

# DEMONSTRATING LINE CODING & EYE DIAGRAM USING MATLAB GUI

- Tanmay Kapil (PC47)
- Jay Srivastava (PC46)
- Atul Kumar (PC44)



# INDEX

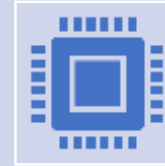
- **BASICS OF AN EYE DIAGRAM**
- **INTERPRETING AN EYE DIAGRAM**
- **LINE CODING**
- **WHAT IS JITTER?**
- **JITTER IN DETAIL**
- **IMPORTANCE OF AN EYE DIAGRAM**
- **OBTAINING THE EYE DIAGRAM**
- **PROJECT CONTENT**
- **REFERENCES**



# BASICS OF AN EYE DIAGRAM



An eye diagram is a common indicator of the quality of signals in high-speed digital transmissions.



An eye diagram is used in electronic engineering to get a good idea of signal quality in the digital domain.



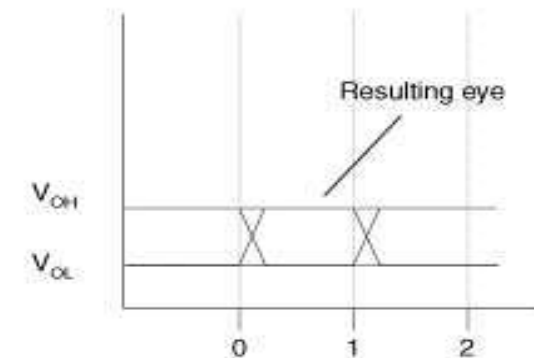
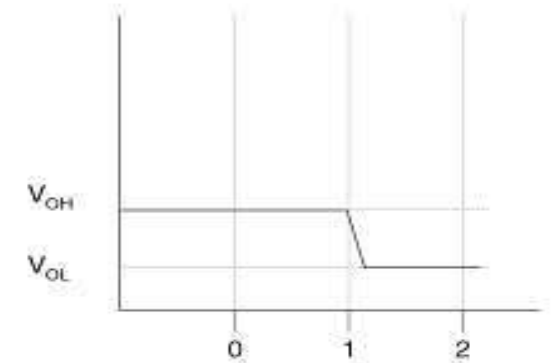
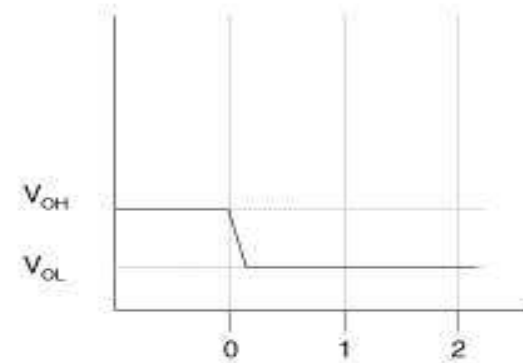
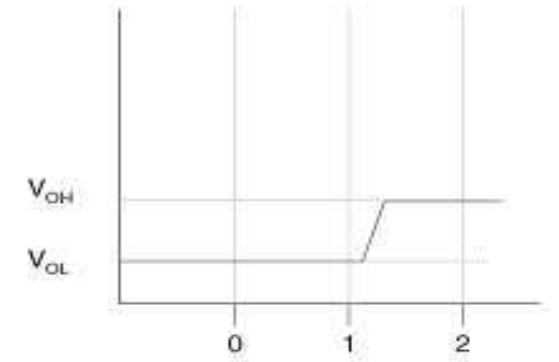
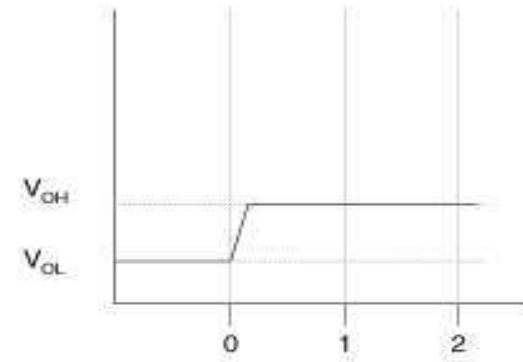
The eye diagram takes its name from the fact that it has the appearance of a human eye.



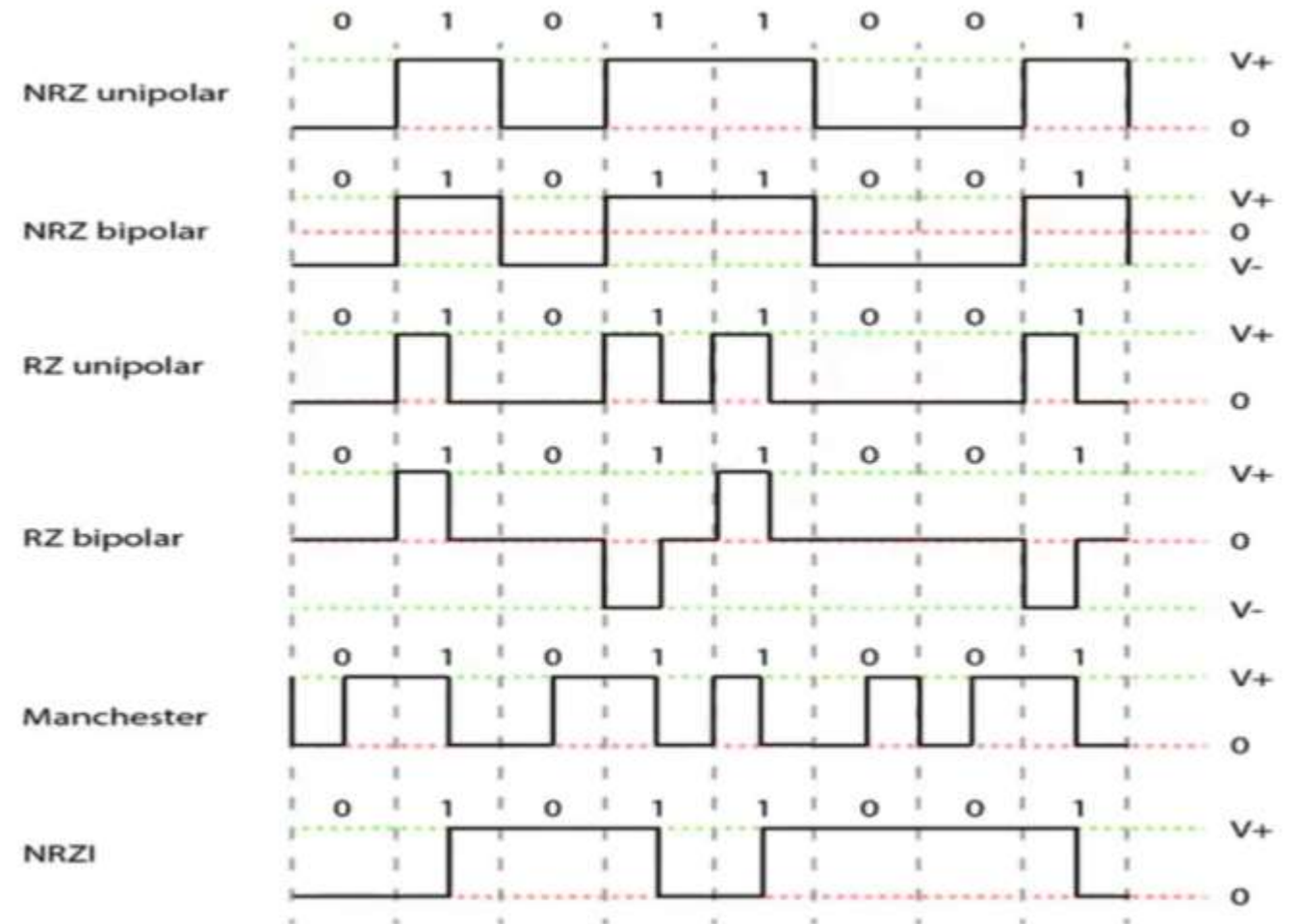
The eye diagram is used primarily to look at digital signals for the purpose of recognizing the effects of distortion and finding its source

