LTE eNodeB/UE PHY Layer Implementation on General purpose CPU and GPU

By

Ahmed Mostafa Hosny

Ahmed Nour El-Deen Shahatah

Khaled Ahmed Ali

Mohamed Mostafa Nasr

Mohamed Osama Mohamed

Under supervision of

Dr. Ahmed Hesham

A Graduation Project Report Submitted to
the Faculty of Engineering at Cairo University
in Partial Fulfillment of the Requirements for the
Degree of
Bachelor of Science
in
Electronics and Communications Engineering

Faculty of Engineering, Cairo University

Giza, Egypt

July 2017

Table of Contents

List of Tables

List of Figures

List of Symbols and Abbreviations

3GPP	3rd Generation Partnership Projects
3GPP2	3rd Generation Partnership Project 2
AMPS	Advance Mobile Phone System
AWGN	Additive white Gaussian noise
CB	Code Blocks
CDMA	Code Division Multiple Access
CM	Cubic Metric
CP	Cyclic Prefix
DFE	Decision feedback equalizer
DL	Downlink
DM-RS	Demodulation Reference Signals
EDGE	Enhanced Data Rates for GSM Evolution
EM	Electromagnetic
E-UTRA	Evolved UMTS Terrestrial Radio Access
E-UTRAN	Evolved UMTS Terrestrial Radio Access Network
EV-DO	Evolution-Data Optimized
FDD	Frequency Division Duplexing
FFT	Fast Fourier Transform
FIR	Finite Impulse Response
GPRS	General Packet Radio Service
GSM	Global system for Mobile communications
HSDPA	High Speed Downlink Packet Access
IFFT	Inverse Fast Fourier Transform
ISI	Intersymbol interference
ITU	International Telecommunication Union
LFSR	Linear-feedback shift register
LMMSE	Linear Mean Minimum Square Error
LOS	Line of sight
LS	Least square
LTE	Long Term Evolution

MIMO	Multiple Input Multiple Output
ML	Maximum Likelihood
MSE	Mean Square Error
OFDM	Orthogonal Frequency Division Multiplexing
OFDMA	Orthogonal Frequency Division Multiple Access
PHY	Physical Layer
PSK	Phase-Shift Keying
PUSCH	Physical Uplink Shared Channel
QOS	Quality of Service
RB	Resource Block
RE	Resource Element
SC-FDMA	Single Carrier Frequency Division Multiple Access
SIMD	Single Instruction Multiple Data
SIMO	Single Input Multiple Output
SNR	Signal-to-Noise Ratio
TACS	Total Access Communication System
TB	Transport Block
TDD	Time Division Duplexing
TDMA	Time Division Multiple Access
UE	User Equipment
UL	Uplink
UL-SCH	Uplink shared channel
UMB	Ultra-Mobile Broadband
UMTS	Universal Mobile Telecommunication System
W-CDMA	Wideband Code Division Multiple Access

Acknowledgment

Firstly, we would like to express our sincere gratitude to our supervisor Dr. Ahmed Hesham for the continuous support of our Graduation Project, for his patience, motivation, and immense knowledge. His guidance helped us a lot.

Besides our supervisor, we would like to thank Eng. Waleed Tag El-Din for his insightful suggestions, comments and encouragement.

Finally, we would like to thank Axxcelera Egypt, for the project idea, training sessions, and the opportunity to join such a great experience. Especially Eng. Mahmoud Abubakr.

Abstract

Nowadays wireless communication has influenced the world in many important ways. It had a dramatic impact on how companies conduct business, making it easier to keep in touch with customers around the world. Our project thesis is about the fourth generation of mobile communications also known as Long Term Evolution (LTE), especially on the physical uplink shared channel (PUSCH). The target of our project is to move the processing of the PUSCH from DSP to a general-purpose CPU/GPU. The PUSCH transmitter and receiver are implemented using two different platforms: CUDA and Intel AVX with Intel math kernel library (MKL). First, moving the implementation from DSPs to general-purpose processor is discussed in the first chapter. Second, the used platforms are presented in chapters 2 and 3. Then, LTE introduction, PUSCH blocks and communication channel are presented in chapters 4 and 5. The resources used like the HW platform and SW framework are presented in chapter 6. Finally, the BER plots and timing results are presented in chapter 7.

Chapter 1: Introduction

1.1 Problem Overview

Communications and DSP applications are known to be computational intensive. In communication systems, we have to perform complex operations on the transmitted/received data. These complex operations increased in LTE PHY layer due to the use of FFT (Fast Fourier Transform) so a fast and powerful hardware is needed.

Due to the high computational requirements of the LTE eNodeB/UE, many specific platforms that are normally combinations of DSPs, general purpose processors and ASICS are available.

1.1.1 DSP Advantages

- Instruction set contains complex mathematical operations (Like FFT).
- Ability to fetch multiple data or instructions from memory at the same time.

These advantages made DSPs commonly used in communication systems.

1.1.2 DSP Disadvantages

- The problem of using these platforms is that the platform may be totally unusable for subsequent LTE releases or for upgrading to a newer technology.
- Upgrading the hardware may lead to software upgrading which means a high waste of time.
- The reuse of the codes over different hardware may lead to several conflicts.

This upgradability problem will be translated into cost as the platform needs to be changed.

1.2 Problem Solution

Implementing the LTE Physical layer on a general-purpose processor will eliminate the need for a specific platform.

1.2.1 Project Objective

Evaluating the feasibility of implementing the LTE eNodeB-L1 PUSCH chain on a general-purpose CPU and GPU and check if we can achieve the LTE 1ms sub-frame constrain.

1.2.2 General-Purpose Processor Advantages

- The general-purpose CPU/GPU will be reusable and expandable.
- Instruction set of the processor contains **SIMD** operations like DSPs.
- The support for general purpose CPU development can be obtained easier than that of specific platforms.

1.2.3 General-Purpose Processor Disadvantages

- **SIMD** operations can't perform complex mathematical operations (like FFT) so they are performed using basic mathematical operations.
- This means that, it would be slower than the specific platform (i.e. DSP) in doing the required operations,

However, we can use parallelism techniques to reduce the execution time on the general-purpose processors like:

- AVX instructions (Supported in Intel Core-ix Processors).
- Intel MKL (Math Kernel Library).
- Intel Cilk Plus operations.
- Offloading parts of the program to the GPU, to use its parallel computing capability.

These techniques are discussed later in chapter 2 and chapter 3 respectively.

Chapter 2: Processors

2.1 General Purpose Processor

General purpose means that it can be used for many different tasks and not designed for a special purpose. A microprocessor can be programmed by the user. It is necessary for the user to know about the internal resources and features of the microprocessor. The programmers must also understand the instructions that a microprocessor can support. Every microprocessor will have its own associated set of instructions that it supports and this list is given by all the microprocessor manufacturers.

A microprocessor can either have a CISC, or complex instruction set computer, architecture or a RISC, or reduced instruction set computer, architecture. The CISC architecture is more complex and can perform complex commands. The RISC architecture is simpler, smaller and faster.

2.2 Intel Processors

Intel® is the world's oldest and most established microprocessor company, producing the world's most popular microprocessor chips. Although perhaps best known for its PC processors, Intel devices are used in many field of electronics including automotive, robotics, consumer electronics, image processing, networking, encryption, military, and other industries.

Since the first tiny Intel 4004 microprocessor chip was made in 1971, Intel has produced an unbroken series of upgrades and improvements to the world's best known microprocessor family. From its early 8-bit beginnings, the Intel architecture now encompasses a range of 32-bit and 64-bit microprocessors that address a range of applications, performance requirements, power levels, and price points.

The cornerstone of Intel architecture's popularity is its compatibility. Each new generation of Intel architecture microprocessor is a superset of its predecessors, providing backward compatibility with older chips and older software, while also adding new or enhanced features. This compatibility allows engineers, programmers,

and development teams to reuse the software and software-development tools from earlier projects, protecting their investment in time and talent.

Intel architecture chips have obviously undergone many changes over the past 40+ years. Early chips were given technical part numbers, such as 8086, 80386, or 80486. This led to the commonly used shorthand of "x86 architecture," in reference to the last two digits of each chip's part number. Beginning in 1993, the "x86" naming convention gave way to more memorable (and pronounceable) product names such as Intel® Pentium® processor, Intel® Celeron® processor, Intel® Core™ processor, and Intel® Atom™ processor.

Although every branch of the broad Intel architecture (or x86) family tree retains the same basic features and functionality as the earlier chips, each new generation also adds its own unique features. For example, Intel Pentium processor added multimedia extensions (called MMX™ technology) that accelerated audio and video processing. Extended temperature Intel Pentium processor with MMX technology is with more streaming-media capabilities known as Intel® Streaming SIMD Extensions (Intel® SSE) and Intel® Streaming SIMD Extensions 2 (Intel® SSE2). Floating-point units (FPUs) went from optional upgrade to standard feature of Intel architecture processors, and today encryption/decryption extensions, power-management features, and multilevel caches are now found on most Intel architecture processors. Data paths have widened from 8 bits to 32 bits, 64 bits, and even 128 bits and more. Operating frequencies have jumped from a few megahertz to 3 GHz (three billion cycles per second) and beyond. [1]

2.3 Single Instruction Multiple Data Instructions

SIMD represents a class of computation where multiple processing engines perform the same operation on multiple data elements simultaneously. SIMD Extensions Support is another example of the commodity general-purpose CPU adding specialized hardware support for domain-specific applications. Like floating-point operations, SIMD operations used to be the domain of highly specialized vector supercomputers (e.g., Control Data STAR 100, Cray 1 Computer System, and the Connection Machine® Model CM-1 from Thinking Machines Corporation).

Over time, the CPU increased in performance, thanks to Moore's law and Dennard scaling. In 1997, SIMD processing was introduced to commodity CPUs with the inclusion of MMX™ technology extensions to a model of the Pentium® processor. The MMX extensions enabled the calculation of 8 bytes or 4 words in parallel using 64-bit registers that supported vector integer operations. These enabled improvements in multimedia and communications workloads, in addition to improving graphics realism and full-screen, full-motion video.

As CPUs became more powerful, software running on them increased in complexity. The Streaming SIMD Extensions (SSE) in 1999 expanded the SIMD capabilities in the architecture to include 128-bit wide architectural registers and packed single precision calculations. To take advantage of this new architectural capability, implementations introduced dedicated hardware to support SSE such that performance gains could be capitalized on. These extensions also addressed some of the MMX limitations by supporting the floating point data type. This enabled mixed integer and floating-point SIMD. In 2001, the SSE2 extensions added support for 8-bit, 16-bit, and 32-bit integer vectors in addition to double-precision data types. These extensions also provided programmers greater control to access and cache data. Since streaming data types do not always have cache locality, the extensions also provided software direct control over cacheability to minimize cache pollution, as well as support for software directed prefetching of data. With the increasing sophistication of the SSE extensions, they supplanted x87 for most floating-point operations (except 80-bit extended precision) and MMX for media applications.

The instruction set architectures remain fluid to match the needs of applications, so it is not a typical for every CPU generation to add new instruction capabilities. The SSE3 extensions in 2004 provided additional instructions for format conversion, and supported unaligned address loads while the SSSE3 extensions in 2006 accelerated operations on packed integers. The SSE4.1 and SSE4.2 extensions in 2007 and 2008 provided further enhancements to improve compiler vectorization, support for packed double word computation and for string and text processing. [2]

In 2010, the Intel® Advanced Vector Extensions (AVX) widened the vector registers to 256 bits and introduced the first set of instructions to utilize these registers, focused on floating point. In 2012, the Intel AVX2 extensions widened the

integer data type to 256 bits. More details about Intel AVX will be provided in the next sub-sections.

