UWB Report

Code Explanation:-

```
clc;
clear all;
close all;


currentFolder = pwd;
disp(currentFolder);


% Specify the path to the new directory
newFolder = 'C:\Users\Intern8\Documents\UWB';  % Modify this path as needed


% Change to the new directory
cd(newFolder);


% Verify the change
currentFolder = pwd;
disp(['New current directory: ', currentFolder]);


% Prompt user to select a file
[filename, filepath] = uigetfile({'*.csv', 'CSV Files (*.csv)'}, 'Select CSV
File');


% Check if the user canceled the dialog
if isequal(filename,0)
    disp('User canceled the dialog');
    return;
end


% Read the data from the selected CSV file into an array
dataArray = readmatrix(fullfile(filepath, filename));


% Ensure there are enough rows in dataArray
if size(dataArray, 1) < 500000
    disp('Not enough rows in the data file.');
    return;
end
```

```
% Extract the second column for the real part
realPart = dataArray(1:500000, 1);


% Extract the third column for the imaginary part
imagPart = dataArray(1:500000, 2);




% Form the complex signal
complexSig = realPart + 1i * imagPart;


% Compute the FFT
N = length(complexSig);  % Length of the signal
X = fftshift(fft(complexSig));  % Compute the Fourier Transform
```

This Section of code includes:-

- Clear the workspace and close all figures.
- Display the current working directory.
- Change to a specified directory.
- Prompt the user to select a CSV file.
- Read data from the selected CSV file.
- Extract the real and imaginary parts of a complex signal from the data.
- Compute the FFT of the complex signal.

This script will read the first 500,000 rows from the selected CSV file, form a complex signal from the specified columns, and then compute its FFT. Make sure to update the path in `newFolder` as needed for your environment.

```
fs=50e6;


% Frequency axis
df = fs / N;  % Frequency resolution
f = -fs/2 : df : fs/2 - df;  % Frequency axis


% Compute the power spectrum
power_spectrum = abs(X).^2 / N;  % Squared magnitude of FFT divided by N


% Convert to dBm
power_spectrum_dBm = 10 * log10(power_spectrum / 0.001);  % Convert to dBm


% Spectrogram parameters
window = hamming(256);  % Window size
noverlap = 0;  % Overlap between consecutive segments
nfft = 4096;  % FFT length
```

```
% Compute the STFT
[S, F, T, P] = spectrogram(complexSig, window, noverlap, nfft, fs, 'yaxis',
'power', 'centered');


P=P/(10^13.2);


% Plot the time-varying spectrum (spectrogram)
figure;
imagesc(T, F / 1e6, 10*log10(P*1000));
xlabel('Time (s)');
ylabel('Frequency (MHz)');
title('Spectrogram of Signal');
colorbar;
axis xy;


% Add labels and title
xlabel('Time (s)');
ylabel('Frequency (MHz)');
title('Spectrogram of Signal');
```

This part of the MATLAB script extends the previous script by computing and visualizing the power spectrum and the spectrogram of the complex signal.

- `abs(X).^2 / N` calculates the power spectrum.
- The power spectrum is then converted to dBm by taking 10 * log10(power_spectrum / 0.001).
- `abs(X).^2 / N` calculates the power spectrum.
- The power spectrum is then converted to dBm by taking 10 * log10(power_spectrum / 0.001).
- `imagesc` is used to plot the spectrogram.
- The time axis `T` is on the x-axis, and the frequency axis `F` (converted to MHz) is on the y-axis.
- The power `P` is converted to dBm by `10 * log10(P * 1000)` before plotting.
- `colorbar` adds a color scale to the plot.
- `axis xy` ensures the correct orientation of the plot.

```
% Order of STFT

[m,n]=size(P);
```

This line retrieves the number of rows `m` and columns `n` of the spectrogram power matrix `P`.

Form the Frequency Detection Matrix:

```
% Forming frequency detection Matrix
for i=1:m
    for j=1:n
        if P(i,j)>0.25e-11    % detecting frequencies above threshold -89dBm
           A(i,j)=1;
        else
           A(i,j)=0;
        end
    end
end
```

- This nested loop iterates over each element of the power matrix `P`.
- For each element `P(i, j)`, it checks if the power is above the threshold of 0.25×10−11Watts (which corresponds to -86 dBm).
- If the power is above the threshold, it sets the corresponding element in matrix `A` to 1.
- If the power is below the threshold, it sets the corresponding element in matrix `A` to 0.

The resulting matrix `A` is a binary matrix where 1 indicates that the corresponding frequency bin has power above the threshold, and 0 indicates it is below the threshold. This matrix can be used for frequency detection and analysis based on the power levels in the spectrogram.

```
fre=[];
for i=1:m
    b=[];
    for j=1:n
        if A(i,j)==1
           arr(i,j)=F(i)/1e6;
           b=[b,F(i)/1e6];
        else
           arr(i,j)=0;
        end
    end
    fre=[fre,b];
end
```

This part of the MATLAB script is creating a frequency array `fre` based on the frequency detection matrix `A` and the frequency vector `F` from the spectrogram. Here is a detailed explanation:

Explanation:

1. **Outer Loop:** The outer loop iterates over each row `i` of the matrix `A`.
   - `b` is initialized as an empty array at the beginning of each row.
2. **Inner Loop:** The inner loop iterates over each column `j` of the current row `i`.
   - If `A(i, j)` is 1, it means the power at this frequency bin is above the threshold.
     - The frequency value `F(i)`, converted to MHz, is stored in the corresponding position of the `arr` matrix.
     - The frequency value `F(i)`, converted to MHz, is appended to the temporary array `b`.
   - If `A(i, j)` is 0, it sets `arr(i, j)` to 0.

3. **Update Frequency Array:**
   - After processing all columns of the current row, the temporary array `b` (containing frequencies above the threshold) is appended to the `fre` array.

Summary:

- The resulting `fre` array contains all the detected frequencies (in MHz) where the power was above the specified threshold.
- The `arr` matrix stores the detected frequencies in their respective positions, with zeros elsewhere.

```
% Define a threshold to group nearby frequencies or frequency resolution in GHz

threshold = 1; % 1MHz is the resolution



% Initialize variables to store groups and center frequencies

groups = {};

current_group = fre(1);



% Group the frequencies

for i = 2:length(fre)

    if abs(fre(i) - fre(i-1)) <= threshold

        current_group = [current_group, fre(i)];

    else

        groups{end+1} = current_group;

        current_group = fre(i);

    end

end

groups{end+1} = current_group; % Add the last group



% Calculate the center frequencies

center_frequencies = cellfun(@mean, groups);
```

Explanation:

- **Initialize `groups` as an empty cell array** to store frequency groups.
- **Start the first group** with the first frequency in `fre`.

- **Loop through `fre` starting from the second frequency:**
    - If the difference between the current frequency and the previous frequency is less than or equal to the threshold (1 MHz), it is considered part of the same group, and the frequency is added to `current_group`.
    - If the difference exceeds the threshold, the current group is saved to `groups`, and a new group is started with the current frequency.