# INTERPRETING AN EYE DIAGRAM

- A properly constructed eye should contain every possible bit sequence from simple alternate 1's and 0's to isolated 1's after long runs of 0's, and all other patterns that may show up weaknesses in the design.
- In the figure shown, the bit sequences 011, 001, 100, and 110 are superimposed over one another to obtain the final eye diagram.



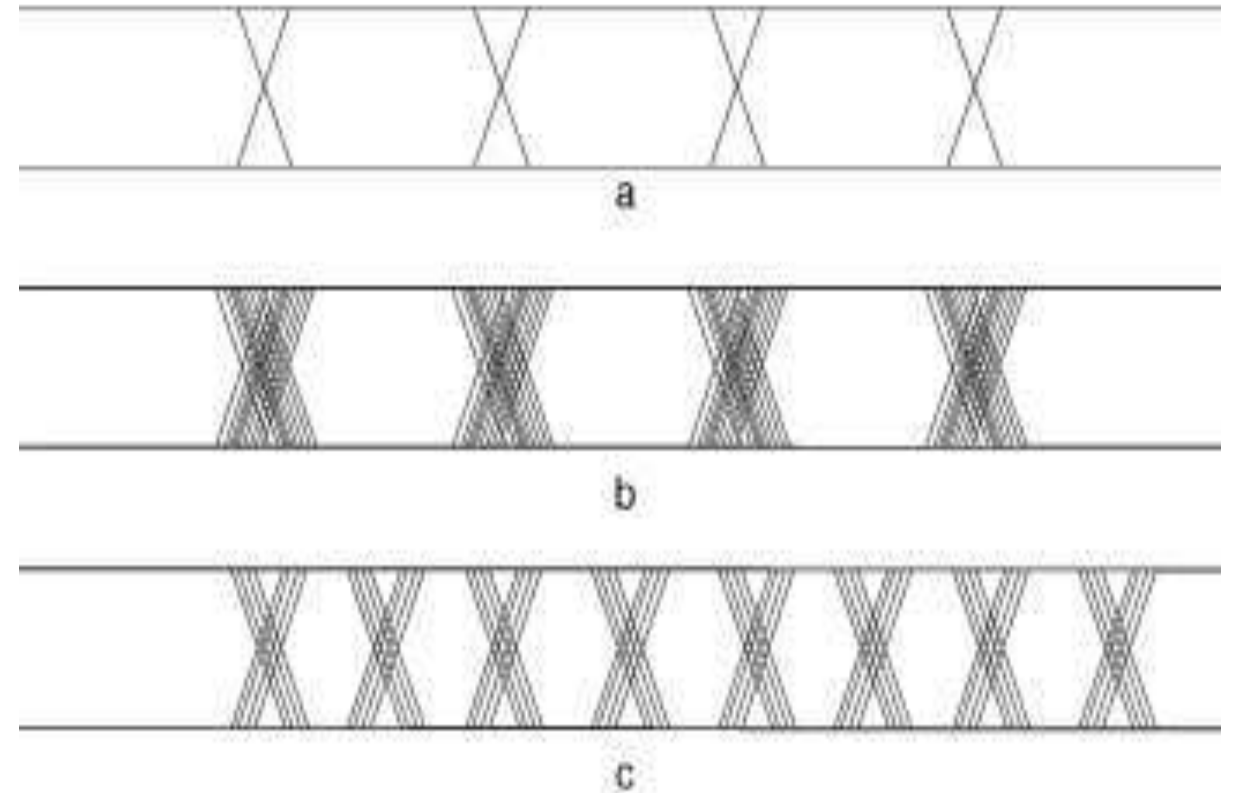
# LINE CODING





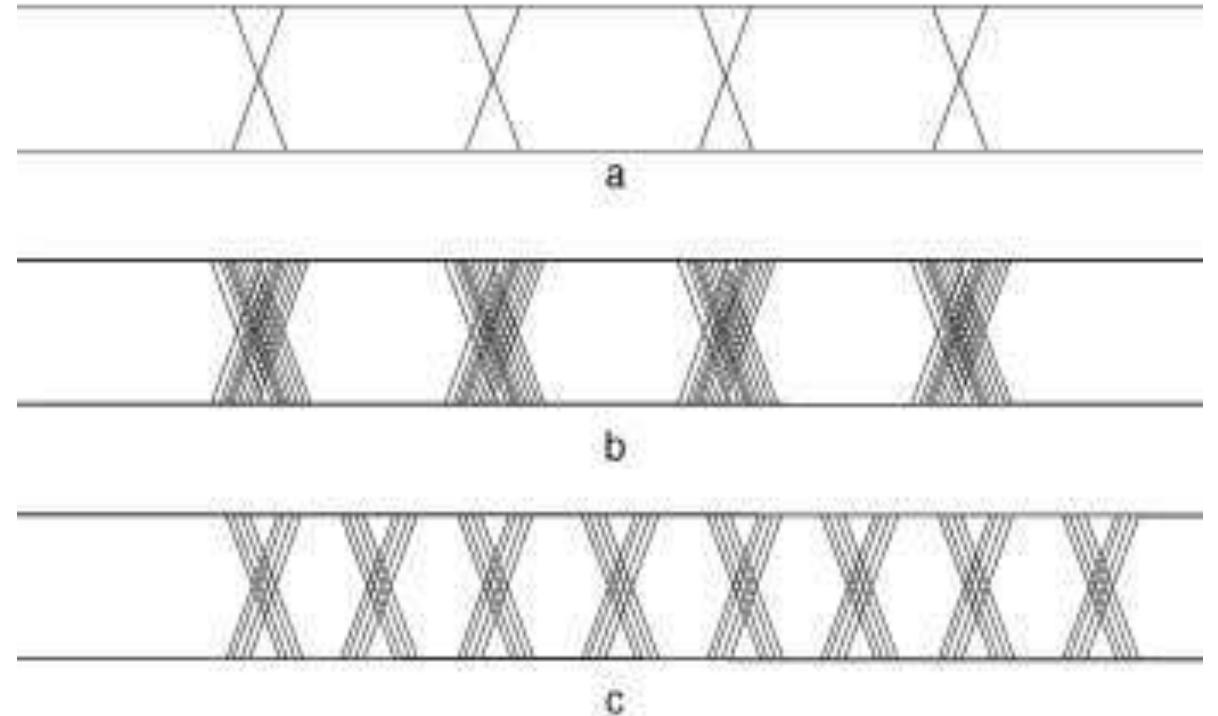
# WHAT IS JITTER?

