**TABLE OF CONTENTS**

[**CHAPTER 1** 4](#_Toc89012494)

[**INTRODUCTION** 4](#_Toc89012495)

[**1.1** **Introduction** 4](#_Toc89012496)

[**1.2** **Objective** 7](#_Toc89012497)

[**CHAPTER 2** 8](#_Toc89012498)

[**DESIGN AND FLOW OF ROUTINES** 8](#_Toc89012499)

[**CHAPTER 3** 10](#_Toc89012502)

[**FORMAT AND STYLE OF ROUTINES** 10](#_Toc89012503)

[**CHAPTER 4** 14](#_Toc89012504)

[**RESULT AND ANALYSIS** 14](#_Toc89012505)

[**CONCLUSION** 23](#_Toc89012506)

**REFERENCES** 22

**TABLE OF FIGURES**

**FIGURE NO TITLE PAGE NO**

[Figure 1 Different types of waveform 4](#_Toc82614937)

Figure 2 Sine wave & Sine wave generator circuit. 5

Figure 3 Triangle wave & triangle wave generator circuit. 5

Figure 4 Square wave & Square wave generator circuit. 6

Figure 5 Sawtooth wave & wave generator circuit. 6

Figure 6 Flowchart 8

Figure 7 Basic GUI 14

Figure 8 Second tab. 15

Figure 9 Error validation. 15

Figure 10 Input error 16

Figure 11 Stop time visibility off 17

Figure 12 Input from user 17

Figure 13 Sine Output 18

Figure 14 Sine output with variations. 18

Figure 15 Square wave output. 19

Figure 16 Triangle wave output. 19

Figure 17 All variations output. 20

# **CHAPTER 1**

# **INTRODUCTION**

## **Introduction**

In signal processing, a signal is a function that communicates information about an occurrence to a receiving station. The term "information signal" refers to any time-varying voltage, current, or electromagnetic wave that carries information in electronics and telecommunications. A signal is a current of electricity or an electromagnetic field that is used to transmit data from one location to another. In electronics, a signal generator is a type of electronic device that generates electronic signals with certain characteristics such as amplitude, frequency, and waveform. In electronic measurements, these generated signals are utilized as a stimulus. They are primarily used in the design, testing, debugging, and repair of electrical or electroacoustic devices, though they are also frequently used in artistic endeavors as well.

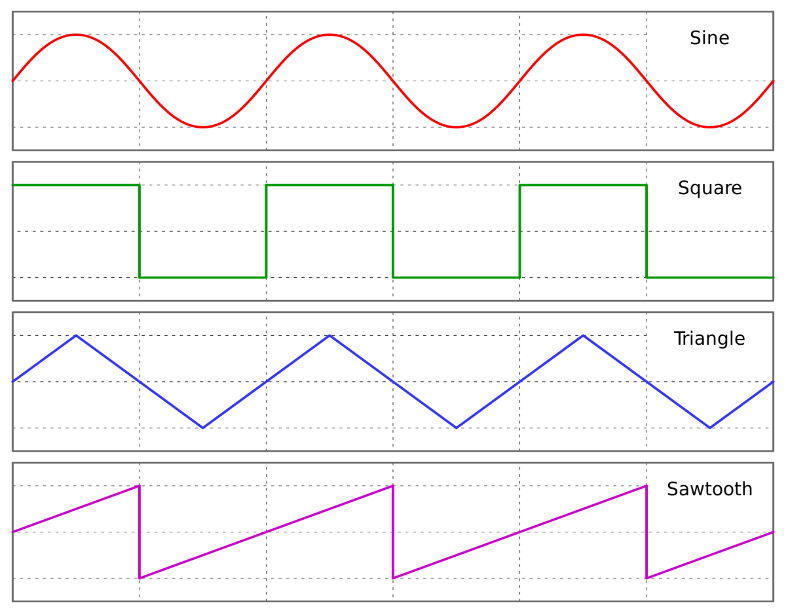


Figure 1:- Different types of waveform

**Types of waveform**

A sine wave is a geometric waveform that oscillates (moves up, down or side-to-side) frequently, and is defined by the function y = sin x. In other terms, it is an s-shaped, smooth wave that oscillates above and below zero.

Chart, line chart

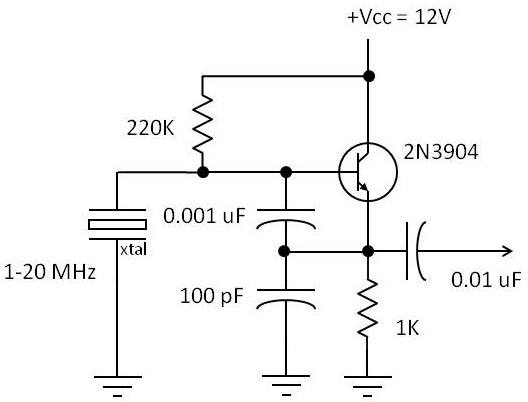
Description automatically generated 

Figure 2:- Sine wave & Sine wave generator circuit.

A triangular wave is a non-sinusoidal waveform that gets its name from its triangular shape. It is a real function that is periodic, piecewise linear, and continuous. The triangle wave, like the square wave, is composed solely of odd harmonics. The higher harmonics, on the other hand, decay considerably more quickly than in a square wave (proportional to the inverse square of the harmonic number as opposed to just the inverse).

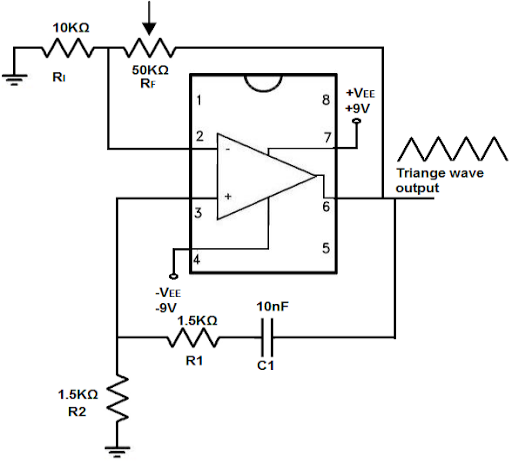
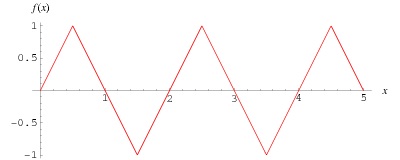


Figure 3:- Triangle wave & triangle wave generator circuit.

