ECE 598 HH **Spring 2018**

Lab Assignment 1

TX & RX OFDM Over USRPs

TA: Suraj Jog

In this lab assignment, we will learn to transmit, capture, and decode OFDM symbols with software defined radios. Specifically, we will use USRPs (Universal Software Radio Peripheral). For basic understanding of USRPs and OFDM, please refer to Lecture 3, Part 2 and Lecture 4 in the course. We will first begin with some basics on how to use USRPs, and then talk about specific tasks for the lab assignment.

1 Basics - Working with USRPs

USRPs are software defined radios. They consist of the digital and RF front-end which can be controlled in software using UHD (USRP Hardware Driver) from a host machine. UHD can be used to control various parameters and functionalities in a USRP. We will first begin by detailing the installation process of UHD. This document covers details for Ubuntu. For other operating systems, one should refer to the link https://files.ettus.com/manual/page_install.html.

1.1 Installing UHD

The step by step instructions for Ubuntu can be found at the link https://files.ettus.com/manual/page_build_guide.html.

Step 1 - Set up Dependencies

Run the following command -

```
>> sudo apt-get install libboost-all-dev libusb-1.0-0-dev python-mako doxygen
   python-docutils cmake build-essential
```

Step 2 - Getting the source code

The UHD source is stored in a Git repository. Download it in a respository in the *'home'* directory using the following command -

```
>> git clone --recursive git://github.com/EttusResearch/uhd.git
```

Step 3 - Build Instructions

First generate Makefiles with CMake

```
>> cd uhd/host
>> mkdir build
>> cd build
>> cmake ../
```

The next step is to build and install.

```
>> make
>> make test
>> sudo make install
```

Finally, we set the library path with the command

```
>> sudo ldconfig
```

1.2 Configuring USRP

Now that UHD has been installed, we need to configure the USRP to communicate with the host computer. This communication takes place over the ethernet, and we need to appropriately set the IP addresses of the host and the USRP port.

Step 1 - IP address of Host

We will modify the IP address of the ethernet port on the host machine to be `192.168.10.1`. Different IP addresses for the host machine could also be chosen. In 'Network Connections', you must identify the Ethernet connection on your host machine, and click on the 'Edit' tab (Fig. 1(a)). Go to the 'IPv4 Settings' tab and change Method to 'Manual'. Then add the new address for the host machine ethernet port as shown in Fig. 1(b).

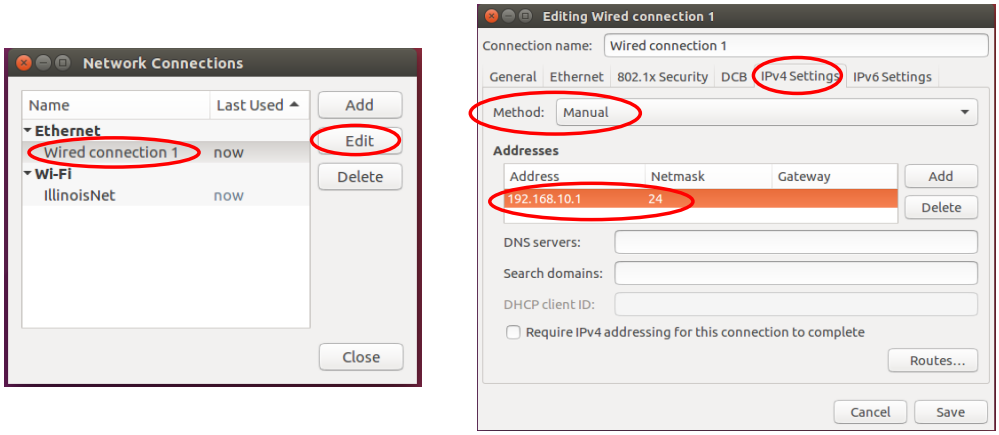

Figure 1: Configuring Host IP Address

Step 2 - IP address of USRP

Since USRP is part of the host's ethernet port subnet, it must have an IP address of the form '192.168.10.X'. Here we set its IP address to `192.168.10.2`, by using the function `/home/uhd/host/utils/usrp2_recovery.py`. The inputs required for the function are — 1) Ethernet Interface Name of host machine, and 2) New IP Address to set. The New IP Address to set is `192.168.10.2` in our case. To identify the Ethernet Interface Name, run command 'ifconfig' and find the interface with the IP address `192.168.10.1`. In our case, it is `'enp3s0f1'` (Fig. 2). On other machines, this can be 'eth0' or 'eth1'...

