# 4G LTE Simulation Project Pt. 1

Group:
Comms Platform 4

Teacher:
Dr. Mark Mahon

Course:
CMPEN 462

# Abstract

LTE was first introduced in 2004 by NTT Docomo, and was finalized as a wireless communications standard in 2008. The goal of LTE was to increase the network throughput with a new architecture which relied on newer DSP techniques to boost the capacity and speed that the network was capable of delivering compared to previous generation 3G technologies.

According to the LTE specifications, the network is capable of delivering up to 300 Mbps downlink, and 75 Mbps uplink. The transfer latency is 5 ms, along with carrier bandwidths scaling from 1.4 MHz to 20 MHz. The standard utilizes frequency division duplexing and time division duplexing. [3]

# Introduction

Our group was tasked with creating a simulation of how a bitstream of data is processed through the various stages of the LTE architecture, and then transmitted. The language our simulation scripts are written in is MATLAB.

We broke the LTE structure into a nine-stage pipeline, as shown in Figure 1. The bitstream was given to us as input, then the input was Encrypted, run through QPSK, Resource Element Mapping, IFFT, OFDM, Cyclic Prefix, IF Conversion, and finally DAC. The colors represent our initial anticipated difficulty, and was used as the metric to divide tasks between each member.

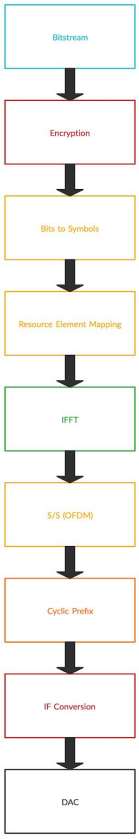


Figure 1. LTE Architecture

The following table (Table 1) includes the specifications we utilized in our implementation.

Our final design is broken into six MATLAB function files and one main MATLAB file. The six function files in order are:

- createPnSequence.m
- encrypt.m
- QPSK.m
- trans.m
- CP.m
- ifConvert.m

The main.m file calls all of the functions and passes the output of one function as the input to the next one. It also saves the output of each function in a .mat file.

| | |
|---|---|
| **System Bandwidth** | 10.24 MHz |
| **Subcarrier Bandwidth** | 10 kHz |
| **IF Frequency** | 100 MHz |
| **Cyclic Prefix Length** | 6.84 μs |
| **Max Transmit Power** | 100 μW |
| **Input (binary data)** | 1 s |
| **Number of Subcarriers** | 1024 |
| **Symbols per Second** | 10,000 |
| **Symbol Duration** | 0.1 ms |

Table 1. Specifications

# Functional Blocks

## Bitstream

The first stage of our pipeline was the input bitstream. This bitstream represents binary data to be sent over the 4G LTE communication standard. As 4G LTE is an IP-based service, the bitstream is actually composed of a stream of IP packets. For our project, we were given an MATLAB file titled Proj1InputData.mat to utilize as the input.

## Encryption

Once the input stream is passed down from the upper layer of the protocol stack, the data must be encrypted. According to NIST LTE specification[2], there are 3 variations of cryptography algorithms utilized, with EPS Encryption Algorithms (EEA) and EPS Integrity Algorithms (EIA) being their basis. The three algorithms are titled EEA1 and EIA1 (based on SNOW 3G), EEA2 and EIA2 (based on AES), EEA3 and EIA3 (based on ZUC). Other algorithms such as PN sequences are also utilized for this stage, as in our case. The encryption method does not provide end-to-end encryption, as the data is only encrypted when traveling through the radio waves, then once in the core-network, this encryption is removed. Therefore it is up to the user to implement a different form of application layer encryption such as SSL or SSH to ensure privacy of data.