- Although in theory eye diagrams should look like rectangular boxes, the finite rise and fall times of signals and oscilloscopes cause eye diagrams to actually look more like the image in **Figure (a)**.
- When high-speed digital signals are transmitted, the impairments introduced at various stages lead to timing errors. One such timing error is “jitter,” which results from the misalignment of rise and fall times (**Figure b**).



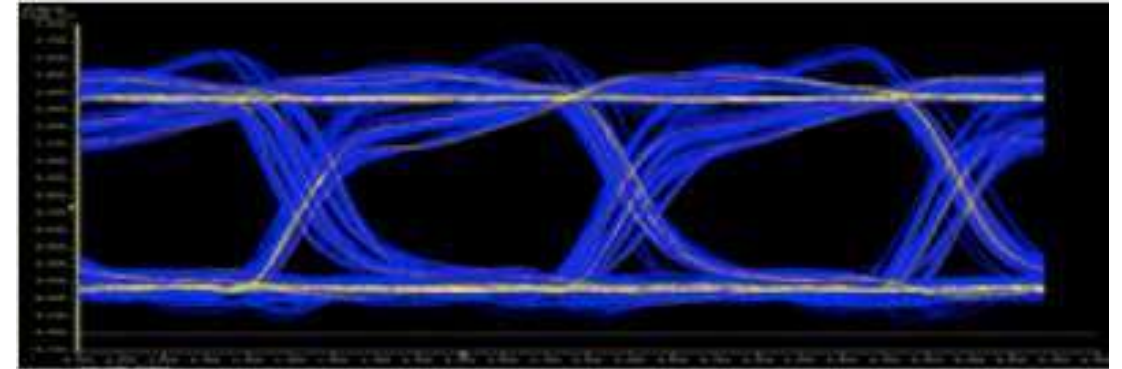
# JITTER IN DETAIL

- Jitter occurs when a rising or falling edges occur at times that differ from the ideal time. Some edges occur early, some occur late.
- In **Figure (c)**, the absolute timing error or jitter margin is less than that in **Figure (b)**, but the eye opening in **Figure (c)** is smaller because of the higher bit rate.
- With the increase in bit rate, the absolute time error represents an increasing portion of the cycle, thus reducing the size of the eye opening. This may increase the potential for data errors.

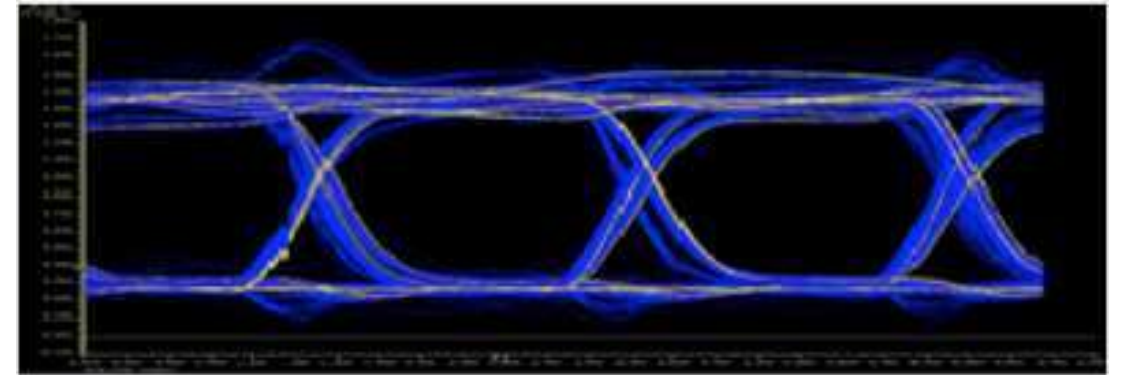


# JITTER IN DETAIL

- The effect of termination is clearly visible in the eye diagrams generated. With improper termination, the eye looks constrained or stressed (**Figure (a)**), and with improved termination schemes, the eye becomes more relaxed (**Figure (b)**).
- A poorly terminated signal line suffers from multiple reflections. The reflected waves are of significant amplitude, which may severely constrict the eye.
- Typically, this is the worst-case operating condition for the receiver, and if the receiver can operate error-free in the presence of such interference, then it meets specifications.



(a)



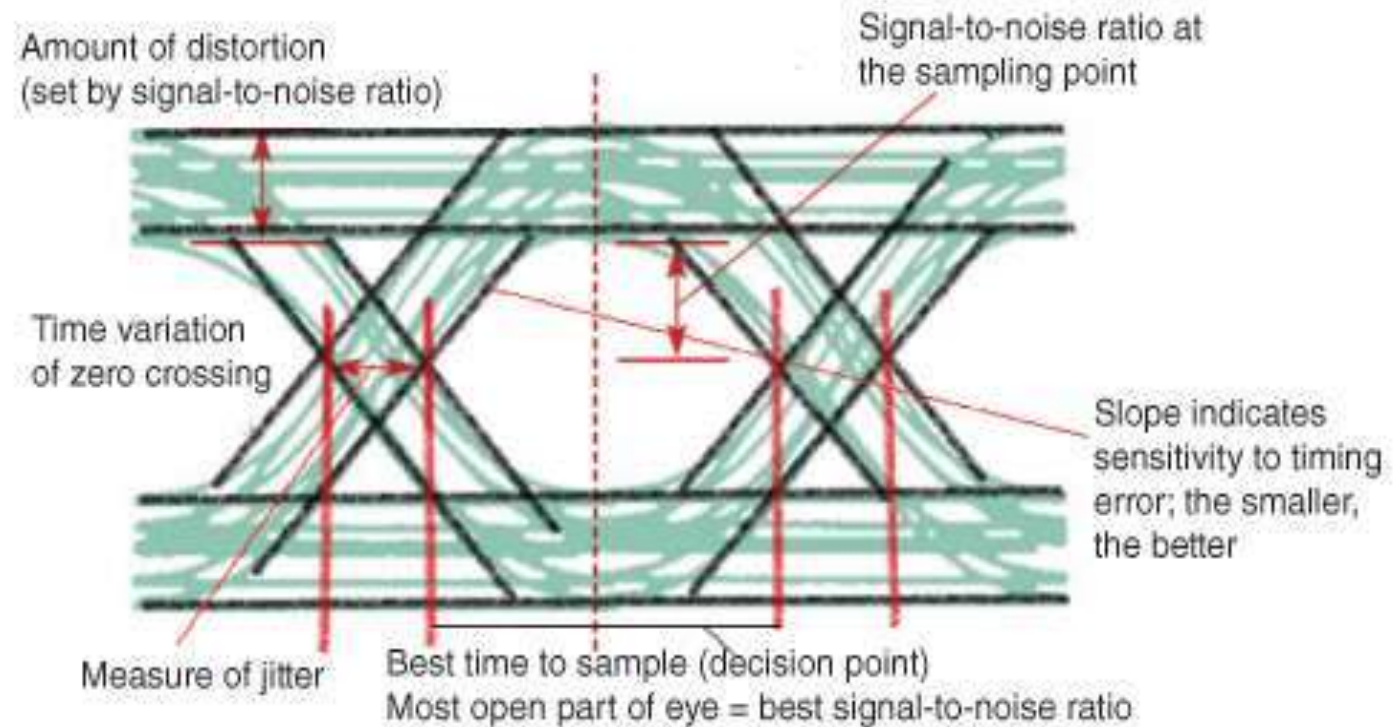
(b)