To set the USRP IP address, run the command -

```
>> sudo python usrp2_recovery.py --ifc = enp3s0f1 --new-ip = '192.168.10.2'
```

After running the command, turn the USRP on and off to ensure proper functioning with the new IP address. You might have to run this multiple times.

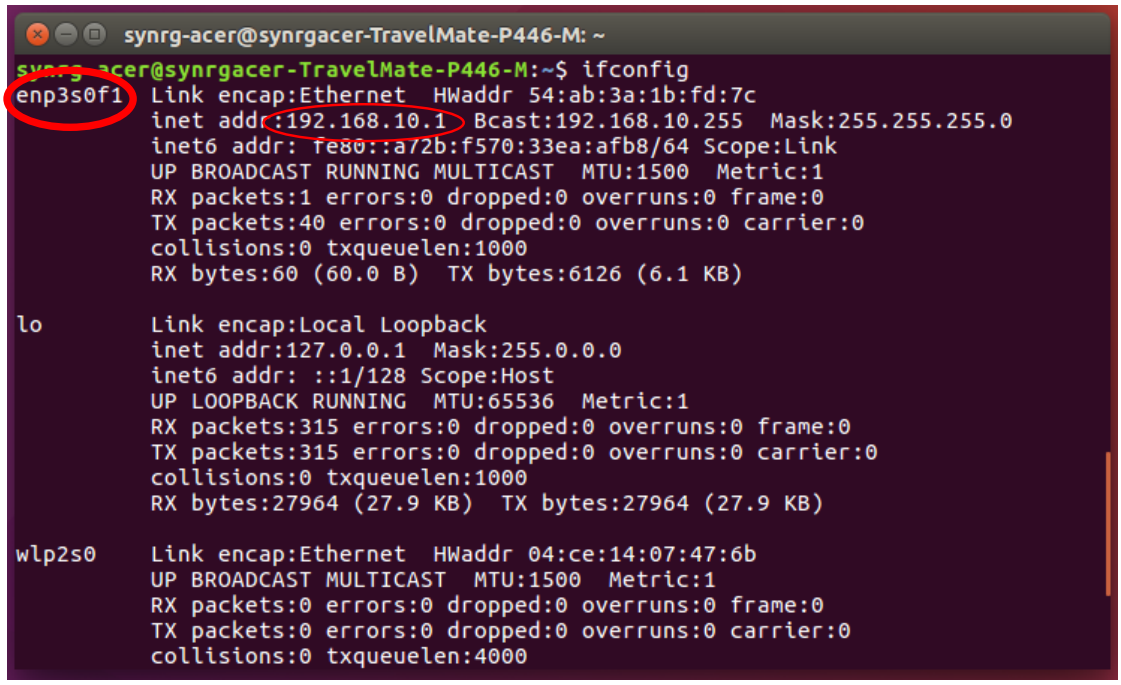

Figure 2: Configuring USRP IP Address

Step 2 - Testing the Configuration Process

In order to test if the above process was performed correctly, we run the command `'ping 192.168.10.2'`. This command sends echo packets to the USRP and displays the responses from the USRP. If the USRP is properly configured, then we will see a series of messages on the terminal showing the Round-Trip Time (RTT) for each echo packet. Another command you can use is `'find_usrp'` which will display the USRPs connected to the machine.

Now the USRP is set to communicate with the host machine through UHD.

1.3 Multiple USRPs

You can connect multiple USRPs to the same machine by connecting them to different ethernet ports. In this case, you will need to configure each ethernet port separately and you will need to use different IP addresses for different ports. For example, the first ethernet port can be set to `192.168.10.1` and its corresponding USRP will be configured to `192.168.10.2`. The second ethernet port can be set to `192.168.20.1` and its corresponding USRP will be configured to `192.168.20.2`.

1.4 Transmitting and Receiving Signals with USRP

The USRP in itself is just the Digital Frontend, and it needs to be augmented with an RF front end, which is the daughterboard. Each daughterboard has its own center frequency range and bandwidth. In this assignment, we will use the 900 MHz ISM band for signal transmission and reception with the SBX daughterboard. Unlike, the 2.45 GHz and 5 GHz ISM bands which have a lot of interference from WiFi, the 900 MHz ISM has relatively much lower interference.