2.3.1 Intel® Advanced Vector Extensions

Intel Advanced Vector Extensions (Intel AVX) is a set of instructions for doing Single Instruction Multiple Data (SIMD) operations on Intel architecture CPUs. These instructions extend previous SIMD offerings (MMX and Intel SSE) by adding the following new features:

- The 128-bit SIMD registers have been expanded to 256 bits. Intel AVX is designed to support 512 as in AVX 512 that was already launched in 2016
- Three-operand, non-destructive operations have been added. Previous two-SIMD extensions performed operations such as A = A + B, which overwrites a source operand; AVX can perform operations like A = B + C, leaving the original source operands unchanged.
- A few instructions take four-register operands, allowing smaller and faster code by removing unnecessary instructions.
- Gather support, enabling vector elements to be loaded from non-contiguous memory locations (Supported by AVX2).
- Three-operand Fused Multiply Add operations (FMA3) support as performing this operation (A = A * B + C) in a single instruction (Supported by AVX2).
- Memory alignment requirements for operands are relaxed.

The AVX instruction set has performance higher than the legacy SSE, and conventional non-vectored (serial) codes, the following result is for running Mandelbrot set example codes (a Mandelbrot set is a computationally intensive operation on complex numbers) which were implemented using AVX, SSE, and conventional float and complex data types.

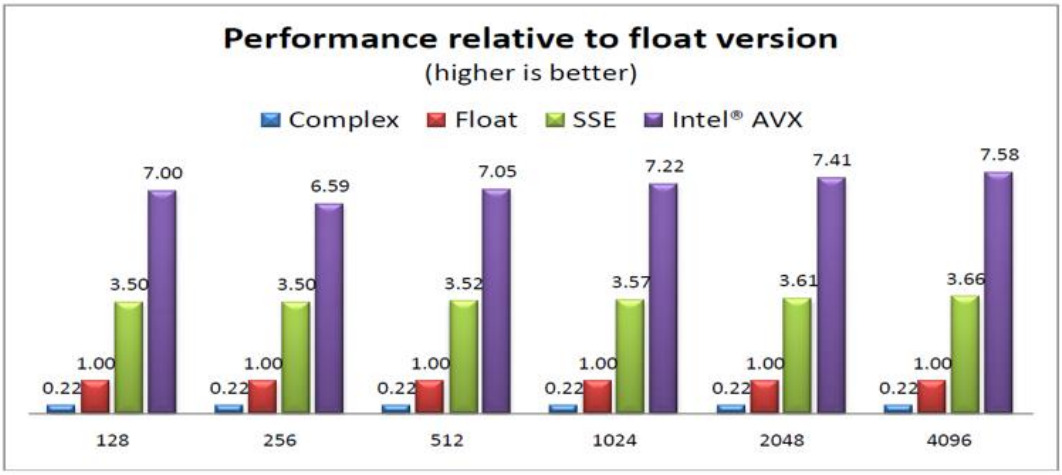


Figure 2.1: Performance comparison between Serial, AVX and SSE.

2.3.2 Programming with Intel® AVX

There are two ways of programming using AVX:

- Using the Assembly instructions.
- Using intrinsic Functions provided by Intel compiler.

The easier way is to use the 2nd method or the intrinsic functions. Intrinsic functions are assembly-coded. They allow the programmer to use C function calls and variables in place of assembly instructions. [3]

2.4 Intel® Math Kernel Library:

The Intel® Math Kernel Library (Intel® MKL) provides a comprehensive set of math functions that are optimized and threaded to exploit all the features of the latest Intel® processors. Intel MKL improves performance with math routines for software applications that solve large computational problems as Intel MKL linear algebra routines, fast Fourier transforms, vectored math functions, random number generation functions, Matrix-Matrix operations and other functionality.

Intel MKL is built using the Intel® C++ and Fortran Compilers and threaded using OpenMP. Its algorithms are constructed to balance data and tasks for efficient use of multiple cores and processors.

2.4.1 Performance Enhancements:

The Intel® Math Kernel Library has been optimized by exploiting both processor and system features and capabilities as those routines that most profit from cache-management techniques.

The major optimization techniques used throughout the library include:

- Loop unrolling to minimize loop management costs
- Copying to reduce chances of data eviction from cache
- Data prefetching to help hide memory latency
- Multiple simultaneous operations to eliminate stalls due to arithmetic unit pipelines
- Use of hardware features such as the SIMD arithmetic units, where appropriate

To achieve all the above, the first time a function from the library is called, a runtime check is performed to identify the hardware on which the program is running. Based on this check, a code path is chosen to maximize use of instruction- and-register level SIMD parallelism and to choose the best cache-blocking strategy. Intel MKL is also designed to be thread safe, which means that its functions operate correctly when simultaneously called from multiple application threads.

2.4.2 Parallelism:

Intel® MKL offers performance gains through parallelism provided by the symmetric multiprocessing performance (SMP) feature. You can obtain improvements from SMP in the following ways:

- One way is based on user-managed threads in the program and further distribution of the operations over the threads based on data decomposition, domain decomposition, control decomposition, or some other parallelizing technique. Each thread can use any of the Intel MKL functions because the library has been designed to be thread-safe.
- Another method is to use the FFT and BLAS level 3 routines. They have been parallelized and require no alterations of your application to gain the performance enhancements of multiprocessing. Performance using multiple processors on the level 3 BLAS shows excellent scaling. Since the threads

are called and managed within the library, the application does not need to be recompiled thread-safe.

2.5 Intel Cilk Plus

Intel Cilk Plus adds fine-grained task support to C and C++, making it easy to add parallelism to both new and existing software to efficiently exploit multiple processors and the vector instructions available on modern CPUs. It provides simple language extensions to express data and task parallelism to the C and C++ language implemented by the Intel C++ Compiler.

Intel Cilk Plus is made up of the following features:

- Task Parallelism
- Data Parallelism

2.5.1 Task Parallelism

1. Intel Cilk Plus adds three keywords to C and C++ to allow developers to express opportunities for parallelism:
   - Cilk_spawn: specifies that a function call can execute asynchronously, without requiring the caller to wait for it to return. This is an expression of an opportunity for parallelism, not a command that mandates parallelism. The Intel Cilk Plus runtime will choose whether to run the function in parallel with its caller.
   - Cilk_sync: specifies that all spawned calls in a function must complete before execution continues. There is an implied cilk_sync at the end of every function that contains a cilk_spawn.
   - Cilk_for: allows iterations of the loop body to be executed in parallel.

As stated above, the cilk_spawn and cilk_for keywords express opportunities for parallelism. Which portion of the application that actually runs in parallel is determined by Intel Cilk Plus runtime that implement task parallelism with an efficient work stealing scheduler.

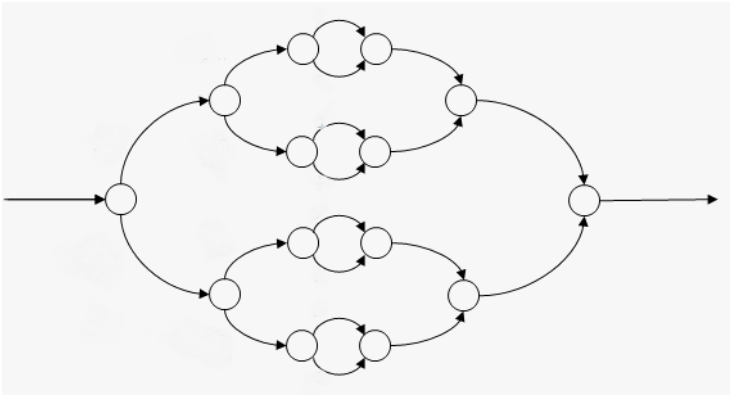


Figure 2.2: Cilk for example for multi-threads.

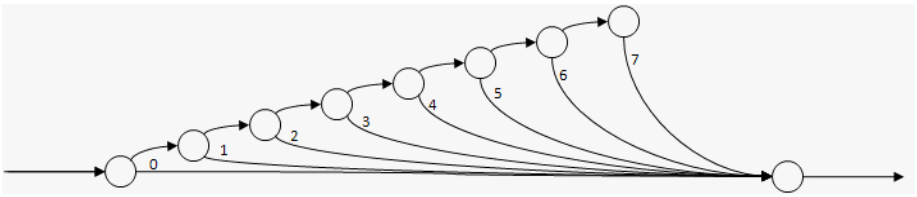


Figure 2.3: Example of Cilk spawn in a loop.

2. Reducers

   Intel Cilk Plus includes reducers to help make parallel programming easier. Traditional parallel programs use locks to protect shared variables, which can be problematic. Incorrect lock use can result in deadlocks. Contention for locked regions of code can slow a program down. And while locks can prevent races, there is no way to enforce ordering, resulting in non-deterministic results. Reducers provide a lock-free mechanism that allows parallel code to use private "views" of a variable which are merged at the next sync. The merge is done in an ordered manner to maintain the serial semantics of the Intel Cilk Plus application.

2.5.2 Data parallelism

1. Array notation: which provide data parallelism for sections/whole of array.

2. SIMD pragma: it is used to guide the compiler to vectorize more loops. Vectorization using the SIMD pragma complements (but does not replace) the fully automatic approach. [4] [5]

Chapter 3: Parallel Computing

3.1 Parallel computing vs Serial computing

Traditionally, software has been written for **serial computation** (see Figure 3.1):

- A problem is broken into a discrete series of instructions.
- Instructions are executed sequentially one after another.
- Executed on a single processor.
- Only one instruction may execute at any moment in time.

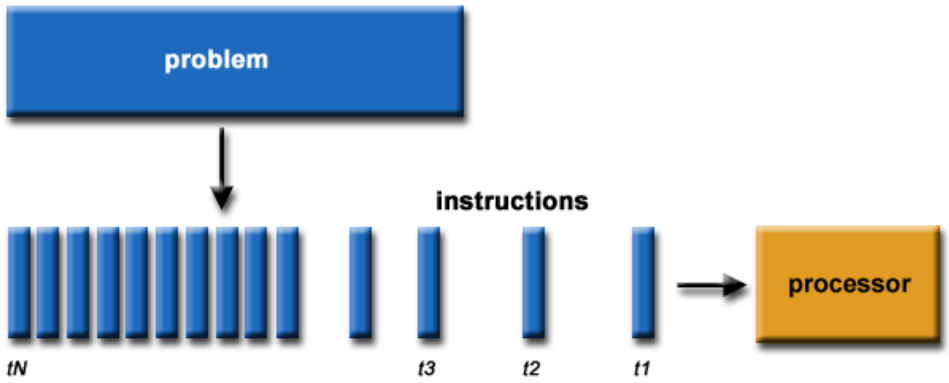


Figure 3.1: Serial Computing.

In the simplest sense, **parallel computing** (see Figure 3.2) is the simultaneous use of multiple compute resources to solve a computational problem [6]:

- A problem is broken into discrete parts that can be solved concurrently.
- Each part is further broken down to a series of instructions.
- Instructions from each part execute simultaneously on different processors.
- An overall control/coordination mechanism is employed.

The computational problem should be able to:

- Be broken apart into discrete pieces of work that can be solved simultaneously;
- Execute multiple program instructions at any moment in time;

- Be solved in less time with multiple compute resources than with a single compute resource.

The compute resources are typically:

- A single computer with multiple processors/cores.
- An arbitrary number of such computers connected by a network.