# IMPORTANCE OF AN EYE DIAGRAM

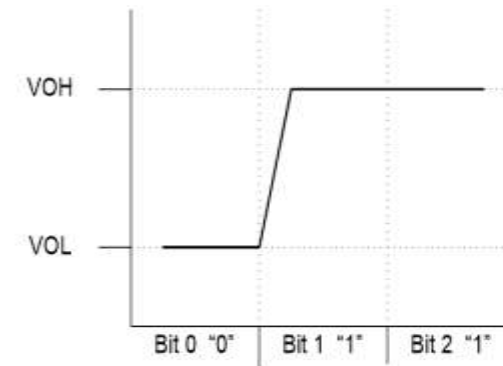
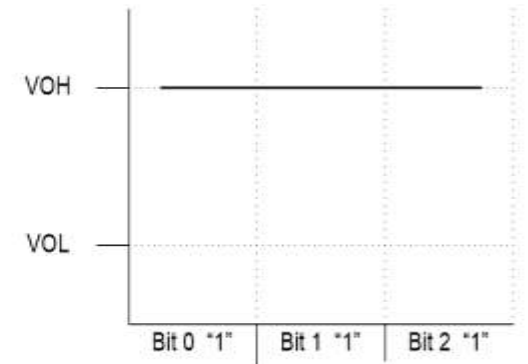
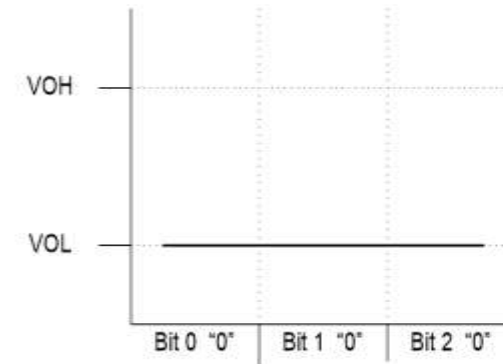
An eye pattern provides the following information about a particular system.

- Actual eye patterns are used to estimate the bit error rate and the signal-to-noise ratio.
- The width of the eye opening defines the time interval over which the received wave can be sampled without error.
- The instant of time when the eye opening is wide, will be the preferred time for sampling.
- The rate of the closure of the eye, according to the sampling time, determines how sensitive the system is to the timing error.
- The height of the eye opening, at a specified sampling time, defines the margin over noise



# OBTAINING THE EYE DIAGRAM

- A Sampling Oscilloscope is used to create an Eye Diagram.
- It is used for displaying and overlaying the results of a continuous sampling of the signal.
- This is done at a resolution of a single Unit Interval (UI) of a one data bit to allow for all combinations of low-to-high and high-to-low transitions to occur.





