. It generates a triangular wave where the width is a scalar parameter between 0 and 1 that determines point between 0 and 2π at which maximum occurs. Function increases from -1 to +1 on the intervals of 0 to 2π then decreases linearly from 1 to –1 on the interval 2π × width to 2π. The given code below generates a sawtooth wave of frequency 10Hz.

```
T = 10*(1/50);

Fs = 1000;

dt = 1/Fs;

t = 0:dt:T-dt;

x = sawtooth(2*pi*50*t);

plot(t,x)

grid on
```

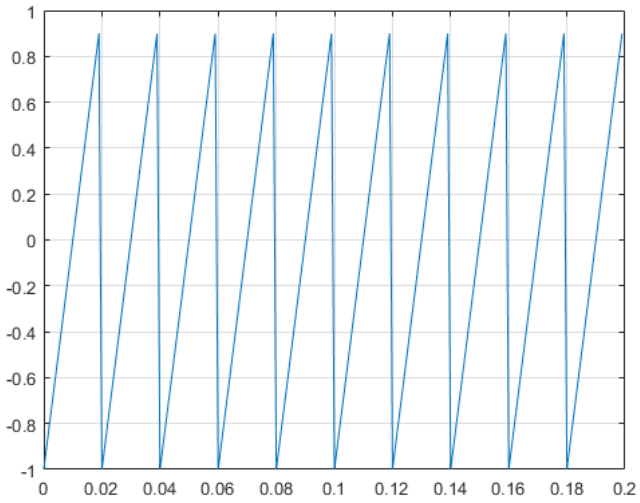


Figure 2.1.10

Sawtooth wave is also generated using the similar algorithm of sine wave.

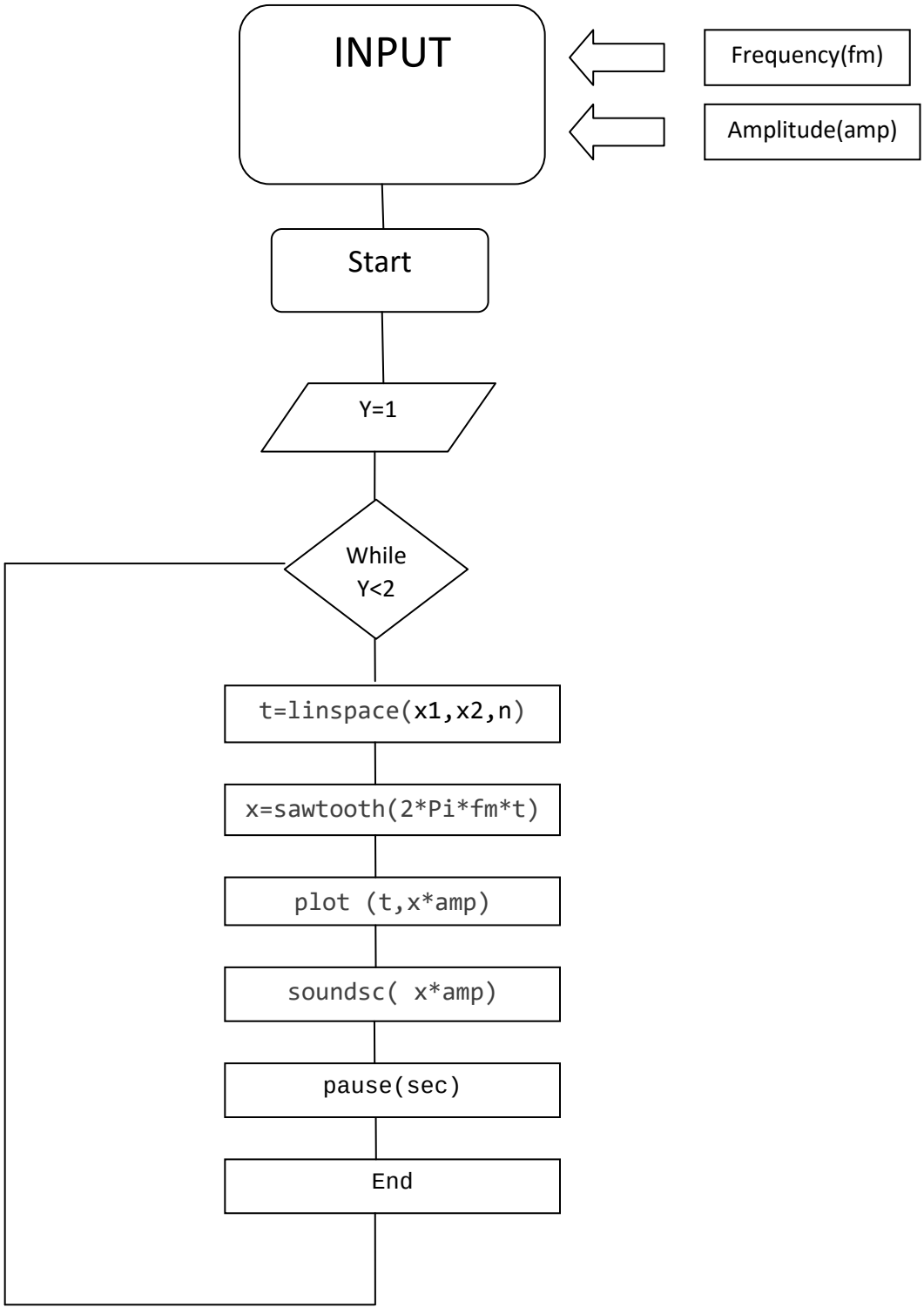

INPUT
Frequency(fm)
Amplitude(amp)
Start
Y=1
While
Y<2
t=linspace(x1,x2,n)
x=sawtooth(2*Pi*fm*t)
plot (t,x*amp)
soundsc(x*amp)
pause(sec)
End

