Theory:

Universal Filtered Multi-Carrier (UFMC) is a multi-carrier modulation technique which divides the available frequency spectrum into smaller subbands, each containing a group of subcarriers. These subbands are individually filtered to reduce out-of-band emissions and improve spectral efficiency. It addresses the limitations of traditional Orthogonal Frequency Division Multiplexing (OFDM) and Filter Bank Multi-Carrier (FBMC) systems.

In OFDM, the spectrum is divided into multiple orthogonal subbands. Here, orthogonal implies that the subbands will be overlapping but there will be no interference produced. This is because the subbands are arranged in such a manner that at any specific time, when one signal reaches its peak, the other signals have zero values. This is shown in figure- 02. From figure- 01, it can be seen that OFDM saves bandwidth due to orthogonal positioning of subbands. A cyclic prefix is added to the beginning of each OFDM symbol to combat some interference. It is a copy of the end of the OFDM symbol and helps in maintaining orthogonality but ends up reducing spectral efficiency.

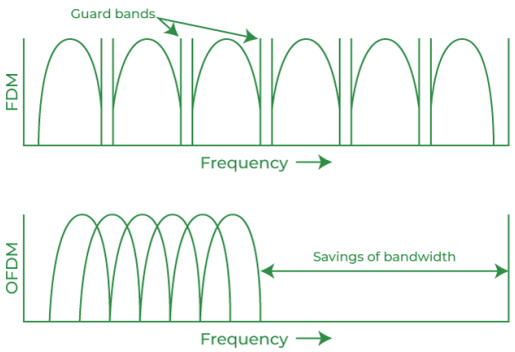

Figure- 01: FDM Vs OFDM

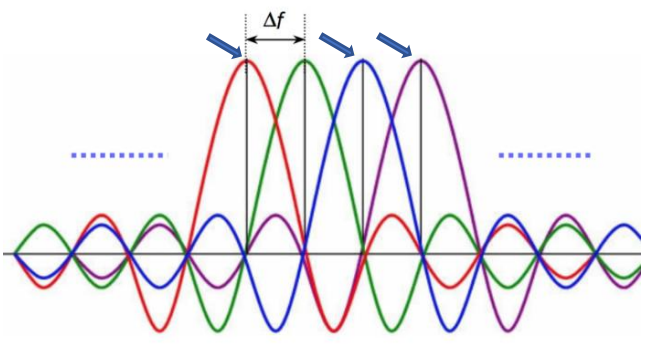

Figure- 02: OFDM frequency spectra

In OFDM the out-of-band emissions remain in the subbands because filtering is applied to the entire band. To resolve this issue, FBMC technique is used which applies filtering to each individual subcarrier. There is no use of cyclic prefix unlike OFDM since we can filter out unnecessary frequencies. But when the number of subcarriers in each band is huge, it is very complex to apply FBMC.

Based on these above techniques and limitations, UFMC is developed. It fulfills the shortcomings of both OFDM and FBMC.

Firstly, the subbands are orthogonally arranged as it is done in OFDM. Then filtering is used. In this case, filter is not applied to individual subcarriers. A group of subcarriers undergoes filtering together, meaning we apply filter to multiple subbands together. This solves the issue of FBMC. Finally, since filtering is used, there is no need to add a cyclic prefix.

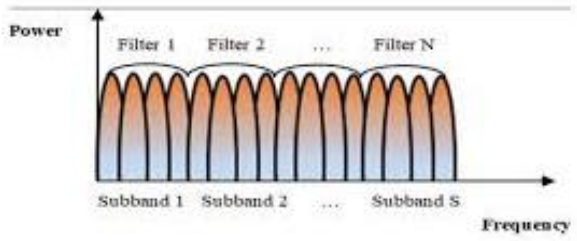

Figure- 03: UFMC Vs OFDM

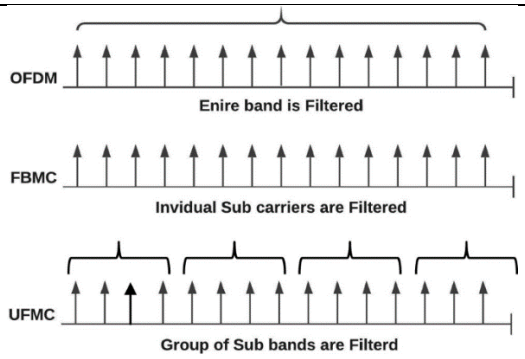

Figure- 04: OFDM vs FBMC Vs UFMC

Steps included in UFMC technique:

The main steps of Universal Filtered Multi-Carrier system are:

1. Data is modulated onto subcarriers within each subband using techniques like Quadrature Amplitude Modulation (QAM).
2. The modulated subcarriers are combined using an IFFT to create a time-domain signal.
3. A filter is applied to each subband to reduce interference between adjacent subbands and minimize out-of-band emissions.
4. The filtered subbands are combined and transmitted over the communication channel.
5. At the receiver end, the signal undergoes the reverse process, including filtering, FFT, and demodulation, to retrieve the original data.

UFMC Transmitter:

The following block diagram shows the transmitter system of UFMC:

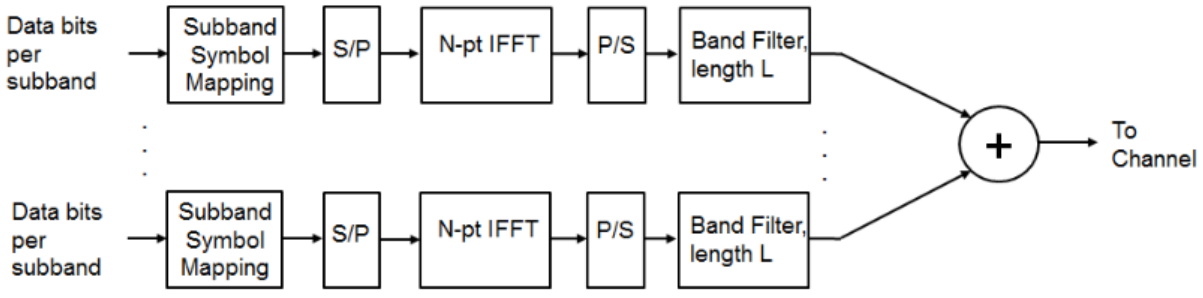

Figure- 05: Block diagram of UFMC transmitter

1. **Input:** Data Bits per Subband: Each subband is a collection of closely spaced subcarriers which contains data bits. The transmitter processes the data bits where multiple subbands are processed in parallel which allows for effective use of frequency spectrum.