- **Add the final group** to `groups` after the loop completes.
- `cellfun(@mean, groups)` calculates the mean of each group of frequencies, resulting in the center frequencies.

Summary:

- The script groups frequencies in `fre` that are within 1 MHz of each other.
- The center frequencies of these groups are calculated and stored in `center_frequencies`.

This script effectively groups frequencies that are close to each other and computes the average frequency for each group, providing a simplified representation of the detected frequency components.

```
a=[]; % Initialize array to store coordinates
for p=1:length(center_frequencies)
    for i = 1:m
        for j = 1:n
            if abs(arr(i,j) - center_frequencies(p)) <= threshold &&
arr(i,j)~=0
                a(end+1,p) =j; % Appending an element in ithe pth column of a
matrix
            end
        end
    end
end
```

This part of the MATLAB script creates a matrix `a` to store the coordinates (column indices) of the frequencies in the spectrogram that are close to the center frequencies of the detected groups. Here's a detailed explanation:

Explanation:

- The outer loop iterates over each center frequency.
- The nested loops iterate over each element of the spectrogram matrix `arr`.
    - For each element, it checks if the difference between the frequency at `arr(i, j)` and the current center frequency is within the threshold and if the frequency is not zero.
    - If the condition is met, it appends the column index `j` to the `p`-th column of matrix `a`.

Summary:

- The matrix `a` will have `p` columns corresponding to each center frequency.
- Each column contains the column indices `j` from the spectrogram matrix `arr` where the frequency values are close to the respective center frequency.

```
% Initialize an empty cell array to store increasing subsequences
increasing_subsequences = cell(length(center_frequencies), 1);


% Loop through each column of the matrix 'a'
for j = 1:length(center_frequencies)
    % Initialize a temporary array to store the current subsequence for the current column
    current_subsequence = [];
    column_subsequences = {}; % Reset the subsequences for each column

    for i = 1:length(a(:,j))
        if i == 1 || a(i,j) > a(i-1,j)
            % If the current element is greater than the previous or it's the first element, add it to the current subsequence
            current_subsequence = [current_subsequence a(i,j)];
        else
            % If the current element is not greater, save the current subsequence if it is valid and start a new one
            if length(current_subsequence) > 1
                column_subsequences{end+1} = current_subsequence; % Save the subsequence
            end
            current_subsequence = a(i,j); % Start a new subsequence
        end
    end

    % Save the last subsequence if it is valid
    if length(current_subsequence) > 1
        column_subsequences{end+1} = current_subsequence;
    end
    % Store the increasing subsequences for the current column
    increasing_subsequences{j} = column_subsequences;
end
```

This part of the MATLAB script is designed to find and store all increasing subsequences of column indices in matrix `a` for each center frequency. Here's a detailed explanation of the code:

Explanation:

1. **Outer Loop:**
   - Iterates over each column `j` of the matrix `a`, corresponding to each center frequency.
2. **Initialize `current_subsequence` and `column_subsequences`:**
   - `current_subsequence` is an array to store the current increasing subsequence.
   - `column_subsequences` is a cell array to store all valid increasing subsequences for the current column.
3. **Inner Loop:**
   - Iterates over each row `i` of the current column `a(:, j)`.
   - If it's the first element (`i == 1`) or the current element `a(i, j)` is greater than the previous element `a(i-1, j)`, it adds the current element to `current_subsequence`.
   - If the current element is not greater, it checks if the current subsequence is valid (length > 1). If valid, it saves the current subsequence to `column_subsequences` and starts a new subsequence with the current element.
4. **After the Inner Loop:**
   - The last `current_subsequence` is saved if it is valid.
5. **Store the Increasing Subsequences:**
   - The `column_subsequences` for the current column are stored in the `increasing_subsequences` cell array.

Summary:

- The script finds all increasing subsequences of column indices for each center frequency and stores them in a cell array `increasing_subsequences`. Each cell in `increasing_subsequences` contains a cell array of increasing subsequences for the corresponding center frequency.

```
for i = 1:length(center_frequencies)
    seq = increasing_subsequences{i, 1};  % Get the subsequence cell array


    % Find the subsequence with the maximum size
    max_length = -1;
    max_subsequence = [];
    for j = 1:length(seq)
        current_sequence = seq{j};
        current_length = length(current_sequence);

        if current_length > max_length
            max_length = current_length;
            max_subsequence = current_sequence;
        end
    end
    r(i)=1;
    for j=2:length(max_subsequence)
        if abs(max_subsequence(j)-max_subsequence(j-1))==1
           r(i)=j;
        end
        if abs(max_subsequence(j)-max_subsequence(j-1))~=1
               pulse_width(i)=max_subsequence(j)-max_subsequence(1);
               break;
        end
    end
end
```

Explanation:

1. **Outer Loop:**
   - Iterates over each center frequency.
2. **Get the Subsequence Cell Array:**
   - Retrieves the cell array of increasing subsequences for the current center frequency.
3. **Find the Longest Increasing Subsequence:**
   - Initializes `max_length` and `max_subsequence` to track the longest subsequence.
   - Loops through each subsequence in `seq` to find the one with the maximum length.
4. **Initialize `r(i)` and Process the Maximum Subsequence:**
   - Sets `r(i)` to 1 initially.
   - Loops through the elements of the longest subsequence `max_subsequence` starting from the second element.

    - Checks if the difference between consecutive elements is 1, indicating a continuous sequence.
    - Updates `r(i)` to the current index `j` if the difference is 1.
    - If the difference is not 1, it calculates the pulse width as the difference between the current element and the first element of the subsequence, then breaks the loop.

Summary:

- The script identifies the longest increasing subsequence of indices for each center frequency.
- It then calculates the pulse width based on the longest continuous part of the subsequence.

```
% Define the center frequency and sampling frequency


% Frequency resolution

freq_resolution = fs/ nfft;

Time_resolution=length(window)/fs;


for i=1:length(center_frequencies)

    center_freq = round(center_frequencies(i)) * 1e6; % Convert to Hz


    % Calculate the approximate index

    ind = (nfft/2)+round(center_freq / freq_resolution);


    % Access the magnitude spectrum at the approximated index

    center_freq_magnitude = max(P(ind, :));


    % Find the maximum value in dBm for the selected frequency

    power_dBm(i) = 10 * log10(center_freq_magnitude*1000);  % Convert to
dBm

end
```

Explanation:

1. **Frequency Resolution and Time Resolution:**
    - `freq_resolution` is the frequency resolution of the FFT, calculated by dividing the sampling frequency `fs` by the FFT length `nfft`.

    - `Time_resolution` is the time resolution of the window, calculated by dividing the length of the window by the sampling frequency `fs`.
2. **Center Frequency Conversion:**
    - Converts each center frequency from MHz to Hz by multiplying by 1e61e61e6.
3. **Approximate Index Calculation:**
    - Calculates the approximate index `ind` in the frequency domain corresponding to the center frequency.
    - The center frequency index is found by adding half the FFT length (`nfft/2`) to the rounded value of the center frequency divided by the frequency resolution.
4. **Magnitude Spectrum Access:**
    - Accesses the magnitude spectrum `P` at the approximated index `ind` for all time points.
    - Finds the maximum value in the magnitude spectrum for the center frequency.
5. **Conversion to dBm:**
    - Converts the maximum magnitude value to dBm by multiplying by 1000 (to convert from Watts to milliwatts) and then using 10log1010 \log_{10}10log10 to convert to dBm.