1.4.1 Transmission of File

The command for transmission of a data file is -

```
>> sudo /home/uhd/host/build/examples/tx_samples_from_file --args
   addr=<usrp_ip_address> --file <path_to_tx_file> --type=float
   --rate <Sampling_rate> --freq <Centre_Frequency> --ant TX/RX
   --gain 10 --ref internal --repeat
```

The path `/home/uhd/host/build/examples/tx_samples_from_file` corresponds to the code where the function for transmitting a file through the USRP is implemented. This is followed by a list of arguments, each of which is explained below -

- `-args addr`: This field represents the IP address of the USRP used for transmission. In our case it is `192.168.10.2`.
- `-file`: This is the path for the data file which is to be transmitted.
- `-type`: Represents the data type.
- `-rate`: Represents the sampling rate for transmission. This parameter needs to be set according to the permissible ranges for the corresponding daughterboard.
- `-freq`: Represents the center frequency for the upcoversion of the baseband signal. This parameter needs to be set according to the permissible ranges for the corresponding daughterboard. In this lab, we set this parameter to `900000000` (900 MHz).
- `-ant`: Selection of antenna on the daughterboard. For the SBX daughterboard (which we will use in Lab 1), we can set it to `TX/RX`.
- `-gain`: Represents gain added to the baseband signal prior to transmission.

- `-ref`: Represents choice of clock reference for the USRP. Here we set the USRP to use its internal clock frequency. In cases when synchronization between multiple USRPs is required, we could set the parameter to `external`. In the `external` mode, the USRP could use the clock from an external clock or a master USRP.
- `-repreat`: This parameter denotes that the USRP should keep repeating the transmission of the data file.

1.4.2 Reception of File

The command for reception of a file is -

```
>> sudo /home/uhd/host/build/examples/rx_samples_to_file  --args
   addr=<usrp_ip_address> --file <path_to_rx_file> --type=float
   --nsamps <No._samples> --rate <Sampling_rate> --freq <Centre_
   Frequency> --ant TX/RX  --gain 20  --ref internal
```

The path `/home/uhd/host/build/examples/rx_samples_to_file` corresponds to the code with the functionality for receiving a file through the USRP and saving it as a data file. Most parameters in the command for reception are same as their counterparts in the command for transmission. The few that differ are

- `-args addr`: This field represents the IP address of the USRP being used for reception.
- `-file`: This field corresponds to the file name (and path) where the received data will be stored.
- `-nsamps`: This field represents the number of samples that the USRP should capture. If the field is omitted from the command, then the USRP will keep capturing samples until the program is terminated.

In this lab, we will use the above functions to transmit and receive OFDM packets on USRPs. While using USRPs you might encounter some common problems:

- *Overflow:* The USRP receiver code outputs 'O'. This means that the buffers are overflowing. This happens when you are receiving at a high rate such that the received samples are not being transferred from the USRP to the machine fast enough and end up overflowing the buffer and getting dropped. In this case, reduce your sampling rate.
- *Underflow:* The USRP transmitter code outputs 'U'. This means that the buffers are empty. This happens when you are transmitting at a high rate such that the transmitted samples are not being transferred from the machine to the USRP fast enough. In this case the buffers will underflow since there are no samples to transmit. Both underflow and overflow corrupt the transmitted and received signals. Reduce your sampling rate is one possible solution for both.
- *Cannot detect USRP:* In this case, try to run usrp2_recovery.py as described above.

2 OFDM Code

In this lab, our goal is to understand and design an OFDM physical layer. You will need to write the code for each block of the RX chain shown in Fig. 3. The code for the TX chain in Fig. 3 will be provided to you and you do not need to implement it. You will need the TX chain code to understand how the packet was constructed.

The code is to be written in MATLAB. We will provide you with skeleton code and you will only need to modify the places where you see *"... add your code here..."*. The lab will be divided into three parts. In each part, you will need to perform some subtasks and generate corresponding output files and figures for evaluation. For debugging purposes, we also provide some test signals with the expected output. The final goal is to use the RX chain to decode the OFDM packets captured by you in the lab with the USRPs.