2. **Subband Symbol Mapping:** The input data bits are converted to symbols using techniques like QAM (Quadrature Amplitude Modulation), PSK (phase Shift Key) etc. Each symbol consists of multiple bits. Based on the modulation scheme, the binary data bits are mapped to symbols. For example, if QAM is used, then either 4-QAM, 16-QAM, or 64-QAM can be used. After mapping, for each subband, a sequence of modulated symbols is produced.

QAM (Quadrature Amplitude Modulation): QAM consists of two sinusoidal carrier signals. In-phase carrier(I) represents one component of the signal which is a sine wave. While a cosine wave which is 90 degree out-of-phase with I is the Quadrature carrier(Q). QAM combines both amplitude modulation and phase modulation. Then a constellation diagram is used to represent the modulation. The diagram plots all possible values of I and Q components on Cartesian plane, x-axis and y-axis representing I and Q component respectively. Each point in the constellation corresponds to a unique combination of I and Q, which represents a specific bit pattern. For example:

- **4-QAM**: Uses 4 points (2 bits per symbol)
- **16-QAM**: Uses 16 points (4 bits per symbol)
- **64-QAM**: Uses 64 points (6 bits per symbol)

3. **Serial-to-Parallel (S/P) Conversion:** S/P converter is used to convert a serial stream of data into parallel data blocks. This step in necessary for the next step. The IFFT algorithm requires data to be provided in parallel format. When serial stream of modulated symbols arrives at S/P converter, it groups the data into blocks of size N (N= size of IFFT). Each block represents a set of frequency domain data.

4. **N-point IFFT (Inverse Fast Fourier Transform):** IFFT converts frequency domain symbols into time domain symbols. N= size of IFFT determines the resolution of the frequency components.

5. **Parallel-to-Serial (P/S) Conversion:** P/S converter transforms the parallel time domain data back to a serial stream for filtering.

6. **Band Filter (Length L):** Each subband is processed by a bandpass filter. L (filter length) determines how sharp the filter is and affects the performance of the system. It is ensured in this step that the signal for each subband is well-contained within its frequency range.

7. **Summation:** All the filtered outputs are added together to form the final composite signal which is now ready for transmission.

UFMC Receiver:

The receiver works in reverse process of the transmitter.

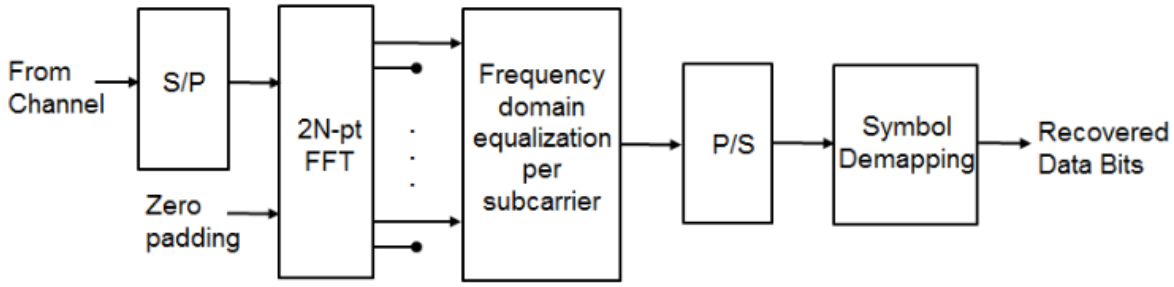

Figure- 06: Block diagram of UFMC receiver

Here, zero padding is necessary because FFT algorithms are most efficient when the input length is a power of 2. It is a technique where additional zeros are added at the beginning or end of a signal. Each subband of subcarriers is processed separately. Zero padding ensures that the subband signal matches the desired FFT size.

Frequency domain equalization is performed to compensate for disturbances introduced by the channel. There is a equalizer which applies a correction factor to each subcarrier hence reducing distortions, attenuation or any fading.

Simulation:

We tried to simulate the transmitter system of UFMC in MATLAB as per the block diagram of transmitter. It is given as follows:

```
clear; clc;
s=rng(211); %To get the same random data each time the code is run

%% Parameters
N = 64;                 % IFFT size
LenFilter = 16;                 % Band filter length
numSubbands = 4;        % Number of subbands
subbandSize = N / numSubbands;       % Number of subcarriers per
subband
modOrder = 16;          % Modulation order

% Adjust the number of symbols per subband to ensure divisibility
numSymbols = ceil(10 / subbandSize) * subbandSize; % Rounded up to fit
subband size
```

```
%% Step 1: Data bits for Each Subband
dataBits = randi([0 1], numSubbands, numSymbols * log2(modOrder)); %
Random data

%% Step 2: Subband Symbol Mapping (16-QAM Modulation)
subbandSymbols = zeros(numSubbands, numSymbols); % Initialize
modulated symbols
for i = 1:numSubbands
    subbandSymbols(i, :) = qammod(dataBits(i, :).', modOrder,
'InputType', 'bit', 'UnitAveragePower', true).';
end

%% Step 3: Serial-to-Parallel (S/P) Conversion
% Reshape symbols into parallel format
s2pData = zeros(numSubbands, subbandSize, numSymbols / subbandSize); %
Parallel data
for i = 1:numSubbands
    s2pData(i, :, :) = reshape(subbandSymbols(i, :), subbandSize, []);
% Reshape into parallel blocks
end

%% Step 4: N-point Inverse Fast Fourier Transform (IFFT)
ifftOutput = zeros(numSubbands, N, size(s2pData, 3)); % IFFT output
for all subbands
for i = 1:numSubbands
    % Pad the parallel subband data to fit the IFFT size
    paddedData = zeros(N, size(s2pData, 3)); % Zero-padding
    startIdx = (i-1) * subbandSize + 1; % Index for subband placement
    paddedData(startIdx:startIdx+subbandSize-1, :) = s2pData(i, :, :);
    % Perform IFFT
    ifftOutput(i, :, :) = ifft(paddedData, N);
end

%% Step 5: Parallel-to-Serial (P/S) Conversion
psData = zeros(numSubbands, N * size(ifftOutput, 3)); % Serialized
IFFT output
for i = 1:numSubbands
    psData(i, :) = reshape(ifftOutput(i, :, :), 1, []); % Flatten IFFT
output into serial form
end

%% Step 6: Bandpass Filtering (Length L)
% Design a band filter (FIR filter)
bandFilter = fir1(LenFilter-1, 1/numSubbands); % Simple FIR filter
filteredOutput = zeros(size(psData)); % Filtered output
for i = 1:numSubbands
    filteredOutput(i, :) = conv(psData(i, :), bandFilter, 'same'); %
Apply filter to each subband
end

%% Step 7: Summation of Subbands
```

```
ufmcSignal = sum(filteredOutput, 1); % Combine all subbands into one
signal

%% Step 8: Plot Results
figure;

% Plot the UFMC signal in the time domain
subplot(3, 1, 1);
plot(real(ufmcSignal));
title('UFMC Signal (Time Domain)');
xlabel('Sample Index'); ylabel('Amplitude');

% Plot the frequency spectrum of the UFMC signal
subplot(3, 1, 2);
plot(abs(fftshift(fft(ufmcSignal))));
title('UFMC Signal (Frequency Domain)');
xlabel('Frequency Bin'); ylabel('Magnitude');

% Plot the first subband's filtered output (adjust for length)
numSamplesToPlot = min(500, length(filteredOutput(1, :))); % Ensure we
don't exceed array bounds
subplot(3, 1, 3);
plot(real(filteredOutput(1, 1:numSamplesToPlot))); % Plot the first
subband's output
title('Filtered Output of First Subband');
xlabel('Sample Index'); ylabel('Amplitude');
```