Summary:

- The script calculates the power in dBm for each center frequency by finding the maximum value in the magnitude spectrum at the approximate frequency index.
- The resulting power values in dBm are stored in the `power_dBm` array.

```
% Display Pulse duration and Duty cycle

for i = 1:length(center_frequencies)

    disp(['Pulse Width(msec): ',
num2str(pulse_width(i)*Time_resolution*1000)]); % Display pulse width

    duty_cycle = r(i) / pulse_width(i); % Calculate duty cycle

    disp(['Duty Cycle(%): ', num2str(duty_cycle*100)]); % Display duty
cycle

    disp(['Power(in dBm): ', num2str(power_dBm(i))]);

end
```

This part of the MATLAB script calculates and displays the pulse width in milliseconds, duty cycle, and power in dBm for each center frequency. Here's a detailed explanation:

- The pulse width in milliseconds is calculated by multiplying the `pulse_width` by the time resolution and converting to milliseconds.
- The duty cycle is calculated as the ratio of the length of the maximum increasing subsequence (`r(i)`) to the pulse width.
- It is then converted to a percentage by multiplying by 100.

Summary:

- The script calculates the pulse width in milliseconds, duty cycle in percentage, and power in dBm for each center frequency and displays these values.

Input Data:-

Input we considered here is "Modulated pulsed continuous wave" (MPCW) which is a type of radar waveform that combines aspects of both continuous wave (CW) and pulse radar techniques.

Recorded IQ data from Signal Generator at center frequency 2.45GHz.

Pulse Width is considered as 0.1msec and Pulse duration is 1msec. Therefore, duty cycle is 10%.

RF Bandwidth is considered as 40MHz which is as per according to ADALM-PLUTO SDR.

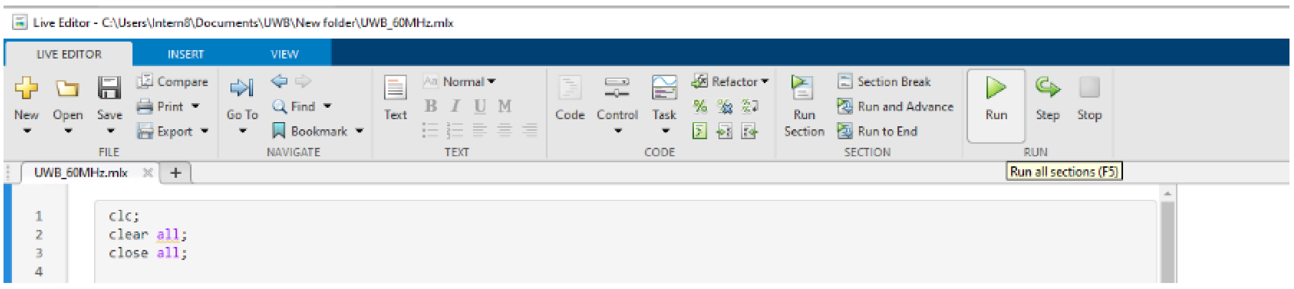
Sampling Rate is considered as 50MHz.

RF Noise Floor is -98dBm according to NF=KTB.

Peak Power is considered as -55dBm.

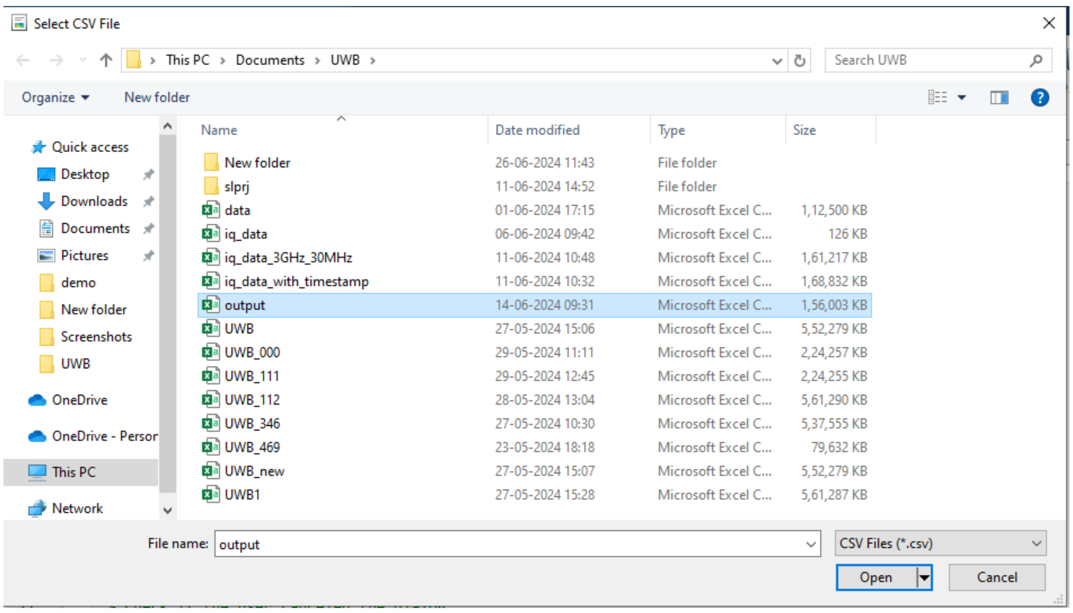
Steps To Run .mlx File:-

1. After open the .mlx file clink on the Run button in the Live Editor Bar as shown bellow:



```
clc;
clear all;
close all;
```

2. After that, a dialog box will pop up to prompt the user to select a .csv file. In the dialog box, select the file named "output.csv" and then click on "Open".