Non-sinusoidal periodic waveform with amplitude alternating at a constant frequency between defined minimum and maximum values, and with the same duration at both minimum and maximum, a square wave is formed. The transitions between the minimum and maximum values of an ideal square wave are instantaneous in nature. The square wave is a special instance of the pulse wave in that it allows for variable durations at both the minimum and maximum amplitudes of the waveform. The duty cycle of a pulse wave is defined as the ratio of the high period to the entire period of the pulse wave. A true square wave has a duty cycle of 50 percent, which is optimal (equal high and low periods). Square waves are frequently seen in electronic and signal processing applications, notably in digital electronics and digital signal processing applications.

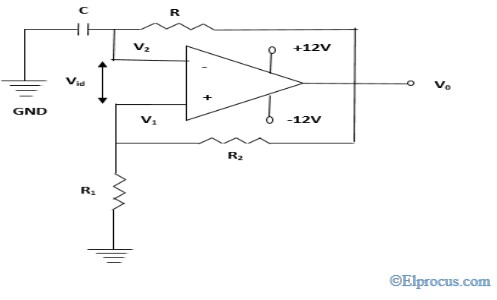
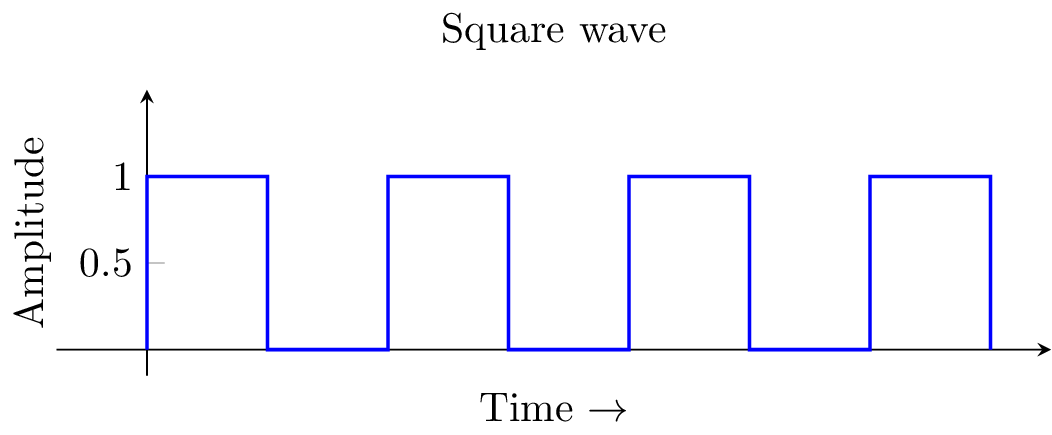


Figure 4:- Square wave & Square wave generator circuit.

The sawtooth wave (also known as the saw wave) is a type of non-sinusoidal waveform that can be generated. It was given this name because it resembles the teeth of a plain-toothed saw with a zero-rake angle, which is how it got its name. A ramp waveform is a single sawtooth, or a sawtooth that is produced intermittently, that is activated only once. According to convention, a sawtooth wave begins with an upward ramp and then abruptly lowers. A reverse (or inverse) sawtooth wave is characterized by a gradual decline followed by a sudden ascent. Alternatively, it can be thought of as the extreme example of an asymmetric triangular wave.

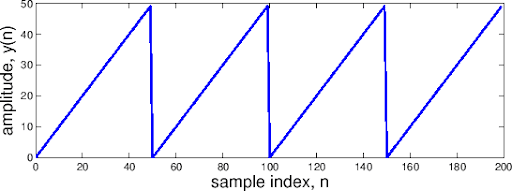
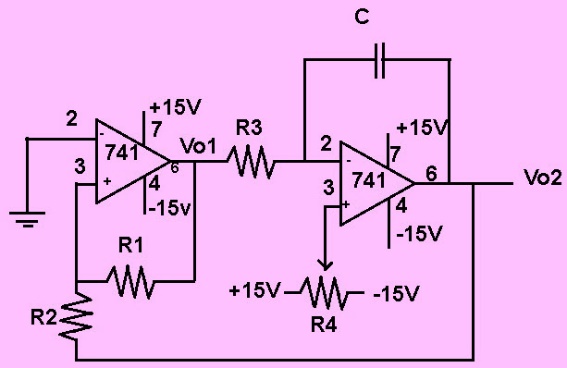
 

Figure 5:- Sawtooth wave & wave generator circuit

Mathematical modelling and simulation (MATLAB) are the programming language utilized for this project. MATLAB is a high-performance programming language for technical computer applications. In a simple-to-use environment, computing, visualization, and programming are all combined into a problem-solving environment in which problems and answers are expressed in mathematical notation that users are familiar with. A Graphical User Interface software can be developed with MATLAB in a variety of ways, including using GUIDE, APP designer, and Simulink, among others. For this task, we'll be working with a program called APP Designer. With the help of the MATLAB App Designer function, which is included in the MATLAB program, it is possible to design interactive applications. App designer simplifies the process of designing a graphical user interface (GUI) by adding drag and drop work and providing auto functions for them; as a result, the user only needs to implement the logic of their software rather than separate coding for the GUI.

## **1.2 Objective**

* User can select the type of signal generator to be simulated in a drop-down list.
* User can key in the input amplitude and the frequency.
* User can view the plot of the output waveform by clicking the simulation button.
* Different variation of different signal generator should be displayed accordingly.
* Your system should be able to perform error checking routine for invalid entries by users.
* You may enhance the appearance of the GUI by proper styling and changing the properties of the UI-objects contained in the GUI.
* Kindly provide at least one logical innovative idea and justify it in the report.