Code Explanation:

```
clear; clc;
s=rng(211); %To get the same random data each time the code is run
```

The function randi generates random integer values. Setting rng makes sure that the sequence of random numbers generated by randi is the same every time you run your code.

```
4   %% Parameters
5   N = 64;                % IFFT size
6   LenFilter = 16;                % Band filter length
7   numSubbands = 4;       % Number of subbands
8   subbandSize = N / numSubbands;       % Number of subcarriers per subband
9   modOrder = 16;         % Modulation order
10
11  % Adjust the number of symbols per subband to ensure divisibility
12  numSymbols = ceil(10 / subbandSize) * subbandSize; % Rounded up to fit subband size
13
```

In this part of the code, we first initialized the required parameters as

N =64, Lenfilter=16, numSubbands=4, subbandSize=N/numSubbands, modOrder=16.

Then we used ceil function to rounded up the symbol . Using ceil ensures that even if the initial division does not produce an integer, the number of symbols (numSymbols) is rounded up to the nearest multiple of subbandSize. This guarantees that the number of symbols is always a whole number that can be evenly divided by the subband size, which simplifies further operations like reshaping and processing the data in blocks.

```
14  %% Step 1: Data bits for Each Subband
15  dataBits = randi([0 1], numSubbands, numSymbols * log2(modOrder)); % Random data
16
17  %% Step 2: Subband Symbol Mapping (16-QAM Modulation)
18  subbandSymbols = zeros(numSubbands, numSymbols); % Initialize modulated symbols
19  for i = 1:numSubbands
20      subbandSymbols(i, :) = qammod(dataBits(i, :).', modOrder, 'InputType', 'bit', 'UnitAveragePower', true).';
21  end
22
```

The purpose of the first step is to generate random binary data bits for each subband that will be used in the modulation process. Each row in this matrix represents the random binary data for a particular subband. The number of columns is the total number of bits required for the specified number of symbols and modulation order.

For the next step where we will do subband symbol modulation mapping. For this we first need to initialize the subband symbols. Then we will use quadrature amplitude modulation where Each group of 4 bits(symbol) gets mapped to a complex number. To do so we first need to do that transpose of the data since qammod expects column. **qammod** is the function that takes your binary instructions and picks the correct flag for each set of 4 bits. This makes sure the message can be transmitted and understood correctly. Also By using both amplitude and phase to encode data, complex numbers allow modulation schemes to pack more information into a given bandwidth.

```
23  %% Step 3: Serial-to-Parallel (S/P) Conversion
24  % Reshape symbols into parallel format
25  s2pData = zeros(numSubbands, subbandSize, numSymbols / subbandSize); % Parallel data
26  for i = 1:numSubbands
27      s2pData(i, :, :) = reshape(subbandSymbols(i, :), subbandSize, []); % Reshape into parallel blocks
28  end
29
```

At first in this step we initialized the s2pdata parameter as a 3-dimensional matrix of zeros. Then we iterates the loop for i=1: numsubbands. Subbandsymbols(1,:) Selects the row corresponding to the i-th subband. And then the function reshape reshapes the selected row in a 2 dimensional array with subbandSize rows. By carefully restructuring the data into parallel blocks, this step ensures that each subband's symbols are prepared for the Inverse Fast Fourier Transform (IFFT) and other signal processing tasks, maintaining the necessary alignment and structure.

```
%% Step 4: N-point Inverse Fast Fourier Transform (IFFT)
ifftOutput = zeros(numSubbands, N, size(s2pData, 3)); % IFFT output for all subbands
for i = 1:numSubbands
    % Pad the parallel subband data to fit the IFFT size
    paddedData = zeros(N, size(s2pData, 3)); % Zero-padding
    startIdx = (i-1) * subbandSize + 1; % Index for subband placement
    paddedData(startIdx:startIdx+subbandSize-1, :) = s2pData(i, :, :);
    % Perform IFFT
    ifftOutput(i, :, :) = ifft(paddedData, N);
end