Select CSV File

This PC › Documents › UWB ›

Search UWB

Organize ▾ New folder

Quick access
Desktop
Downloads
Documents
Pictures
demo
New folder
Screenshots
UWB
OneDrive
OneDrive - Persor
This PC
Network

Name	Date modified	Type	Size
New folder	26-06-2024 11:43	File folder	
slprj	11-06-2024 14:52	File folder	
data	01-06-2024 17:15	Microsoft Excel C...	1,12,500 KB
iq_data	06-06-2024 09:42	Microsoft Excel C...	126 KB
iq_data_3GHz_30MHz	11-06-2024 10:48	Microsoft Excel C...	1,61,217 KB
iq_data_with_timestamp	11-06-2024 10:32	Microsoft Excel C...	1,68,832 KB
output	14-06-2024 09:31	Microsoft Excel C...	1,56,003 KB
UWB	27-05-2024 15:06	Microsoft Excel C...	5,52,279 KB
UWB_000	29-05-2024 11:11	Microsoft Excel C...	2,24,257 KB
UWB_111	29-05-2024 12:45	Microsoft Excel C...	2,24,255 KB
UWB_112	28-05-2024 13:04	Microsoft Excel C...	5,61,290 KB
UWB_346	27-05-2024 10:30	Microsoft Excel C...	5,37,555 KB
UWB_469	23-05-2024 18:18	Microsoft Excel C...	79,632 KB
UWB_new	27-05-2024 15:07	Microsoft Excel C...	5,52,279 KB
UWB1	27-05-2024 15:28	Microsoft Excel C...	5,61,287 KB

File name: output

CSV Files (*.csv)

Open Cancel

3. Now see the results

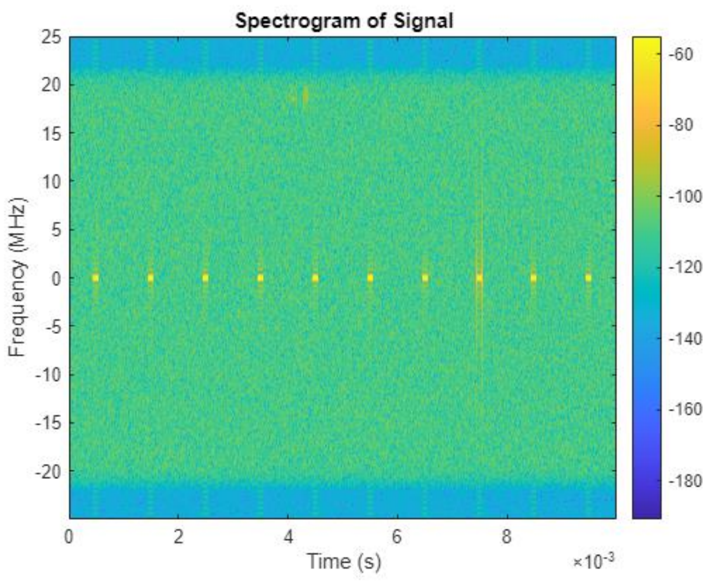
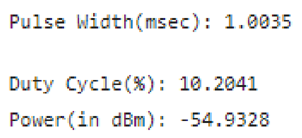


Fig: Spectrogram



```
Pulse Width(msec): 1.0035

Duty Cycle(%): 10.2041
Power(in dBm): -54.9328
```