In our design the encryption takes place in two of the function files createPnSequence.m and encrypt.m. The first file creates a Pseudo Noise (PN) sequence using a Modular Shift Register Generator (MSRG). A diagram of the MSRG used is shown below in Figure 2. The characteristic polynomial used was $f(x) = 1 + x + x^2 + x^5 + x^{19}$ . The initialization vector (IV) used was [1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]. We achieved this in MATLAB by first creating a vector containing the IV. The last element of this vector was stored as the first element of a new vector that stores the entire PN sequence. Next, there is a loop that goes through the rest of the sequence, $2^n - 1$ total values, updating all the registers for each value and storing the last element as the next element in the PN sequence. After this loop there is another loop that allows you to change the size of the sequence by looping through and creating a new array with as many of the values from the sequence as needed. If more values are needed then the length of the sequence it starts back at the beginning of the sequence. This part was not utilized because the sample .mat file provided included just one iteration of the PN sequence. The encrypt.m file is much simpler. It takes the PN sequence and input data as inputs and XORs them together repeating the PN sequence if needed and stores the result in a new array.

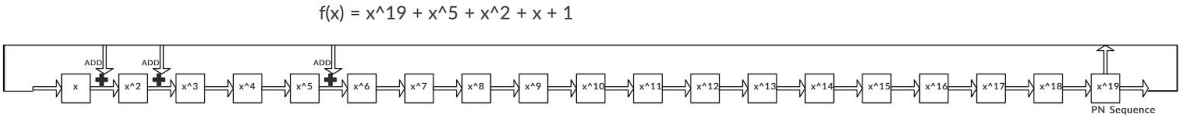


Figure 2. MSRG Encryption

# Bits to Symbols

When transmitting data over radio waves through a transmit antenna, we must think of the wave as a sinusoid (with time dependent phase, magnitude, and frequency components) as opposed to the literal bits we want to transmit. LTE specification[1] can utilize QPSK, QAM16, and QAM64 to make modifications to the sinusoid and map the bits to the wave. Quadrature Phase Shift Keying (QPSK) is a modulation scheme that involves shifting the sinusoid phase based on the binary input stream, taking in packets of 2 bits and mapping them to one symbol (the state of the carrier). We call the rate at which the carrier changes states the symbol rate. The LTE spec QPSK diagram is shown in Figure 3. Quadrature Amplitude Modulation (QAM) is another form of digital signal modulation that uses a combination of signal phase and magnitude to determine the symbol output. QAM offers higher data rates than QPSK, and can map 16, 64, or 256 symbols, however LTE spec only supports QAM16 and QAM64. In the case of QAM64, each symbol can encode 6 bits ( $2^6 = 64$ ), and QAM16 can encode 4 bits per symbol ( $2^4 = 16$ ).