```

In this part pf code, we are performing the Inverse Fast Fourier Transform (IFFT) on the data from multiple subbands, padding them to fit the IFFT size of N. First, we initialize ifftOutput, a three-dimensional array, to store the IFFT results for each subband. For each subband, represented by the index i, we create a zero-padded array paddedData with dimensions N by the size of the third dimension of s2pdata, ensuring that the data fits the IFFT size. The subband data from s2pdata is placed into the appropriate section of paddedData starting at startindex calculated as (i-1) *subbandsize+1; This process aligns the subband data correctly within the larger padded array. Finally, we perform the IFFT on paddedData and store the resulting time-domain signal in ifftoutput, for each subband. This transformation prepares the data for subsequent stages of processing or transmission.

```
%% Step 5: Parallel-to-Serial (P/S) Conversion
psData = zeros(numSubbands, N * size(ifftOutput, 3)); % Serialized IFFT output
for i = 1:numSubbands
    psData(i, :) = reshape(ifftOutput(i, :, :), 1, []); % Flatten IFFT output into serial form
end

%% Step 6: Bandpass Filtering (Length L)
% Design a band filter (FIR filter)
bandFilter = fir1(LenFilter-1, 1/numSubbands); % Simple FIR filter
filteredOutput = zeros(size(psData)); % Filtered output
for i = 1:numSubbands
    filteredOutput(i, :) = conv(psData(i, :), bandFilter, 'same'); % Apply filter to each subband
end

```

The goal is to process the output of an Inverse Fast Fourier Transform (IFFT) through parallel-to-serial conversion and bandpass filtering. In Step 5, we first initialize psData, a matrix sized to hold serialized IFFT output for all subbands. The loop iterates over each subband, flattening the IFFT output matrix ifftOutput for each subband into a single row using the reshape function, thus converting parallel data to serial form. This reorganization makes the data easier to handle for subsequent processing steps. In Step 6, a bandpass filter is designed using the fir1 function, which creates a simple FIR filter with length LenFilter-1 and a cutoff frequency of 1/numSubbands. This filter helps isolate specific frequency bands for each subband. Another loop then initializes filteredOutput, a matrix to store the filtered data. For each subband, the serialized data from psData is convolved with the designed FIR filter using the conv function with the 'same' option, ensuring the output remains the same length as the input. This filtering step refines the signal, preparing it for further analysis or transmission by attenuating unwanted frequencies.

```
%% Step 7: Summation of Subbands
ufmcSignal = sum(filteredOutput, 1); % Combine all subbands into one signal

%% Step 8: Plot Results
figure;

% Plot the UFMC signal in the time domain
subplot(3, 1, 1);
plot(real(ufmcSignal));
title('UFMC Signal (Time Domain)');
xlabel('Sample Index'); ylabel('Amplitude');

% Plot the frequency spectrum of the UFMC signal
subplot(3, 1, 2);
plot(abs(fftshift(fft(ufmcSignal))));
title('UFMC Signal (Frequency Domain)');
xlabel('Frequency Bin'); ylabel('Magnitude');

% Plot the first subband's filtered output (adjust for length)
numSamplesToPlot = min(500, length(filteredOutput(1, :))); % Ensure we don't exceed array bounds
subplot(3, 1, 3);
plot(real(filteredOutput(1, 1:numSamplesToPlot))); % Plot the first subband's output
title('Filtered Output of First Subband');
xlabel('Sample Index'); ylabel('Amplitude');
```

In Step 7, the code combines all filtered subband signals into one unified signal by summing them along the first dimension, resulting in ufmcSignal. This step consolidates the contributions from each subband into a single signal ready for transmission or analysis. In Step 8, the code generates a figure with three subplots to visualize the processed signal. The first subplot plots the real part of the ufmcSignal in the time domain, showing how the signal varies over time. The second subplot displays the frequency spectrum of ufmcSignal by computing its FFT, then using fftshift to center the zero-frequency component, providing insights into the signal's frequency content. The third subplot plots the real part of the first subband's filtered output, adjusted to plot up to 500 samples, to ensure the data's length is within bounds. This step gives a clear view of the filtered subband's time-domain signal, helping in verifying the filtering process. Together, these steps provide a comprehensive view of the signal both in time and frequency domains, ensuring the processed signal is correctly formed and ready for subsequent use.

Effect of different parameters on the output:

IFFT size: Larger N (IFFT size) increases the frequency resolution meaning each subcarrier occupies a smaller bandwidth. The time-domain signal also becomes smoother while the subbands of frequency-domain are better separated. The opposite effect is seen if N is reduced.