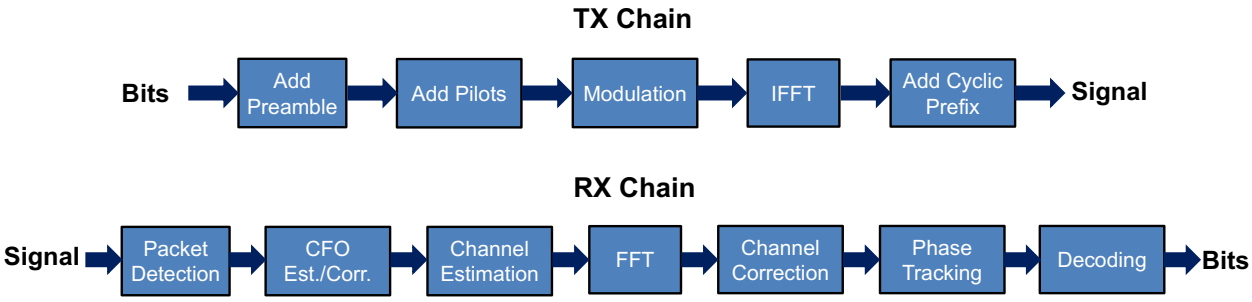

Figure 3: OFDM Physical Layer Chain

The code has 4 subfolders:

- **SRC_CODE:** These functions implement the various blocks in Fig. 3. This folder also contains the code `tx_ofdm_chain.m` which generates the transmission file. In addition, it has a subfolder named `functions` which implements some auxiliary functionality to help you with the lab.
- **Test_Scripts:** Code to test the implementations of individual receiver chain blocks.
- **Mat_Files:** Test Signals for debugging and evaluation. It has two subfolders `Debug` and `Test`. The files in `Debug` are to help you debug your code. The test scripts will use the files from the folder `Test` and we will evaluate your results on these data files.
- **Results:** This folder is initially empty. You must save the results to be submitted in this folder.

In this lab, we will only work with BPSK modulation. We will use the OFDM parameters corresponding to WiFi, as shown in Table 1. The main difference is the center frequency. We will run our experiments at $fc = 900$ MHz instead of 2.4 GHz to avoid interference. We will also use a lower sampling rate to avoid *overflow* and *underflow* at the USRP. In particular, we will run our experiments with a sampling rate $fs = 10$ MHz. The test code, however, was generated with a sampling frequency of $ft = 20$ MHz. Hence for decoding the received packets from the USRP, you should use fs. For decoding test data and running test code, you should use ft.

All parameters are stored in the matlab file `./Mat_Files/Parameters.mat`. The code loads these variables into the workspace so you can use them as is.

Parameter	Variable	Value
Carrier Frequency	fc	900 MHz
Sampling Frequency	fs	10 MHz
Sampling Frequency for debugging	ft	20 MHz
Subcarrier width	w	312.5 MHz
Number of Subcarriers / Symbol length	num_bins	64 bins/samples
Cyclic Prefix	cp	16 samples
Number of preamble symbols	num_syms_preamble	8
Number of data symbols	num_syms_data	72
Signal Length	num_samples	6288
Number of data subcarriers	num_bins_data	48
Number of pilot subcarriers	num_bins_pilots	4
Pilot Subcarriers	pilots	[-21, -7, 7, 21]
Gaurd/ Unused Subcarriers	guard_bins	[-32,...,-27, 0, 27,...,31]
Preamble Bits	bits_preamble	...
Pilot Bits	bits_pilots	[1 0 1 0]

Table 1: Parameters

3 Part 1 - Transmitting and Capturing OFDM packets

During your respective lab slots, you will transmit an OFDM packet through a USRP transmitter, and capture the packet over the wireless channel using a USRP receiver. You must capture packets at 10 different locations in the lab. The OFDM file to be used for transmission will be provided to each group at the beginning of the lab. The OFDM file is in the form of a '.dat' file, which can be read using the function `read_complex_binary2.m` (present in subfolder `/SRC_CODE/functions`).

In order to understand how the OFDM packet is generated, we provide the code for the packet generation in the file `'./SRC_CODE/tx_ofdm_chain.m'`. The code takes as input the data bits, and outputs the complex signal as per the TX chain in Fig. 3. The function `read_complex_binary.m` is then used to write the complex signal into the '.dat' file.