Figure 2.1.11

3.Signal Detection

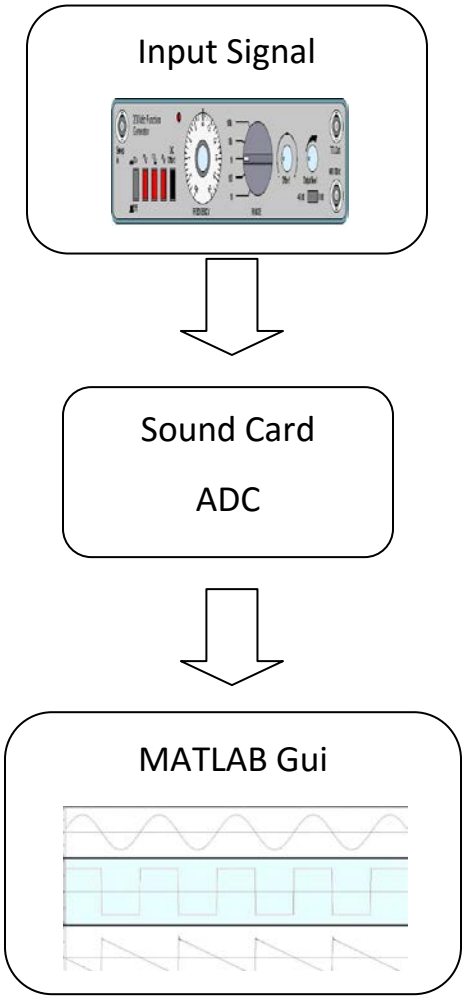


Figure 3.1

The process of signal detection was also done through programming language on MATLAB, here to detect the signal we used the soundcard microphone port and the result was shown on our laptop or PC on the graphical user interface plot axes. Various modifications of the codes were required to get the ultimate desired results.

First and foremost the data acquisition toolbox is required for the proper detection of the signal; the soundcard provides us with a continuous analog electrical signal, here the computer/laptop's microphone is being used to sample the signal. The final code used to detect the signals is provided below. We created few additional m.script file and compiled them too in order to ensure that our final codes run properly.

3.1. Allowance for High Pass Signal: The main task of these codes was to filter the incoming signal and only allow the high pass signals to pass through and resulting signal we get to see on the GUI was basically a signal above a maximum cut off frequency. The output of this filter is directly proportional to rate of change of the input signal. " High-pass filters are often used to clean up low-frequency noise, remove humming sounds in audio signals, redirect higher frequency signals to appropriate speakers in sound systems, and remove low-frequency trends from time series data thereby highlighting the high-frequency trends."

```
function hpcb

global highpass hhp

if get(hhp,'value')

highpass=true;

else

highpass=false;

end
```

3.2.Allowance for Low Pass Signal

Here the main task of the code is to allow only low pass signal from the signal being detected to pass and show the final result on GUI. The task of low pass filter is usually to pass signal with lower than cut off frequency. By removing some frequency a smoothing effect is created by the filter that is the filter provides us with slow changes in output so that we can thoroughly observe the trends and also boost the signal to noise ratio consisting minimal signal degradation.

```
function lpcb

global lowpass hlp

if get(hlp,'value')

lowpass=true;

else

lowpass=false;

end
```

3.3.Pausing the Signal

We are well aware that the command “pause” by itself, causes the currently executing function to stop and wait for you to press any key before continuing.

```
function pau

global paus

paus=~paus;
```

3.4.Stopping the signal

This code below stops further detected signal from being processed and from appearing on the GUI axes, basically it stops the program from running.

```
function stp

global st

st=true;
```

The interface below is the final interface used for signal detection through a microphone port, we did not use much knobs as it may confuse the user and tried keeping it at it's simplest form