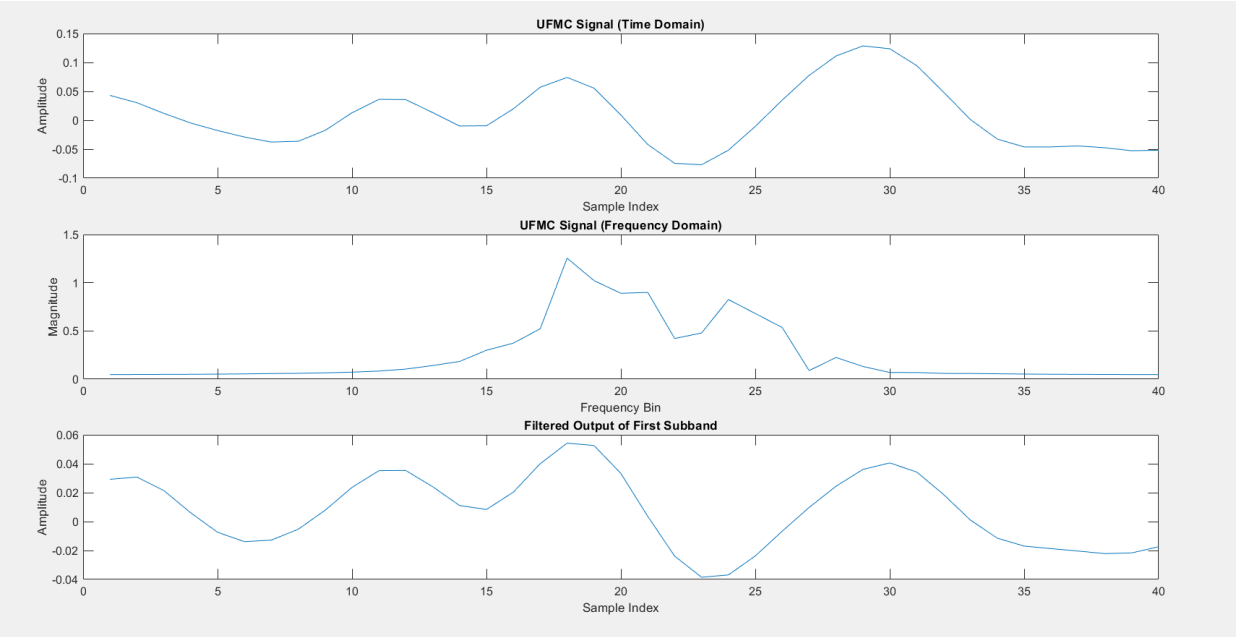

Figure- 07: IFFT size= 40

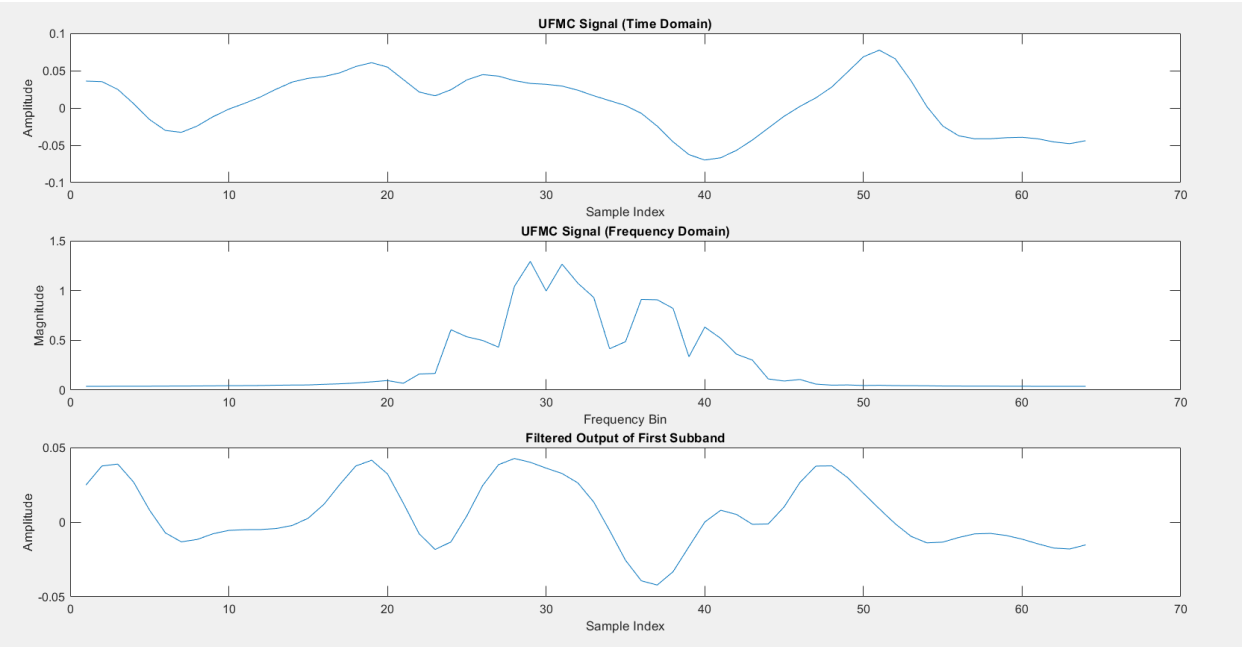

Figure- 08: IFFT size= 64

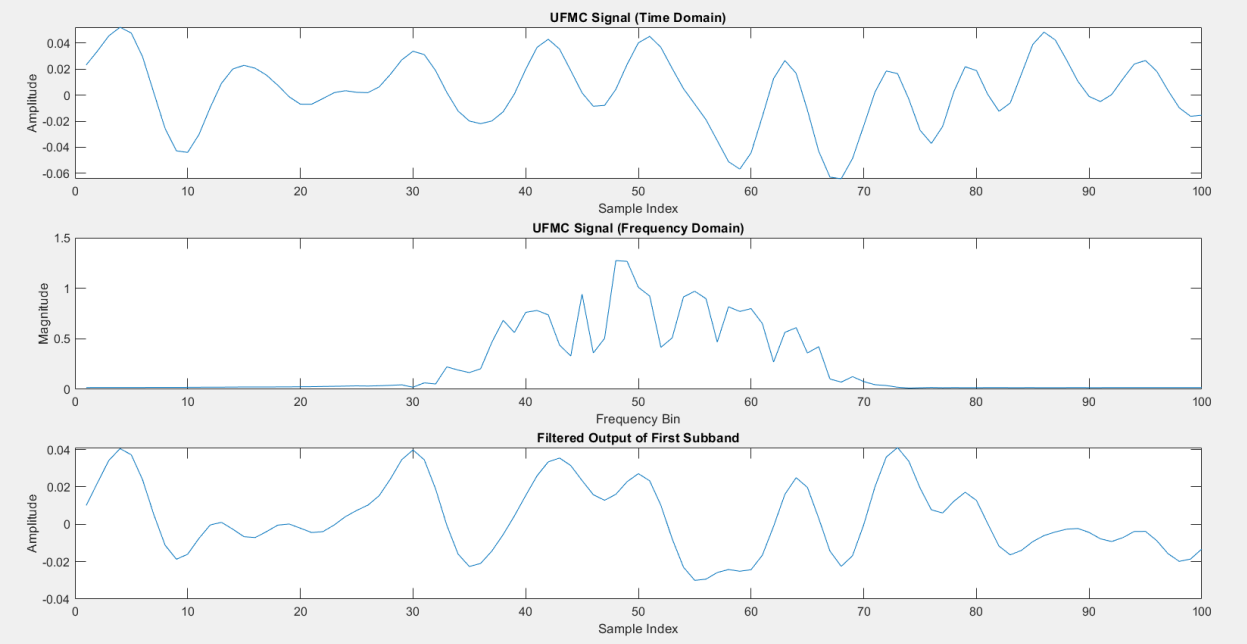

Figure- 09: IFFT size= 100

Band filter length: Longer filters provides precise subband but increase computational time. They cut the frequency portions smoothly. Shorter subbands overlap more creating interference and make the signal noisier.