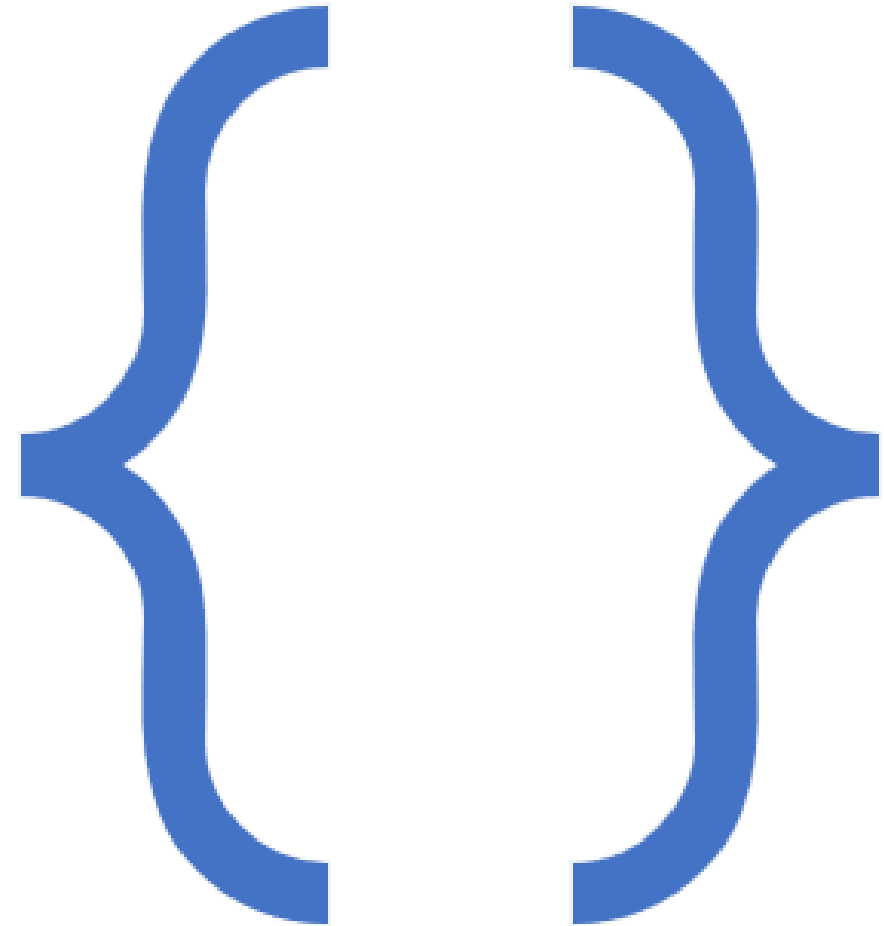
# PROJECT CONTENT

**Demonstrating Line  
Coding and Eye  
Diagram using  
MATLAB GUI**

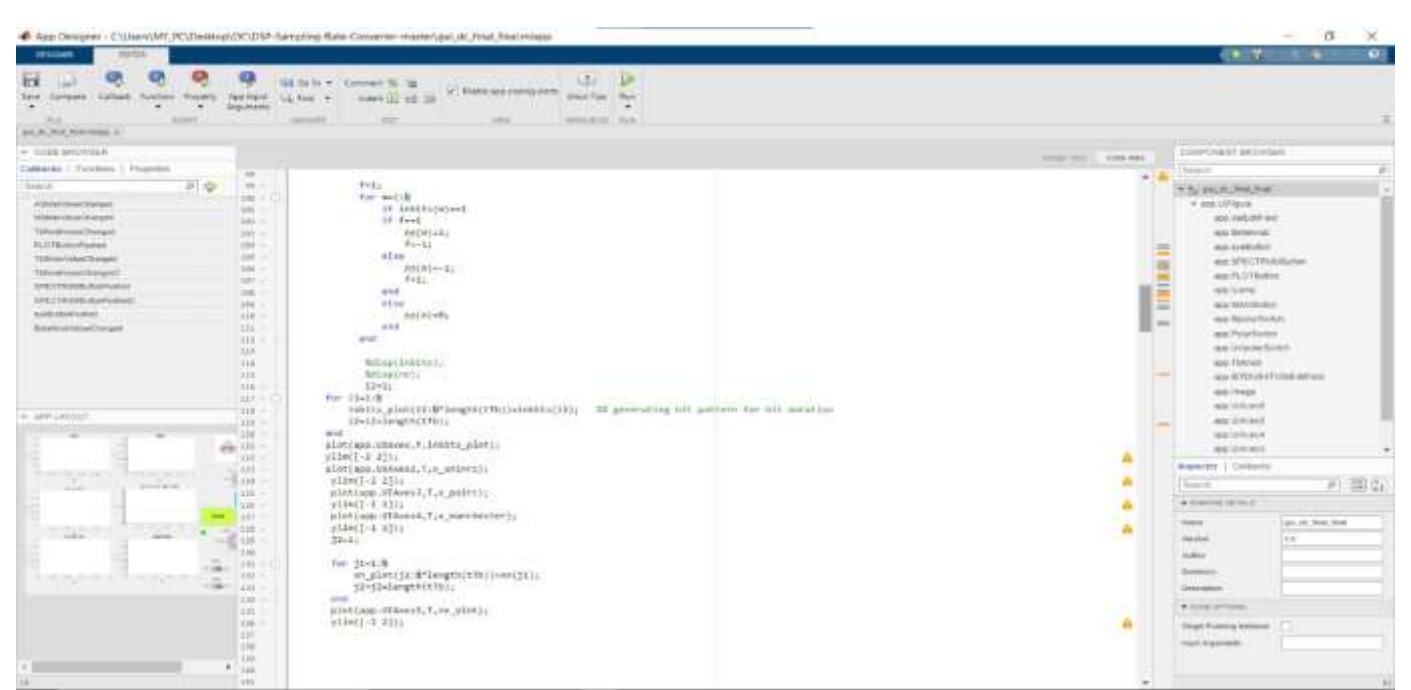




# MATLAB CODE & OUTPUT



# CODE FOR PLOT BUTTON

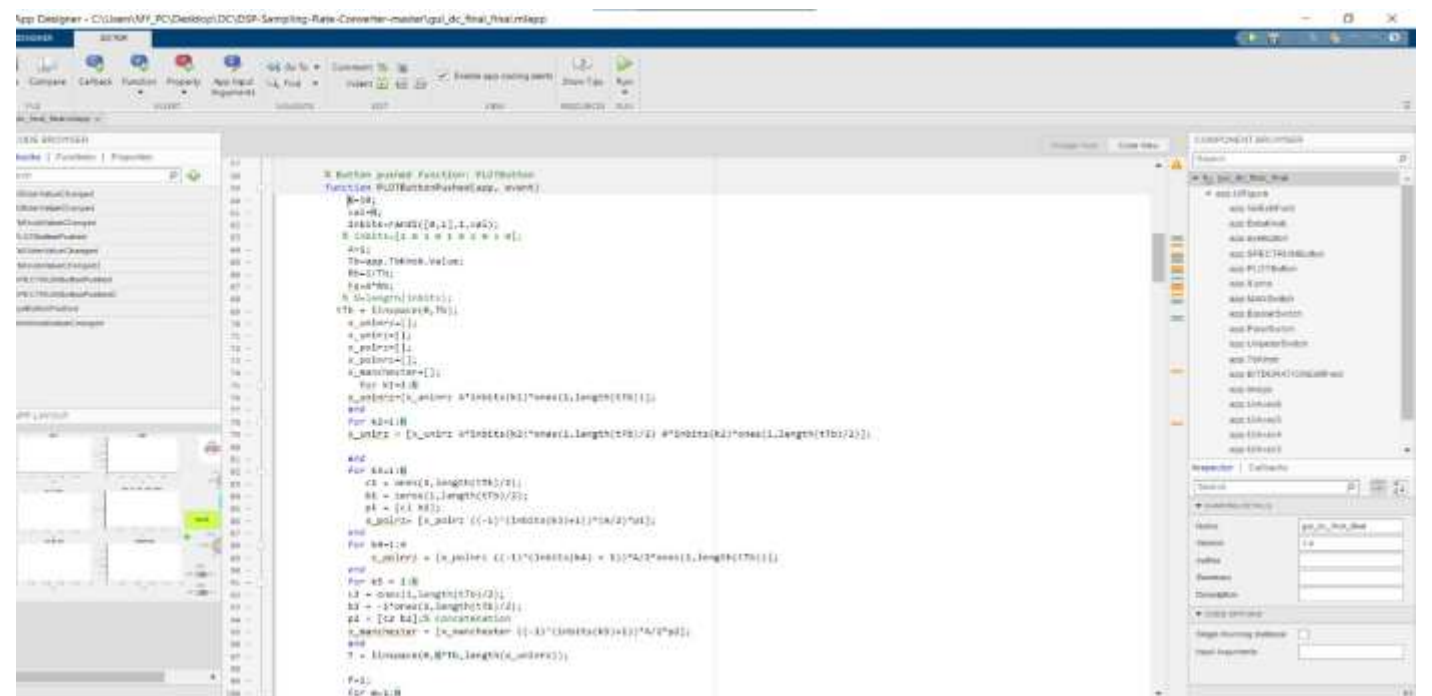


```

function plotButtonPushed(app, event)
    % Generate the signal x[n]
    N = 100; % Number of samples
    f = 0.1; % Frequency in Hz
    Ts = 0.01; % Sampling period in seconds
    t = 0:Ts:(N*Ts-1); % Time vector
    x = cos(2*pi*f*t); % Signal x[n]

    % Compute the magnitude spectrum
    X = fft(x, N); % FFT of x[n]
    X_mag = abs(X)/length(X); % Magnitude spectrum
    X_freq = linspace(0, 1/(2*Ts), length(X_mag)); % Frequency vector

    % Plot the signal and its magnitude spectrum
    figure; % Create a new figure
    plot(X_freq, X_mag, 'b'); % Plot the magnitude spectrum
    hold on; % Hold the plot
    plot(t, x, 'r'); % Plot the signal x[n]
    title('Signal x[n] and its magnitude spectrum'); % Set the title
    xlabel('Frequency (Hz)'); % Set the x-axis label
    ylabel('Magnitude'); % Set the y-axis label
    legend('Signal x[n]', 'Magnitude Spectrum'); % Add a legend
    grid on; % Turn on the grid
  