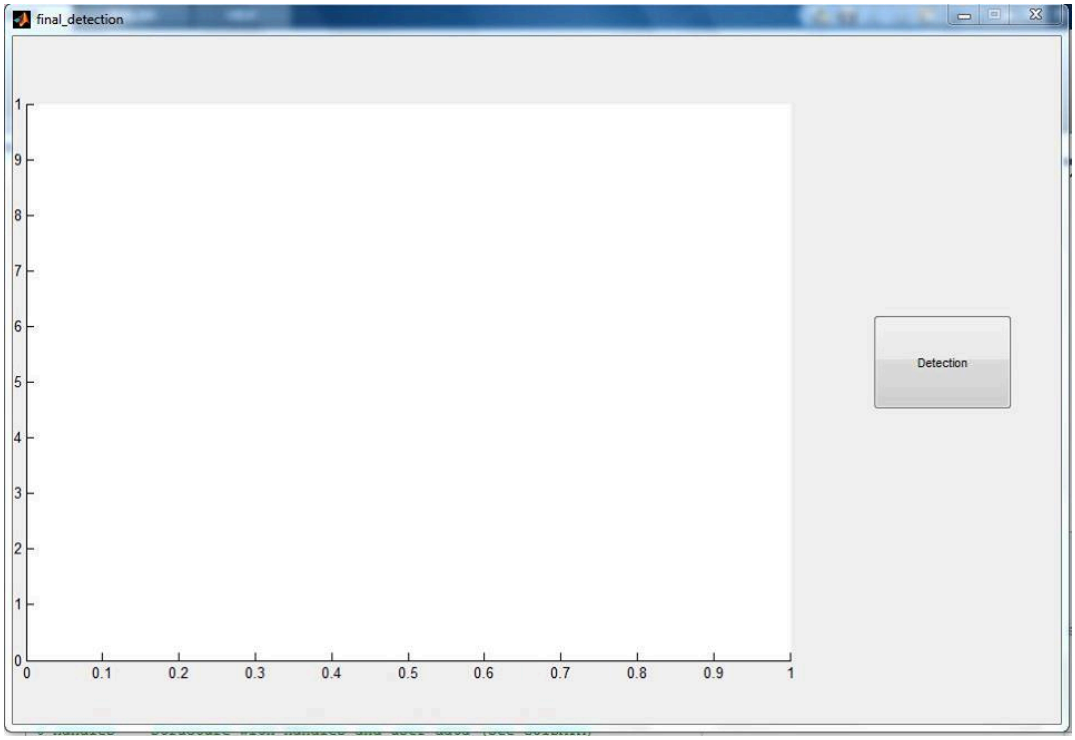
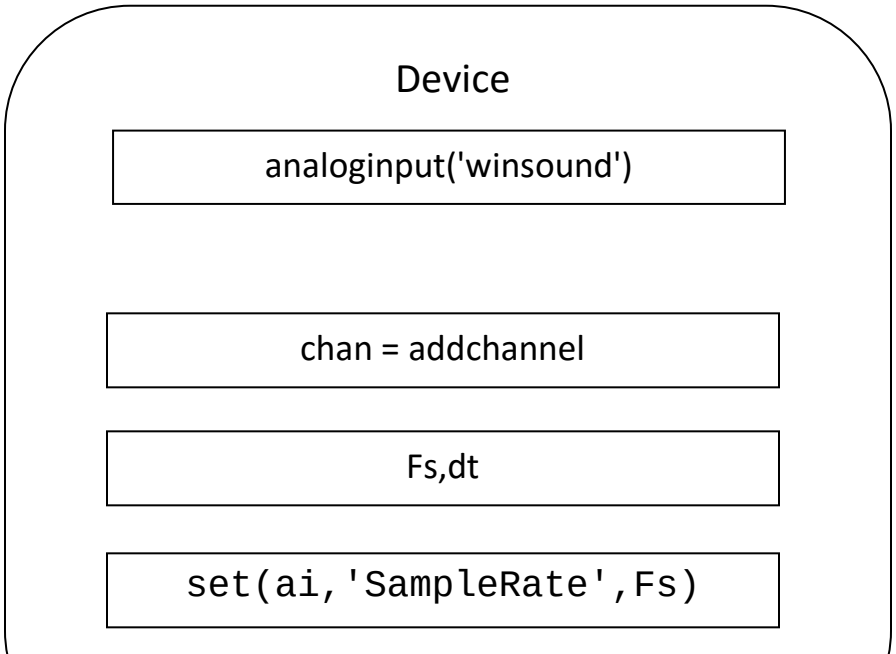


Figure 3.2

When a particular signal of specified voltage was given to the microphone port of the soundcard the following signals were generated.



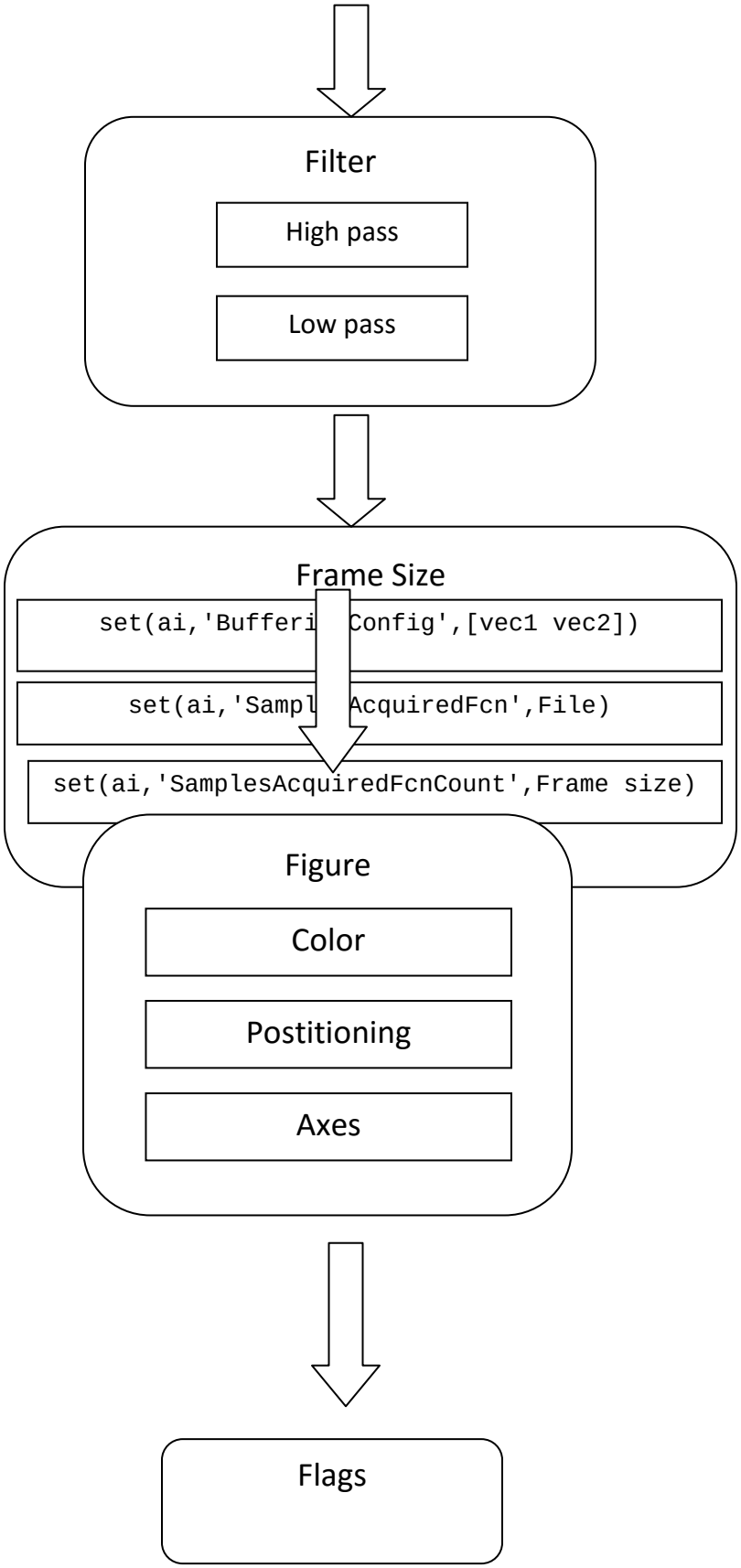

Filter
High pass
Low pass
Frame Size
set(ai,'Bufferi Config',[vec1 vec2])
set(ai,'Sampl AcquiredFcn',File)
set(ai,'SamplesAcquiredFcnCount',Frame size)
Figure
Color
Postitioning
Axes
Flags