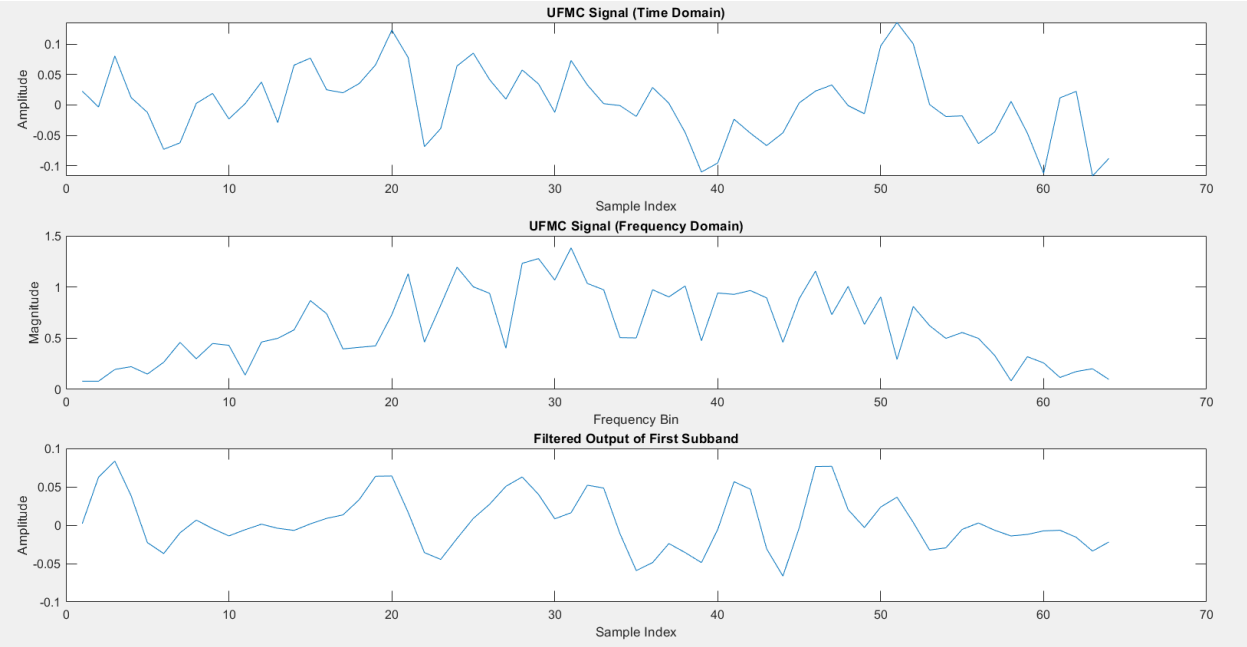

Figure- 10: Filter length= 2

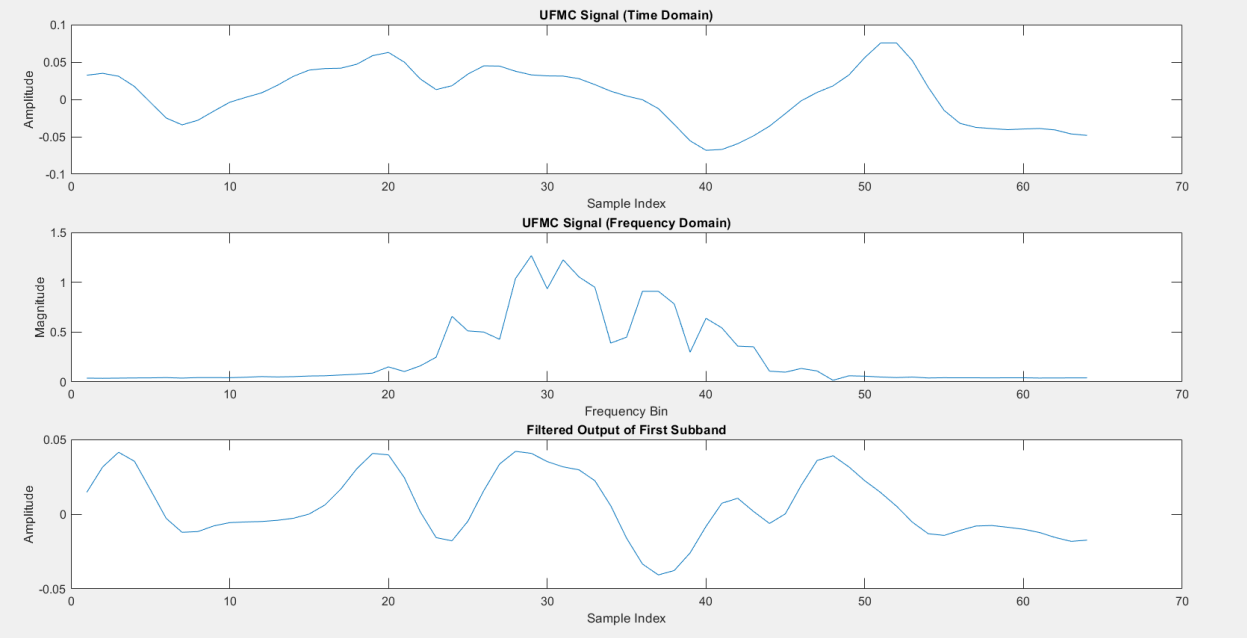

Figure- 11: Filter length= 13

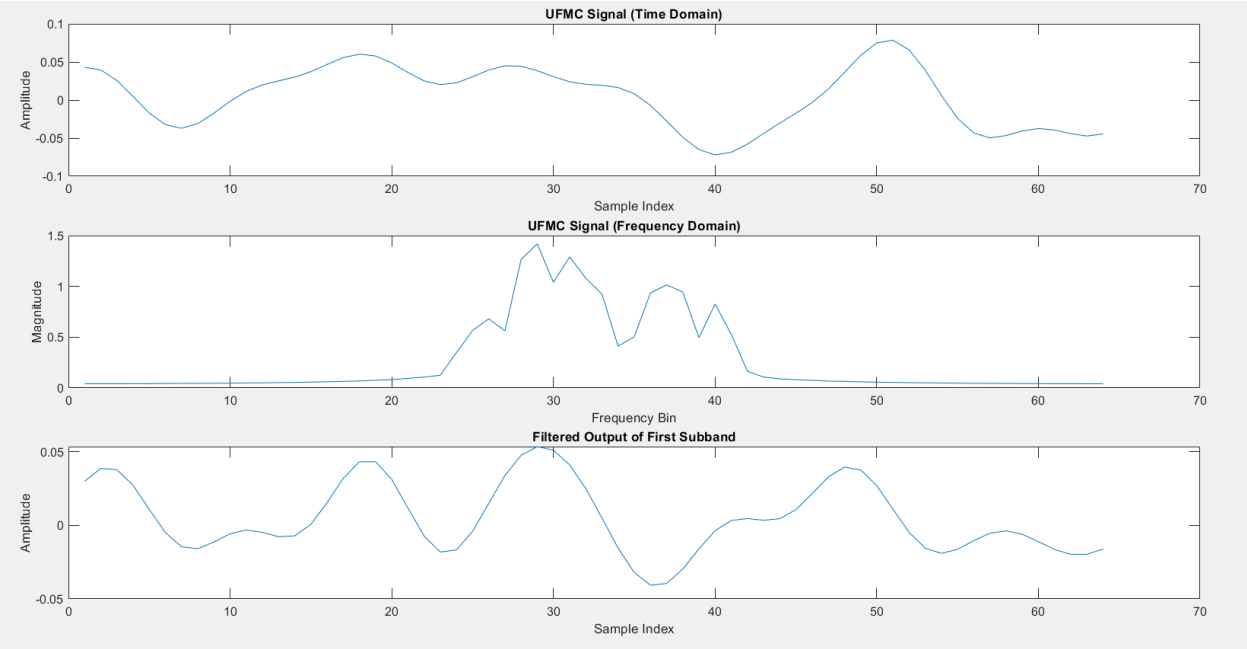

Figure- 12: Filter length= 50

Number of subbands: With more subbands, the signal becomes more complex. So, for a simpler waveform fewer subbands are preferable. Fewer subbands can also create more sparse spectrum whereas more subbands produces densely packed spectrum and has higher number of peaks.