# **CHAPTER 2**

# **DESIGN AND FLOW OF ROUTINES**

# Diagram Description automatically generated

Figure 6: Flowchart.

Based on the program's requirements, an algorithm was developed, and the algorithm was then translated into a flowchart for use in the program's implementation. MATLAB program flow chart illustrating the entire procedure of the MATLAB program in detail

The implementation of the graphical user interface (GUI) will be the initial part of the software development process. The program has a number of components, such as an edit text area for entering the frequency, amplitude, and stop time. A drop-down menu was added to allow for the selection of the type of signal to be made. In the code portion, the implementation of properties was done first, before anything else. All of the variables that would be used in the program were declared, defined, and initialized before the program began. The varied waveform values are stored in variables, as is the simulation of button status, as well as temporary waveform values, among other things. The time variable was initialized to the length of the simulation time-period; this variable determined how long the simulation would run in seconds, measured in seconds.

The waveform will be selected by the user from a drop-down menu. The if statements are activated based on the user's selection. All of the distinct waveforms have their own conditional loops, and the operation of the loop will be detailed in further detail in the following section. The several variations of the waveform will be presented alongside the current waveform plot at the conclusion of the analysis. If the user presses the simulated all button, the waveform will be plotted in its entirety at the same time.

# **CHAPTER 3**

# **FORMAT AND STYLE OF ROUTINES**

This section will contain the code of the developed system, the explain for some of the part of codes will be done. The codes which are explained in this section are the codes which are not similar and required an explanation for the code.

properties (Access = public)

%included all the variables required for the users

Property % Description

%below four variables hold the value of four wavetype

sawtooth = 0;

traingle = 0;

sine = 0;

square = 0;

set = 0;

frequency = 50;%frequency variable for signal generator

amplitude = 5;%frequency variable for signal generator

nofsamples = 1000;%number of samples variable for signal generator

stoptime = 0.25;%for number of time the simulation will be performed

buttonstate = 0;%hold the simulation state

waveformtype = 'SINE' ; %waveform state variable from the drop down

sinetemp=0 ; % holding temporary value of the sine for variations

traingletemp = 0 ; % holding temporary value of the traingle for variations

sawtoothtemp= 0 ; % holding temporary value of the sawtooth for variations

sqauretemp = 0 ; % % holding temporary value of the square for variations

iterationsine=0; % holding iteration value of the sine for variations

iterationtraingle=0; % holding iterations value of the traingle for variations

iterationsquare=0; % holding iterations value of the square for variation

iterationsawtooth=0; % holding iterations value of the sawtooth for variation

buttonstateall = 0; %holding button state for all simulation tab

end

It is necessary to declare all variables used in the programme in the properties created above, because they will be used in different components in the app designer. If the variables are to be used in multiple components in the app designer, they must be specified in the properties. All of the variables that were required for the programme were defined, including variables to store each type of waveform, variables to hold for user selections and input, variables to hold the state of buttons, variables to store the values of waveforms that had been previously executed, and so on. Comments were supplied in each needed line, allowing the scripts to be easily comprehended by the reader.

% Value changed function: AmplitudeEditField

function AmplitudeEditFieldValueChanged(app, event)

value = app.AmplitudeEditField.Value;

if value <= 0

errordlg('Enter a Positive Value','Input error');

else

app.amplitude = value;

end

waveform(app)

end

The above line of code is for the amplitude value which the user will input. Since the edit field type is numeric any string type will not be taken as input and will show an error that the value must be numeric. To make the input error free an algorithm was made to allow users to only input positive value, this was implemented by using a validation variable ‘value’ and compared it with 0 using the less than equal to operator. If the value is less than 0, an error dialog will be displayed mentioning that user can enter only positive value.

function SimulateAllButtonValueChanged(app, event)

app.buttonstateall = app.SimulateAllButton.Value;

waveformall(app)

end

The above line of code is for the ‘Simulate all’ button, the ‘Simulate all’ button was used for calculating all the types of waveform based on the amplitude, frequency and other parameters required for the calculation. To do this the state of the button is stored in a variable and the function for the operation is called.

function StopTimesecondsEditFieldValueChanged(app, event)

value = app.StopTimesecondsEditField.Value;

if value <= 0

errordlg('Enter a Positive Value','Input error');

else

app.stoptime = value;

end

app.StopTimesecondsEditField.Visible = 'off'

app.StopTimesecondsEditFieldLabel.Visible ='off'

waveform(app)

end

The above line of code is for the stop time edit value, where the similar algorithm for input error was implemented. But since the Stop time for an execution needs to be inputted for one time, after the user will provide a valid input the visibility for edit field and the label is turned off, so user can provide a valid input for once.

function waveform(app)%function for simulation button

%unless the button is in on state executions wont work

if app.buttonstate == 1

dt = 1/app.nofsamples;%derivate of time

t= (0:dt:app.stoptime-dt)';%time function for the graph

if strcmpi(app.waveformtype, 'SINE') %drop down variable for sine

if(app.iterationsine ~=0)%checking whether the first iteration is performed or not

app.sinetemp = app.sine;% assign the previous sine value to temporary value

end

app.sine = app.amplitude\*(cos(2\*pi\*app.frequency\*t));% calcution for sine values based

%on the formula

%below lines plot the values on all the differnt axis

%required and hold the plot for variation comparisons

plot(app.UIAxes,t,app.sine)

plot(app.UIAxes\_2,t,app.sine)

hold(app.UIAxes\_2,'on')

plot(app.UIAxes\_2,t,app.sinetemp)

app.iterationsine = 1;%this variable will make sure that the next execution onwards