Some points to be noted for generating the OFDM packet are:

- There are 8 preamble symbols. The first 4 are for packet detection. The next 2 are for CFO estimation and the last 2 are for channel estimation.
- Cyclic Prefixes are added to the end of each data symbol. For the preamble symbols however, the cyclic prefix is added only to the last preamble symbol. This is because the preamble symbols are identical.
- In the modulation step, bit 0 maps to 1, and bit 1 maps to -1.
- Preamble symbols do not need pilots, and as a result the variable `bits_preamble` has 52 bits.
- Care must be taken to use the correct subcarrier indices. Because of the fftshift, the index of the subcarriers in the MATLAB fft index (1 to 64) is not the same as its normal index (-32 to 31). For example, the subcarrier with index -1 becomes 64 after the fftshift. Use the supplied functions (present in the `functions` subfolder) `convert_bin_index_fft_to_normal.m` and `convert_bin_index_normal_to_fft.m` to move between the two. To understand the usage better, look at the `'./SRC_CODE/tx_ofdm _chain.m'` code.

- The variables `bits_preamble`, `bits_data`, and `bits_pilots` are all asigned as per the normal indexing of subcarriers. For instance, the bits in the variable `bits_pilots` (1,0,1 and 0) should be assigned to subcarriers with indices -21, -7, 7 and 21 respectively. Similar mapping between bits and subcarriers should take place for the preamble and data bits. To understand this better, take a look at `'./SRC_CODE/tx_ofdm_chain.m'` code.

The above points are important because you will need to account for them while implementing the code for the receiver chain. At the end of the lab, each group will take their captured packets to further process them offline. For this purpose, each group should come equipped with a USB or an external hard disk with about 4 GB spare memory. Further, each group must implement the functionality to decode the captured packets to retrieve the original data bits.

4 Part 2 - Receiver Chain Blocks

In this part, you will work to implement the various receiver chain blocks shown in Fig. 3. This part is divided into the following subtasks:

4.1 Packet Detection

Implement the function `'packet_detection.m'`. It should take a complex signal as input and return the index of the first sample in the packet. You can use the signals in `/Mat_Files/Debug/Packet_Detection.mat` to check if your packet detection works. This file has two variables: `signals` which contains 100 different received complex wireless signals and `start_index` which contains the 100 corresponding indices to the start of each packet. Depending on the exact parameters that you use, your packet detection output could vary slightly. However this will not affect the ability to decode the packet due to the presence of cyclic prefixes as long as the offset in packet start is less than a cyclic prefix.

Here are a few hints to help you.

- You can implement either the double sliding window algorithm or correlation to detect the start of the packet. You know the preamble structure, so you can make use of that.
- In order to detect the start of the packet, you can use a threshold of 6. You must however play around with this threshold to see if you can get better packet detection. Plot the signal superimposed with the packet start index to see how well your packet detection is performing.

Finally, you must run the script `/Test_Scripts/test_packet_detection.m` which tests your packet detection algorithm on the data file `/Mat_Files/Test/Packet_Detection.mat`. It will output the file `/Results/Packet_Detection_Test.mat` which should be submitted for evaluation.

4.2 CFO Estimation and Correction

Implement the function `estimate_cfo.m`. The function should take as input two consecutive received OFDM symbols (rx_1 and rx_2), and output an estimate of the CFO. Given the length of the OFDM symbol to be L and a sampling rate of f_s, the CFO estimate can be given by the equation

$$f_\Delta = \frac{-f_s}{2\pi L} \ \angle\left(\sum_{j=1}^{L} rx_1(j) \cdot rx_2(j)^*\right)$$

Then, you must implement `correct_cfo.m`. The function should take a signal in the time domain along with an estimate of the CFO and should output the signal compensating for the CFO across the samples.

For evaluation you must do the following -

- For evaluating `estimate_cfo.m`, run the script `/Test_Scripts/test_estimate_cfo.m`. The script outputs the file `/Results/Result_CFO_estimation.mat`.
- For evaluating `correct_cfo.m`, run the script `/Test_Scripts/test_correct_cfo.m`. It will output the file `/Results/Result_CFO_Correction.mat` and the figure `/Results/Result_Cons_CFO_Corr.fig` which shows the constellation before and after CFO correction but before channel estimation.

Both above outputs should be submitted for evaluation.

4.3 Channel Estimation and Correction

Implement the function `estimate_channel.m`. The function should take as input two received OFDM preamble symbols and should output the complex wireless channel estimate. While the channel estimation could be performed using just one received preamble symbol, you should use two here for averaging. Also, make sure to ignore the unused subcarriers in the preamble while dividing the received symbol with the transmitted preamble.

After that, implement `correct_channel.m`. The function should take an OFDM symbol in the frequency domain along with an estimate of the channel and should output the symbol after compensating for the channel. Again, be careful to avoid the unused subcarriers.