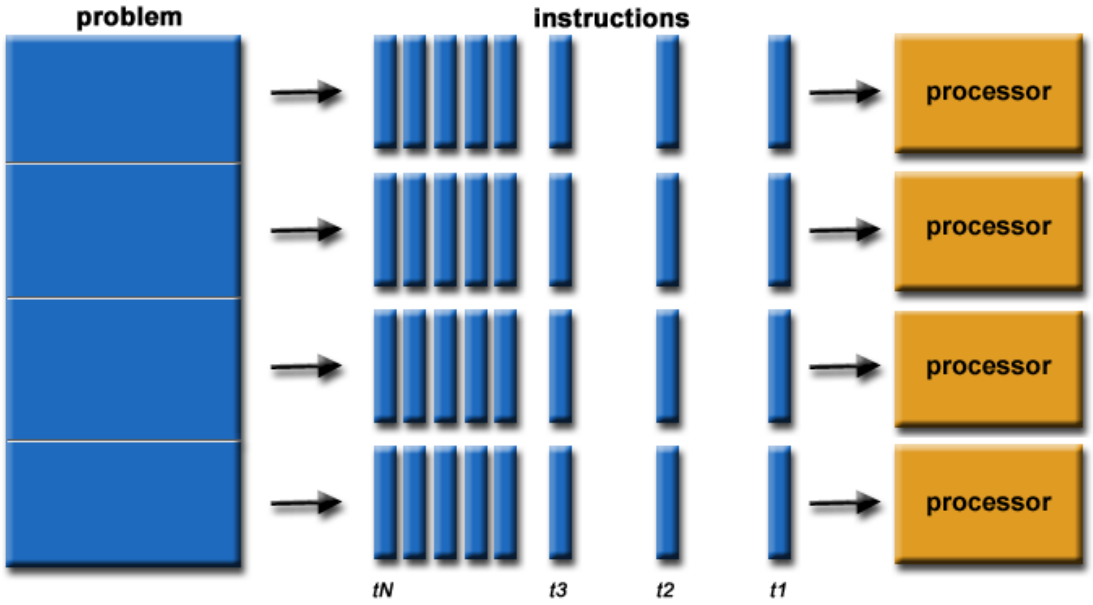


Figure 3.2: Parallel Computing.

3.2 Limits to serial computing

Both physical and practical reasons pose significant constraints to simply building ever faster serial computers. Today, billions of transistors can fit on a chip the size of a five pence coin. This has meant that every couple of years, devices become more powerful, smaller and cheaper. But designing a processor to deliver high performance is far more than just increasing the clock rate.

- Clock rates can't increase indefinitely
- Overheating
- Transmission delays
- Parasitic capacitance
- Slowness of the other system components

In order to overcome the problems caused by increasing CPU clock rate without affecting the performance, today's microprocessors focus on making the most effective use of each processing cycle instead of increasing processors clock speed which generate too much heat. For example, properly utilized CPU instruction sets can make some operations process more efficiently and reducing the amount of cycles needed to perform an operation, increasing the effectiveness of parallel processing, as well as increasing the efficiency of code execution.

3.3 Types of Parallel computing

There are several Types of Parallel Computing which are used worldwide.

- Bit-Level Parallelism.
- Instruction-Level Parallelism.
- Data Parallelism.
- Task parallelism.

3.4 Graphics Processing Unit

August 31, 1999 marks the introduction of the Graphics Processing Unit (GPU) for the PC industry. The technical definition of a GPU is "a single chip processor with integrated transform, lighting, triangle setup/clipping, and rendering engines that is capable of processing a minimum of 10 million polygons per second" [7].

The GPU's advanced capabilities were originally used primarily for 3D game rendering. But now those capabilities are being harnessed more broadly to accelerate computational workloads in areas such as financial modeling, cutting-edge scientific research and oil and gas exploration.

GPUs are optimized for taking huge batches of data and performing the same operation over and over very quickly, unlike PC microprocessors, which tend to skip all over the place.

Architecturally, the CPU is composed of just few cores with lots of cache memory that can handle a few software threads at a time. In contrast, a GPU is composed of hundreds of cores that can handle thousands of threads simultaneously (see Figure 3.3). The ability of a GPU with 100+ cores to process thousands of threads can

accelerate some software by 100x over a CPU alone. What's more, the GPU achieves this acceleration while being more power- and cost-efficient than a CPU [8].

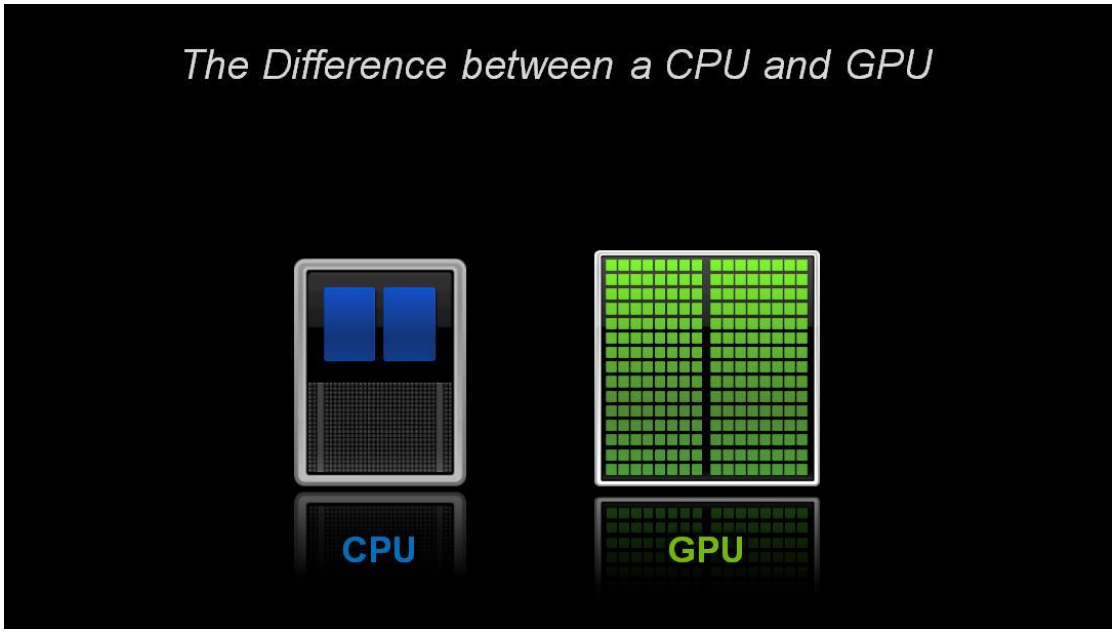


Figure 3.3: The difference between a CPU and GPU.

3.5 CUDA

CUDA is a parallel computing platform and programming model invented by NVIDIA that makes using a GPU for general purpose computing simple and elegant. The developer still programs in the familiar C, C++, Fortran, or an ever expanding list of supported languages, and incorporates extensions of these languages in the form of a few basic keywords.

Using CUDA allows the programmer to take advantage of the massive parallel computing power of an NVIDIA graphics card in order to do general purpose computation. CPUs like Intel Core 2 Duo and AMD Opteron are good at doing one or two tasks at a time, and doing those tasks very quickly. Graphics cards, on the other hand, are good at doing a massive number tasks at the same time, and doing those tasks relatively quickly.

To put this into perspective, suppose you have a 20-inch monitor with a standard resolution of 1920 x 1200. NVIDIA graphics card has the computational ability to

calculate the color of 2,304,000 different pixels, many times a second. In order to accomplish this feat, graphics cards use dozens, even hundreds of ALUs.

Fortunately, NVIDIA's ALUs are fully programmable, which enables us to harness an unprecedented amount of computational power into the programs that we write.

As stated previously, CUDA lets the programmer take advantage of the hundreds of ALUs inside a graphics processor, which is much more powerful than the handful of ALUs available in any CPU. However, this does put a limit on the types of applications that are well suited to CUDA.

3.5.1 When to Use CUDA

CUDA is only well suited for highly parallel algorithms: In order to run efficiently on a GPU, you need to have many hundreds of threads. Generally, the more threads you have, the better. If you have an algorithm that is mostly serial, then it does not make sense to use CUDA. Many serial algorithms do have parallel equivalents, but many do not. If you can't break your problem down into at least a thousand threads, then CUDA probably is not the best solution for you.

CUDA is extremely well suited for number crunching: If there is one thing that CUDA excels at, it's number crunching. The GPU is fully capable of doing 32-bit integer and floating point operations. In fact, it GPUs are more suited for floating point computations, which makes CUDA an excellent for number crunching. Some of the higher end graphics cards do have double floating point units, however there is only one 64-bit floating point unit for every 16 32-bit floating point units. So, using double floating point numbers with CUDA should be avoided if they aren't absolutely necessary for your application.

CUDA is well suited for large datasets: Most modern CPUs have a couple megabytes of L2 cache because most programs have high data coherency. However, when working quickly across a large dataset, say 500 Megabytes, the L2 cache may not be as helpful. The memory interface for GPUs is very different from the memory interface of CPUs. GPUs use massive parallel interfaces in order to connect with its memory. This type of interface is approximately 10 times faster than a typical CPU to memory interface, which is great.

3.5.2 CUDA Programming Model Basics

The CUDA programming model is a heterogeneous model in which both the CPU and GPU are used. In CUDA, the host refers to the CPU and its memory, while the device refers to the GPU and its memory. Code run on the host can manage memory on both the host and device, and also launches kernels which are functions executed on the device. These kernels are executed by many GPU threads in parallel. Given the heterogeneous nature of the CUDA programming model, a typical sequence of operations for a CUDA C program is:

1. Declare and Initialize host data.
2. Declare and allocate device memory.
3. Transfer data from the host to the device.
4. Execute one or more kernels.
5. Transfer results from the device to the host.

3.5.3 CUDA Limitations

- Unlike OpenCL, CUDA-enabled GPUs are only available from Nvidia.
- Copying between host and device memory may incur a performance hit due to system bus bandwidth and latency.
- Exception handling is not supported in CUDA code.
- Threads should be running in groups of at least 32 for best performance, with total number of threads numbering in the thousands.

Chapter 4: Communications

4.1 Introduction

Mobile communication has become an everyday commodity. In the last decades, it has evolved from being an expensive technology for a few selected individuals to today's ubiquitous systems used by a majority of the world's population. From the first experiments with radio communication by Guglielmo Marconi in the 1890s, the road to truly mobile radio communication has been quite long. To understand the complex mobile-communication systems of today, it is important to understand where they came from and how cellular systems have evolved. The task of developing mobile technologies has also changed, from being a national or regional concern to becoming an increasingly complex task undertaken by global standards-developing organizations such as the Third Generation Partnership Project (3GPP) and involving thousands of people.

Mobile communication technologies are often divided into generations (see Figure 4.1), with 1G being the analog mobile radio systems of the 1980s, 2G the first digital mobile systems, and 3G the first mobile systems handling broadband data. The next generation, 4G or Long-Term Evolution (LTE), provides even better support for mobile broadband. Further evolution steps of 4G LTE will be taken within the next few years. In a longer term perspective, around 2020 one may enter into what some would call "5G" radio access. This continuing race of increasing sequence numbers for mobile system generations is in fact just a matter of labels. What is important is the actual system capabilities and how they have evolved. [9]

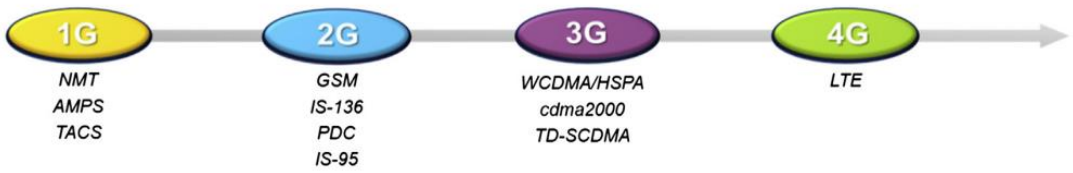


Figure 4.1: Generations of mobile communication systems.