Fig: Parameters

Code For MATLAB App:-

```
classdef new_app < matlab.apps.AppBase

    % Properties that correspond to app components
    properties (Access = public)
        UIFigure                       matlab.ui.Figure
        Frequency3GHzEditField         matlab.ui.control.NumericEditField
        Frequency3GHzEditFieldLabel    matlab.ui.control.Label
        DutyCycle3EditField            matlab.ui.control.NumericEditField
        DutyCycle3EditFieldLabel       matlab.ui.control.Label
        PulseWidth3nsecEditField       matlab.ui.control.NumericEditField
        PulseWidth3usecLabel           matlab.ui.control.Label
        Signal3Label                   matlab.ui.control.Label
        DutyCycle2EditField            matlab.ui.control.NumericEditField
        DutyCycle2EditFieldLabel       matlab.ui.control.Label
        PulseWidth2nsecEditField       matlab.ui.control.NumericEditField
        PulseWidth2nsecEditFieldLabel  matlab.ui.control.Label
        Signal2Label                   matlab.ui.control.Label
        DutyCycle1EditField            matlab.ui.control.NumericEditField
        DutyCycle1EditFieldLabel       matlab.ui.control.Label
        PulseWidth1nsecEditField       matlab.ui.control.NumericEditField
        PulseWidth1nsecEditFieldLabel  matlab.ui.control.Label
        Signal1Label                   matlab.ui.control.Label
        EditField                      matlab.ui.control.EditField
        Frequency2GHzEditField         matlab.ui.control.NumericEditField
        Frequency2GHzEditFieldLabel    matlab.ui.control.Label
        Frequency1GHzEditField         matlab.ui.control.NumericEditField
        Frequency1GHzLabel             matlab.ui.control.Label
        UITable                        matlab.ui.control.Table
        GeneratePDWButton              matlab.ui.control.Button
        StartButton                    matlab.ui.control.Button
        UIAxes                         matlab.ui.control.UIAxes
    end


    properties (Access = public)
         dataArray; % Description
        S;
        F;
        t;
        fs;
        P;
        nfft;
```

```
            magnitude;
            peak_power_dBm
            duty_cycle;
            center_frequencies;
            pulse_duration;
            power_dBm;
        end


        % Callbacks that handle component events
        methods (Access = private)

            % Button pushed function: StartButton
            function StartButtonPushed(app, event)
                 clc;

                % Checking current path directory

                newText='';
                app.EditField.Value=newText;


     app.fs = 36e9; % Sampling frequency (Hz)
     F1=app.Frequency1GHzEditField.Value;
     F2=app.Frequency2GHzEditField.Value;
     F3=app.Frequency3GHzEditField.Value;

     PW1=app.PulseWidth1nsecEditField.Value;
     PW2=app.PulseWidth2nsecEditField.Value;
     PW3=app.PulseWidth3nsecEditField.Value;

     DC1=app.DutyCycle1EditField.Value;
     DC2=app.DutyCycle2EditField.Value;
     DC3=app.DutyCycle3EditField.Value;

     % Set the frequencies in the workspace
     assignin('base', 'F1', F1);
     assignin('base', 'F2', F2);
     assignin('base', 'F3', F3);
     assignin('base', 'PW1', PW1);
     assignin('base', 'PW2', PW2);
     assignin('base', 'PW3', PW3);
     assignin('base', 'DC1', DC1);
     assignin('base', 'DC2', DC2);
     assignin('base', 'DC3', DC3);
     assignin('base', 'fs', app.fs);

     mdl='UWB_2023';
     out=sim(mdl);

     O=out.simout;
     app.dataArray(:,1)=O.time;
     app.dataArray(:,2)=real(O.data);
     app.dataArray(:,3)=imag(O.data);

     newText='Ready To Go';
     app.EditField.Value=newText;
            end
            % Button down function: UIAxes
            function UIAxesButtonDown(app, event)
                % Check if dataArray has at least 3 columns
        if size(app.dataArray, 2) < 3
            error('dataArray does not have enough columns. It should have at least 3 
    columns.');
        end

        realPart = app.dataArray(:,2);
        imagPart = app.dataArray(:,3);

        % Form the complex signal
        complexSig = realPart + 1i * imagPart;

        % Calculating magnitude of complex Signals
        app.magnitude = abs(complexSig);

        % Compute the STFT parameters

        N = length(complexSig); % Length of the signal

        % Create spectrogram
        window = hamming(512); % Window size
        noverlap = 0; % Overlap between consecutive segments
        app.nfft = 4096; % FFT length

        % Compute the STFT
        [app.S, app.F, app.t,app.P] = spectrogram(complexSig, window, noverlap, 
    app.nfft, app.fs, 'yaxis','power');

        % Plot the time-varying spectrum on the UIAxes
        freq_range = [2e9, 18e9]; % Frequency range from 2 GHz to 18 GHz
        freq_indices = app.F >= freq_range(1) & app.F <= freq_range(2);
        imagesc(app.UIAxes, app.t, app.F(freq_indices),10*log10(app.P(freq_indices, 
    :)*1000));
        axis(app.UIAxes, 'xy');

        % Set x and y axis labels
        xlabel(app.UIAxes, 'Time (s)');
        ylabel(app.UIAxes, 'Frequency (Hz)');
        title(app.UIAxes, 'Spectrogram of Signal');
        colorbar(app.UIAxes);

        % Set the axis limits
    xlim(app.UIAxes, [min(app.t), max(app.t)]); % Adjust the x-axis limits
    ylim(app.UIAxes, [min(app.F(freq_indices)), max(app.F(freq_indices))]); % Adjust 
    the y-axis limits

    % Customize color axis (colorbar)
    % Adjust color axis limits to cover the range of power values in dBm
    caxis(app.UIAxes, [-65, max(max(10*log10(app.P(freq_indices, :) * 1000)))]);
    %
    % % Reverse the colormap for darker colors for high amplitudes and brighter colors 
    for lower amplitudes
    % colormap(app.UIAxes, flipud(parula)); % Use reversed parula colormap
            end

            % Button pushed function: GeneratePDWButton
            function GeneratePDWButtonPushed(app, event)
```

```
% Order of STFT
[m,n]=size(app.P)

% Forming frequency detection Matrix
for i=1:m
    for j=1:n
        if app.P(i,j)>=1e-8     % detecting frequencies above threshold
            A(i,j)=1;
        else
            A(i,j)=0;
        end
    end
end

fre=[];
for i=1:m
    b=[];
    for j=1:n
        if A(i,j)==1 & app.F(i)<18e9
           arr(i,j)=app.F(i)/1e9;
           b=[b,app.F(i)/1e9];
        else
           arr(i,j)=0;
        end
    end
    fre=[fre,b];
end

% Define a threshold to group nearby frequencies or frequency resolution in GHz
threshold = 0.01; % 10 MHz is the resolution

% Initialize variables to store groups and center frequencies
groups = {};
current_group = fre(1);

  % Group the frequencies
    for i = 2:length(fre)
        if abs(fre(i) - fre(i-1)) <= threshold
            current_group = [current_group, fre(i)];
        else
            groups{end+1} = current_group;
            current_group = fre(i);
        end
    end
    groups{end+1} = current_group; % Add the last group


% Calculate the center frequencies
app.center_frequencies = cellfun(@mean, groups);

a=[]; % Initialize array to store coordinates
for p=1:length(app.center_frequencies)
    for i = 1:m
        for j = 1:n
            if abs(arr(i,j) - app.center_frequencies(p)) <= threshold
               a(end+1,p) =j; % Appending an element in ithe pth column of a
matrix
```

```
                end
            end
        end
    end

    l=length(a(:,1));
    for i=1:length(app.center_frequencies)
        initial(i)=0;
        pulse_width(i)=1;
        for j=1:l
            if a(j,i)~=0
                if initial(i)==0
                   initial(i)=a(j,i);
                   r=j;
                end
                if j~=1 & abs(a(j,i)-a(j-1,i))~=1
                   pulse_width(i)=a(j,i)-initial(i);
                   cw_width(i)=j-r;
                   break;
                end
            end
        end
    end
    app.center_frequencies
    cw_width
    pulse_width

        % Display Pulse duration and Duty cycle
        for i = 1:length(app.center_frequencies)
            app.duty_cycle(i) = app.t(cw_width(i)) / app.t(pulse_width(i));  %
    Calculate duty cycle
            app.pulse_duration(i)=app.t(pulse_width(i))*1e6;
        end

    % Frequency resolution
    freq_resolution = app.fs/ app.nfft;