%the temporary vriable will store the previous value

The above line of code is for the waveform function which is the main functions and gets called after time a user enters a input for any of the parameters. function first check for the condition that whether user pressed the simulate all button or not. For the derivative of time the half the value of no of samples was taken. For the time variable an array was interpolated, the array started form 0 with an increment of dt sample and end variable of stop time – dt. The nested if else condition loop is for the validation of the drop-down components. If the waveform selected is sine type, the nested loop will check that whether the execution is for the first time or not, if the execution is first time it will executes the rest of the body, else it will first store the current sine value to a temporary variable and then executes rest of the loop. Sine waveform is calculated using the formula and stored in the sine variable. Graph was plotted using the plot command, two graph components was used for plot. One of the component was for the current sine wave plot while the other was for variation of the sine value. This was achieved by using the hold function, after the first value is plotted the graph was hold, this made the next value to be plotted on top of it.

function waveformall(app)

if app.buttonstateall == 1

dt = 1/app.nofsamples;

t= (0:dt:app.stoptime-dt)';

app.sine = app.amplitude\*(cos(2\*pi\*app.frequency\*t));

app.square = app.amplitude\*square(2\*pi\*app.frequency\*t);

app.traingle = app.amplitude\*sawtooth(2\*pi\*app.frequency\*t, 1/2);

app.sawtooth = app.amplitude\*sawtooth(2\*pi\*app.frequency\*t);

plot(app.UIAxes5,t,app.sine)

hold(app.UIAxes5,'on')

plot(app.UIAxes5,t,app.traingle)

plot(app.UIAxes5,t,app.square)

plot(app.UIAxes5,t,app.sawtooth)

hold(app.UIAxes5,'off')

end

end

The above line of code is Waveformall function, this function first check for the condition that whether user pressed the simulate all button or not. For the derivative of time the half the value of no of samples was taken. For the time variable an array was interpolated, the array started form 0 with an increment of dt sample and end variable of stop time – dt. The time variable is for the x axis in the plot. As per formula the four waveform were initialized. Graph was plotted in the ALL tab, hold function was used for holding the graph and displaying all the graphs in one plot.

# **CHAPTER 4**

# **RESULT AND ANALYSIS**

A picture containing calendar

Description automatically generated

Figure 7: Basic GUI.

Figure 7 & Figure 8 shows the developed program GUI, where all the required parameters are included in the program. Six tabs were included. First tab is for input which contains all the components required for the input from the user, it also contains simulate and simulate all buttons which is necessary for simulation. Next five tabs are for the graph for different conditions and waveforms.

Timeline

Description automatically generated

Figure 8: Second tab.

Calendar

Description automatically generated with medium confidence

Figure 9: Error validation.

Figure 9 shows the error validation, since in figure it is mentioned that the value must be in numeric. This shows that if the user will enter any input other than numeric it will display an error, and this is valid for all the entries. Figure 10 shows another input validation where if the user will provide input of value less than 1, error dialog will be thrown mentioning to enter a positive value.

Graphical user interface

Description automatically generated

Figure 10: Input error.

Figure 11 & 12 shows the visibility of the stop time, in figure 12 all the input parameters are seen, while in the figure 11 it can be seen that after the user enters the stop time parameters the label and the stop time edit field value component was hidden or its visibility was off.

Calendar

Description automatically generated

Figure 11: Stop time visibility off.

A picture containing calendar

Description automatically generated

Figure 12: Input from user.

A picture containing calendar

Description automatically generated

Figure 13: Sine Output.

A picture containing timeline

Description automatically generated

Figure 14: Sine output with variations.

Figure 13 shows the output of the sine wave, while the figure 14 shows the output of the sine with variations. Since the axis can be seen that the first output amplitude is 10 while the another output is 15.

Timeline

Description automatically generated

Figure 15: Square wave output.

`Calendar

Description automatically generated with low confidence

Figure 16: Triangle wave output.

A picture containing text

Description automatically generated

Figure 17: All variations output.

Figure 15 and Figure 16 shows different waveform outputs, figure 15 shows the square wave generator output while the figure 16 shows triangle wave generator output. Figure 17 shows the all variations output, where all the four waveforms were plotted.

**CHAPTER 6**

**DISCUSSION**

First objective was to allow user to select the type of signal generator to be simulated in a drop-down list, this was achieved by implementing drop down component, the callback function was then initialized with a variable to hold the drop-down selection. Second objective was to user can key in the input amplitude and the frequency, this objective was achieved through implementation of numeric edit text field and the callback function stored the input into a variable which later was used for waveform initialization. Third objective was user can view the plot of the output waveform by clicking the simulation button, this was implemented by implementing two state buttons one for simulation and another for all waveform simulation. Fourth objective was to implement variation of different signal generator that should be displayed accordingly, this was implemented by creating a temp variable for all the waveform and an iteration variable, the temp variable hold the previous value of the respective waveform, and hold function of MATLAB was used to plot the variation in the respective graph. Fifth objective was to make sure that the system should be able to perform error checking routine for invalid entries by users, this was implemented and as shown in the result section that any wrong entries by the user displayed the error message. The error valid implementation was done by using a validation variable which checks for the input, if the validation condition is not true then the error dialog will show the error message. Sixth objective was to enhance the GUI for the system, the GUI as shown in the results and analysis section the GUI contains tab, images, several graphs, the background color and even the background color for components are changed. The last objective was to implement an innovative idea, there were several innovative ideas that were implemented, first one by showing the graph of each waveforms in each different tabs which makes the GUI more interactive to the user. Another innovative idea was the implementation of simulate all button which shows all the different waveform graph in one plot which lets the user see the changes in the waveform for each type.

There were several error which were occurred during the implementation of the program, first being that once the user entered the stop time and the graph is plotted, after if the user enters the stop time error was thrown related to different dimension. This was fixed by making the visibility off for stop time after each operation until the next program execution. Another error occurred was for the iteration which was fixed by implementing a iteration validation variation and after the execution change the validation variable to another value.