Figure 3.3

3.5.Sinewave:

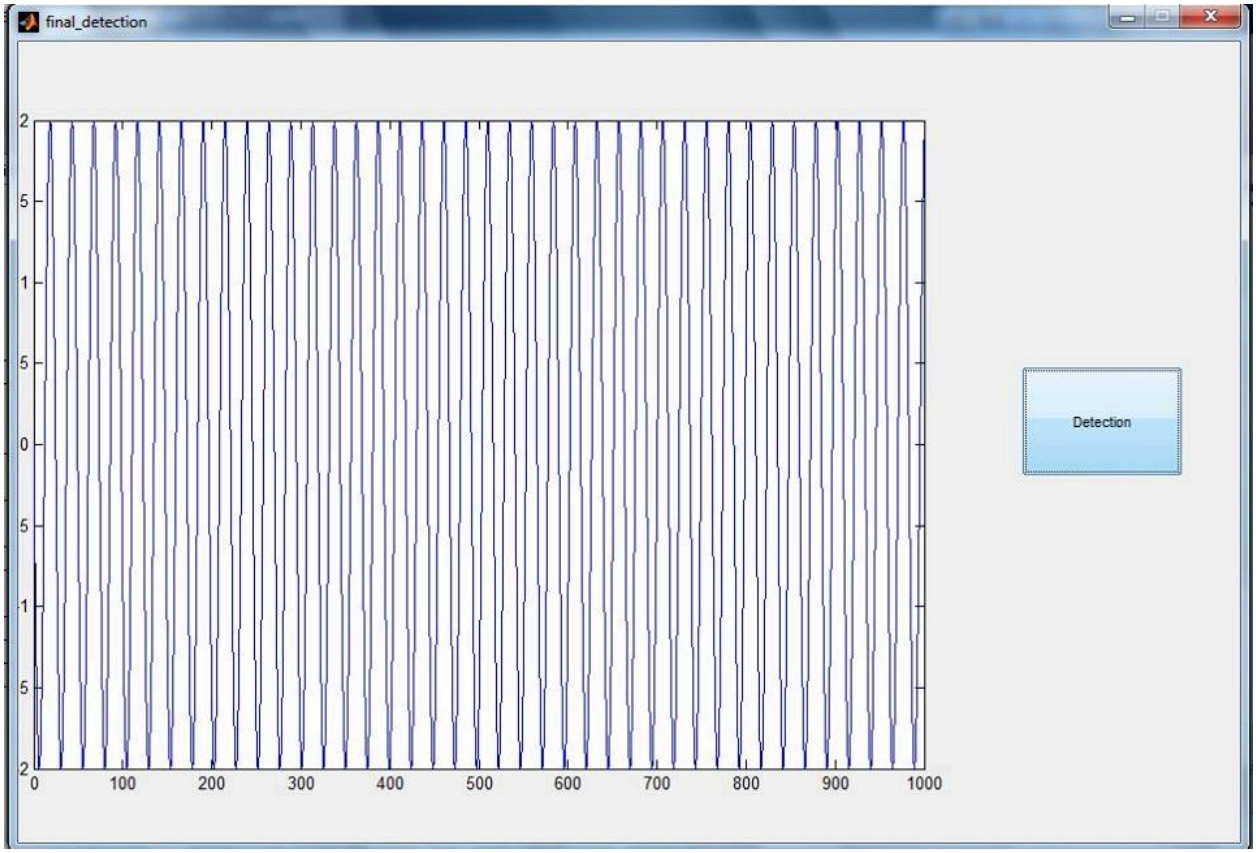


Figure 3.4

When a sinewave is passed from signal generator through the microphone port of the soundcard of the PC which we are working with the corresponding sinewave appears on the GUI axes window. Pressing the button of detection helps us get this final sine wave once the final codes are run.