    for i=1:length(app.center_frequencies)
        center_freq = app.center_frequencies(i) * 1e9; % Convert to Hz

        % Calculate the approximate index
        ind = round(center_freq / freq_resolution);

        % Access the magnitude spectrum at the approximated index
        center_freq_magnitude = max(app.P(ind, :));

        % Find the maximum value in dBm for the selected frequency
        app.power_dBm(i) = 10 * log10(center_freq_magnitude*1000);  % Convert to dBm
    end
    % Initialize the data and column headers
    data = [app.center_frequencies', app.pulse_duration',
    app.duty_cycle',app.power_dBm'];
    columnHeaders = {'Frequency (GHz)', 'Pulse Duration (usec)', 'Duty Cycle',
    'Power(dBm)'};

    % Add column headers to the data
    data = [columnHeaders; num2cell(data)]

    % Set the data to the table
```

```
app.UITable.Data = data;

        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 = [100 100 636 494];
            app.UIFigure.Name = 'MATLAB App';

            % Create UIAxes
            app.UIAxes = uiaxes(app.UIFigure);
            title(app.UIAxes, 'Receive Spectrogram')
            xlabel(app.UIAxes, 'Time')
            ylabel(app.UIAxes, 'Frequency')
            zlabel(app.UIAxes, 'Z')
            app.UIAxes.ButtonDownFcn = createCallbackFcn(app, @UIAxesButtonDown,
true);
            app.UIAxes.Position = [256 226 365 236];

            % Create StartButton
            app.StartButton = uibutton(app.UIFigure, 'push');
            app.StartButton.ButtonPushedFcn = createCallbackFcn(app,
@StartButtonPushed, true);
            app.StartButton.Position = [256 461 100 23];
            app.StartButton.Text = 'Start';

            % Create GeneratePDWButton
            app.GeneratePDWButton = uibutton(app.UIFigure, 'push');
            app.GeneratePDWButton.ButtonPushedFcn = createCallbackFcn(app,
@GeneratePDWButtonPushed, true);
            app.GeneratePDWButton.Position = [271 155 100 23];
            app.GeneratePDWButton.Text = 'Generate PDW';

            % Create UITable
            app.UITable = uitable(app.UIFigure);
            app.UITable.ColumnName = '';
            app.UITable.RowName = {};
            app.UITable.Position = [25 30 591 117];

            % Create Frequency1GHzLabel
            app.Frequency1GHzLabel = uilabel(app.UIFigure);
            app.Frequency1GHzLabel.HorizontalAlignment = 'right';
            app.Frequency1GHzLabel.Position = [23 419 107 22];
            app.Frequency1GHzLabel.Text = 'Frequency 1 (GHz)';

            % Create Frequency1GHzEditField
            app.Frequency1GHzEditField = uieditfield(app.UIFigure, 'numeric');
            app.Frequency1GHzEditField.ValueDisplayFormat = '%0.2f';
            app.Frequency1GHzEditField.Position = [144 419 100 22];
            app.Frequency1GHzEditField.Value = 11;

            % Create Frequency2GHzEditFieldLabel
```

```matlab
app.Frequency2GHzEditFieldLabel = uilabel(app.UIFigure);
app.Frequency2GHzEditFieldLabel.HorizontalAlignment = 'right';
app.Frequency2GHzEditFieldLabel.Position = [26 322 104 22];
app.Frequency2GHzEditFieldLabel.Text = 'Frequency 2(GHz)';

% Create Frequency2GHzEditField
app.Frequency2GHzEditField = uieditfield(app.UIFigure, 'numeric');
app.Frequency2GHzEditField.ValueDisplayFormat = '%0.2f';
app.Frequency2GHzEditField.Position = [145 322 100 22];
app.Frequency2GHzEditField.Value = 13;

% Create EditField
app.EditField = uieditfield(app.UIFigure, 'text');
app.EditField.Position = [398 195 83 21];

% Create Signal1Label
app.Signal1Label = uilabel(app.UIFigure);
app.Signal1Label.Position = [125 440 48 22];
app.Signal1Label.Text = 'Signal 1';

% Create PulseWidth1nsecEditFieldLabel
app.PulseWidth1nsecEditFieldLabel = uilabel(app.UIFigure);
app.PulseWidth1nsecEditFieldLabel.HorizontalAlignment = 'right';
app.PulseWidth1nsecEditFieldLabel.Position = [13 398 116 22];
app.PulseWidth1nsecEditFieldLabel.Text = 'Pulse Width 1 (nsec)';

% Create PulseWidth1nsecEditField
app.PulseWidth1nsecEditField = uieditfield(app.UIFigure, 'numeric');
app.PulseWidth1nsecEditField.ValueDisplayFormat = '%0.2f';
app.PulseWidth1nsecEditField.Position = [144 398 100 22];
app.PulseWidth1nsecEditField.Value = 500;

% Create DutyCycle1EditFieldLabel
app.DutyCycle1EditFieldLabel = uilabel(app.UIFigure);
app.DutyCycle1EditFieldLabel.HorizontalAlignment = 'right';
app.DutyCycle1EditFieldLabel.Position = [56 377 73 22];
app.DutyCycle1EditFieldLabel.Text = 'Duty Cycle 1';

% Create DutyCycle1EditField
app.DutyCycle1EditField = uieditfield(app.UIFigure, 'numeric');
app.DutyCycle1EditField.ValueDisplayFormat = '%0.2f';
app.DutyCycle1EditField.Position = [144 377 100 22];
app.DutyCycle1EditField.Value = 0.3;

% Create Signal2Label
app.Signal2Label = uilabel(app.UIFigure);
app.Signal2Label.Position = [129 343 48 22];
app.Signal2Label.Text = 'Signal 2';

% Create PulseWidth2nsecEditFieldLabel
app.PulseWidth2nsecEditFieldLabel = uilabel(app.UIFigure);
app.PulseWidth2nsecEditFieldLabel.HorizontalAlignment = 'right';
app.PulseWidth2nsecEditFieldLabel.Position = [14 301 116 22];
app.PulseWidth2nsecEditFieldLabel.Text = 'Pulse Width 2 (nsec)';

% Create PulseWidth2nsecEditField
app.PulseWidth2nsecEditField = uieditfield(app.UIFigure, 'numeric');
app.PulseWidth2nsecEditField.ValueDisplayFormat = '%0.2f';
app.PulseWidth2nsecEditField.Position = [145 301 100 22];
```

```
            app.PulseWidth2nsecEditField.Value = 200;

            % Create DutyCycle2EditFieldLabel
            app.DutyCycle2EditFieldLabel = uilabel(app.UIFigure);
            app.DutyCycle2EditFieldLabel.HorizontalAlignment = 'right';
            app.DutyCycle2EditFieldLabel.Position = [57 280 73 22];
            app.DutyCycle2EditFieldLabel.Text = 'Duty Cycle 2';

            % Create DutyCycle2EditField
            app.DutyCycle2EditField = uieditfield(app.UIFigure, 'numeric');
            app.DutyCycle2EditField.ValueDisplayFormat = '%0.2f';
            app.DutyCycle2EditField.Position = [145 280 100 22];
            app.DutyCycle2EditField.Value = 0.5;

            % Create Signal3Label
            app.Signal3Label = uilabel(app.UIFigure);
            app.Signal3Label.Position = [129 236 48 22];
            app.Signal3Label.Text = 'Signal 3';

            % Create PulseWidth3usecLabel
            app.PulseWidth3usecLabel = uilabel(app.UIFigure);
            app.PulseWidth3usecLabel.HorizontalAlignment = 'right';
            app.PulseWidth3usecLabel.Position = [12 194 116 22];
            app.PulseWidth3usecLabel.Text = 'Pulse Width 3 (nsec)';

            % Create PulseWidth3nsecEditField
            app.PulseWidth3nsecEditField = uieditfield(app.UIFigure, 'numeric');
            app.PulseWidth3nsecEditField.ValueDisplayFormat = '%0.2f';
            app.PulseWidth3nsecEditField.Position = [143 194 100 22];
            app.PulseWidth3nsecEditField.Value = 1000;

            % Create DutyCycle3EditFieldLabel
            app.DutyCycle3EditFieldLabel = uilabel(app.UIFigure);
            app.DutyCycle3EditFieldLabel.HorizontalAlignment = 'right';
            app.DutyCycle3EditFieldLabel.Position = [55 173 73 22];
            app.DutyCycle3EditFieldLabel.Text = 'Duty Cycle 3';

            % Create DutyCycle3EditField
            app.DutyCycle3EditField = uieditfield(app.UIFigure, 'numeric');
            app.DutyCycle3EditField.ValueDisplayFormat = '%0.2f';
            app.DutyCycle3EditField.Position = [143 173 100 22];
            app.DutyCycle3EditField.Value = 0.7;

            % Create Frequency3GHzEditFieldLabel
            app.Frequency3GHzEditFieldLabel = uilabel(app.UIFigure);
            app.Frequency3GHzEditFieldLabel.HorizontalAlignment = 'right';
            app.Frequency3GHzEditFieldLabel.Position = [25 215 104 22];
            app.Frequency3GHzEditFieldLabel.Text = 'Frequency 3(GHz)';

            % Create Frequency3GHzEditField
            app.Frequency3GHzEditField = uieditfield(app.UIFigure, 'numeric');
            app.Frequency3GHzEditField.ValueDisplayFormat = '%0.2f';
            app.Frequency3GHzEditField.Position = [144 215 100 22];
            app.Frequency3GHzEditField.Value = 17;

            % 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 = new_app

            % 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
```

