Lab 3a - OFDM Simulation

Pranavi Boyalakuntla
Principles of Wireless
Spring 2021

I. Introduction

One of the downsides to a single carrier communication system is that the symbol period must be high in order to avoid inter-symbol interference (ISI). When the symbol period is higher, the data rate of the system is lower.

One way of increasing data rate is using a multiple carrier communication system. Using more than one carrier allows for information to be transmitted at multiple frequency intervals at the same time. In order to ensure there is no intercarrier interference (ICI), guard bands, or unused bandwidth between channels, are used. This partitioning scheme is called Frequency Division Multiplexing (FDM). Fig. 1 shows a multiple carrier communication system that is using FDM.

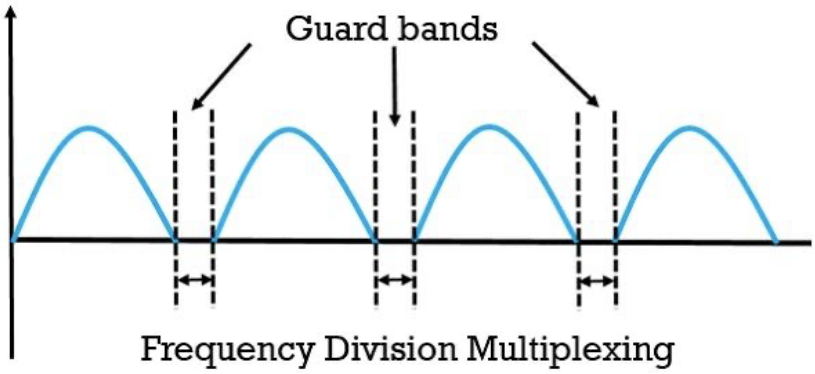

Fig. 1: A multiple carrier communications system that utilizes guard bands.

The use of guard bands requires the available bandwidth to be greater than the required bandwidth. In the case where bandwidth is more limited, orthogonal frequency division multiplexing (OFDM) can be used.

OFDM utilizes the concept of orthogonality to allow channels to overlap while minimizing ICI. When two channels are orthogonal to one another their zero crossings align and the peak of one channel does not interfere with the peak of another channel. In Fig. 2, each channel is represented by a sinc function. All of the zero crossings align which means that the channels are orthogonal. The peak of one channel also exists over lower values of other channels. Although these channels overlap, the ICI is minimized by maintaining orthogonality.

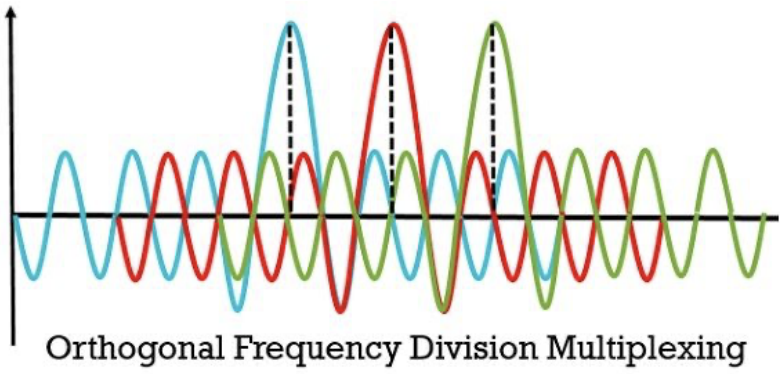

Fig. 2: Orthogonal channels that overlap.

OFDM uses the available bandwidth more efficiently than FDM as there are no unused portions of the spectrum. As a result, OFDM is used in popular communication systems like WiFi and LTE.

The goal of this project is to implement an OFDM system on an SDR platform. This paper will cover the first step of that: simulating an OFDM system with synchronized clocks in MATLAB.

II. Overview of OFDM

A. Flat-Fading Assumption

The frequency flat-fading assumption means that the channel's transfer function can be assumed to be constant over a frequency band. This is advantageous because all frequencies can be demodulated in the same way. Most channels do not allow the flat-fading assumption to be made across the entire channel. However, when looking at a smaller frequency band, the channel effects vary in smaller degrees . This allows the flat-fading assumption to hold true. In Fig. 3, it is shown how given any channel response, flat-fading can be assumed if it is chunked into smaller portions. The x axis is frequencies and the y axis is signal amplitude.