4.2 Long Term Evolution (4G)

LTE or Long Term Evolution is the brand name given to the efforts of 3GPP 4th Generation technology development efforts mostly in Europe and UMB (Ultra-Mobile Broadband) is the brand name for similar efforts by 3GPP2 in North America.

LTE have been first time introduced in 3GPP Release 8. This essential evolution will enable networks to offer the higher data throughput to mobile terminals needed in order to deliver new and advanced mobile broadband services. The primary objectives of this network evolution are to provide these services with a quality at least equivalent to what an end-user can enjoy today using their fixed broadband access at home, and to reduce operational expenses. LTE is also referred to as E-UTRA (Evolved UMTS Terrestrial Radio Access) or E-UTRAN (Evolved UMTS Terrestrial Radio Access Network).

Although 3G/3.5G technologies such as HSPA/EV-DO deliver significantly higher bit rates than 2G technologies, they do not fully satisfy the wireless broadband requirements of instant-on, always-on and multi-megabit throughput. With LTE delivering even higher peak throughput and much lower latency, mobile operators (either 3GPP or 3GPP2 based) have a unique opportunity to evolve their existing infrastructure to next generation wireless networks.

4.3 LTE Requirements

LTE is focusing on an optimum support of Packet Switched (PS) services. Main requirements for the design of an LTE system were identified in the beginning of the standardization work on LTE in 2004. They can be summarized as follows [10]:

Data Rate: Peak data rates target 100 Mbps (downlink) and 50 Mbps (uplink) for 20 MHz spectrum allocation, assuming 2 receive antennas and 1 transmit antenna at the terminal.

Throughput: Target for downlink average user throughput per MHz is 3-4 times better than 3GPP Release 6. Target for uplink average user throughput per MHz is 2-3 times better than 3GPP Release 6.

Spectrum Efficiency: Downlink target is 3-4 times better than 3GPP Release 6. Uplink target is 2-3 times better than 3GPP Release 6. Table 4.1 summarizes the data rate and spectrum efficiency requirements set for LTE.

Table 4.1: Data rate and spectrum efficiency requirements defined for LTE.

Downlink (20 MHz)			Uplink (20 MHz)		
Unit	**Mbps**	**bps/Hz**	**Unit**	**Mbps**	**bps/Hz**
Requirement	100.0	5.0	**Requirement**	50.0	2.5
2×2 MIMO	172.8	8.6	**16QAM**	57.6	2.9
4×4 MIMO	326.4	16.3	**64QAM**	86.4	4.3

Latency: User plane latency. The one-way transit time between a packet being available at the IP layer in either the device or radio access network and the availability of this packet at IP layer in the radio access network/device shall be less than 30 ms.

Control plane latency. Also C-plane, that means the time it takes to transfer the device from a passive connection with the network (IDLE state) to an active connection (CONNECTED state) shall be further reduced, e.g. less than 100 ms to allow fast transition times.

Bandwidth: LTE supports a subset of bandwidths of 1.4, 3, 5, 10, 15 and 20 MHz.

Mobility: The system should be optimized for low mobile speed (0-15 km/h), but higher mobile speeds shall be supported as well including high speed train environment as special case.

Spectrum allocation: Operation in paired (Frequency Division Duplex / FDD mode) and unpaired spectrum (Time Division Duplex / TDD mode) is possible.

Quality of Service: End-to-end Quality of Service (QoS) shall be supported. Voice over Internet Protocol (VoIP) should be supported with at least as good radio and backhaul efficiency and latency as voice traffic over the UMTS circuit switched networks.

4.4 LTE Advantages

- IP network based protocol
- Higher data rate
- UE battery life enhancement
- Total user capacity increase
- Quality improvement

4.5 Interference

One of the most challenging issues facing wireless communication systems is the interference, Interference is anything which modifies, or disrupts a signal as it travels along a channel between a source and a receiver. The term typically refers to the addition of unwanted signals to a useful signal.

There are two types of interference, which are:

- Inter-symbol Interference
- Inter Channel Interference

4.6 Multiplexing & Multiple Access Techniques

Multiplexing is a way of sending multiple signals or streams of information over a communications link at the same time in the form of a single complex signal, the receiver recovers the separate signals (Demultiplexing). Two basic forms of multiplexing are Time Division Multiplexing (TDM), and Frequency Division Multiplexing (FDM).

Multiple access is a technique that lets multiple mobile users share the allotted spectrum in the most effective manner. Since the spectrum is limited, the sharing is necessary to improve the overall capacity over a geographical area. This is carried out by permitting the available bandwidth to be used simultaneously by different users.

One of the key elements of LTE is the use of OFDM (Orthogonal Frequency Division Multiplexing) as the signal bearer, as well as OFDM's associated access schemes, OFDMA (Orthogonal Frequency Division Multiple Access) and SC-FDMA (Single Carrier Frequency Division Multiple Access).

4.6.1 OFDM

OFDM (Orthogonal Frequency Division Multiplexing) is a form of signal modulation is being used for many of the latest wireless and telecommunications standards which divides a high data rate modulating stream placing them onto many slowly modulated narrowband close-spaced subcarriers, and in this way is less sensitive to frequency selective fading as they have a bandwidth smaller than the mobile channel coherence bandwidth. This obviates the need for complex frequency equalizers which are featured in 3G technologies. The sub-carriers are mutually orthogonal in the frequency domain which mitigates Inter-Symbol Interference (ISI) as shown in Figure 4.2.

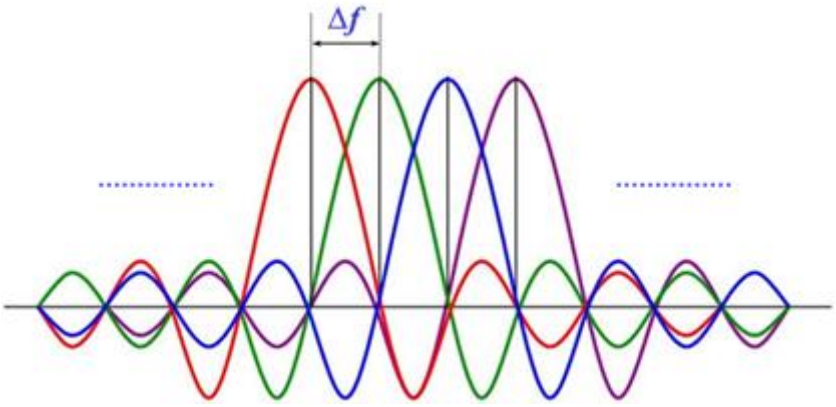


Figure 4.2: OFDM Spectrum.

The data to be transmitted on an OFDM signal is spread across the subcarriers of the signal, each subcarrier taking part of the payload. This reduces the data rate taken by each subcarrier. The lower data rate has the advantage that interference from reflections is much less critical. This is achieved by adding a guard band time or guard interval into the system.

Input bits are first grouped and assigned for transmission over different frequencies (sub-carriers) and then summed up this can be done using IFFT (Inverse Fast Fourier Transformation) and the resulting signal could then be sent over the air as shown in Figure 4.3.

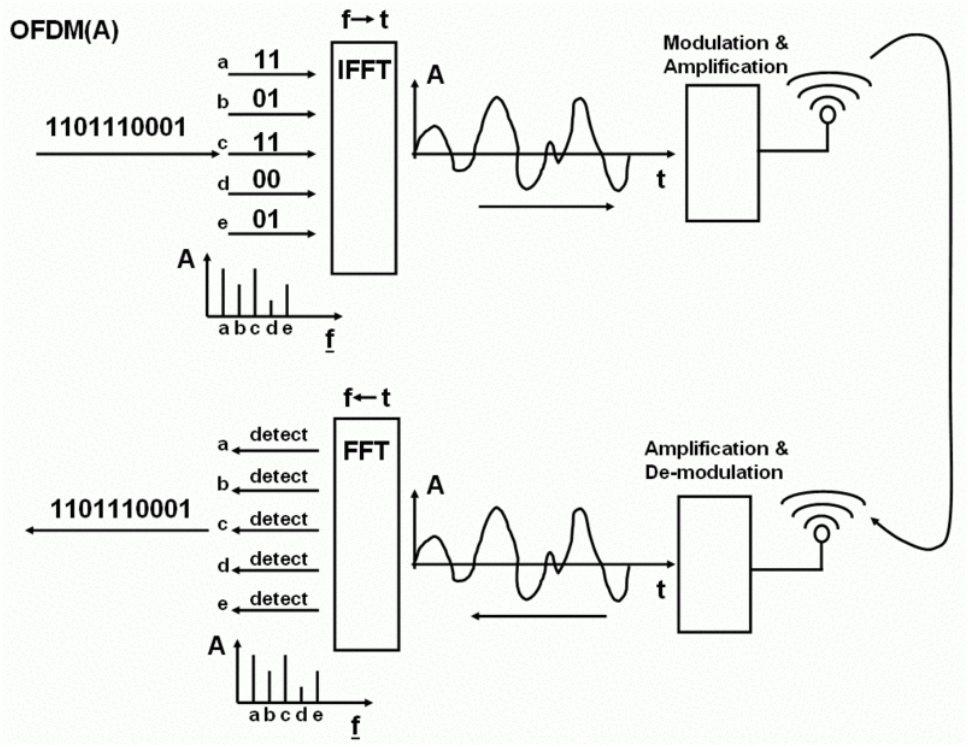


Figure 4.3: OFDM(A) Architecture.

OFDM has been adopted in the Wi-Fi arena where the standards like 802.11a, 802.11n, 802.11ac and more. It has also been chosen for the cellular telecommunications standard LTE / LTE-A, and in addition to this it has been adopted by other standards such as WiMAX. [11]

4.6.1.1 Cyclic Prefix

In telecommunications, the term cyclic prefix refers to the prefixing of a symbol with a repetition of the end. Although the receiver is typically configured to discard the cyclic prefix samples, the cyclic prefix serves two purposes.

- As a guard interval, it eliminates the intersymbol interference from the previous symbol.
- As a repetition of the end of the symbol, it allows the linear convolution of a frequency-selective multipath channel to be modelled as circular convolution, which in turn may be transformed to the frequency domain using a discrete Fourier transform. This approach allows for simple frequency-domain processing, such as channel estimation and equalization.

In order for the cyclic prefix to be effective (i.e. to serve its aforementioned objectives), the length of the cyclic prefix must be at least equal to the length of the multipath channel. Different OFDM cyclic prefix lengths are available in various systems. For example, within LTE a normal length and an extended length are available and after Release 8 a third extended length is also included, although not normally used.

Although the concept of cyclic prefix has been traditionally associated with OFDM systems, the cyclic prefix is now also used in single carrier systems to improve the robustness to multipath propagation.