Explanation :-

This MATLAB code defines an App Designer class named `new_app` that provides a graphical user interface (GUI) for generating and analyzing radar signals. The app includes fields to input signal parameters (frequency, pulse width, and duty cycle) for three different signals, a spectrogram display, and a button to generate Pulse Descriptor Words (PDWs).

Here is a detailed explanation of the code:

Properties

1. **UI Components:**
    - `UIFigure`: The main figure of the app.
    - `NumericEditField`, `Label`: Input fields and labels for frequency, duty cycle, and pulse width of three signals.
    - `EditField`: A text field to display status messages.
    - `UITable`: A table to display the generated PDWs.
    - `Button`: Buttons to start the simulation and generate PDWs.
    - `UIAxes`: Axes to display the spectrogram of the received signal.
2. **App Data Properties:**
    - `dataArray`: Array to store the real and imaginary parts of the simulated signal.
    - `S`, `F`, `t`, `fs`, `P`, `nfft`, `magnitude`: Variables related to the Short-Time Fourier Transform (STFT) and spectrogram.
    - `peak_power_dBm`, `duty_cycle`, `center_frequencies`, `pulse_duration`, `power_dBm`: Variables for PDW calculation.

Methods

1. **StartButtonPushed:**
    - Clears the command window.
    - Retrieves and assigns signal parameters from the input fields.
    - Assigns these parameters to the base workspace for use in a Simulink model (`UWB_2023`).
    - Runs the Simulink model and stores the output in `dataArray`.
2. **UIAxesButtonDown:**
    - Computes the spectrogram of the complex signal from `dataArray`.
    - Plots the spectrogram on `UIAxes` with appropriate labels and limits.
3. **GeneratePDWButtonPushed:**
    - Analyzes the spectrogram to detect frequencies above a certain threshold.
    - Groups the detected frequencies and calculates center frequencies.
    - Computes pulse widths and duty cycles for each detected frequency.
    - Converts the maximum magnitude at the center frequency to dBm.
    - Populates `UITable` with the calculated PDWs.

Component Initialization

- **createComponents:**
    - Initializes the UI components and their properties.
    - Positions the components on the figure.

App Creation and Deletion

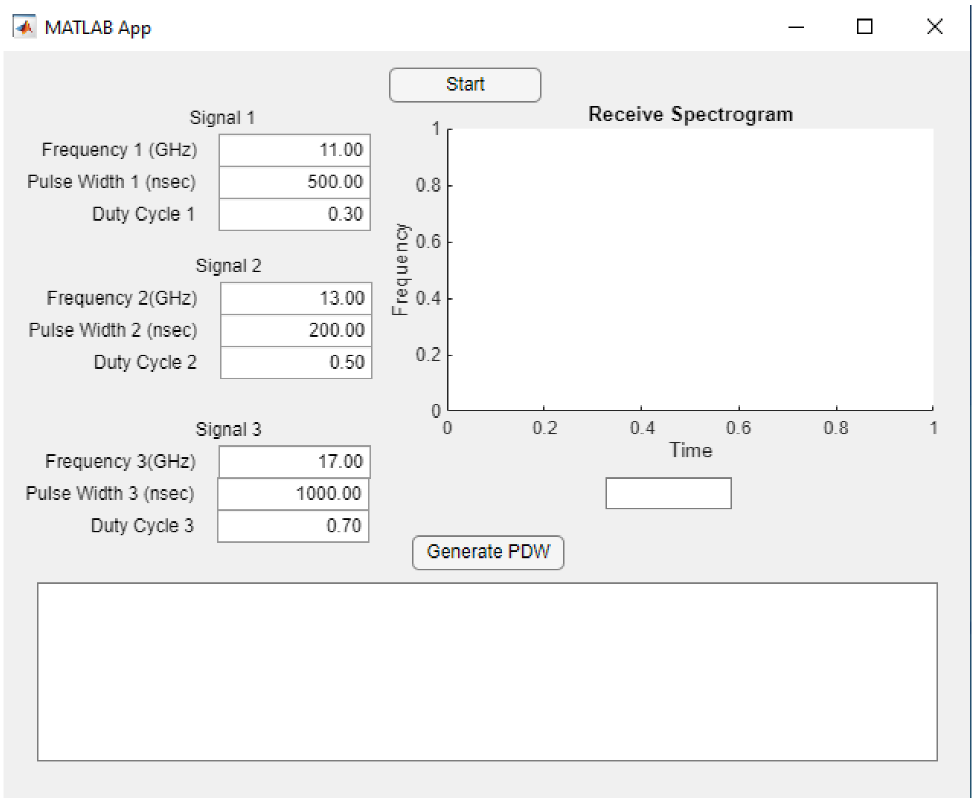
- **App Construction:**
    - Creates the app, initializes components, and registers the app.
- **App Deletion:**
    - Deletes the `UIFigure` when the app is deleted.

Key Functionalities

- **User Input:** Users can input parameters for three different radar signals.
- **Simulation:** The app uses Simulink to simulate the radar signals and store the result.
- **Spectrogram Analysis:** The app calculates and displays the spectrogram of the received signal.
- **PDW Generation:** The app detects frequencies, calculates pulse width, duty cycle, and power, and displays this information in a table.

This app is useful for radar signal analysis, providing a visual representation of the signal's spectrogram and extracting key parameters in the form of PDWs.

GUI:-


MATLAB App

Start

Receive Spectrogram

Signal 1

Frequency 1 (GHz)	11.00
Pulse Width 1 (nsec)	500.00
Duty Cycle 1	0.30

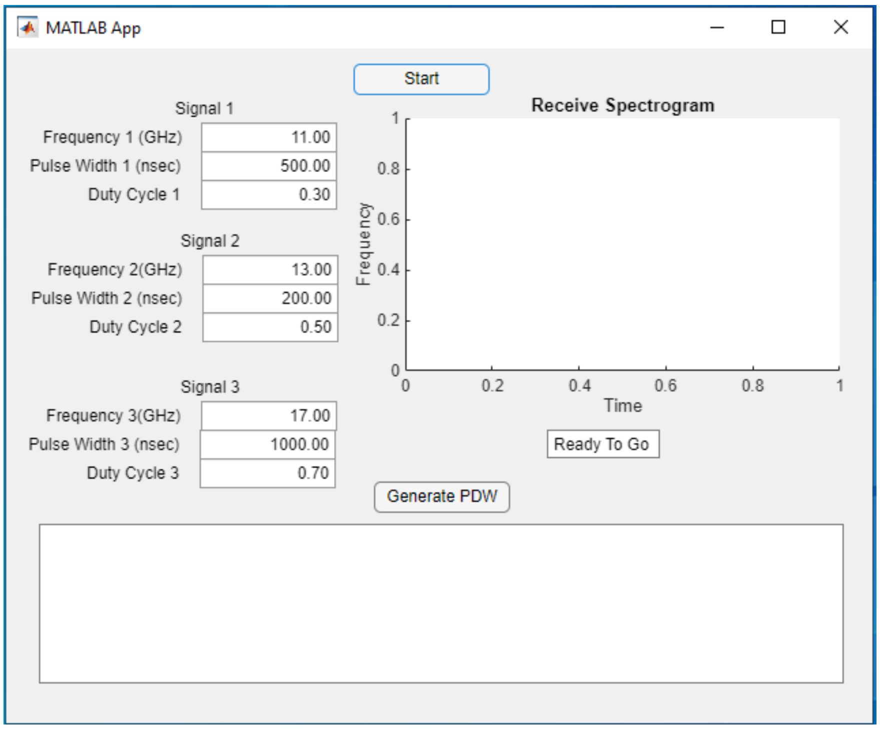
Signal 2

Frequency 2(GHz)	13.00
Pulse Width 2 (nsec)	200.00
Duty Cycle 2	0.50

Signal 3

Frequency 3(GHz)	17.00
Pulse Width 3 (nsec)	1000.00
Duty Cycle 3	0.70

- Input the frequency with 2-18GHz range, Pulse width in nsec and Duty cycle for 3 different MPCW signals which will interfere at receiver which after reception undergoes processing.
- Once IQ date at receiver is generated a “Ready To Go” message is popped up in the Display bellow axis.