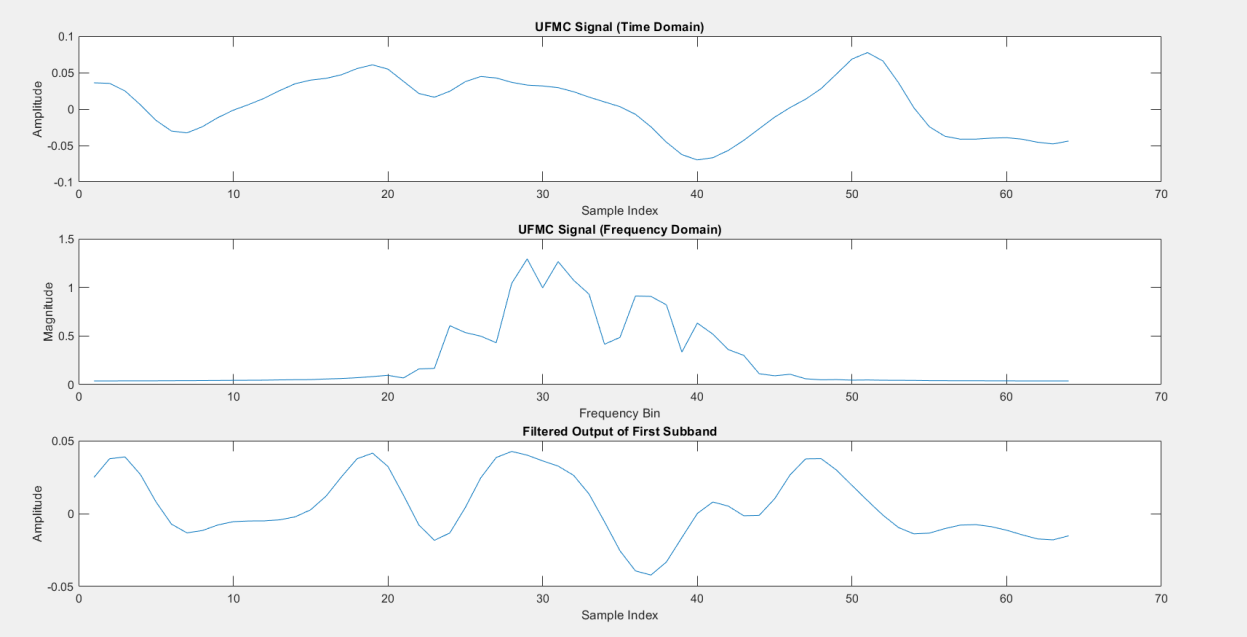

Figure- 13: Number of Subbands = 4

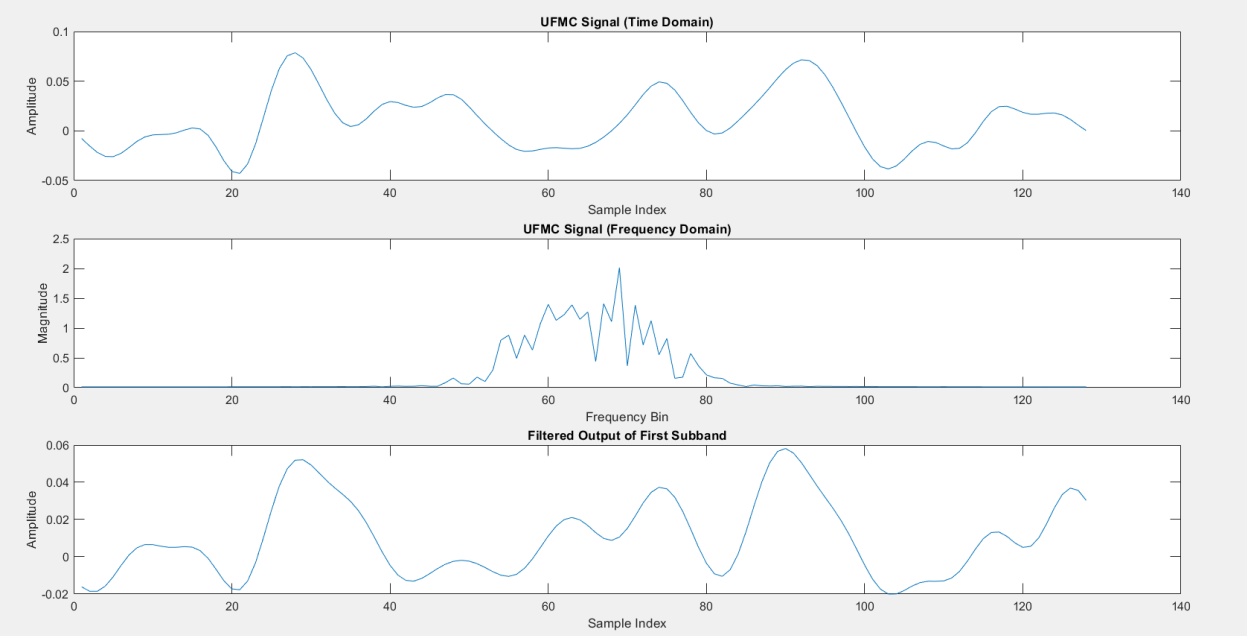

Figure- 14: Number of Subbands = 8

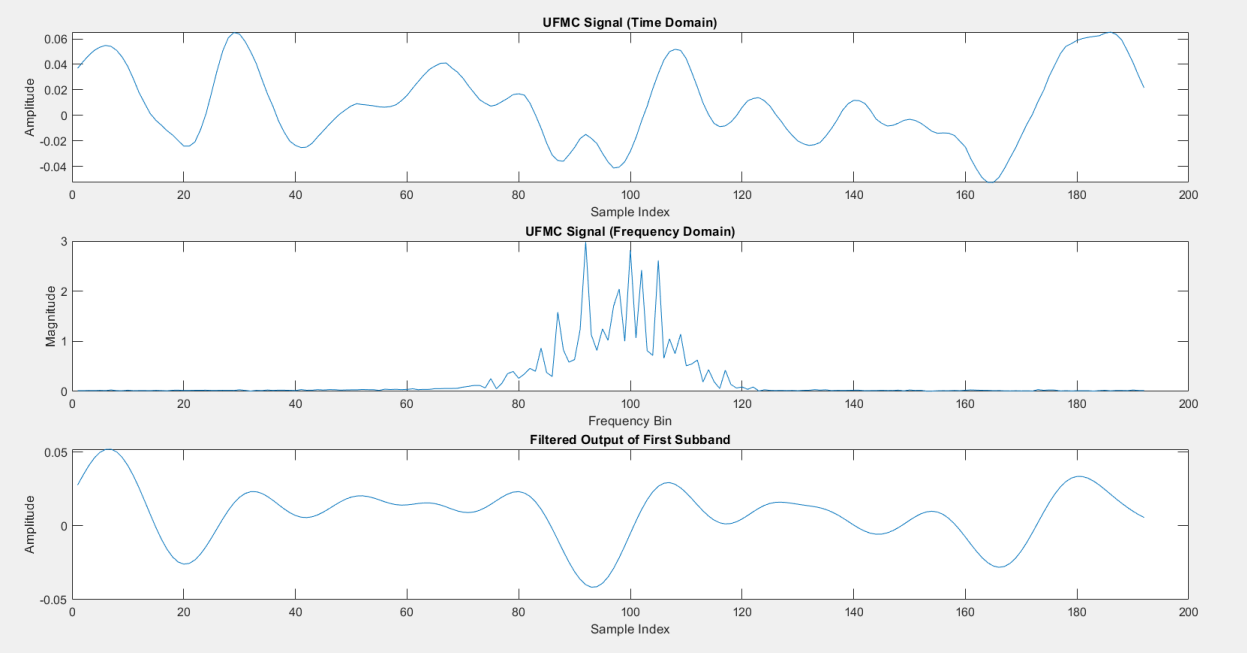

Figure- 15: Number of Subbands = 16

Discussion:

Since UFMC is developed based on OFDM and FBMC, we first tried to clear our basics of these two topics. We learned the shortcomings of both the techniques and realized how UFMC fulfills those. Both transmitter and receiver block diagrams of UFMC were obtained. Following the diagram of transmitter system, and with a lot of help from the AI, we were able to generate a code in MATLAB representing the transmission system. We faced a lot of issues while generating the code since there were very little resources available on the Internet on Universal Filtered Multi-carrier. We tried changing codes resolving errors for a long time. Finally, we were successful in developing a transmitter system. The next issue we faced was to understand the code. So, we took help from the AI again and tried to understand the code line by line as much as possible. We obtained three graphs as the output: first one shows the UFMC signal in time-domain, second one in frequency-domain and the third one is the filtered output of first subband. After that, we tried changing some parameters like the IFFT size, filter length, number of subbands etc. to analyze the changes in the output graphs. Since the output matched with our theoretical knowledge, we may say that it was a successful project for us.