**CHAPTER 7**

# **CONCLUSION**

It is possible to conclude that the assignment's required goal has been met. The assignment makes use of an application created for the construction of the graphical user interface (GUI), with a drop-down list being provided for the type of waveform selection. The editfield text component was created in order to allow the user to enter the frequency, amplitude, and stop time of the signal. The simulation was enabled and disabled using a two-state button that had two different states. Every possible error that users can make, such as entering negative or zero values for input parameters, has been addressed via the implementation of error handling. The results and analysis section, in addition, reveal that the developed system generated the values in the manner that had been predicted. In order to make it easier for other users to understand, comments have been placed in the application as well..

**REFERENCES**

|  |  |  |
| --- | --- | --- |
| **Resources** | **In-Text Citation** | **End-Text Referencing (Reference List)** |
| **Article:**  **Four.**  **Author** | MATLAB App Designer is a feature that allows MATLAB code to be packaged into an interactive software. Harun, N. H., Hambali, H. A., Hassan, M. G., & Karim, K. N. (2017). Learn by example: MATLAB app design. | Harun, N. H., Hambali, H. A., Hassan, M. G., & Karim, K. N. (2017). Learn by example: MATLAB app design. |
| **Book:**  **Two Authors** | Chaparro, L., & Akan, A. (2018). *Signals and Systems using MATLAB*. Academic Press. | Chaparro, L., & Akan, A. (2018). *Signals and Systems using MATLAB*. Academic Press. |

APPENDICES

classdef signalgeneratorv2 < matlab.apps.AppBase

% Properties that correspond to app components

properties (Access = public)

UIFigure matlab.ui.Figure

TabGroup matlab.ui.container.TabGroup

INPUTTab matlab.ui.container.Tab

WAVEFORMDropDownLabel matlab.ui.control.Label

WAVEFORMDropDown matlab.ui.control.DropDown

FrequencyEditFieldLabel matlab.ui.control.Label

FrequencyEditField matlab.ui.control.NumericEditField

AmplitudeEditFieldLabel matlab.ui.control.Label

AmplitudeEditField matlab.ui.control.NumericEditField

NumberofsamplesSliderLabel matlab.ui.control.Label

NumberofsamplesSlider matlab.ui.control.Slider

StopTimesecondsEditFieldLabel matlab.ui.control.Label

StopTimesecondsEditField matlab.ui.control.NumericEditField

SimulateButton matlab.ui.control.StateButton

SIGNALGENERATORLabel matlab.ui.control.Label

Image matlab.ui.control.Image

Image\_2 matlab.ui.control.Image

Image\_3 matlab.ui.control.Image

Image\_4 matlab.ui.control.Image

SawtoothCircuitLabel matlab.ui.control.Label

TraingleCircuitLabel matlab.ui.control.Label

SquareCircuitLabel matlab.ui.control.Label

SineCircuitLabel matlab.ui.control.Label

SimulateAllButton matlab.ui.control.StateButton

SINETab matlab.ui.container.Tab

UIAxes matlab.ui.control.UIAxes

UIAxes\_2 matlab.ui.control.UIAxes

SQUARETab matlab.ui.container.Tab

UIAxes2 matlab.ui.control.UIAxes

UIAxes2\_2 matlab.ui.control.UIAxes

TRAINGLETab matlab.ui.container.Tab

UIAxes3 matlab.ui.control.UIAxes

UIAxes3\_3 matlab.ui.control.UIAxes

SAWTOOTHTab matlab.ui.container.Tab

UIAxes3\_2 matlab.ui.control.UIAxes

UIAxes3\_4 matlab.ui.control.UIAxes

ALLTab matlab.ui.container.Tab

UIAxes5 matlab.ui.control.UIAxes

end

properties (Access = public)

%included all the variables required for the users

Property % Description

%below four variables hold the value of four wavetype

sawtooth = 0;

traingle = 0;

sine = 0;

square = 0;

set = 0;

frequency = 50;%frequency variable for signal generator

amplitude = 5;%frequency variable for signal generator

nofsamples = 1000;%number of samples variable for signal generator

stoptime = 0.25;%for number of time the simulation will be performed

buttonstate = 0;%hold the simulation state

waveformtype = 'SINE' ; %waveform state variable from the drop down

sinetemp=0 ; % holding temporary value of the sine for variations

traingletemp = 0 ; % holding temporary value of the traingle for variations

sawtoothtemp= 0 ; % holding temporary value of the sawtooth for variations

sqauretemp = 0 ; % % holding temporary value of the square for variations

iterationsine=0; % holding iteration value of the sine for variations

iterationtraingle=0; % holding iterations value of the traingle for variations

iterationsquare=0; % holding iterations value of the square for variation

iterationsawtooth=0; % holding iterations value of the sawtooth for variation

buttonstateall = 0; %holding button state for all simulation tab

end

methods (Access = public)

function waveform(app)%function for simulation button

%unless the button is in on state executions wont work

if app.buttonstate == 1

dt = 1/app.nofsamples;%derivate of time

t= (0:dt:app.stoptime-dt)';%time function for the graph

if strcmpi(app.waveformtype, 'SINE') %drop down variable for sine

if(app.iterationsine ~=0)%checking whether the first iteration is performed or not

app.sinetemp = app.sine;% assign the previous sine value to temporary value

end

app.sine = app.amplitude\*(cos(2\*pi\*app.frequency\*t));% calcution for sine values based

%on the formula

%below lines plot the values on all the differnt axis

%required and hold the plot for variation comparisons

plot(app.UIAxes,t,app.sine)

plot(app.UIAxes\_2,t,app.sine)

hold(app.UIAxes\_2,'on')

plot(app.UIAxes\_2,t,app.sinetemp)

app.iterationsine = 1;%this variable will make sure that the next execution onwards

%the temporary vriable will store the previous value

%all the other waveform executions are similar to the sine

%while the difference being the formula and calculation for

%different waveform

elseif strcmpi(app.waveformtype, 'TRAINGLE')

if(app.iterationtraingle ~=0)

app.traingletemp = app.traingle;

end

app.traingle = app.amplitude\*sawtooth(2\*pi\*app.frequency\*t, 1/2);% the different between