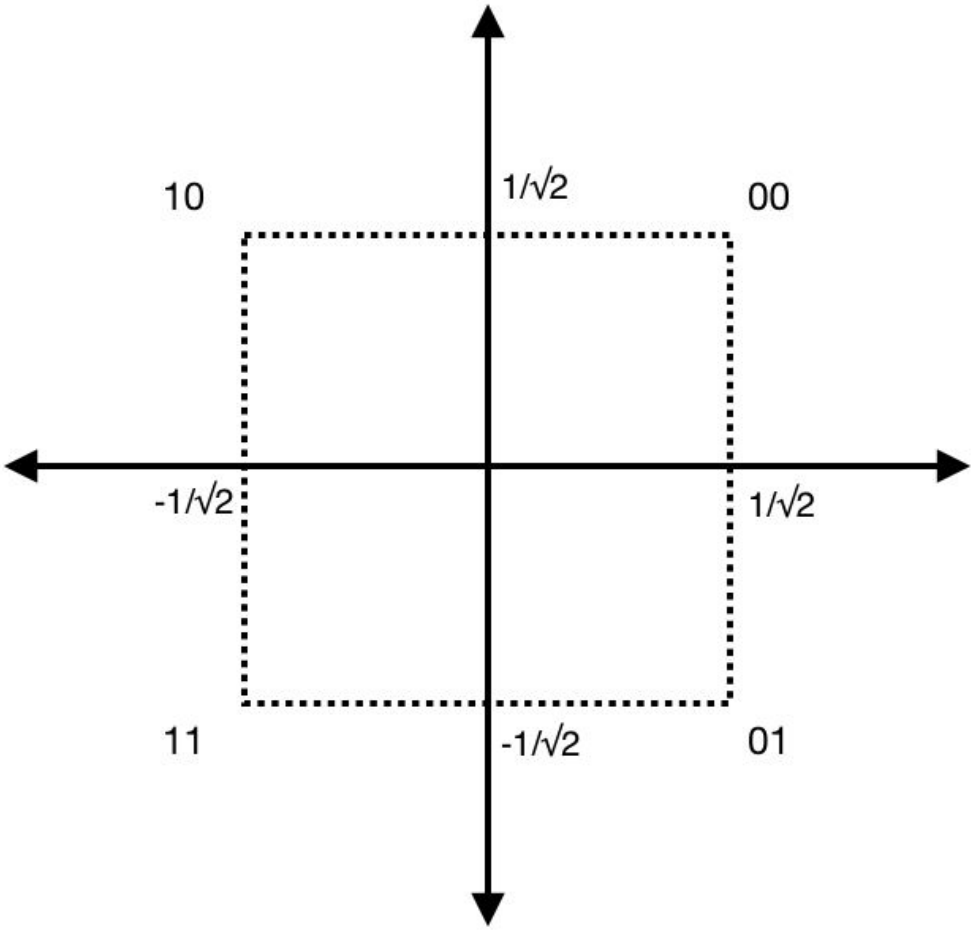


Figure 3. QPSK LTE Spec. Constellation Diagram

The bits to symbols conversion in our system was done in QPSK.m. To do this we first converted our data from an array to a 2 row matrix which split the data into symbols with each column representing a two byte symbol needed for QPSK. We then looped through this two row matrix column by column and assigned the real and imaginary parts of each complex element. The two parts of the complex element were put together and added as the next element of the output array.

# Resource Element Mapping

According to the LTE spec[4], the number of subcarriers ranges from 128 to 2048, depending on the desired channel bandwidth. The subcarrier spacing is 15kHz. Resource element mapping allows us to have multiple users utilizing the network simultaneously. The transmission of data can be scheduled by resource blocks (RBs), which are composed of 12 subcarriers with 180kHz of bandwidth per time slot. A resource element (RE) is an atomic unit within an RB, and is composed of one OFDM subcarrier. Figure 4 shows how an RB and RE exists in context to each other, within LTE spec.

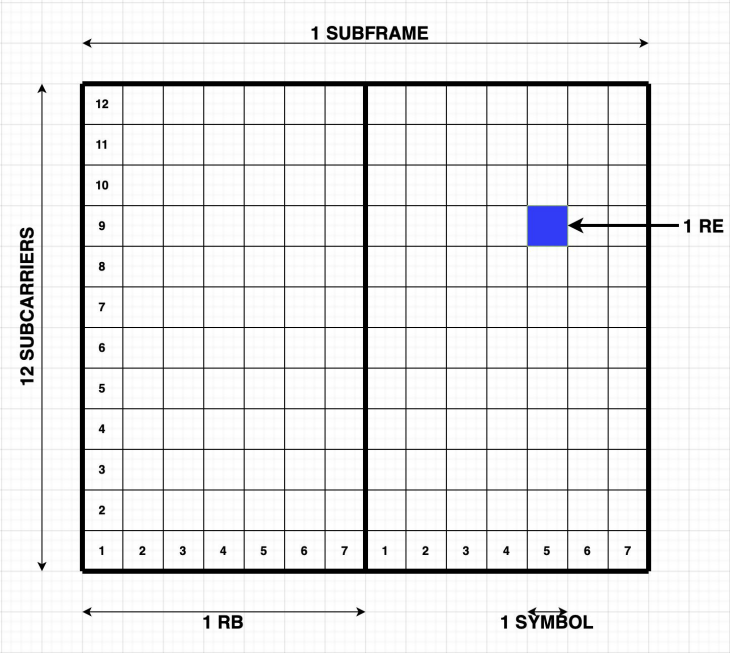


Figure 4. RE and RB Diagram

In our implementation this was done in the same function as the IFFT which will be described in more detail in the next section.

# IFFT

The Inverse Fast Fourier Transform (IFFT) is used to convert signals from the frequency domain to the time domain, as the signal must be represented in the time domain for transmission.

In our system the IFFT along with the Resource Element Mapping was performed in the trans.m file. For resource element mapping all that's needed in our system is to break the data vector into groups of 1024 for the IFFT. This was achieved by reshaping the data into a matrix with each row being one group of 1024 symbols. We then looped through those rows and performed an IFFT on each row. Each row was then appended to a new vector for the output. In a real life application the data would not be put back in serial but this was done to compare our values to the test data given.