```



```

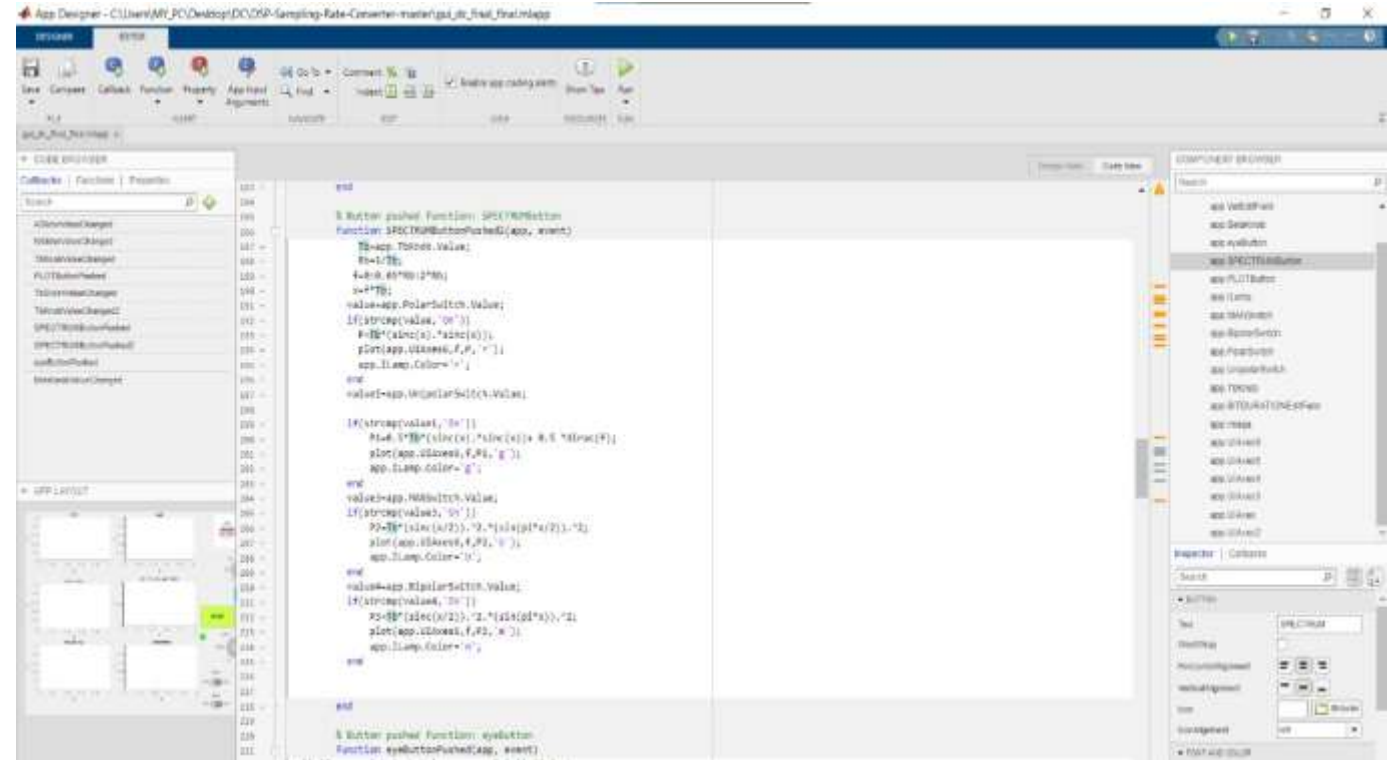
function plotButtonPushed(app, event)
    % Generate the signal x[n]
    N = 100; % Number of samples
    f = 0.1; % Frequency in Hz
    Ts = 0.01; % Sampling period in seconds
    t = 0:Ts:(N*Ts-1); % Time vector
    x = cos(2*pi*f*t); % Signal x[n]

    % Compute the magnitude spectrum
    X = fft(x, N); % FFT of x[n]
    X_mag = abs(X)/length(X); % Magnitude spectrum
    X_freq = linspace(0, 1/(2*Ts), length(X_mag)); % Frequency vector

    % Plot the signal and its magnitude spectrum
    figure; % Create a new figure
    plot(X_freq, X_mag, 'b'); % Plot the magnitude spectrum
    hold on; % Hold the plot
    plot(t, x, 'r'); % Plot the signal x[n]
    title('Signal x[n] and its magnitude spectrum'); % Set the title
    xlabel('Frequency (Hz)'); % Set the x-axis label
    ylabel('Magnitude'); % Set the y-axis label
    legend('Signal x[n]', 'Magnitude Spectrum'); % Add a legend
    grid on; % Turn on the grid
  
```



# CODE FOR SPECTRUM



The image shows a MATLAB App Designer window titled "App Designer - C:\Users\MI\_PC\Desktop\DC\DCP-Sampling-Rate-Converter-master\gui\_rtr\_final\_final.mapp". The interface includes a toolbar with icons for Save, Compiler, Callback, Function, Property, App Input Arguments, Find, Index, and a link to the live app. The main workspace is divided into three panes: Component Browser, Inspector, and Code Editor.

**Component Browser:** Lists various UI components such as app.VariablePlaceholder, app.Decimal, app.Evaluate, app.SPECTRUMButton, app.PLOTButton, app.Limits, app.RMWButton, app.RotateButton, app.PolarButton, app.UnpolarButton, app.TONES, app.BTDRatioCheckBox, app.Toggle, app.UIHandle, app.UIAxis, app.UIAxis2, app.UIAxis3, app.UIAxis4, app.UIAxis5, and app.UIAxis6.

**Inspector:** Shows the properties of the selected component, "SPECTRUMButton".

**Code Editor:** Contains the following MATLAB code:

```
end

% Button pushed function: SPECTRUMButton
function SPECTRUMButtonPushed(app, event)

    % Update the value of the slider
    f0 = app.Tones.Value;
    f0 = 1/f0;
    f0 = f0 * 1000;
    w = 2*pi*f0;
    value = app.PolarSwitch.Value;
    if strcmp(value, 'On')
        P = 10*(abs(x)/2);
        plot(app.UIAxes, f/P, 'r');
        app.UIAxes.Color = 'r';
    end
    value = app.UnpolarSwitch.Value;
    if strcmp(value, 'On')
        P = 10*(abs(x)/2);
        plot(app.UIAxes, f/P, 'g');
        app.UIAxes.Color = 'g';
    end
    value = app.RMWButton.Value;
    if strcmp(value, 'On')
        P2 = 10*(abs(x/2)/2);
        plot(app.UIAxes, f/P2, 'b');
        app.UIAxes.Color = 'b';
    end
    value = app.HilbertSwitch.Value;
    if strcmp(value, 'On')
        P3 = 10*(abs(x/2)/2);
        plot(app.UIAxes, f/P3, 'm');
        app.UIAxes.Color = 'm';
    end
end

% Button pushed function: evaluate
function evaluatePushed(app, event)
```

# CODE FOR EYE DIAGRAM

App Designer - C:\Users\MY\_PC\Desktop\DC\_DSP-Sampling-Rate-Converter-master\gui\_dc\_final\_final.mapp

DESIGNER EDITOR

Save Compare Callback Function Property App Input Arguments Find Indent Go To Comment Enable app coding alerts Show Tips Run