%sawtooth and traingle wave is traingle wave is half of

%the sawtooth wave

plot(app.UIAxes3,t,app.traingle)

plot(app.UIAxes3\_3,t,app.traingle)

hold(app.UIAxes3\_3,'on')

plot(app.UIAxes3\_3,t,app.traingletemp)

app.iterationtraingle = 1;

elseif strcmpi(app.waveformtype, 'SQUARE')

if(app.iterationsquare ~=0)

app.sqauretemp = app.square;

end

app.square = app.amplitude\*square(2\*pi\*app.frequency\*t);

plot(app.UIAxes2,t, app.square);

plot(app.UIAxes2\_2,t,app.square)

hold(app.UIAxes2\_2,'on')

plot(app.UIAxes2\_2,t,app.sqauretemp)

app.iterationsquare = 1;

elseif strcmpi(app.waveformtype,'SAWTOOTH')

if(app.iterationsawtooth ~= 0)

app.sawtoothtemp = app.sawtooth;

end

app.sawtooth = app.amplitude\*sawtooth(2\*pi\*app.frequency\*t);

plot(app.UIAxes3\_2,t, app.sawtooth)

plot(app.UIAxes3\_4,t,app.sawtooth)

hold(app.UIAxes3\_4,'on')

plot(app.UIAxes3\_4, t,app.sawtoothtemp)

app.iterationsawtooth = 1;

end%end of drop down comparison algo

else

end%end of button state algo

end%end of wavefunction

%below function is for the simulation all button this works

%similar to the previous function while difference being this

%function plots all the waveform at once.

function waveformall(app)

if app.buttonstateall == 1

dt = 1/app.nofsamples;

t= (0:dt:app.stoptime-dt)';

app.sine = app.amplitude\*(cos(2\*pi\*app.frequency\*t));

app.square = app.amplitude\*square(2\*pi\*app.frequency\*t);

app.traingle = app.amplitude\*sawtooth(2\*pi\*app.frequency\*t, 1/2);

app.sawtooth = app.amplitude\*sawtooth(2\*pi\*app.frequency\*t);

plot(app.UIAxes5,t,app.sine)

hold(app.UIAxes5,'on')

plot(app.UIAxes5,t,app.traingle)

plot(app.UIAxes5,t,app.square)

plot(app.UIAxes5,t,app.sawtooth)

hold(app.UIAxes5,'off')

end

end

end

% Callbacks that handle component events

methods (Access = private)

% Value changed function: WAVEFORMDropDown

function WAVEFORMDropDownValueChanged(app, event)

app.waveformtype = app.WAVEFORMDropDown.Value;

disp(app.waveformtype)

waveform(app)

end

% Value changed function: FrequencyEditField

function FrequencyEditFieldValueChanged(app, event)

value = app.FrequencyEditField.Value;

%error check routine, if user's input is invalid it will

%no value will be initialized to the waveform function

if value <= 0

errordlg('Enter a positive value','Input error');

else

app.frequency = value;

end

waveform(app)

end

% Value changed function: AmplitudeEditField

function AmplitudeEditFieldValueChanged(app, event)

value = app.AmplitudeEditField.Value;

if value <= 0

errordlg('Enter a Positive Value','Input error');

else

app.amplitude = value;

end

waveform(app)

end

% Value changed function: NumberofsamplesSlider

function NumberofsamplesSliderValueChanged(app, event)

app.nofsamples = app.NumberofsamplesSlider.Value;

waveform(app)

end

% Value changed function: StopTimesecondsEditField

function StopTimesecondsEditFieldValueChanged(app, event)

value = app.StopTimesecondsEditField.Value;

if value <= 0

errordlg('Enter a Positive Value','Input error');

else

app.stoptime = value;

end

app.StopTimesecondsEditField.Visible = 'off'

app.StopTimesecondsEditFieldLabel.Visible ='off'

waveform(app)

end

% Value changed function: SimulateButton

function SimulateButtonValueChanged(app, event)

app.buttonstate = app.SimulateButton.Value;

waveform(app)

end

% Value changed function: SimulateAllButton

function SimulateAllButtonValueChanged(app, event)

app.buttonstateall = app.SimulateAllButton.Value;

waveformall(app)

end

end

% Component initialization

methods (Access = private)

% Create UIFigure and components

function createComponents(app)

% Create UIFigure and hide until all components are created

app.UIFigure = uifigure('Visible', 'off');

app.UIFigure.Position = [100 100 864 623];

app.UIFigure.Name = 'MATLAB App';

% Create TabGroup

app.TabGroup = uitabgroup(app.UIFigure);

app.TabGroup.Position = [1 0 864 624];

% Create INPUTTab

app.INPUTTab = uitab(app.TabGroup);

app.INPUTTab.Title = 'INPUT';

app.INPUTTab.BackgroundColor = [0.702 0.4314 0.3098];

% Create WAVEFORMDropDownLabel

app.WAVEFORMDropDownLabel = uilabel(app.INPUTTab);

app.WAVEFORMDropDownLabel.BackgroundColor = [0 0.4471 0.7412];

app.WAVEFORMDropDownLabel.HorizontalAlignment = 'right';

app.WAVEFORMDropDownLabel.Position = [641 523 75 22];

app.WAVEFORMDropDownLabel.Text = 'WAVEFORM';

% Create WAVEFORMDropDown

app.WAVEFORMDropDown = uidropdown(app.INPUTTab);

app.WAVEFORMDropDown.Items = {'SINE', 'SQUARE', 'TRAINGLE', 'SAWTOOTH'};

app.WAVEFORMDropDown.ValueChangedFcn = createCallbackFcn(app, @WAVEFORMDropDownValueChanged, true);

app.WAVEFORMDropDown.BackgroundColor = [0 0.4471 0.7412];

app.WAVEFORMDropDown.Position = [731 523 100 22];

app.WAVEFORMDropDown.Value = 'SINE';