3.6.Square wave

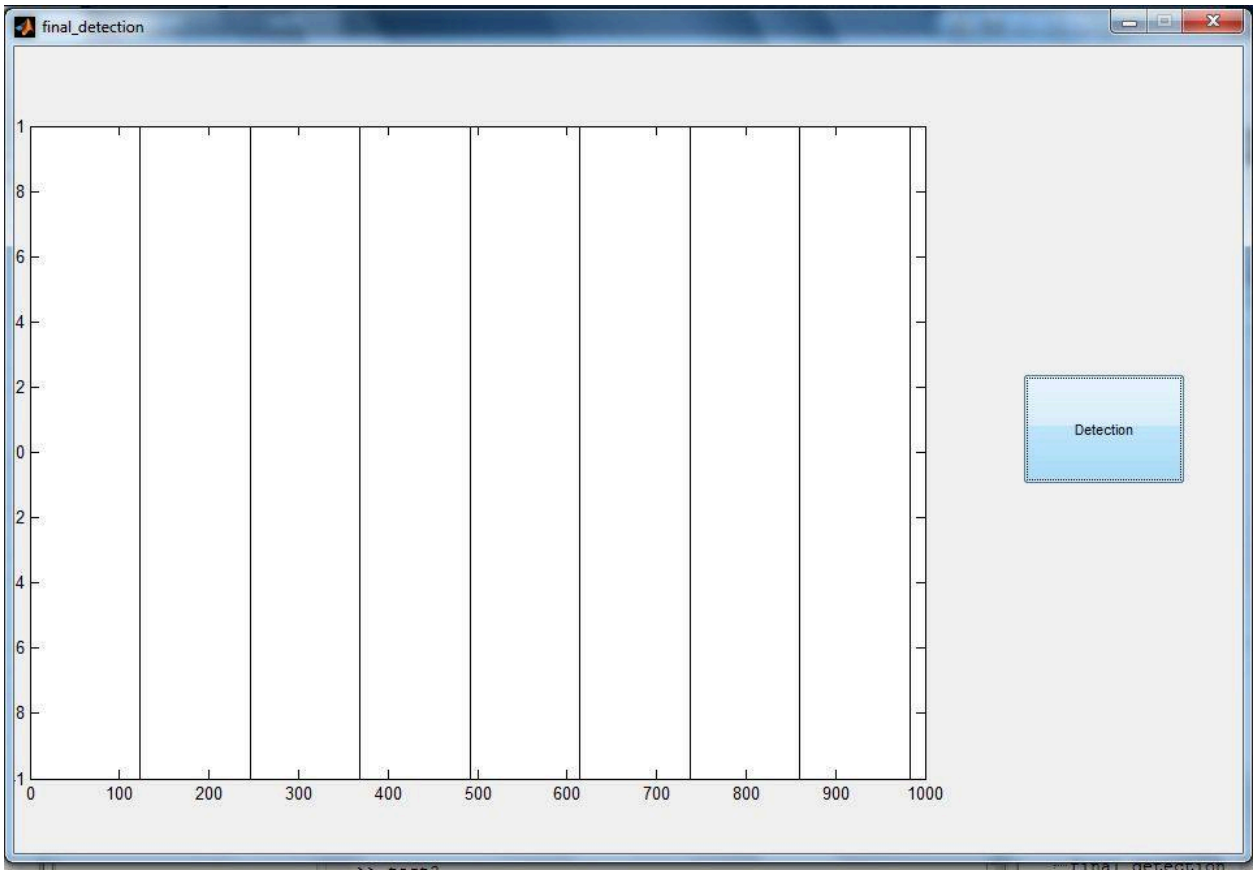


Figure 3.5

Similarly when a square wave is passed from a signal generator through the microphone port of soundcard of the PC, with the help of Data Acquisition toolbox on MATLAB and proper MATLAB coding this final wave is achieved and shown on the GUI axes window.

3.7.Sawtooth wave

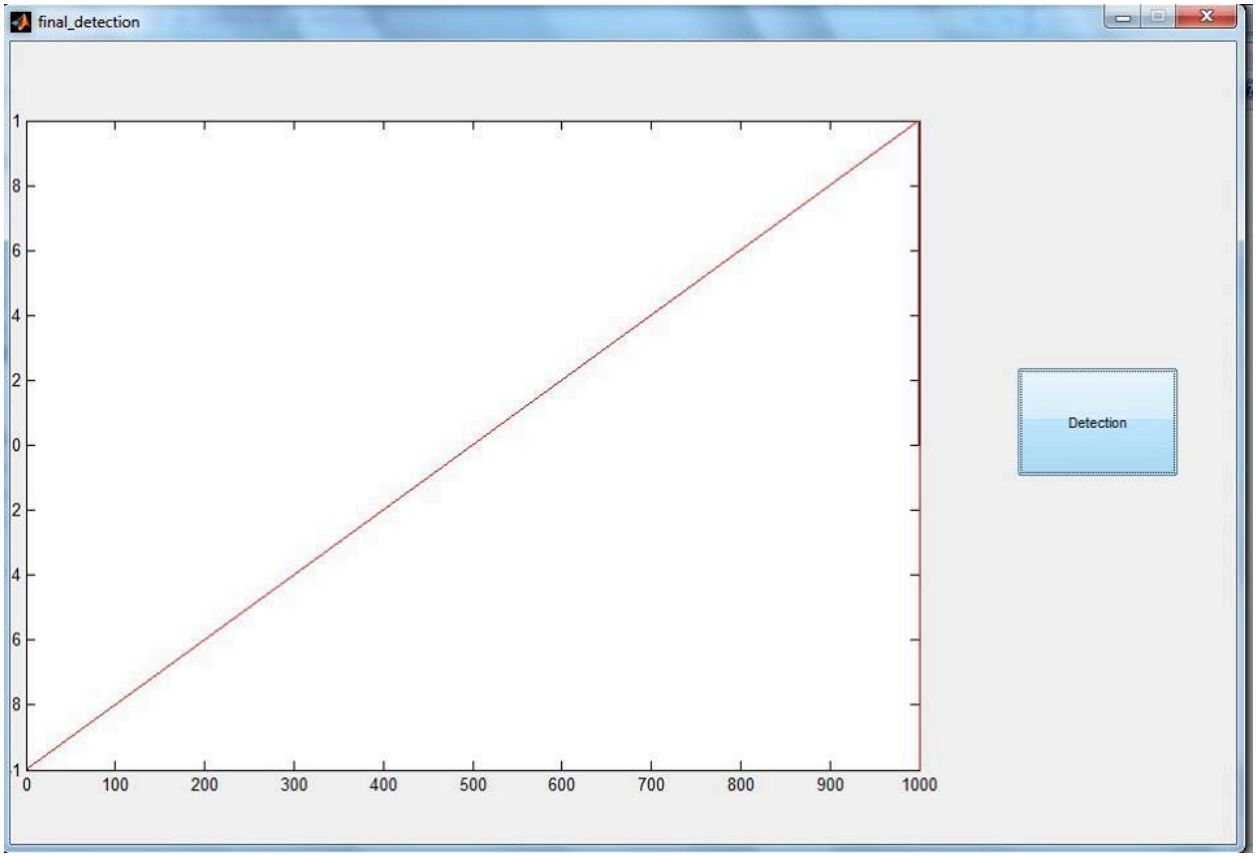


Figure 3.6

Similar to the previous two waves that were being achieved from MATLAB coding here too, when a sawtooth wave is passed from a signal generator through the microphone port of PC's soundcard, running the finals codes the window of GUI appears where after pressing the button of "detection" we get the desired sawtooth wave on the GUI axes window.