MATLAB App

Start

Receive Spectrogram

Signal 1

Frequency 1 (GHz)	11.00
Pulse Width 1 (nsec)	500.00
Duty Cycle 1	0.30

Signal 2

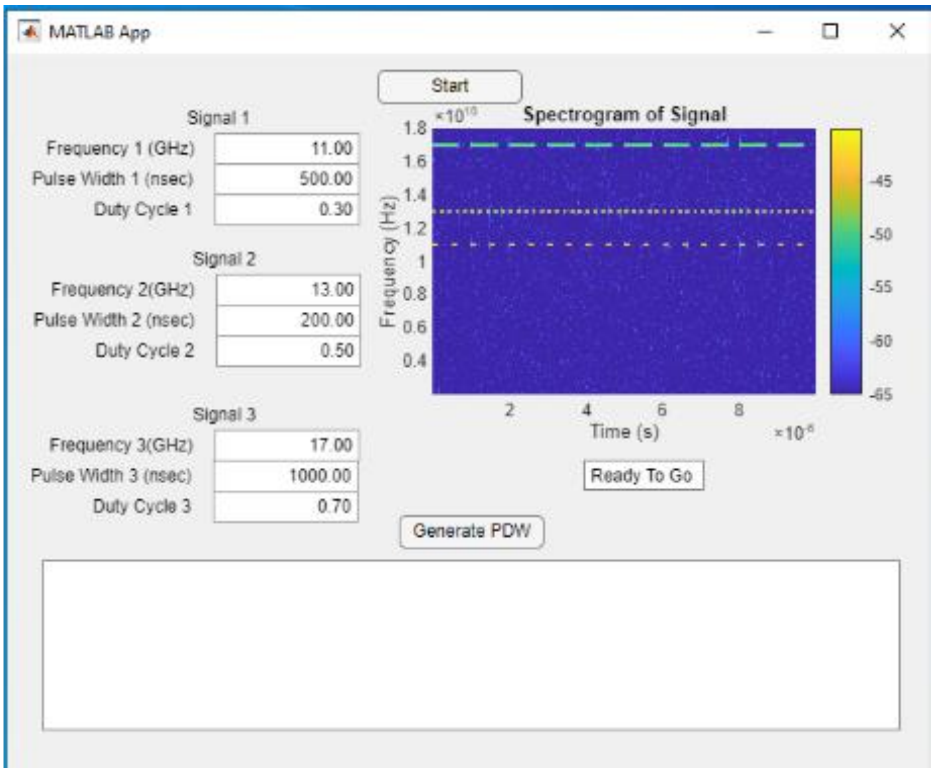
Frequency 2(GHz)	13.00
Pulse Width 2 (nsec)	200.00
Duty Cycle 2	0.50

Signal 3

Frequency 3(GHz)	17.00
Pulse Width 3 (nsec)	1000.00
Duty Cycle 3	0.70

Ready To Go

- After seeing this message click on the axis plot and you will see the spectrogram plot in it.



Signal 1

Frequency 1 (GHz)	11.00
Pulse Width 1 (nsec)	500.00
Duty Cycle 1	0.30

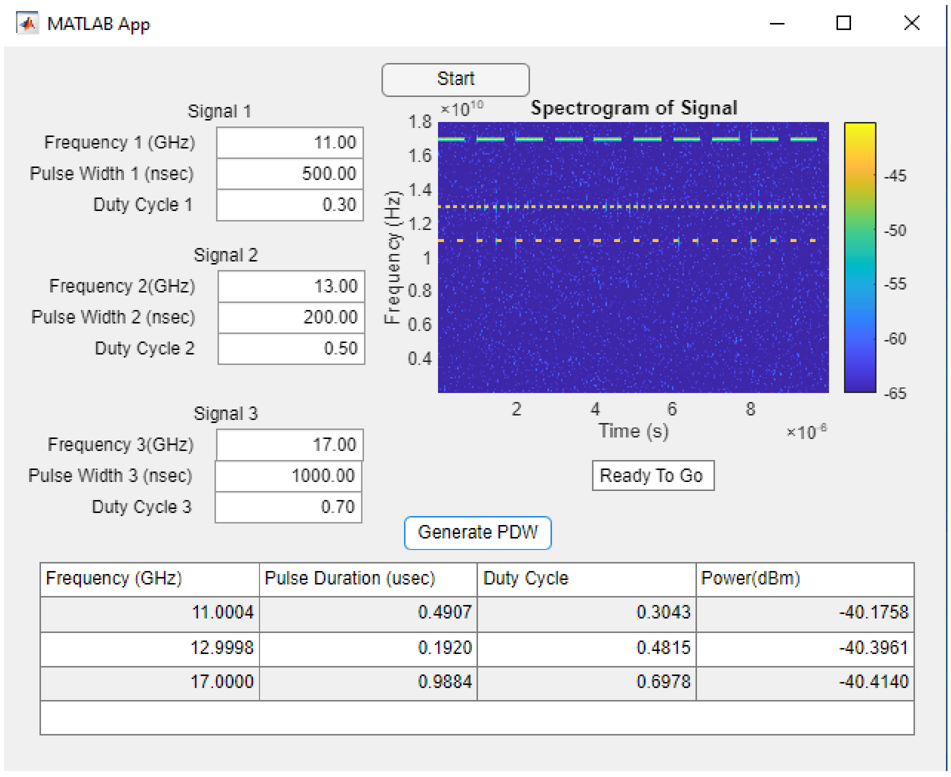
Signal 2

Frequency 2(GHz)	13.00
Pulse Width 2 (nsec)	200.00
Duty Cycle 2	0.50

Signal 3

Frequency 3(GHz)	17.00
Pulse Width 3 (nsec)	1000.00
Duty Cycle 3	0.70

- After generation of Spectrogram click on "Generate PDW" and wait for some time and you will get all the detected parameters in form of a table.



Signal 1

Frequency 1 (GHz)	11.00
Pulse Width 1 (nsec)	500.00
Duty Cycle 1	0.30

Signal 2

Frequency 2(GHz)	13.00
Pulse Width 2 (nsec)	200.00
Duty Cycle 2	0.50

Signal 3

Frequency 3(GHz)	17.00
Pulse Width 3 (nsec)	1000.00
Duty Cycle 3	0.70

Frequency (GHz)	Pulse Duration (usec)	Duty Cycle	Power(dBm)
11.0004	0.4907	0.3043	-40.1758
12.9998	0.1920	0.4815	-40.3961
17.0000	0.9884	0.6978	-40.4140