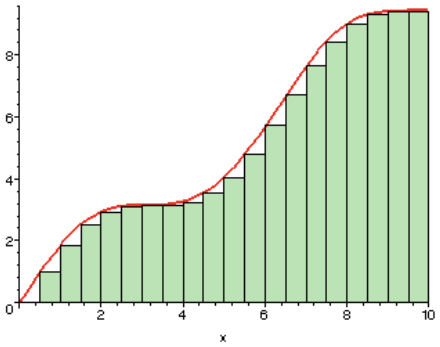

Fig. 3: The frequency response of a channel chunked into smaller portions to uphold the flat-fading assumption.

B. The Discrete Fourier Transform

Any information in the frequency domain has to be converted to the time domain before transmission. This can be done using the Fourier transform.

The Fourier transform is a mathematical operation that represents a time domain signal in the frequency domain. It does this by breaking a signal into its frequency components. The Fourier transform is closely related to the Fourier series which is the expansion of a periodic function into multiple frequencies.

In fact, the Fourier transform is the limit of the Fourier series as $T \rightarrow \infty$. The definition of the Fourier series is shown in Eq. 1 where ω_0 is the fundamental frequency and t is time.

$$x(t) = \sum_{n=-\infty}^{+\infty} c_n e^{jn\omega_o t}$$
$$\text{where } c_n = \frac{1}{T}\int_T x(t)e^{-jn\omega_0 t}dt \tag{1}$$

Eq. 1 can be rearranged into Eq. 2 as shown below.

$$Tc_n = \int_T x(t)e^{-jn\omega_0 t}dt \tag{2}$$

As $T \to \infty$, $\omega_0 \to 0$ and $n\omega_0$ becomes a constant quantity equal to ω. By substituting a few values, the definition of the Fourier transform can be derived as shown in Eq. 3. It can be seen that the Fourier transform of a continuous signal in time will be continous in frequency as well.

$$X(\omega) = \int_{-\infty}^{+\infty} x(t)e^{-j\omega t}dt \tag{3}$$

Conversely, the Inverse Fourier transform of a continuous signal in the frequency domain will be continuous in time. However, when implemented in hardware, continuous signals are sampled and sent discretely in time. The Discrete Time Fourier transform (DTFT) of a discrete-time signal is continuous in the frequency domain. The definition of the DTFT is shown in Eq. 4. Comparing Eq. 4 with Eq. 3 shows that the DTFT is a time discrete version of the Fourier Transform.

$$X\left(e^{j\Omega}\right) = \sum_{\ell=-\infty}^{\infty} x[\ell]e^{-j\Omega} = \sum_{\ell=0}^{L-1} x[\ell]e^{-j\Omega} \tag{4}$$

In the case of OFDM, the frequency domain needs to be chunked to uphold the flat-fading assumption. As a result, the signal will be discrete in the frequency domain. The Inverse Discrete Fourier Transform (IDFT) is a mathematical operation that will take a discrete signal in the frequency domain and return a discrete signal in the time domain. The L-point IDFT is defined in Eq. 5.

$$x[m] = \frac{1}{L}\sum_{k=0}^{L-1} X_k e^{2\pi j\frac{mk}{L}} \tag{5}$$

Conversely, the Discrete Fourier Transform (DFT) takes a discrete signal in the time domain and converts it to a discrete signal in the frequency domain. The definition of the DFT is shown in Eq. 6. Comparing Eq. 6 to Eq. 4 shows that the DFT is a sampled version of the DTFT.

$$X_k = \sum_{\ell=0}^{L-1} x[\ell]e^{-2\pi j\frac{\ell k}{L}} \tag{6}$$

In using the DFT and IDFT in the OFDM algorithm, there are several properties that will be helpful.

- **Linearity of the DFT:** The additive and scaling relationships between signals are maintained after the DFT is taken.
- **Periodicity of the DFT:** The L-point DFT is periodic with period L.
- **Circular Convolution in Time Become Multiplication in Frequency: T**he relationship between circular convolution and multiplication forms the basis of OFDM.

C. Mechanics Behind OFDM

Information in the OFDM algorithm begins in the frequency domain. It is important to note that this information in the frequency domain represents a snapshot of what is being sent across multiple channels in one unit of time.

In order to send this information, it needs to be converted to the time domain using the IDFT. The DFT and IDFT hold the property that circular convolution in the time domain becomes multiplication in the frequency domain. However, when sending a signal through the air, it is convolved with the channel impulse response. In order to maintain that transmitting a signal over the air is equivalent to multiplication it in the frequency domain with the channel frequency response, the system needs to be "tricked" into performing circular convolution.