4.Changing Amplitude and Frequency of Signal Using Arduino Uno

For the amplification of signal being generated and detected one boundary we had was the signal that was being formed had very low voltage amplitude, in practical work such signals won't be of much use if not amplified hence we decided to externally amplify the signal. Arduino maps voltage values from 0-5 into a range of integer values 0-1023

The process of amplifying the signal we first took an attempt of was through Arduino Uno, generally an Aduino Uno looks like the one below.

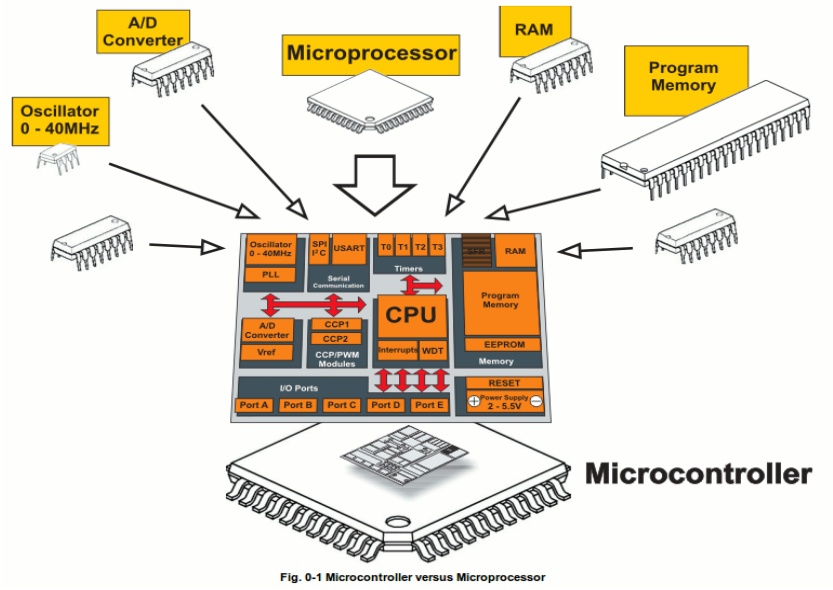


Figure 4.1

It is a small computer on a single chip with processor memory and input/output which is embedded inside typically that they can control and mostly of small and low cost.

Typical components include:

- power circuit
- programming interface
- basic input; usually buttons and LEDs
- I/O pins

An arduino is operated through programming and the way it is done includes few steps which are:

1. **Download & install the Arduino environment (IDE)**
2. **Connect the board to your computer via the UBS cable**
3. **If needed, install the drivers**
4. **Launch the Arduino IDE**
5. **Select your board**
6. **Select your serial port**
7. **Open the blink example**
8. **Upload the program**

After the connecting the USB cable to arduino port shown below the codes gets uploaded on the Arduino Uno chip and start functioning.

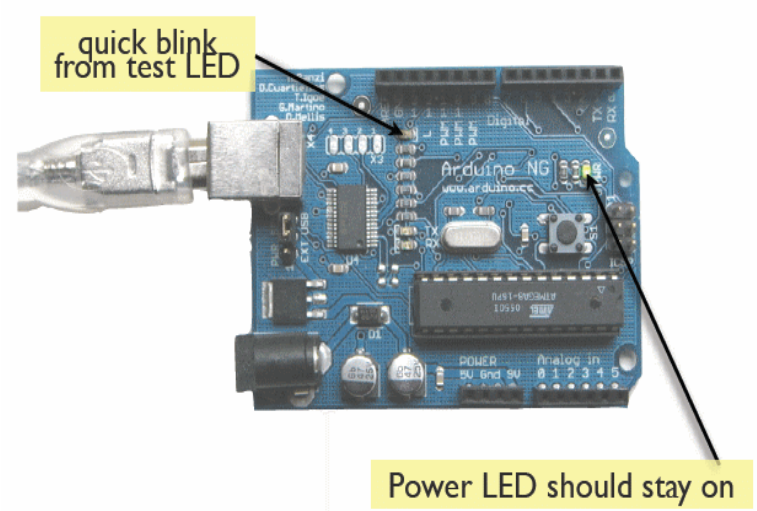


Figure 4.2

For our task we used the Arduino Uno's analog read/write pins where pins are an input or output connected to something for example output to a LEB and input from a knob Output pins can provide 40 mA of currentWriting HIGH to an input pin installs a 20KΩ pull-up. The value for Arduino analog pin ranges from 0-255.

While writing the programming codes on the window of arduino to read a signal externally we have to write the command "analogRead(pin)" which would read the value from a specific arduino pin. Time required reading a signal is estimated to be 100 milliseconds so the maximum reading rate is 10,000 times a second.

The command "analogWrite(pin, value)" writes an analog value to the specified pin. It generates a steady square wave of specified duty cycle until the next call. The frequency of the PWM signal on most pins is approximately 490 Hz.

Initially for amplification we used these commands to externally amplify the detected before showing the results on graphical user interface on MATLAB and also externally amplify the generated signal through MATLAB before being tested on circuitry board do give desired voltage. The problem we faced on doing so was:

- The results had too much noise and the signals being detected was distorted which would complicate the further implication of this oscilloscope which is why we could not proceed further with this process so had to shift to our next option of using a micro-controller.
- If we want to measure the signal with one of the Arduino's analog inputs, the Arduino can only measure voltages between 0 and 5V. Measuring negative voltages in the signal Arduino would read only 0V and we would end up clipping the bottom of the signal.