Cyclic prefix disadvantage is that it takes up system capacity and reduces the overall data rate as the cyclic prefix re-transmits data that is already being transmitted. [12]

4.6.1.2 OFDM Advantages & Disadvantages

OFDM has been used in many high data rate wireless systems because of the many advantages it provides [11]:

- ***Immunity to selective fading:*** OFDM is more resistant to frequency selective fading than single carrier systems because it divides the overall channel into multiple narrowband signals that are affected individually as flat fading sub-channels.
- ***Resilience to Bandlimited Interference:*** Interference appearing on a channel may be bandwidth limited and in this way will not affect all the sub-channels. This means that not all the data is lost.
- ***Spectrum efficiency:*** Using close-spaced overlapping sub-carriers, a significant OFDM advantage is that it makes efficient use of the available spectrum.
- ***Resilient to ISI:*** This results from the low data rate on each of the sub-channels.
- ***Simpler channel equalization:*** One of the issues with CDMA systems was the complexity of the channel equalization which had to be applied across the whole channel. An advantage of OFDM is that using multiple sub-channels, the channel equalization becomes much simpler.

- ***Data rate Optimization:*** Allows optimization of data rates for all users in a cell by transmitting on the best (i.e. non-faded) subcarriers for each user.

This last feature is the fundamental aspect of OFDMA, the use of OFDM technology to multiplex traffic by allocating specific patterns of sub-carriers in the time-frequency space to different users. In addition to data traffic, control channels and reference symbols can be interspersed.

Whilst OFDM has been widely used, there are still a few disadvantages to its use which need to be addressed when considering its use:

- ***High peak to average power ratio:*** An OFDM signal has a noise like amplitude variation and has a relatively high large dynamic range, or peak to average power ratio. This impacts the RF amplifier efficiency as the amplifiers need to be linear and accommodate the large amplitude variations and these factors mean the amplifier cannot operate with a high efficiency level.

- ***Sensitive to carrier offset and drift:*** Another disadvantage of OFDM is that is sensitive to carrier frequency offset and drift. Single carrier systems are less sensitive.

4.6.2 SC-FDMA

For the LTE uplink, a different concept is used for the access technique. Although still using a form of OFDMA technology, the implementation is called Single Carrier Frequency Division Multiple Access (SC-FDMA).

One of the key parameters that affects all mobiles is that of battery life. Even though battery performance is improving all the time, it is still necessary to ensure that the mobiles use as little battery power as possible. With the RF power amplifier that transmits the radio frequency signal via the antenna to the base station being the highest power item within the mobile, it is necessary that it operates in as efficient mode as possible. This can be significantly affected by the form of radio frequency modulation and signal format. Signals that have a high peak to average ratio and require linear amplification do not lend themselves to the use of efficient RF power amplifiers. As a result, it is necessary to employ a mode of transmission that has as near a constant power level when operating. Unfortunately, OFDM has a high peak to average ratio. While this is not a problem for the base station where power is not a

particular problem, it is unacceptable for the mobile. As a result, LTE uses a modulation scheme known as SC-FDMA - Single Carrier Frequency Division Multiple Access which is a hybrid format. This combines the low peak to average ratio offered by single-carrier systems with the multipath interference resilience and flexible subcarrier frequency allocation that OFDM provides [13].

Despite its name, Single Carrier Frequency Division Multiple Access (SC-FDMA) also transmits data over the air interface in many sub-carriers but adds an additional processing step as shown in Figure 4.4. Instead of putting M bits together (e.g. 4 bits representing a 16-QAM modulation) as in the OFDM to form the signal for one sub-carrier, the additional processing block in SC-FDMA spreads the information of each bit over all the sub-carriers. This is done as follows: Again, a number of bits (e.g. 4 bits) are grouped together. In OFDM, these groups of bits would have been the input of the IFFT. In SC-FDMA, however, these bits are now piped into a Fast Fourier Transformation (FFT) function first. The output of the process is the basis for the creation of the sub-carriers for the following IFFT. As not all sub-carriers are used by the mobile station, many of them are set to zero in the diagram. These may or may not be used by other mobile stations.

On the receiver side the signal is demodulated, amplified and treated by the Fast Fourier Transformation function in the same way as in OFDMA. The resulting amplitude diagram, however, is now not analyzed straight away to get the original data stream but fed to the Inverse Fast Fourier Transformation function to remove the effect of the additional signal processing originally done at the transmitter side. The result of the IFFT is again a time domain signal. The time domain signal is now fed to a single detector block which recreates the original bits. Thus, instead of detecting the bits on many different sub-carriers, only a single detector is used on a single carrier [14].

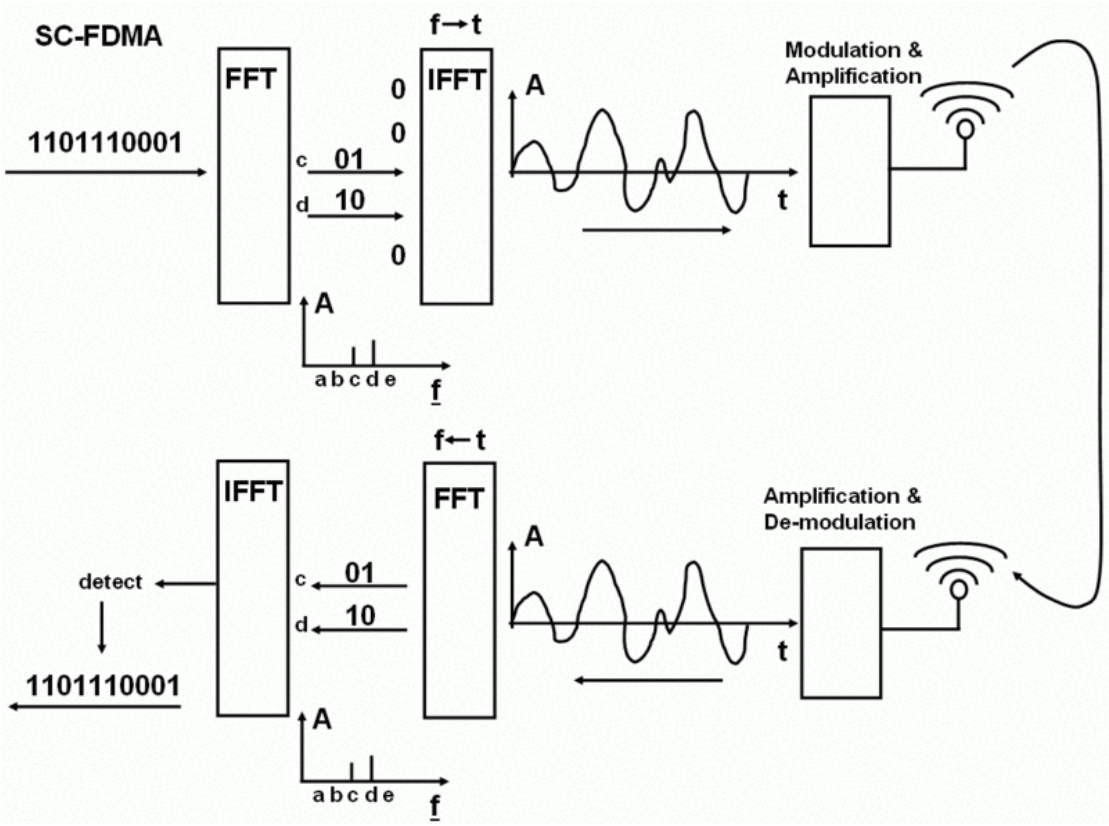


Figure 4.4: SC-FDMA Architecture.

4.7 Physical layer parameters

There are two types of frame structure in the LTE standard, Type 1 and Type 2. Type 1 uses Frequency Division Duplexing (FDD) in which uplink and downlink are separated by frequency. Type 2 uses Time Division Duplexing (TDD) in which uplink and downlink are separated in time.

For full-duplex FDD, uplink and downlink frames are separated by frequency and are transmitted continuously and synchronously as shown in the figure below.

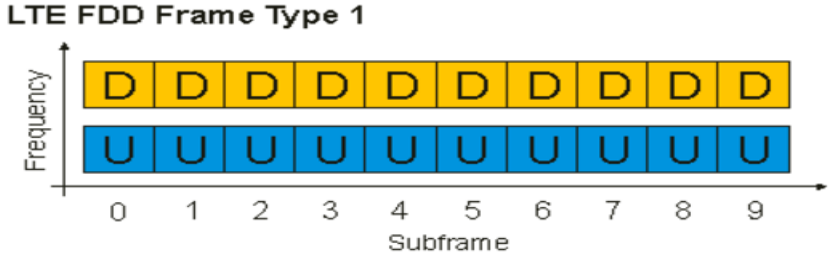


Figure 4.5: LTE FDD frame (Type 1).

In half-duplex FDD operation, the UE cannot transmit and receive at the same time.

For TDD mode, the uplink and downlink subframes are transmitted on the same frequency and are multiplexed in the time domain. The locations of the uplink, downlink and special subframes are determined by the uplink-downlink configuration.

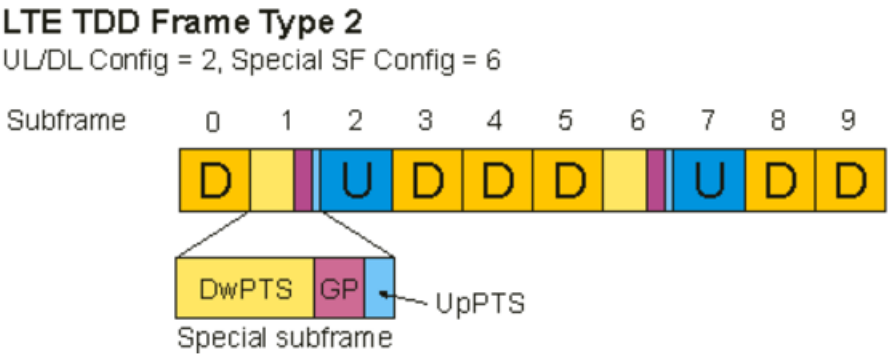


Figure 4.6: LTE TDD Frame (Type 2).

The figure above shows a radio frame with a special subframe in the 2nd and 6th subframes. Special subframes are used for switching from downlink to uplink and contain three sections: DwPTS (downlink pilot time slot), GP (guard period), and UpPTS (uplink pilot time slot). [15]

The radio frame has a length of $10\,ms$ ($T_{frame} = 307200 * T_s$). Each frame is divided into 10 equally sized subframes of $1\,ms$. Each subframe consists of 2 equally sized slots of $0.5\,ms$. Each slot in turn consists of a number of OFDM symbols which can be either 7 (for normal cyclic prefix) or 6 (for extended cyclic prefix). T_s (sampling time) expresses the basic time unit for LTE, corresponding to a sampling frequency of $30.72\,MHz$. This sampling frequency is given due to the defined subcarrier spacing for LTE with $\Delta f = 15\,kHz$ and the maximum size for FFT to generate the OFDM symbols is 2048, $F_s = 2048 * 15\,kHz = 30.72\,MHz$. [10]

The figure below shows the structure of the uplink resource grid for both FDD and TDD. The horizontal axis shows the time domain of the subframe, i.e. the SC-FDMA symbols in the subframe. While the vertical axis shows the frequency domain of one resource block.