Circular convolution is a special case of periodic convolution where two periodic functions with the same period are convolved. To imitate periodicity, the signal can be prepended and postpended with itself ad infinitum. This will then "trick" the air into performing circular convolution instead of regular convolution.

To bring the signal back into the frequency domain, the DFT is taken.

Now, the signal needs to be processed back into a guess of what was originally sent, $\hat{X_k}$. Eq. 7 describes the relationship between the received signal in the frequency domain, Y_k, the channel response, H_k, and the sent signal in the frequency domain, X_k.

$$Y_k = H_k X_k \tag{7}$$

Rearranging Eq. 7 results in Eq. 8 which solves for the guess of what was originally sent, X_k.

$$\hat{X_k} = \frac{Y_k}{H_k} \tag{8}$$

The data rate of the method described is equal to $\frac{blockSize}{\infty}$ because ∞ bits need to be sent to transmit one block. This issue can be circumvented using the properties of convolution. Prepending each block with a cyclic prefix 25% the size of it doesn't include the portions of the signal where the estimated channel convolution has diminished or negligible effects. Note that the 25% cyclic prefix length is a rule of thumb and is not always the case. The cyclic prefix length depends on the channel impulse response length.

The cyclic prefix also helps eliminate ISI as it acts as a guard period. This is because any ISI that would have affected the beginning of the next symbol is now affecting the cyclic prefix. In addition, Fig. 4 shows how the cyclic prefix acts as a guard against time offsets of when the receiver will start capturing the block. This plus point to the cyclic prefix is less relevant to this simulation, but is something that needs to be thought about in hardware.

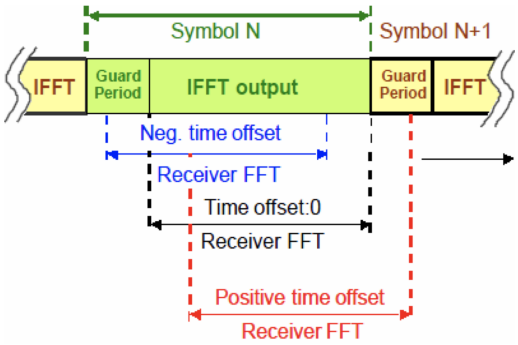

Fig. 4: The cyclic prefix guards helps handle uncertainty of when the receiver will capture the starting point of the block.

III. Implementation

The implementation described in this report uses a block length of 64 channels. As discussed above, the cyclic prefix length will be 25% of the block length or 16 bits. The high level overview of the software implementation is shown in Fig. 5.