5.Procedures

"In developing a project, methodologies are one of the most important elements to be considered to make sure that the development of the project is smooth and get the expected result. Good methodologies can describe the structure or the flow of the project whereby it can be the guideline in managing it. It is also to avoid the project to alter course from the objectives that have been stated or in other words the project follow the guideline based on the objectives."

5.1 Signal Generation

The code gives us continuous signal generation with the help of a while loop. The 'linspace' command ensures the number of samples taken and the sample width. The signal equation is set with the frequency (fm), amplitude (amp) and the time (t) as variable which is set by the user in the Graphical User Interface once the run button is pressed. The 'soundsc' command scales the values of audio signal 'x' to fit in the range from –1.0 to 1.0 V, and then sends the data to the speaker at the default sample rate of 8192 hertz. By first scaling the data, 'soundsc' plays the audio as loudly as possible without clipping. The mean of the dynamic range of the data is set to zero. The 'pause' command allows the user to stop the generation at any point without closing MATLAB.

5.2 Signal Detection

At first we have to create a MATLAB object using ‘analoginput’ command which is a feature of the Data Acquisition Toolbox (DAQ). Then we create a channel with ‘addchannel’ to acquire data through soundcard. The sampling frequency (Fs) is set to 44100 Hz and the sampling period (dt) is determined. The ‘set(ai,'SampleRate',Fs)’ command sets the channel frequency to our desired sample frequency. The ‘set(ai,'SamplesPerTrigger',Inf)’ sets the number of samples to be acquired from the sample to infinite for continuous detection. The parameters for the low pass and high pass filters are saved in a different MATLAB object. When the filters are activated, the plot is drawn to satisfy the object parameters. The ‘set(ai,'BufferingConfig',[ps 32])’ command sets the frame size and number at which we view the plot. Then the ‘set(ai,'SamplesAcquiredFcnCount',ps)’ command sets the rate at which the plot acquires samples for the channel. Finally the ‘set(ai,'SamplesAcquiredFcn','piece_processing')’ command specifies the ‘piece processing’ M-file to execute every time a predetermined number of sample is acquired. Later the color,positioning and axes of the figures are specified along with the Flags(pause and stop).

6.Discussion

6.1.Advantages:

Our Software Based Signal Generator and Oscilloscope has few advantages and it would be

- ✓ It is very cost effective, compared to the oscilloscopes and signal generator we get to buy from market which we use in the laboratory.
- ✓ Generates complete signal of Sine, Square and Sawtooth wave with amplitude of 1 volt.
- ✓ It is easier to operate because it is to be seen than many people faces difficulties while operating a signal generator or oscilloscope practically due to its many knobs and procedures but we ensured to keep our entire design user friendly so that once instructed anyone can use it easily.
- ✓ We made it portable, i.e. as it is a souncard based signal generator and oscilloscope the entire procedure is carried out through programming language and we can have access to it from any PC or Laptop once the codes are run on MATLAB compiler.
- ✓ We can print the waveform present on screen which facility is not provided in the conventional Oscilloscope.
- ✓ PC screen is larger so the waveforms can be seen more clearly.
- ✓ Lower power consumption.

6.2.Drawbacks

- Applying high voltage has a risk of damaging the soundcard of the PC or laptop hence to overcome it in future we have find out a way to externally amplify it through Arduino Uno or external circuits.
- PC based signal generator cannot generate frequency greater than 20 KHz.
- There is no multi-channel facility which is useful for phase measurement of two or more waveform that applied at the input.
- Cannot generate or detect signals with amplitude greater than 1 V.
- Data Acquisition toolbox is mostly available for 32 bit PC.

7.Future Implementation

Unlike other systems our system assures a convenient and cost effective method to generate and detect signals which could be utilized easily for laboratory purposes. However if we do some more research and come up with a proper way of using external circuitry, specifically an operational amplifier (op-amp) then we can amplify the signals generated by signal generator which could increase the usage of signal generator in everyday use. For amplification even if we use Arduino Uno instead then it would be of more benefit but for that first we need to find a proper procedure to remove the excess

noise we get while when the signal is being generated and alongside when attempted to amplify signals using Arduino Uno, one of the major reasons we could not take it further because the amplified signal was distorted and had too much noise.

Creating an execution file for both the codes of generation and detection will minimize huge number of separate m.scripts and once the execution file is run it work smoothly itself as software. This in return will reduce more time to run this program and generation and detection both would become easier. We can add another channel and compare two signals simultaneously.

8.Conclusion

The whole idea of the Software based signal generator and oscilloscope was to ensure the capability of using MATLAB programming language and it's callback commands to create a user friendly interface to the user to get access to a signal generator and oscilloscope from anywhere around the world. Even though both signal generator and oscilloscopes is old instrument already available at the market, but making it Software based so that a user can get access of it from their PC/ laptop, have made it more cost effective and easier to operate. This paper's goal was to familiarize a person with various MATLAB codes that we used on the process of making the signal generator and oscilloscope, to thoroughly give brief explanation of each of the commands so that the person going through the paper can have a clear idea of the codes.

References

1. Roland Szabó, AurelGontean, Ioan Lie, MirceaBăbăiOă,"Oscilloscope Control with PC". INTERNATIONAL JOURNAL OF COMPUTERS AND COMMUNICATIONS,Issue 3, Volume 3, 2009.