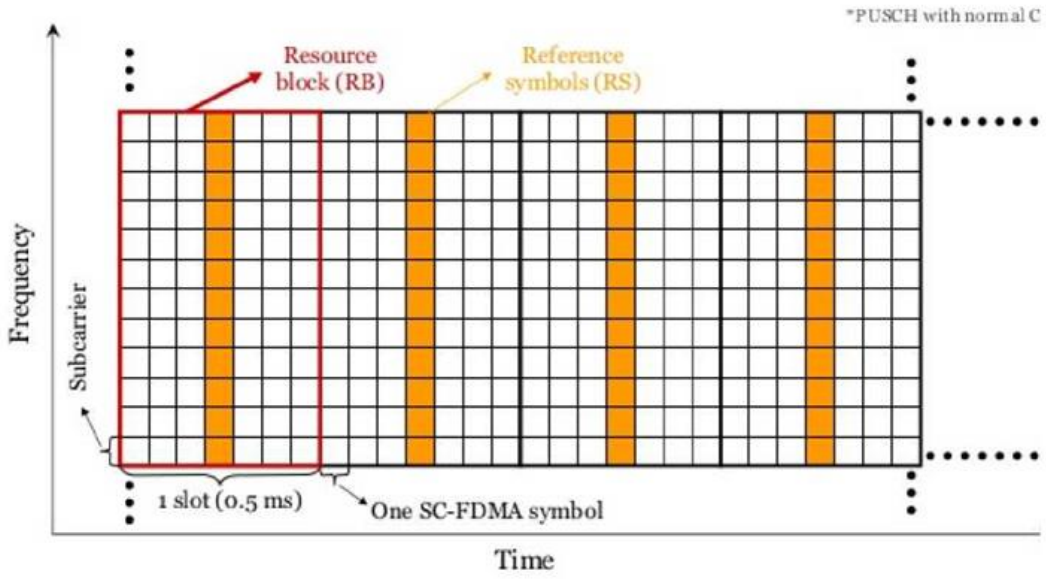


Figure 4.7: LTE uplink grid.

The smallest unit of resource is the Resource Element (RE) which consists of one SC-FDMA data block length on one sub-carrier. Resource element contains a single complex value representing data from a physical channel or signal. A resource block consists of 12 REs for the duration of a slot $(0.5\ ms)$. The resource block (RB) is the smallest unit that can be allocated to a user. In frequency, resource blocks are either 12 subcarriers x 15 kHz subcarriers spacing or 24 subcarriers x 7.5 kHz subcarriers spacing. The resource block is 180 kHz wide in frequency and 1 slot long in time. [16]

Data is allocated to a device (User Equipment, UE) in terms of resource blocks. one UE can be allocated integer multiples of one resource block in the frequency domain. Each UE can use resource blocks assigned to it for a transmission time interval (TTI) of $1ms$. All scheduling decisions for downlink and uplink are done in the base station (eNodeB). [10]

4.7.1 Demodulation reference signal (DMRS)

Uplink demodulation reference signals are used for channel estimation in the eNodeB receiver for coherent demodulation of the Physical Uplink Shared Channel (PUSCH) to which the UL-SCH transport channel is mapped, as well as for the Physical Uplink Control Channel (PUCCH), which carries different types of L1/L2 control signaling. It is located on the 4th symbol in each slot (for normal cyclic prefix)

and spans the same bandwidth as the allocated uplink data. The basic structure for demodulation reference signals is the same for PUSCH and PUCCH transmission, although there are some differences - for example, in terms of the exact set of SC-FDMA symbols in which the reference signals are transmitted. [10]

4.7.2 Bandwidth

The bandwidths defined by the standard are $1.4, 3, 5, 10, 15, and\ 20\ MHz$. The table below shows how many subcarriers and resource blocks in each bandwidth for uplink. [17]

Table 4.2: Number of resource blocks for different LTE bandwidths.

Channel Bandwidth	1.4	3	5	10	15	20
Number of resource blocks	6	15	25	50	75	100

4.8 Channel coding, and interleaving

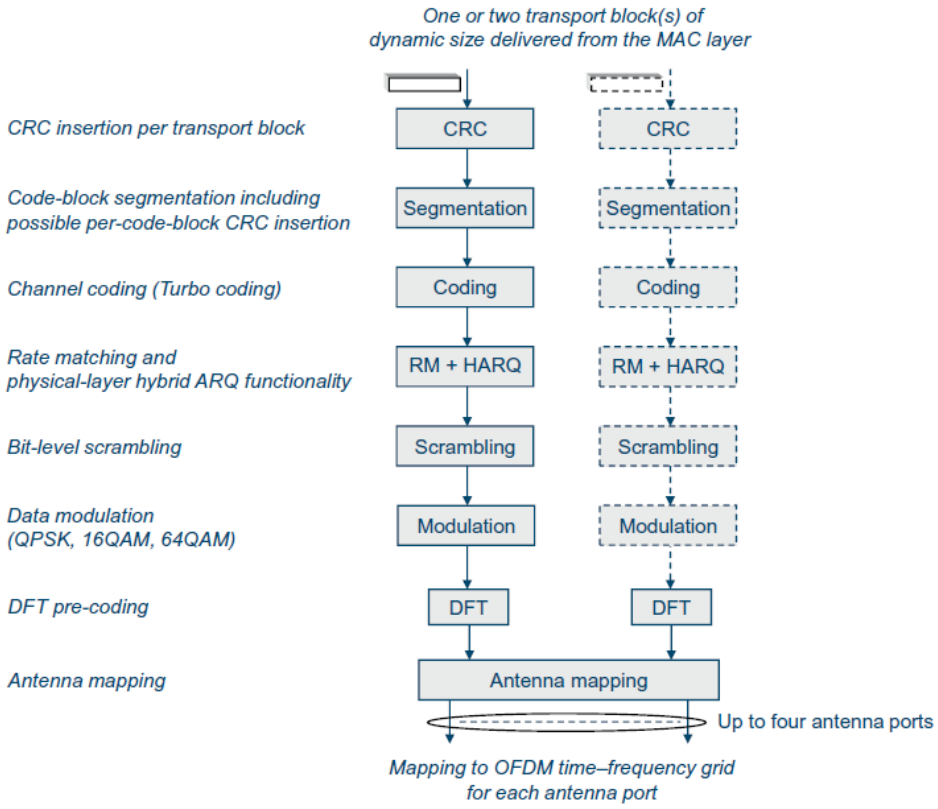


Figure 4.8: Physical-layer processing for UL-SCH.

Data and control streams from/to MAC layer are encoded /decoded to offer transport and control services over the radio transmission link. Channel coding scheme is a combination of error detection, error correcting, rate matching, interleaving and transport channel or control information mapping onto/splitting from physical channels.

4.8.1 Channel coding

The bit sequence input for a given code block to channel coding is denoted by $c_0, c_1, c_2, c_3, ..., c_{K-1}$, where K is the number of bits to encode. After encoding, the bits are denoted by $d_0^{(i)}, d_1^{(i)}, d_2^{(i)}, d_3^{(i)}, ..., d_{D-1}^{(i)}$, where D is the number of encoded bits per output stream and i indexes the encoder output stream. The relation between c_k and $d_k^{(i)}$ and between K and D is dependent on the channel coding scheme.

The following channel coding schemes can be applied to TrCHs:

- tail biting convolutional coding;
- turbo coding.

Usage of coding scheme and coding rate for the different types of TrCH is shown in Table 4.3. Usage of coding scheme and coding rate for the different control information types is shown in Table 4.4.

The values of D in connection with each coding scheme:

- tail biting convolutional coding with rate 1/3: $D = K$;
- turbo coding with rate 1/3: $D = K + 4$.

The range for the output stream index i is 0, 1 and 2 for both coding schemes.

Table 4.3: Usage of channel coding scheme and rate for TrCHs

TrCH	Coding scheme	Coding rate
UL-SCH	Turbo coding	1/3
DL-SCH		
PCH		
MCH		
BCH	Tail biting convolutional coding	1/3

Table 4.4: Usage of channel coding scheme and rate for control information

Control Information	Coding scheme	Coding rate
DCI	Tail biting convolutional coding	1/3
CFI	Block code	1/16
HI	Repetition code	1/3
UCI	Block code	variable
	Tail biting convolutional coding	1/3

Tail biting convolutional coding

A tail biting convolutional code with constraint length 7 and coding rate 1/3 is defined.

The configuration of the convolutional encoder is presented in Figure 4.9.

The initial value of the shift register of the encoder shall be set to the values corresponding to the last 6 information bits in the input stream so that the initial and final states of the shift register are the same. Therefore, denoting the shift register of the encoder by $s_0, s_1, s_2, ..., s_5$, then the initial value of the shift register shall be set to

$$s_i = c_{(K-1-i)}$$

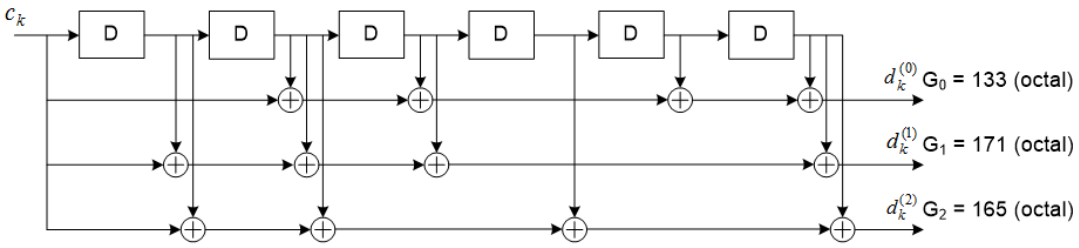


Figure 4.9: Rate 1/3 tail biting convolutional encoder

The encoder output streams $d_k^{(0)}$, $d_k^{(1)}$ and $d_k^{(2)}$ correspond to the first, second and third parity streams, respectively as shown in Figure 4.9.

4.8.2 Interleaver

Interleaving is the reordering of data that is to be transmitted so that consecutive bytes of data are distributed over a larger sequence of data to reduce the effect of burst errors. The use of interleaving greatly increases the ability of error protection codes to correct for burst errors. Many of the error protection coding processes can correct for small numbers of errors, but cannot correct for errors that occur in groups.

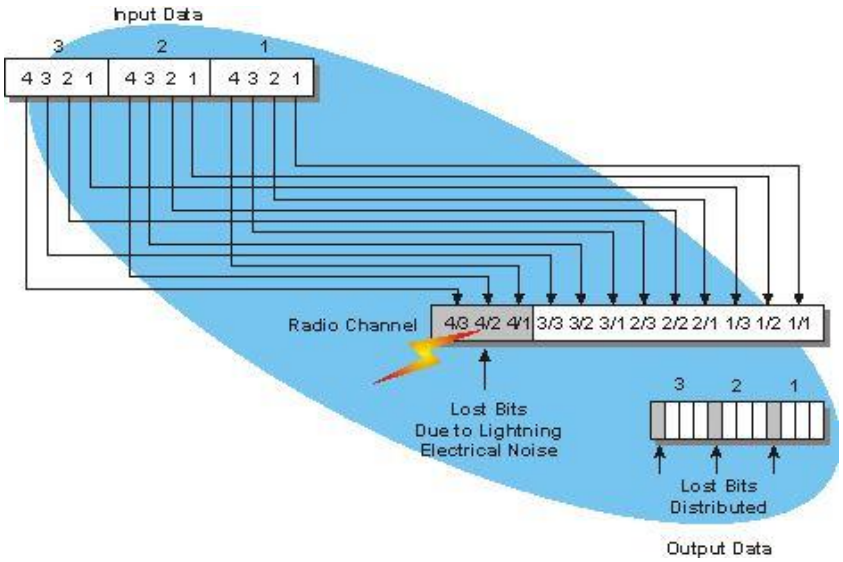


Figure 4.10: Interleaving Operation.