FILE INSERT NAVIGATE EDIT VIEW RESOURCES RUN

gui\_dc\_final\_final.mapp

CODE BROWSER

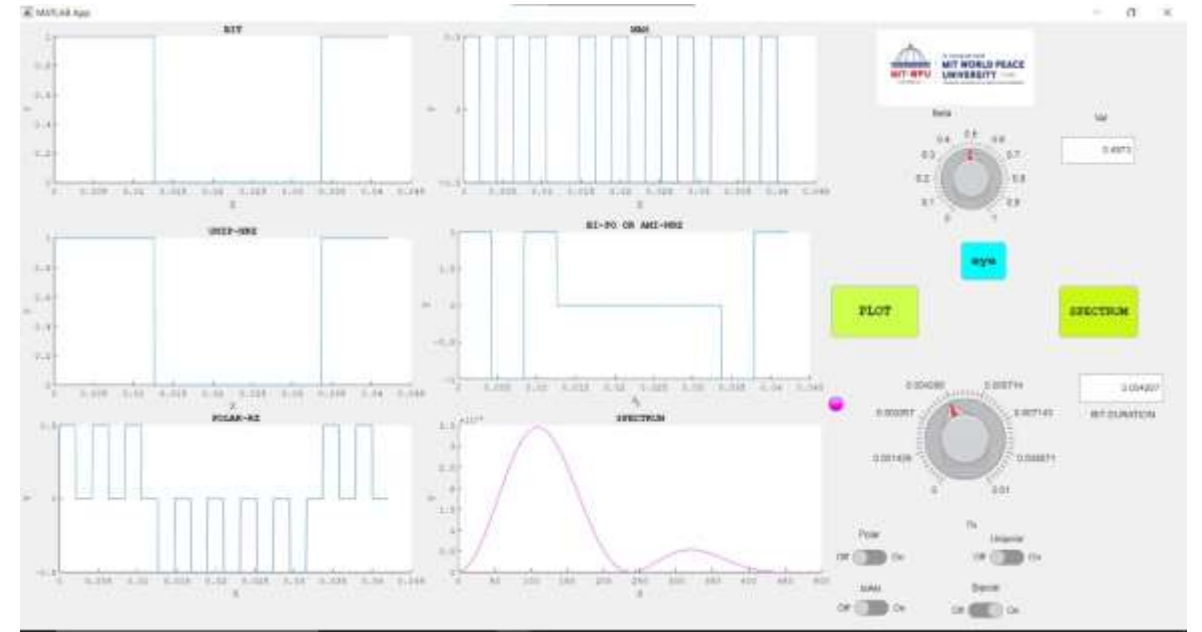
Callbacks Functions Properties

Search

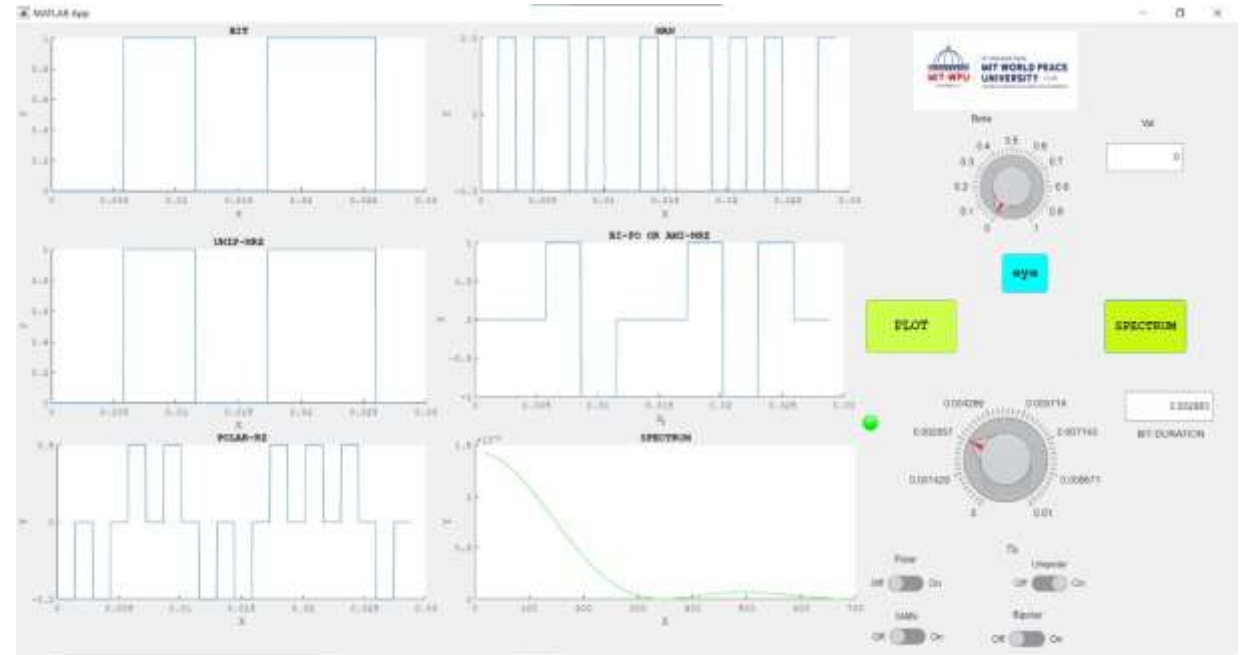
ASliderValueChanged  
NSliderValueChanged  
TkKindValueChanged  
PLOTButtonPushed  
TkSliderValueChanged  
TkKindValueChanged2  
SPECTRUMButtonPushed  
SPECTRUMButtonPushed2  
eyeButtonPushed  
BetaKnobValueChanged