Conclusion:

Throughout this project, we learned how to generate a UFMC transmitter system. We implemented our theoretical understanding from the block diagram to generate a MATLAB code which represented the desired system. Finally, we changed different parameters and visualized the effects of the changes on the output graph. The outcome matched with our theoretical understanding.

Reference:

https://www.google.com/url?sa=i&url=https%3A%2F%2Fwww.geeksforgeeks.org%2Forthogonal-frequency-division-multiplexing-ofdm%2F&psig=AOvVaw2_1GaEF4V-7BJMiZIhVyyU&ust=1733086826940000&source=images&cd=vfe&opi=89978449&ved=0CBQQjRxqFwoTCJjphNj5hIoDFQAAAAAdAAAAABAE

https://www.google.com/url?sa=i&url=https%3A%2F%2Fwww.linkedin.com%2Fpulse%2Fofdm-orthogonal-frequency-division-multiplexing-fpgas-juan-abelaira&psig=AOvVaw2_1GaEF4V-7BJMiZIhVyyU&ust=1733086826940000&source=images&cd=vfe&opi=89978449&ved=0CBQQjRxqFwoTCJjphNj5hIoDFQAAAAAdAAAAABAJ

https://www.google.com/url?sa=i&url=https%3A%2F%2Flink.springer.com%2Fchapter%2F10.1007%2F978-981-16-6624-7_35&psig=AOvVaw18oO7qEU_ZTuMt2xM8FBlF&ust=1733086872079000&source=images&cd=vfe&opi=89978449&ved=0CBQQjRxqFwoTCNjvnPL5hIoDFQAAAAAdAAAAABAE

https://www.google.com/url?sa=i&url=https%3A%2F%2Fwww.youtube.com%2Fwatch%3Fv%3DzhMST8bX_fY&psig=AOvVaw1bSqVRdSNADVqs25LSkkCa&ust=1733086919182000&source=images&cd=vfe&opi=89978449&ved=0CBQQjRxqFwoTCJCr24P6hIoDFQAAAAAdAAAAABAE

https://www.mathworks.com/help/comm/ug/ufmc-vs-ofdm-modulation.html

chatgpt

copilot