This diagram shows that a block of data information may be distributed over multiple time slots or frames in a carrier line to distribute the effect of burst errors on the information signal. In this example, a block of digital audio is being transmitted through a radio channel. The digital audio is divided into blocks of 4 bits and the bits for each block are distributed (interleaved) over a communication channel. During the transmission, a lightning bolt creates a burst of electrical noise that disrupts 3 bits of data transmission. Because these bits are interleaved, the received data has burst errors that are distributed. This allows the audio to be continuously heard with a marginal amount of distortion instead of completely losing the audio during the burst errors. [18]

The inputs to the interleaver are:

1. Data Bits.
2. RI (Rank Information) Control Bits.

3. ACK (Acknowledgement) Control Bits.

The interleaving operation is done as in section 5.2.2.8 in [19].

4.9 Physical uplink shared channel bit level processing

The baseband signal representing the physical uplink shared channel is defined in terms of the following steps:

- scrambling
- modulation of scrambled bits to generate complex-valued symbols
- transform precoding to generate complex-valued symbols
- mapping of complex-valued symbols to resource elements
- generation of complex-valued time-domain SC-FDMA signal for each antenna port

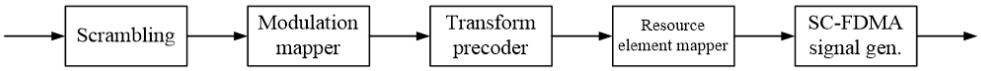


Figure 4.11: Overview of uplink physical channel processing.

4.9.1 Scrambler

Scramblers are circuits that pseudo-randomly change the values of bits in a data block or stream with the purpose of "whitening" its spectrum (spread it so that no strong spectral component will exist, thus reducing electromagnetic interference) or to introduce security (as part of an encryption procedure).

The pseudo-randomness is normally accomplished using an LFSR circuit. In this case, a scrambler is just an LFSR plus an additional modulo-2 adder (XOR gate), and it is specified using the LFSR's characteristic polynomial.

The algorithm of bit level scrambling is mentioned in section 5.3.1 in [20].

4.9.2 Modulation Mapper

The block of scrambled bits $\tilde{b}(0),...,\tilde{b}(M_{\text{bit}}-1)$ shall be modulated resulting in a block of complex-valued symbols $d(0),...,d(M_{\text{symb}}-1)$. the modulation mappings applicable for the physical uplink shared channel are.

- Binary PSK (BPSK), using 2 symbols
- Quadrature PSK (QPSK), using 4 symbols
- 16QAM, using 16 symbols
- 64QAM, using 64 symbols

The modulation mapper takes binary digits, 0 or 1, as input and produces complex-valued modulation symbols, $x=I+jQ$, as output.

4.9.3 Transform precoding

The block of complex-valued symbols $d(0),...,d(M_{\text{symb}}-1)$ is divided into $M_{\text{symb}}/M_{\text{sc}}^{\text{PUSCH}}$ sets, each corresponding to one SC-FDMA symbol. Transform precoding shall be applied according to

$$z(l\cdot M_{\text{sc}}^{\text{PUSCH}}+k)=\frac{1}{\sqrt{M_{\text{sc}}^{\text{PUSCH}}}}\sum_{i=0}^{M_{\text{sc}}^{\text{PUSCH}}-1}d(l\cdot M_{\text{sc}}^{\text{PUSCH}}+i)e^{-j\frac{2\pi ik}{M_{\text{sc}}^{\text{PUSCH}}}}$$
$$k=0,...,M_{\text{sc}}^{\text{PUSCH}}-1$$
$$l=0,...,M_{\text{symb}}/M_{\text{sc}}^{\text{PUSCH}}-1$$

resulting in a block of complex-valued symbols $z(0),...,z(M_{\text{symb}}-1)$. The variable $M_{\text{sc}}^{\text{PUSCH}}=M_{\text{RB}}^{\text{PUSCH}}\cdot N_{\text{sc}}^{\text{RB}}$, where $M_{\text{RB}}^{\text{PUSCH}}$ represents the bandwidth of the PUSCH in terms of resource blocks, and shall fulfil

$$M_{\text{RB}}^{\text{PUSCH}}=2^{\alpha_2}\cdot 3^{\alpha_3}\cdot 5^{\alpha_5}\le N_{\text{RB}}^{\text{UL}}$$

where $\alpha_2,\alpha_3,\alpha_5$ is a set of non-negative integers.

4.9.4 Mapping to physical resources

The block of complex-valued symbols $z(0),...,z(M_{\text{symb}}-1)$ shall be multiplied with the amplitude scaling factor β_{PUSCH} in order to conform to the transmit power P_{PUSCH} and mapped in sequence starting with $z(0)$ to physical resource blocks assigned for transmission of PUSCH. The mapping to resource elements (k,l) corresponding to the physical resource blocks assigned for transmission and not used for transmission of reference signals and not reserved for possible SRS transmission shall be in increasing order of first the index k, then the index l, starting with the first slot in the subframe.

If uplink frequency-hopping is disabled, the set of physical resource blocks to be used for transmission are given by $n_{\mathrm{PRB}} = n_{\mathrm{VRB}}$ where n_{VRB} is obtained from the uplink scheduling grant as described in Section 8.1 in [21].

For simplicity we assumed that uplink frequency-hopping is disabled.

4.9.5 SC-FDMA baseband signal generation

This section applies to all uplink physical signals and physical channels except the physical random access channel.

The time-continuous signal $s_l(t)$ in SC-FDMA symbol l in an uplink slot is defined by

$$s_l(t) = \sum_{k=-\lfloor N_{\mathrm{RB}}^{\mathrm{UL}} N_{\mathrm{sc}}^{\mathrm{RB}}/2 \rfloor}^{\lceil N_{\mathrm{RB}}^{\mathrm{UL}} N_{\mathrm{sc}}^{\mathrm{RB}}/2 \rceil - 1} a_{k^{(-)},l} \cdot e^{j2\pi(k+1/2)\Delta f\left(t - N_{\mathrm{CP},l} T_{\mathrm{s}}\right)}$$

for $0 \le t < (N_{\mathrm{CP},l} + N) \times T_{\mathrm{s}}$ where $k^{(-)} = k + \lfloor N_{RB}^{UL} N_{sc}^{RB}/2 \rfloor$, $N = 2048$, $\Delta f = 15\,\mathrm{kHz}$ and $a_{k,l}$ is the content of resource element (k,l).

The SC-FDMA symbols in a slot shall be transmitted in increasing order of l, starting with $l = 0$, where SC-FDMA symbol $l > 0$ starts at time $\sum_{l'=0}^{l-1} (N_{\mathrm{CP},l'} + N) T_{\mathrm{s}}$ within the slot.

Table 4.5 lists the values of $N_{\mathrm{CP},l}$ that shall be used. Note that different SC-FDMA symbols within a slot may have different cyclic prefix lengths.

Table 4.5: SC-FDMA parameters.

Configuration	Cyclic prefix length $N_{CP,l}$
Normal cyclic prefix	160 for $l = 0$ 144 for $l = 1,2,...,6$
Extended cyclic prefix	512 for $l = 0,1,...,5$

4.10 Decoding of convolutional encoding using Viterbi algorithm

Several algorithms exist for decoding convolutional codes. For relatively small values of k, the Viterbi algorithm is universally used as it provides maximum

likelihood performance and is highly parallelizable. Viterbi decoders are thus easy to implement in software on CPUs with SIMD instruction sets.

Longer constraint length codes are more practically decoded with any of several sequential decoding algorithms, of which the “Fano algorithm” is the best known.

The Viterbi algorithm is very suitable for the requirements of the project for its parallelization capability.

4.10.1 Viterbi overview:

The Viterbi algorithm is best illustrated using the trellis diagram. The trellis diagram is just a way to show the transition from one state to another state when the input to the convolutional encoder during time evolution. First, we will show the trellis diagram when encoding the bits.

The trellis diagram is shown in Figure 4.12:

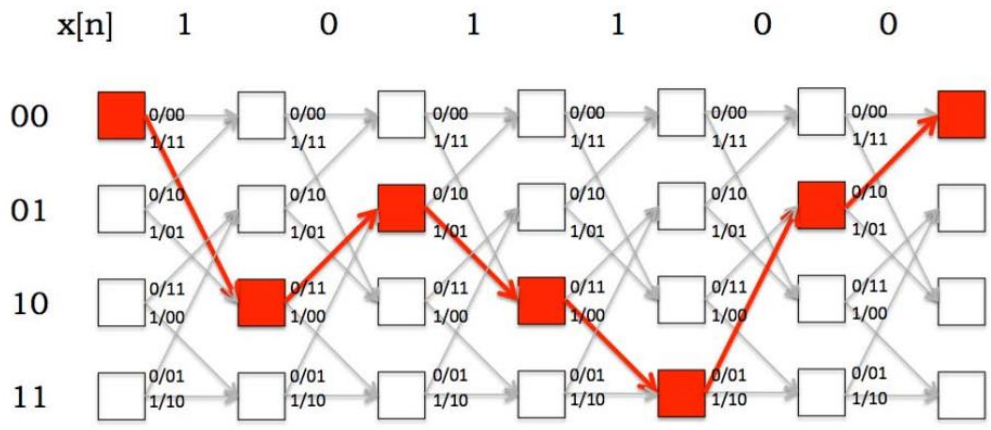


Figure 4.12: A trellis diagram showing the time evolution of the state machine.

In Figure 4.12, the x[n] represents the input to the convolutional encoder at each time clock.

The trellis diagram above corresponds to a convolutional encoder with 2 shift registers and coding rate = $\frac{1}{2}$

Each box represents one of the states for the shift registers. Therefore, we have 4 states which mean 4 vertical boxes. When the input to the convolutional encoder is

“1’, the state in the 1st stage will change from “00” to “10”. The input then completes forming a path and emitting a codeword of 2 bits at each transition.

At the receiver, suppose we now receive a sequence of bits representing the output from the encoder. Then, there will be some path through the trellis that would exactly match the received sequence. But, when there are bit errors, we need to find the *most likely* transmitted message sequence. If we can come up with a way to capture the errors introduced by going from one state to the next, then we can accumulate the errors along a path and come up with an estimate of the total number of errors along the path. Then, the path with the smallest such accumulation of errors is the path we want and it represents the most likely transmitted message.

The Viterbi decoder solves these problems by computing the cost (errors) when going from one state to another. Then, the costs are accumulated and the path with the least cost represents the most likely transmitted message.

4.10.2 Viterbi algorithm Parameters:

The decoding algorithm has two metrics: the branch metric (BM) and the path metric (PM). The branch metric is a measure of the "distance" between what was transmitted and what was received, and is defined for each arc in the trellis. In hard decision decoding, where we are given a sequence of digitized parity bits, the branch metric is the *Hamming distance* between the expected parity bits and the received ones.

Path metric is defined for hard decision and it corresponds to the Hamming distance with respect to the received parity bit sequence over the most likely path from the initial state to the current state in the trellis. By “most likely”, we mean the path with smallest Hamming distance between the initial state and the current state, measured over all possible paths between the two states.

4.10.3 Steps of Viterbi algorithm:

1- Calculating the branch metric for all arcs in the trellis.

As stated previously we will calculate the hamming distance between both the received sequence and all possible values coming out from the Viterbi decoder at all states.

2- ACS calculation (**A**dd, **C**ompare, and **S**hift).