% Create FrequencyEditFieldLabel

app.FrequencyEditFieldLabel = uilabel(app.INPUTTab);

app.FrequencyEditFieldLabel.BackgroundColor = [0 0.4471 0.7412];

app.FrequencyEditFieldLabel.HorizontalAlignment = 'right';

app.FrequencyEditFieldLabel.FontWeight = 'bold';

app.FrequencyEditFieldLabel.Position = [647 447 66 22];

app.FrequencyEditFieldLabel.Text = 'Frequency';

% Create FrequencyEditField

app.FrequencyEditField = uieditfield(app.INPUTTab, 'numeric');

app.FrequencyEditField.ValueChangedFcn = createCallbackFcn(app, @FrequencyEditFieldValueChanged, true);

app.FrequencyEditField.FontWeight = 'bold';

app.FrequencyEditField.BackgroundColor = [0 0.4471 0.7412];

app.FrequencyEditField.Position = [728 447 97 22];

% Create AmplitudeEditFieldLabel

app.AmplitudeEditFieldLabel = uilabel(app.INPUTTab);

app.AmplitudeEditFieldLabel.BackgroundColor = [0 0.4471 0.7412];

app.AmplitudeEditFieldLabel.HorizontalAlignment = 'right';

app.AmplitudeEditFieldLabel.FontWeight = 'bold';

app.AmplitudeEditFieldLabel.Position = [647 383 64 22];

app.AmplitudeEditFieldLabel.Text = 'Amplitude';

% Create AmplitudeEditField

app.AmplitudeEditField = uieditfield(app.INPUTTab, 'numeric');

app.AmplitudeEditField.ValueChangedFcn = createCallbackFcn(app, @AmplitudeEditFieldValueChanged, true);

app.AmplitudeEditField.FontWeight = 'bold';

app.AmplitudeEditField.BackgroundColor = [0 0.4471 0.7412];

app.AmplitudeEditField.Position = [726 383 100 22];

% Create NumberofsamplesSliderLabel

app.NumberofsamplesSliderLabel = uilabel(app.INPUTTab);

app.NumberofsamplesSliderLabel.HorizontalAlignment = 'right';

app.NumberofsamplesSliderLabel.FontWeight = 'bold';

app.NumberofsamplesSliderLabel.Position = [513 322 117 22];

app.NumberofsamplesSliderLabel.Text = 'Number of samples';

% Create NumberofsamplesSlider

app.NumberofsamplesSlider = uislider(app.INPUTTab);

app.NumberofsamplesSlider.Limits = [1000 10000];

app.NumberofsamplesSlider.ValueChangedFcn = createCallbackFcn(app, @NumberofsamplesSliderValueChanged, true);

app.NumberofsamplesSlider.Position = [530 310 279 3];

app.NumberofsamplesSlider.Value = 1000;

% Create StopTimesecondsEditFieldLabel

app.StopTimesecondsEditFieldLabel = uilabel(app.INPUTTab);

app.StopTimesecondsEditFieldLabel.BackgroundColor = [0 0.4471 0.7412];

app.StopTimesecondsEditFieldLabel.HorizontalAlignment = 'right';

app.StopTimesecondsEditFieldLabel.FontWeight = 'bold';

app.StopTimesecondsEditFieldLabel.Position = [705 224 120 22];

app.StopTimesecondsEditFieldLabel.Text = 'Stop Time(seconds)';

% Create StopTimesecondsEditField

app.StopTimesecondsEditField = uieditfield(app.INPUTTab, 'numeric');

app.StopTimesecondsEditField.ValueChangedFcn = createCallbackFcn(app, @StopTimesecondsEditFieldValueChanged, true);

app.StopTimesecondsEditField.BackgroundColor = [0 0.4471 0.7412];

app.StopTimesecondsEditField.Position = [699 203 105 22];

% Create SimulateButton

app.SimulateButton = uibutton(app.INPUTTab, 'state');

app.SimulateButton.ValueChangedFcn = createCallbackFcn(app, @SimulateButtonValueChanged, true);

app.SimulateButton.Text = 'Simulate';

app.SimulateButton.BackgroundColor = [1 1 0];

app.SimulateButton.FontWeight = 'bold';

app.SimulateButton.FontColor = [0 0 1];

app.SimulateButton.Position = [366 20 129 50];

% Create SIGNALGENERATORLabel

app.SIGNALGENERATORLabel = uilabel(app.INPUTTab);

app.SIGNALGENERATORLabel.BackgroundColor = [1 0.4118 0.1608];

app.SIGNALGENERATORLabel.FontSize = 20;

app.SIGNALGENERATORLabel.FontWeight = 'bold';

app.SIGNALGENERATORLabel.Position = [347 557 213 32];

app.SIGNALGENERATORLabel.Text = 'SIGNAL GENERATOR';

% Create Image

app.Image = uiimage(app.INPUTTab);

app.Image.Position = [19 354 209 170];

app.Image.ImageSource = 'SINECIRCUIT.jpg';

% Create Image\_2

app.Image\_2 = uiimage(app.INPUTTab);

app.Image\_2.Position = [19 111 209 170];

app.Image\_2.ImageSource = 'SQUARE CIRCUIT.png';

% Create Image\_3

app.Image\_3 = uiimage(app.INPUTTab);

app.Image\_3.Position = [251 343 209 202];

app.Image\_3.ImageSource = 'SQUARE\_CIRCUIT.jpg';

% Create Image\_4

app.Image\_4 = uiimage(app.INPUTTab);

app.Image\_4.Position = [251 111 209 170];

app.Image\_4.ImageSource = 'SAWTOOTHCIRCUIT.jpg';

% Create SawtoothCircuitLabel

app.SawtoothCircuitLabel = uilabel(app.INPUTTab);

app.SawtoothCircuitLabel.Position = [327 289 93 22];

app.SawtoothCircuitLabel.Text = 'Sawtooth Circuit';

% Create TraingleCircuitLabel

app.TraingleCircuitLabel = uilabel(app.INPUTTab);

app.TraingleCircuitLabel.Position = [107 300 86 22];

app.TraingleCircuitLabel.Text = 'Traingle Circuit';