# OFDM

Orthogonal Frequency Division Multiplexing (OFDM) is a multi-carrier modulation method utilizing frequency division multiplexing (FDM). It allows multiple users to transmit and receive data without having to worry about wasting bandwidth, or intercarrier interference (ICI) between subcarriers. OFDM is used for the downlink, over carriers of size 180 kHz. Each user is given a certain number of RBs for their data transmission, and the number of RBs assigned to a particular user depends on LTE scheduling mechanisms to provide ideal performance in varied radio environments.[5]

This did not need to be implemented into our project because we only had one "user", the one second binary stream.

# Cyclic Prefix

The Cyclic Prefix (CP) is used in LTE communication as a guard between symbols, while preserving the orthogonality of the subcarriers. The CP is a cyclic set of symbols from the end of each symbol set, and added to the front of the transmitted symbol to eliminate intersymbol interference (ISI). The receiver can then identify where each symbol ends, and determine what symbol sets correlate to what data. The normal CP has a duration of 5.2 μs for the first prefix, and then 4.7 μs for the remaining symbols.[6]

The cyclic prefix addition was performed in the CP.m function. To add a cyclic prefix we first split the data back into groups of 1024 symbols. We then looped through these groups and added a copy of the last 70 symbols (our calculated prefix length) to the beginning of each group. This group was then added to a new vector for the output.

## IF Conversion

The Intermediate Frequency (IF) is the carrier wave frequency shift. This is done for signal amplification purposes so that the transmitter antenna has a large enough signal to transmit over a large distance. The data signal is multiplied by the carrier signal (where the real part is multiplied by $cos(2 * \pi * t * f)$, and the imaginary part by $sin(2 * \pi * t * f)$, then sent to the DAC.[7]

In our system this was done in the ifConvert.m function. This function first splits the data into two arrays one with the real parts and one with the imaginary parts of the complex data. It then loops through the arrays and multiplies the real part by $cos(2 * \pi * t * f)$ and the imaginary part by $sin(2 * \pi * t * f)$ with t being time and f being the subcarrier bandwidth. The real and imaginary parts are then put back together and added to the end of the output vector.

## DAC

As a radio transmitter can only transmit analog signals, the DAC must convert the digital signal we have generated so far into an analog signal so that the antenna can transmit the desired data.

This step was going to be performed on the Raspberry Pi but was not needed after the project was modified.

# Testing

All of our testing was performed in a seperate test.m file. The file was broken into sections with each section being for one of the .m files. Each section first calls the function with the test data from the last step. The output is then compared to the correct test data for that step. It then prints if the data was equal or not. If the data was not equal you could uncomment out a block of code that would print a desired range of the values in both the correct test data and the output of the function along with the difference between them. You could then use this data to determine where the function was going wrong and attempt to fix it. The specific problems we encountered will be discussed in the next section.

# Problems

The first major problem that we encountered was getting the correct PN sequence. When comparing the results most of the values in the beginning were correct with a few missing 1's. This was because we had copied the characteristic equation down wrong in class. The $x^2$ term was missing. After the announcement on Canvas we updated the code and it was correct. We did not have any major problems with the encrypt.m function. The QPSK function also had a few problems. The first problem was we were trying to modify our code for the homework which produced graphable data and had multiple points for each symbol at different times to make a smooth graph. This resulted in a running time that was extremely long and did not produce the correct output. We had to modify it to only produce one complex pair for each symbol. Second, the way we were comparing our pairs of data for each symbol (eg. 00, 01, 10, or 11) was not working and we had to modify our if and else statements. The trans.m function which performed the serial to parallel and IFFT had one major problem. When we tried to reshape the data we could not get it to reshape into 1024 columns with each row being a group of 1024 symbols because the MATLAB reshape function fills by columns and not rows, which we did not realize for a while. After we figured that out and did some searching on the MATLAB forums we found that we could make the data into 1024 rows with each column representing a group and transpose it to get the data into the shape we wanted. The CP.m function did not provide many problems because we figured out the reshaping from the last steps and adding the prefixes was pretty straightforward. The only problem we had with the ifConvert.m function was with the time variable which we found an answer to from someone else's post on Piazza.