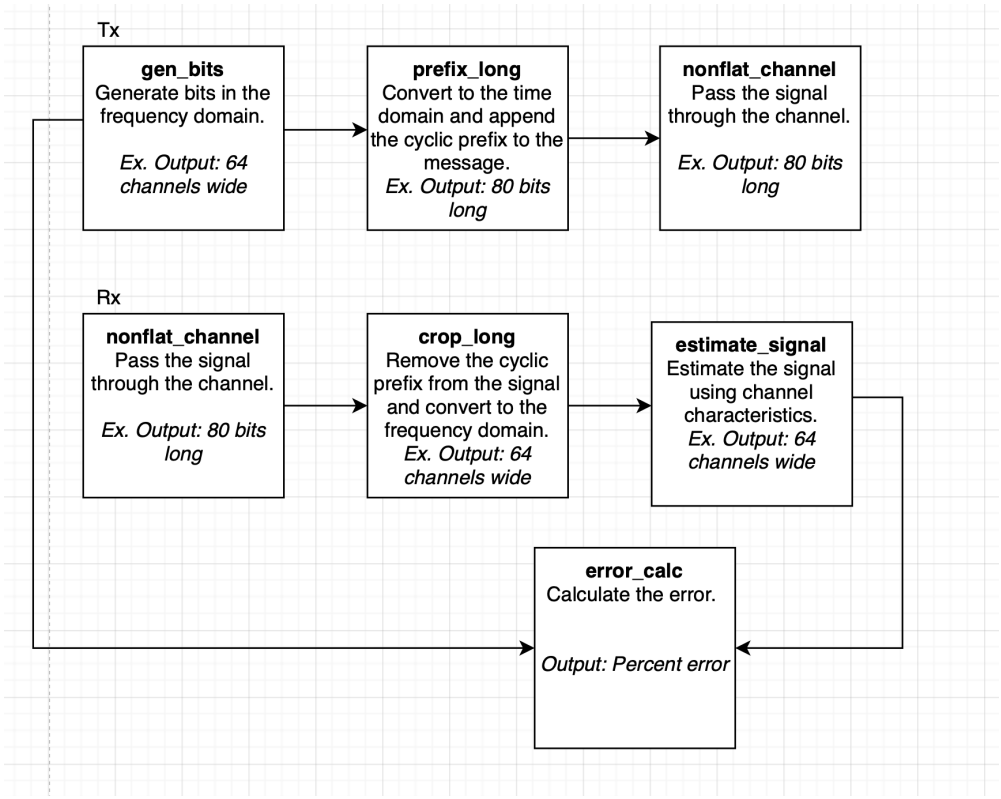

Fig. 5: High level software implementation overview

The first step is generating the data at each of the channels in the frequency domain. The block size is 64, so there will be 64 channels to transmit data across. It is important to remember that the data is in the frequency domain at this point. An example of three generated blocks is shown in Fig. 6.

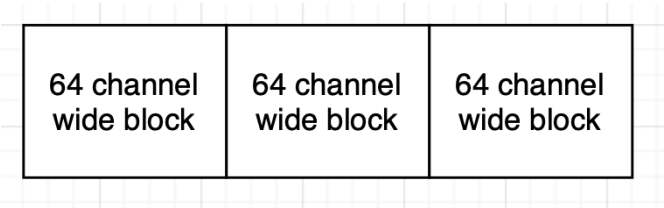

Fig. 6: Generation structure of three 64 channel wide blocks.

This information is then converted to the time domain using the IDFT and the cyclic prefix is prepended to each block. Fig. 7 shows the structure of the data after the cyclic prefixes are prepended. Each prefix is unique to each block and is a copy of the last 16 bits.

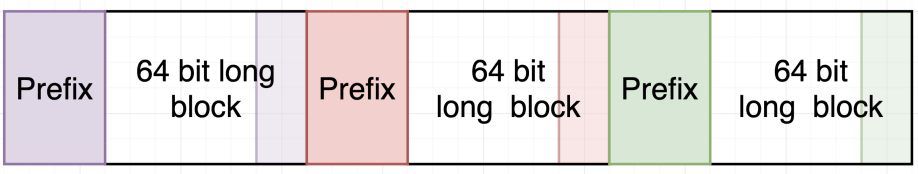

Fig. 7: Data structure after the cyclic prefixes are prepended.

This data is sent through the channel and the cyclic prefixes are removed. Fig. 8 shows how the data might look at this point.

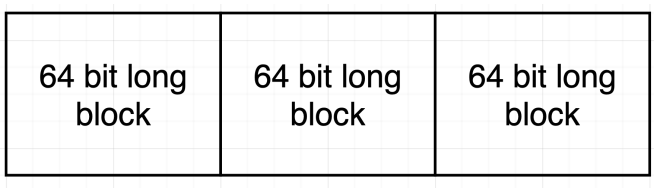

Fig. 8: Example of three blocks of data once cyclic prefixes are removed.

To convert the data back into the frequency domain, the DFT of the data is taken. Although the data at this point is in the frequency domain and looks like Fig. 6, it is not an estimate of what was originally sent.

The blocks are divided by the estimated channel response to receive $\hat{X}_k$ in accordance with Eq. 7. Fig. 9 illustrates how this division process is implemented with a 64-channel wide H matrix across multiple blocks.

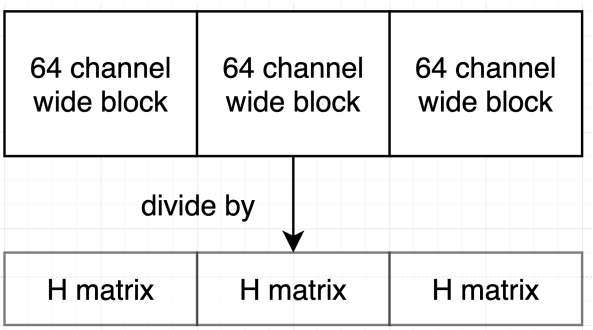

Fig. 9: Dividing a 3 block wide signal with the estimated channel matrix repeated 3 times.

A. Channel Estimation

In order to implement the steps explained above, H_k, or the channel estimation matrix needs to be calculated. This can be done by:

- sending multiple blocks of a known signals across the channel.
- dividing the received signal by the sent signal using Eqn. 7.
- averaging the channel estimation responses for each of the blocks.

The signals are sent across the air using the same OFDM method described above.