% Create SquareCircuitLabel

app.SquareCircuitLabel = uilabel(app.INPUTTab);

app.SquareCircuitLabel.Position = [314 523 82 22];

app.SquareCircuitLabel.Text = 'Square Circuit';

% Create SineCircuitLabel

app.SineCircuitLabel = uilabel(app.INPUTTab);

app.SineCircuitLabel.Position = [104 535 67 22];

app.SineCircuitLabel.Text = 'Sine Circuit';

% Create SimulateAllButton

app.SimulateAllButton = uibutton(app.INPUTTab, 'state');

app.SimulateAllButton.ValueChangedFcn = createCallbackFcn(app, @SimulateAllButtonValueChanged, true);

app.SimulateAllButton.Text = 'Simulate All';

app.SimulateAllButton.BackgroundColor = [0 1 0];

app.SimulateAllButton.FontWeight = 'bold';

app.SimulateAllButton.Position = [542 20 128 50];

% Create SINETab

app.SINETab = uitab(app.TabGroup);

app.SINETab.Title = 'SINE';

app.SINETab.BackgroundColor = [0.702 0.4314 0.3098];

% Create UIAxes

app.UIAxes = uiaxes(app.SINETab);

title(app.UIAxes, 'SINE')

xlabel(app.UIAxes, 'X')

ylabel(app.UIAxes, 'Y')

app.UIAxes.GridColor = [0.6353 0.0784 0.1843];

app.UIAxes.BackgroundColor = [0.0745 0.6235 1];

app.UIAxes.Position = [0 253 862 292];

% Create UIAxes\_2

app.UIAxes\_2 = uiaxes(app.SINETab);

title(app.UIAxes\_2, 'SINE VARIATIONS')

xlabel(app.UIAxes\_2, 'X')

ylabel(app.UIAxes\_2, 'Y')

app.UIAxes\_2.BackgroundColor = [0.0745 0.6235 1];

app.UIAxes\_2.Position = [0 1 862 253];

% Create SQUARETab

app.SQUARETab = uitab(app.TabGroup);

app.SQUARETab.Title = 'SQUARE';

app.SQUARETab.BackgroundColor = [0.702 0.4314 0.3098];

% Create UIAxes2

app.UIAxes2 = uiaxes(app.SQUARETab);

title(app.UIAxes2, 'SQUARE')

xlabel(app.UIAxes2, 'X')

ylabel(app.UIAxes2, 'Y')

app.UIAxes2.BackgroundColor = [0.4667 0.6745 0.1882];

app.UIAxes2.Position = [0 253 864 283];

% Create UIAxes2\_2

app.UIAxes2\_2 = uiaxes(app.SQUARETab);

title(app.UIAxes2\_2, 'SQUARE VARIATIONS')

xlabel(app.UIAxes2\_2, 'X')

ylabel(app.UIAxes2\_2, 'Y')

app.UIAxes2\_2.BackgroundColor = [0.4667 0.6745 0.1882];

app.UIAxes2\_2.Position = [1 0 861 254];

% Create TRAINGLETab

app.TRAINGLETab = uitab(app.TabGroup);

app.TRAINGLETab.Title = 'TRAINGLE';

app.TRAINGLETab.BackgroundColor = [0.702 0.4314 0.3098];

% Create UIAxes3

app.UIAxes3 = uiaxes(app.TRAINGLETab);

title(app.UIAxes3, 'TRAINGLE')

xlabel(app.UIAxes3, 'X')

ylabel(app.UIAxes3, 'Y')

app.UIAxes3.BackgroundColor = [0.9294 0.6941 0.1255];

app.UIAxes3.Position = [8 253 854 283];

% Create UIAxes3\_3

app.UIAxes3\_3 = uiaxes(app.TRAINGLETab);

title(app.UIAxes3\_3, 'TRAINGLE VARIATIONS')

xlabel(app.UIAxes3\_3, 'X')

ylabel(app.UIAxes3\_3, 'Y')

app.UIAxes3\_3.BackgroundColor = [0.9294 0.6941 0.1255];

app.UIAxes3\_3.Position = [8 1 854 253];

% Create SAWTOOTHTab

app.SAWTOOTHTab = uitab(app.TabGroup);

app.SAWTOOTHTab.Title = 'SAWTOOTH';

app.SAWTOOTHTab.BackgroundColor = [0.702 0.4314 0.3098];

% Create UIAxes3\_2

app.UIAxes3\_2 = uiaxes(app.SAWTOOTHTab);

title(app.UIAxes3\_2, 'SAWTOOTH')

xlabel(app.UIAxes3\_2, 'X')

ylabel(app.UIAxes3\_2, 'Y')

app.UIAxes3\_2.BackgroundColor = [1 0 1];

app.UIAxes3\_2.Position = [5 270 854 275];

% Create UIAxes3\_4

app.UIAxes3\_4 = uiaxes(app.SAWTOOTHTab);

title(app.UIAxes3\_4, 'SAWTOOTH VARIATIONS')

xlabel(app.UIAxes3\_4, 'X')

ylabel(app.UIAxes3\_4, 'Y')

app.UIAxes3\_4.BackgroundColor = [1 0 1];

app.UIAxes3\_4.Position = [8 1 854 262];

% Create ALLTab

app.ALLTab = uitab(app.TabGroup);

app.ALLTab.Title = 'ALL';

% Create UIAxes5

app.UIAxes5 = uiaxes(app.ALLTab);

title(app.UIAxes5, 'ALL')

xlabel(app.UIAxes5, 'X')

ylabel(app.UIAxes5, 'Y')

app.UIAxes5.Position = [0 1 859 598];

% Show the figure after all components are created

app.UIFigure.Visible = 'on';

end

end

% App creation and deletion

methods (Access = public)

% Construct app

function app = signalgeneratorv2

% Create UIFigure and components

createComponents(app)

% Register the app with App Designer

registerApp(app, app.UIFigure)

if nargout == 0

clear app

end

end

% Code that executes before app deletion

function delete(app)

% Delete UIFigure when app is deleted

delete(app.UIFigure)

end

end

end