2. Brian Costello, "GRAPHICAL INTERFACE CONCEPT FOR A SIGNAL DETECTION PROCESS"In-House Technical Memorandum, February 2003

3. Brian D. Storey, "Using the MA TLAB Data Acquisition Toolbox".

4. Ritika, PreetiKumari, PremRanjanDubey,"DESIGNING A PC OSCILLOSCOPE USING FREEDUINO",Birla Institute of Technology, Mesra, Students of Department of Electronics and Communication,22nd May, 2013.

5. "LEARNING TO PROGRAMWITH MATLABBuilding GUI Tools",2013,ISBN 978-0-470-93644-3,Printed in the United States of America.

6. Ariffuddin bin Joret, "PC-MATLAB based Signal Generator Development", [Online]. Available:http://eprints.uthm.edu.my/6106/1/PCMATLAB_ based_Signal_Generator_Development.pdf,proceeding of National Conference of Electric and Electronic Engineering, 2012.

7. Debraj, “PC Based Signal Generator”,[Online].

Available:https://sites.google.com/site/hobbydebraj/home/pc-basedsignal-

generator.

8. Muhammad Lokman Al Hakim Bin Abu Bakar,”Graphical User

Interface For Signal Generator”,[Online].Available:

http://umpir.ump.edu.my/76/1/cd2637.pdf .

9. [Online]. Available: https://www.mathworks.com/products/daq/codeexamples.

html?file=%2Fproducts%2Fdemos%2Fdaq%2Facquiring_d

ata%2Facquiring_data.html .

9.Appendix

Final Code for Signal Generation

```
Function pushbutton1_Callback(hObject, eventdata, handles)

amp=str2double(get(handles.edit_amplitude,'string'));

fm=str2double(get(handles.edit_Frequency,'string'));
```

The command below edits the **Amplitude** and **Frequency** of the desired signal being generated, for amplitude:

```
if ispc && isequal(get(hObject,'BackgroundColor'),

get(0,'defaultUicontrolBackgroundColor'))

 set(hObject,'BackgroundColor','white');

end
```

This command below is to change the frequency of the generated signal:

```
if ispc && isequal(get(hObject,'BackgroundColor'),

get(0,'defaultUicontrolBackgroundColor'))

 set(hObject,'BackgroundColor','white');

end
```

The code below generates a sine wave of voltage 1 volts peak to peak

```
y=1;



while (y<2)



 %t=linspace(0,10,time);

  t=linspace(0, 1000, 1*10000);

   x=sin(2*3.14*fm*t);

   %x1=amp;

   plot (t,x*amp);

   soundsc( x*amp);

   pause(0.00000001);

end
```

The code below generates a sine wave of voltage 1 volts peak to peak

```
while (y<2)
  %t=linspace(0,10,time);
  t=linspace(0,1000,10000);
  x=square(2*3.14*fm*t);
   %x1=amp;
  plot (t,x*amp,'k');
  soundsc( x*amp);
  pause(0.00000001);
end
```

Here are the codes we have used for the generation of a sawtooth wave:

```
while (y<2)


  %t=linspace(0,10,time);
  t=linspace(0,1000,10000);
  x=sawtooth(2*pi*fm*t);
   %x1=amp;
  plot (t,x*amp,'r');
  soundsc( x*amp);
  pause(0.00000001);
end
```

Final codes for Signal Detection, i.e. Oscilloscope

```
function varargout = final_detection(varargin)

% FINAL_DETECTION MATLAB code for final_detection.fig

%      FINAL_DETECTION, by itself, creates a new FINAL_DETECTION or raises
the existing

%      H = FINAL_DETECTION returns the handle to a new FINAL_DETECTION
or the handle to

%      FINAL_DETECTION('CALLBACK',hObject,eventData,handles,...) calls the
local

%      function named CALLBACK in FINAL_DETECTION.M with the given
input arguments.

%      FINAL_DETECTION('Property','Value',...) creates a new
FINAL_DETECTION or raises the

%      existing singleton*.  Starting from the left, property value pairs are

%      applied to the GUI before final_detection_OpeningFcn gets called.  An
```

```
%      stop.  All inputs are passed to final_detection_OpeningFcn via
varargin.
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @final_detection_OpeningFcn, ...
                   'gui_OutputFcn',  @final_detection_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end
if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
```

```
end

% End initialization code - DO NOT EDIT

function final_detection_OpeningFcn(hObject, eventdata, handles,

varargin)

handles.output = hObject;

 guidata(hObject, handles);

function varargout = final_detection_OutputFcn(hObject, eventdata,

handles)

varargout{1} = handles.output;

% --- Executes on button press in pushbutton1.

function pushbutton1_Callback(hObject, eventdata, handles)

run_me();
```

Codes for piece processing of the signal

```
function piece_processing

global ai ps st hp pc paus frs datas

global highpass lowpass numerator_hp numerator_lp

if st

   stop(ai);

   delete(ai);

else

   data = getdata(ai,ps);

   %datas(1:ps)=datas(ps+1:2*ps);

   datas(1:2*ps)=datas(1*ps+1:3*ps);

   %datas(ps+1:2*ps)=data;

   datas(2*ps+1:3*ps)=data;

   pc=pc+1;
```

```
if (mod(pc,frs)==0)&&(~paus)
   [tmp ii20]=max(data); % syncrhinize to maximum
   %datas1=datas(ii20:ps+ii20-1);
   if (~highpass)&&(~lowpass)
      datas1=datas(ps+ii20:2*ps+ii20-1);
      %datas1=datas(ps+ii20-0:2*ps+ii20-1);
      %datas1(0+1:end)
   end
   if highpass
      datas10=datas(ps+ii20-300:2*ps+ii20-1);
      datas10=filter(numerator_hp,1,datas10);
      datas1=datas10(301:end);
   end
   if lowpass
      datas10=datas(ps+ii20-300:2*ps+ii20-1);
```

```
            datas10=filter(numerator_lp,1,datas10);

            datas1=datas10(301:end);

        end

        set(hp,'YData',datas1);

        drawnow;

    end

end
```