IV. Results

Fig. 10 shows the received signal in the frequency domain after the cyclic prefixes were removed and the DFT was computed. This graph only displays one block for understanding. The channel effects can be seen here as not flat-fading; some frequencies experience greater fading than others.

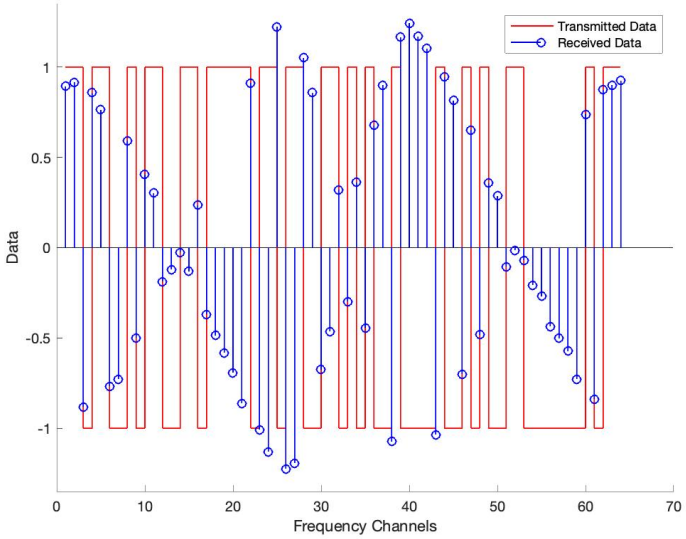

Fig. 10: Received data once cyclic prefixes were removed and the transmitted data plotted in the frequency domain

Fig. 11 shows the received estimate solved by dividing the received data shown in Fig. 10 with the estimated channel matrix as described in Eqn. 8. This graph only displays one block for understanding. There is some error as can be seen by the changes in color.

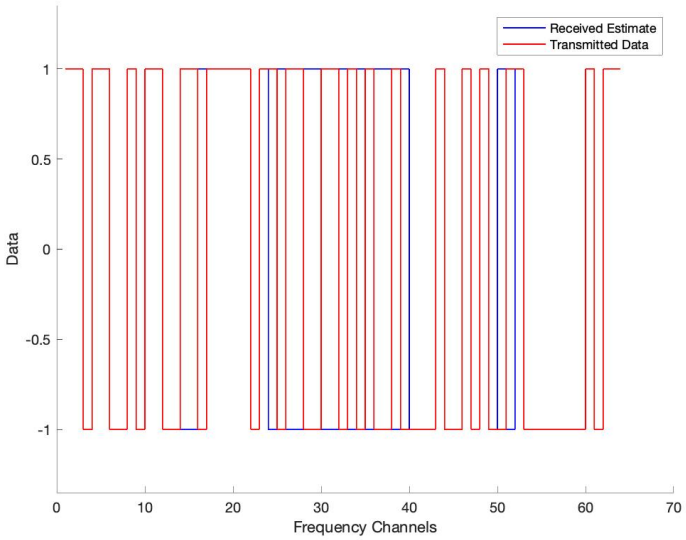

Fig. 11: Received estimate calculated using the channel matrix and the transmitted data plotted in the frequency domain

The final bit error rate (BER) for this implementation was 0.31 or 31%.

V. Discussion and Next Steps

Here are some potential causes for the high BER:

- Incorrect channel estimation: method of averaging and reshaping the matrix is implemented incorrectly.
- Large complex components once the received data is processed

- Error computation is incorrect

VI. Resources

https://digitalcommons.calpoly.edu/cgi/viewcontent.cgi?article=1063&context=eesp

https://techdifferences.com/difference-between-fdm-and-ofdm.html

VII. Appendix

The code is located at: https://github.com/naviatolin/PoW-Labs/tree/master/Lab_3A