For evaluation you must do the following -

- For evaluating the code `estimate_channel.m`, you must run the script `/Test_Scripts/test_estimate_channel.m`. It will output the following files - 1) `/Results/Result_Channel_Estimation.mat`, and 2) `/Results/Result_Channel.fig`.
- For evaluating `correct_channel.m`, you must run the code `/Test_Scripts/test_correct_channel.m`. It will output the figure `/Results/Result_Cons_Channel_Corr.fig`.

All outputs above should be submitted for evaluation.

4.4 Phase Tracking

Implement the function `estimate_residual_cfo_sfo.m`. The function should take as input a received OFDM symbol in the frequency domain along with an estimate of the channel and should output an estimate of the residual CFO and SFO using the pilot bits.

Recall that for an OFDM symbol of length L with C cyclic prefix samples, the residual CFO is the y-intercept and the residual SFO is the slope of the phase as a function of frequency bins. It can be calculated:

$$\delta f_c = \frac{f_s}{2\pi(L+C)} \times y - intercept \qquad\qquad \delta f_s = \frac{f_s L}{2\pi(L+C)} \times slope$$

Then, implement `correct_residual_cfo_sfo.m`. The function should take as input an estimate of the channel along with the residual CFO and SFO and should output a new estimate of the channel after compensating for CFO and SFO.

Here are a few hints to help you.

- Incorporate the phase tracking correction in the channel value, that is, update the channel h from symbol to symbol so that the phase does not accumulate so much that it starts wrapping around 2π. This will also allow us to automatically correct for the initial channel as well as all the accumulated phase when we run correct_channel.m.
- In phase tracking, when estimating the slope make sure that the indicies of the pilot subcarriers are in the normal mode, that is, -32...31.
- In phase tracking, you can use the builtin matlab functions regress( ) or robustfit( ) for estimating the slope of the phase.

For evaluation, you must run the script `/Test_Scripts/test_phase_tracking.m`. It will output the result `/Results/Result_Phase_Tracking.mat` and the figure `Result_Cons_Phase_Track.fig` which shows the constellation with and without phase tracking.

4.5 RX Chain

Implement the function `rx_ofdm_chain.m`. The function should take as input a complex signal and output the data bits. While generating the data bits, be careful about accounting for the conversion between fft indices and normal indices.

Here are a few hints to help you.

- Make sure to compenstate for the cyclic prefix bofore you run the FFT in the RX chain. Before you process a data symbol, make sure to skip one cyclic prefix.
- Make sure to skip enough samples (3/4 of the cyclic prefix) at the beginning to avoid inter symbol interference at the start of processing the packet.

To test and debug your code, use the file `/Mat_Files/Debug/RX_chain.mat`. This file has 2 variables: `bits` and `signal`. The i^{th} row of `bits` corresponds to the decoded data bits of the i^{th} row in `signal`.

For evaluation, you must run the script `/Test_Scripts/test_rx_ofdm_chain.m`. It will output the file `/Results/RX_Chain_Test.mat` which must be submitted.

5 Part 3 - Decoding OFDM Packets

The final part of the lab comprises using the RX chain designed by you to decode the packets captured in the lab. The captured signal should be passed as an argument to `rx_ofdm_chain.m`, which will output the decoded data bits. The decoded data bits for each of the OFDM files captured at the 10 locations should be saved as `'Result_RX_Chain_TrueOFDM_i.mat'`, where i ($1 \leq i \leq 10$) denotes the location number for the corresponding packet. Each .mat file should have both the captured signal and the corresponding decoded bits in the variables `signal` and `bits_data` respectively.

6 Submission

You should submit the following for evaluation as a single .zip on compass2g.illinois.edu.

1. The folder `SRC_CODE` with all functions that you implement.
2. The `Results` folder which includes the following files
    - `Packet_Detection_Test.mat`
    - `Result_CFO_estimation.mat`
    - `Result_CFO_Correction.mat`
    - `Result_Cons_CFO_Corr.fig`
    - `Result_Channel_Estimation.mat`
    - `Result_Channel.fig`
    - `Result_Cons_Channel_Corr.fig`
    - `Result_Phase_Tracking.mat`
    - `Result_Cons_Phase_Track.fig`
    - `RX_Chain_Test.mat`
    - `Result_RX_Chain_TrueOFDM_i.mat where 1 ≤ i ≤ 10`