APP LAYOUT

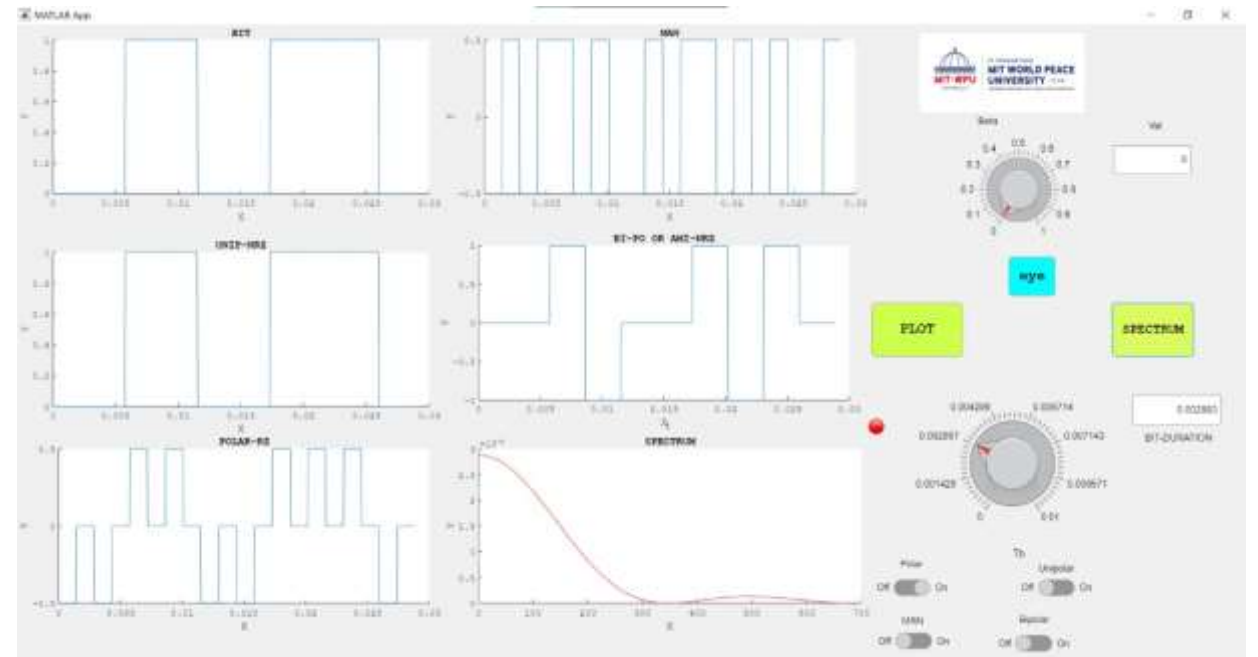
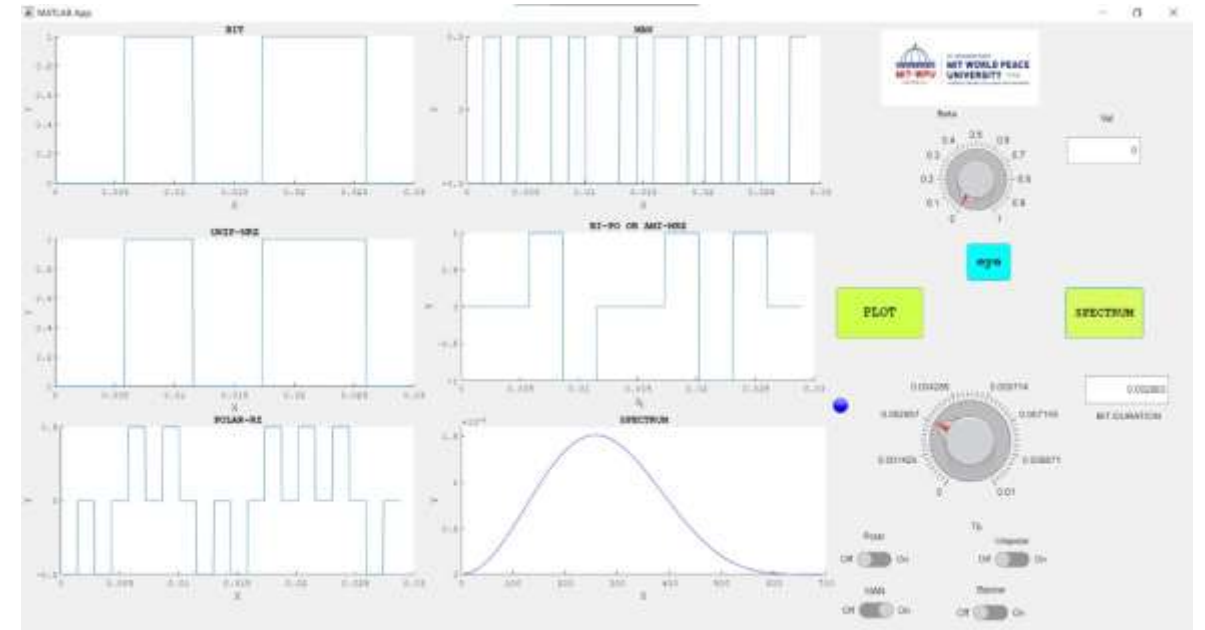
```
116  
117  
118 end  
119  
120 % Button pushed function: eyeButton  
121 function eyeButtonPushed(app, event)  
122 rolloff = app.BetaKnob.Value; % Rolloff factor  
123 span = 6; % Filter span in symbols  
124 sps = 10;  
125 b = rcosdesign(rolloff, span, sps);  
126 bnrz=ones(1,sps);  
127 brz=[ones(1,sps/2) zeros(1,sps/2)];  
128 len=250;  
129 d = awgn( 5*randi([0 1], len, 1) - 5,10); % 10 dB is the AWGN SNR, 0 being poor, 20 being good SNR  
130 subplot(2,1,1)  
131 stem(d)  
132 x = upfirdn(d, b, sps); % replace bnrz with b, brz for raised cosine, bipolar return zero etc  
133  
134 eyediagram(x,sps);  
135 end  
136
```



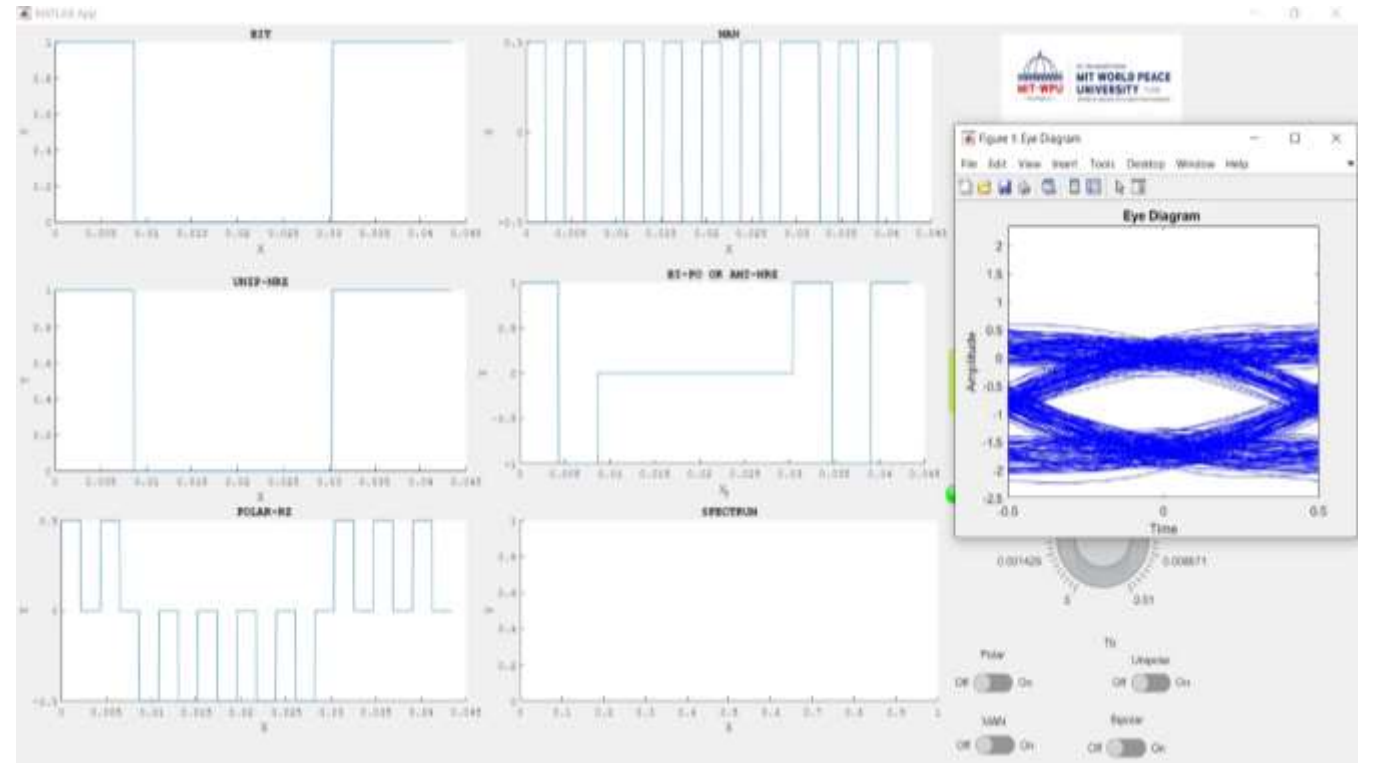
# OUTPUT



# OUTPUT



# EYE DIAGRAM





# REFERENCES

- [https://www.tutorialspoint.com/digital\\_communication/digital\\_communication\\_pulse\\_shaping.htm](https://www.tutorialspoint.com/digital_communication/digital_communication_pulse_shaping.htm)
- <https://www.edn.com/eye-diagram-basics-reading-and-applying-eye-diagrams/>
- <https://www.testandmeasurementtips.com/basics-eye-diagrams/>



# THANK YOU

- Tanmay Kapil(PC47)
- Jay Srivastava(PC46)
- Atul Kumar(PC44)