# Results and Functionality

Most of the blocks produced the exact values as the sample data. trans.m and ifConvert.m were the only two that didn't, however they were within an extremely small margin of error probably due to rounding somewhere along the line. We determined that these values were probably close enough and could not get them any closer even after extensive testing. The results are summarized in Table 2 below.

| Our Function | .mat File Compared To | Margin of Error |
|---|---|---|
| createPnSequence.m | Proj1PNSeq.mat | None |
| encrypt.m | Proj1TestEncryptedData.mat | None |
| QPSK.m | Proj1ModSymb.mat | None |

| | | |
|---|---|---|
| trans.m | ProjectIFFTOfdmSymbolStream.mat | ~ $10^{-18}$ |
| CP.m | Proj1TransSymbStream.mat | None |
| ifConvert.m | Proj1UpCnvrt2IFData.mat | ~ $10^{-16}$ to $10^{-17}$ |

Table 2. Results

# Work Performed and Roles

| Task | Completed by |
|---|---|
| MATLAB Simulations | Nikolas Collina, Praharsh Verma |
| Document Writeup (MATLAB Parts) | Nikolas Collina |
| Document Writeup (Theory Parts) | Praharsh Verma |

Table 3. Worked Performed & Roles

For the MATLAB scripts, Nikolas and Praharsh worked together over FaceTime and the screen sharing capability on MacOS to allow us to collaborate on the code in real-time, debugging and ensuring quality of output.

For the document writeup, Nikolas and Praharsh worked together using Google Docs and FaceTime to collaborate on the document in real-time. Nikolas wrote up the sections pertaining to how the MATLAB scripts were implemented in each section, as well as the Testing, Problems, and Results & Functionality sections. Praharsh wrote up the sections pertaining to the theory on how LTE architecture functions, as well as the Introduction and Abstract.

Originally, our group created a GroupMe chat to remain in contact with each other about the project. Once the group chat was made, we (Nikolas and Praharsh) tried to set a meeting time when we could all discuss the project, and get a general idea on what needs to be done, set deadlines, and divide tasks up. We had all decided on a time to meet, and

booked a room in the library right before spring break and divided up the tasks. We had tried to set deadlines for getting all the parts done by the Monday after break. We reached out multiple times and were not receiving responses from our team members. We decided to work on as much of the project as we could get done in case we never heard back from them. The night before the deadline, we heard back from one of them, who had no functional code written for the project. We asked him to explain what he had written and got no response after that. We decided to finish up the project on our own to avoid getting a bad grade, and then worked on the write up ourselves as well as we did not hear anything back from them until after it was written.

# Bibliography

[1] StackPath. [Online]. Available: https://www.electronicdesign.com/technologies/4g/article/21796272/an-introduction-to-lteadvanced-the-real-4g#âModulationâ.

[2] J. Cichonski, J. M. Franklin, and M. Bartock, “Guide to LTE security,” 2017.

[3] “The MobileBroadband Standard,” LTE. [Online]. Available: https://www.3gpp.org/technologies/keywords-acronyms/98-lte.

[4] “LTE in a Nutshell: The Physical Layer.” [Online]. Available: https://home.zhaw.ch/kunr/NTM1/literatur/LTE in a Nutshell - Physical Layer.pdf.

[5] “LTE OFDM Technology,” Tutorialspoint. [Online]. Available: https://www.tutorialspoint.com/lte/lte_ofdm_technology.htm. [Accessed: 01-Apr-2020].

[6] L. Z. Pedrini, “What is CP (Cyclic Prefix) in LTE?,” telecomHall Forum, 10-Nov-2019. [Online]. Available: https://www.telecomhall.net/t/what-is-cp-cyclic-prefix-in-lte/6369. [Accessed: 01-Apr-2020].

[7] “The Benefits of an Intermediate Frequency in RF Systems: Selected Topics: Electronics Textbook,” All About Circuits. [Online]. Available: https://www.allaboutcircuits.com/textbook/radio-frequency-analysis-design/selected-topics/the-benefits-of-an-intermediate-frequency-in-rf-systems/. [Accessed: 01-Apr-2020].