A. Main OFDM Implementation

```
%% Clear
clear all;

%% Estimate the channel
block_len = 64;
prefix_len = 16;
block_num = 2;

% generate channels in the frequency domain
tx_train = gen_data(block_num, block_len);

% take the ifft and add the cyclic prefix to the signal
prefixed_train = prefix_long(tx_train, block_len, prefix_len, block_num);

% pass the signal through the channel
rx_train = nonflat_channel(prefixed_train);

% find the delay
delay = find_delay(rx_train, prefixed_train);

% apply the delay
rx_delay = rx_train(delay:end);

% crop the signal by the prefix length
rx_hat = crop_long(rx_delay, block_len, prefix_len, block_num);

% estimate channel
h_multiple_run = rx_hat./tx_train;
h_stacked = reshape(h_multiple_run, [block_num, block_len]);
H = mean(h_stacked);

%% Clear all but channel estimate
clearvars -except H

%% OFDM Process
block_len = 64;
prefix_len = 16;
block_num = 1000;

% generate channels in the frequency domain
tx = gen_data(block_num, block_len);

% add the cyclic prefix to the signal
prefixed_tx = prefix_long(tx, block_len, prefix_len, block_num);

% pass the signal through the channel
rx = nonflat_channel(prefixed_tx);

% find the delay
delay = find_delay(rx, prefixed_tx);

% apply the delay
rx_delay = rx(delay:end);

% crop the signal by the prefix length
rx_data = crop_long(rx_delay, block_len, prefix_len, block_num);
```

```
% solve for x_hat
X = estimate_signal(H, rx_data, block_num);

X_hat = sign(real(X));


% Compute the error.
error = compute_error(X_hat, tx)

%%
figure
hold on
stem(tx(1:64), 'LineWidth', 1, 'Color', 'red')
stem(rx_data(1:64), 'LineWidth', 1, 'Color', 'blue')
ylim([-1.35 1.35])
xlabel('Frequency Channels')
ylabel('Data')
legend('Transmitted Data', 'Received Data');
hold off

figure
hold on
stairs(X_hat(1:64), 'LineWidth', 1,'Color', 'blue')
stairs(tx(1:64), 'LineWidth', 1,'Color', 'red')
ylim([-1.35 1.35])
xlabel('Frequency Channels')
ylabel('Data')
legend('Received Estimate', 'Transmitted Data');
hold off
```

B. Helper Functions

```
function freq_signal = time_to_freq(time_signal)
    freq_signal = fft(time_signal);
end
```

```
function [ y ] = nonflat_channel( x )

SNR_dB = 30;  %nominal SNR in dB

SNR = 10^(SNR_dB/10); % nominal SNR

% make an impulse response
% this is done by upsampling and interpolating several
% discrete points
% the impulse response is shifted in time to insert a delay
%
tmp = [0 -0.1 1 -0.1 0.05 -0.01 0 0 0 0 ];
tmp = resample(tmp, 10,9);
h = zeros(64,1);
h(8:8+length(tmp)-1) = tmp;

% convolve input signal with channel impulse response
y = conv(x,h);

% compute appropriate noise variance to achive SNR
noise_var = var(y)/SNR;

% add noise of the appropriate variance
y = y + sqrt(noise_var/2)*(randn(size(y)) + 1i*randn(size(y)));


end
```

```
function data = gen_data(block_num, block_len)
    data_len = block_num * block_len;
    data = (round(rand(1,data_len)) * 2) - 1;
end
```

```
function time_signal = freq_to_time(freq_signal)
    time_signal = ifft(freq_signal);
end
```

```
function [delay, lags, cor] = find_delay(y, x)
    [cor, lags] = xcorr(y, x);
    [~,index]=max(cor);
    delay = lags(index);
end
```

```
function X_hat = estimate_signal(channel, signal, block_num)
    repeated_channel = repmat(channel, 1, block_num);
    X_hat = signal./repeated_channel;
end
```

```
function cropped = crop_signal(data, prefix_length, block_len)
    starting = prefix_length + 1;
    cropped = data(starting:end);
end
```

```
function data = crop_long(signal, block_len, prefix_len, block_number)
    data = [];
    chunk_len = block_len + prefix_len;
    for i = 1 : chunk_len : ((block_number * chunk_len) - (chunk_len - 1))
        ending = i+((block_len + prefix_len) - 1);
        portion = signal(i:ending);
        cropped = crop_signal(portion, prefix_len, block_len);
        time = time_to_freq(cropped);
        data = [data time];
    end
end
```

```
function data = crop_long(signal, block_len, prefix_len, block_number)
    data = [];
    chunk_len = block_len + prefix_len;
    for i = 1 : chunk_len : ((block_number * chunk_len) - (chunk_len - 1))
        ending = i+((block_len + prefix_len) - 1);
        portion = signal(i:ending);
        cropped = crop_signal(portion, prefix_len, block_len);
        time = time_to_freq(cropped);
        data = [data time];
    end
end
```

```
function output = add_cyclic_prefix(signal, prefix_length)
    delta = prefix_length - 1;
    starting = length(signal) - delta;
    cyclic_prefix = signal(starting : end);
    output = [cyclic_prefix signal];
end
```