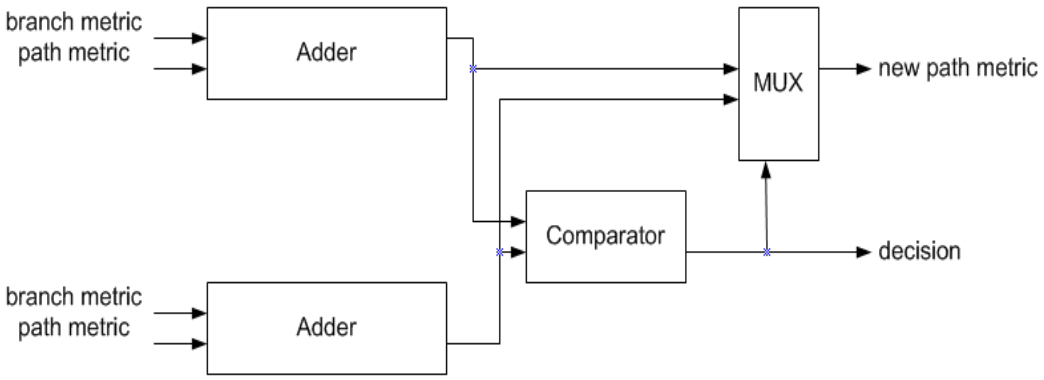


Figure 4.13: Block diagram of ACS.

As shown in Figure 4.13 we can sum up the ACS algorithm in each stage in the trellis as follows:

Add the predecessor state path metric to the current state branch metric for both predecessor states. **Compare** between the cost of the two addition results. By "cost", we mean the Hamming distance. **Select** the minimum cost to be the new path metric for the current state which will be predecessor in the next iteration.

Mathematically, at any state 'S' at time 'N':

$$PM[S,N] = \min(PM[\alpha, N-1] + BM[\alpha, N-1], PM[\beta, N-1] + BM[\beta, N-1])$$

PM[S, N]: path metric at state S at time N.

α and β are the two predecessor states.

3- Finding the most likely path.

After all data have been processed, we must find the most likely path through the trellis done by searching through the final “column” in PM and selecting the most likely (lowest cost) operation done infrequently compared to other calculations.

4- Traceback and Estimating the output.

In the 4th step we will find the total path from the final minimum cost state that we have already found it in the 3rd step to the first state.

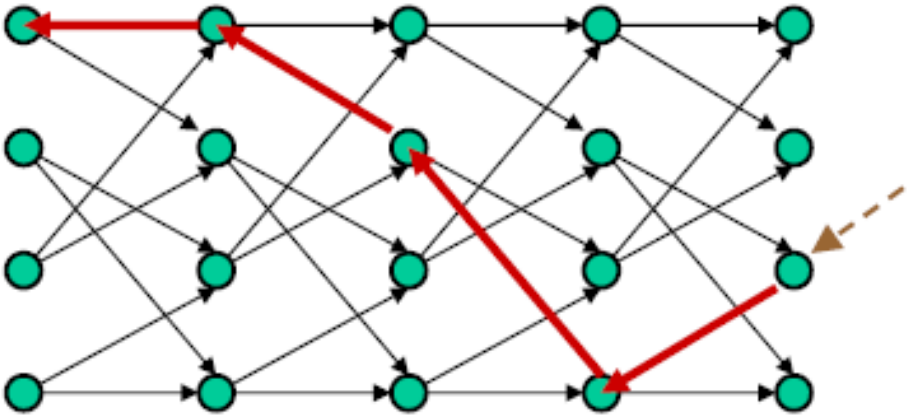


Figure 4.14: Traceback on trellis.

We can perform traceback by:

- Read the Path metric of the both predecessor states
- Select the minimum and add the new backward state into our traceback path.
- By comparing both the previous state and the current state, we can guess the input bit.

After that, we can reorder both the output bits and the traceback states.

Chapter 5: Channel

The communication channel provides the connection between the transmitter and the receiver. The physical channel may be a pair of wires that carry the electrical signal, or an optical fiber that carries the information on a modulated light beam, or an underwater ocean channel in which the information is transmitted acoustically, or free space over which the information-bearing signal is radiated by use of an antenna.

One common problem in signal transmission through any channel is additive noise and interference which may arise externally to the system, such as interference from other users of the channel. When such noise and interference occupy the same frequency band as the desired signal, its effect can be minimized by proper design of the transmitted signal and its demodulator at the receiver.

The effects of noise may be minimized by increasing the power in the transmitted signal. However, equipment and other practical constraints limit the power level in the transmitted signal. Another basic limitation is the available channel bandwidth. A bandwidth constraint is usually due to the physical limitations of the medium and the electronic components used to implement the transmitter and the receiver. These two limitations result in constraining the amount of data that can be transmitted reliably over any communications channel. Shannon's basic results relate the channel capacity to the available transmitted power and channel bandwidth as shown in the following equation [22]:

$$C = B.log_2(1 + SNR) \tag{5.1}$$

5.1 Additive White Gaussian Noise (AWGN)

To simulate the effect of the noise component, AWGN model is used. Its name didn’t come to existence by coincidence, however due to the following justifications:

- Additive: as the noise component is added into the received signal at the receiver side.
- White: as it has uniform power across the entire frequency band.
- Gaussian: as it has normal distribution in time domain with zero average value.

The noise effect presents due to various reasons (e.g. the thermal noise by the virtue of electrons movement in the electronic circuit being used for transmission and reception of the signal).

5.2 Channel fading

Basic Electromagnetic (EM) wave Common propagation phenomena that affect the channel between the transmitter and the receiver are reflection, diffraction, and scattering.

But the main problem in the wireless channel that needs to be mitigated is the channel fading.

In wireless communications, fading is variation of the attenuation of a signal with various variables. These variables include time, geographical position, and radio frequency. Fading is often modeled as a random process. A fading channel is a communication channel that experiences fading. In wireless systems, fading may either be due to multipath propagation, referred to as multipath induced fading, weather (particularly rain), or shadowing from obstacles affecting the wave propagation, sometimes referred to as shadow fading. [23]

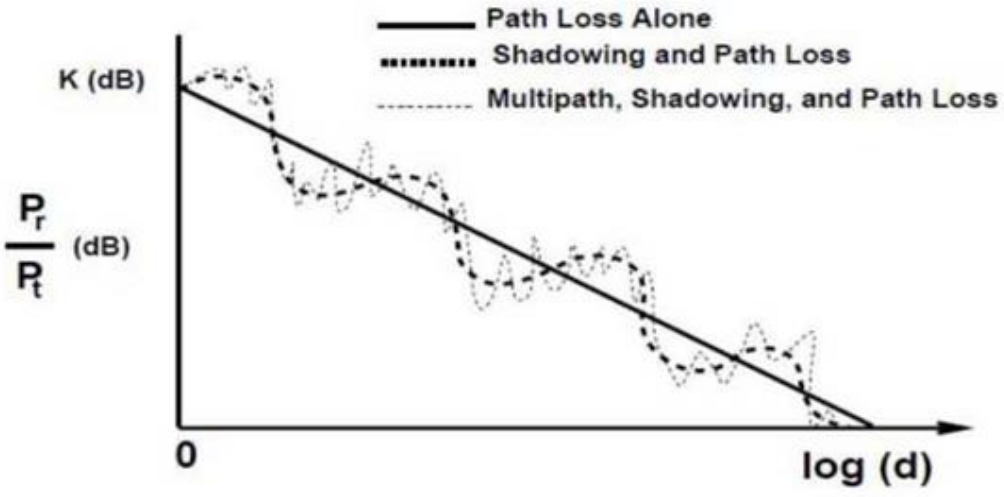


Figure 5.1: main sources of wireless channel fading

As shown in Figure 5.1, there are three main sources of fading:

- Large scale fading or path-loss.
- Medium scale fading or shadowing.
- Small scale fading or multipath effect.

5.2.1 Large scale fading (path-loss)

Path-loss is a reduction of the power density of the transmitted EM signal due to different effects, such as:

- FSPL (Free Space Path Loss) due to the distance between the transmitter and the receiver passing through the medium.

$$FSPL = \left(\frac{4\pi df}{c}\right)^2 \tag{5.2}$$

- Refraction if there are different media in the propagation path.
- Reflection.
- Diffraction losses when part of the radio-wave front is obstructed by an opaque obstacle.

Equation (5.2) mentioned above was for free-space medium. For general medium

$$PL(d)\alpha\left(\frac{d}{d_0}\right)^n \tag{5.3}$$

As shown in equation (5.3), the path-loss exponent n is a variable that depends on the medium:

Table 5.1: Path-loss exponents for different environments

Environment	Path-loss exponent, n
Free space	2
Urban area cellular radio	2.7 to 3.5
Shadowed urban cellular radio	3 to 5
In building line-of-sight	1.6 to 1.8
Obstructed in building	4 to 6
Obstructed in factories	2 to 3

5.2.2 Medium scale fading (shadowing)

The reasons of shadowing are not so far from that of the large scale fading. It may be considered as the average of the small scale fading as shown in Figure 5.1.

5.2.3 Small scale fading (multi-path fading)

Small scale fading is a characteristic of radio propagation resulting from the presence reflectors and scatters that cause multiple versions of the transmitted signal to arrive at the receiver, each distorted in amplitude, phase and angle of arrival. The superposition of all of these versions produces a distorted received signal.

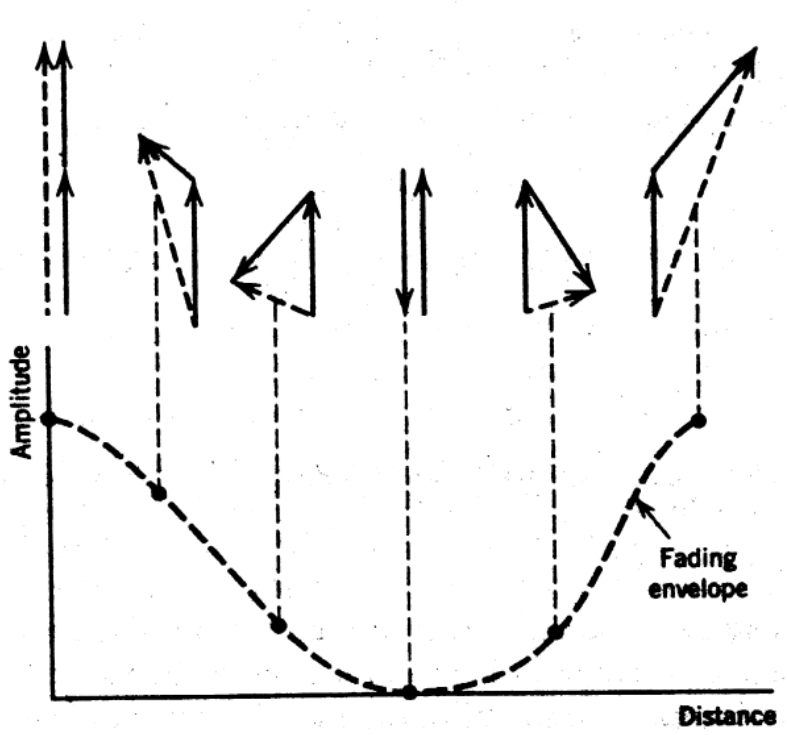


Figure 5.2: Illustrate how the envelop fades as two incoming signals combine with different phase

The small scale fading affects the communication link between transmitters and receivers, by the following:

- Rapid changes in signal strength over a small travel distance or time interval.
- Time dispersion (echoes) caused by multipath propagation.
- Random frequency modulation due to varying Doppler shifts.

The above effects can be reduced or mitigated by tuning the following factors:

- Speed of mobile and speed of surrounding objects (that affect Doppler shift).
- Signal Transmission BW.

5.2.4 Types of fading channels

Channels maybe classified using different concepts but with the concept introduced in the previous subsection of fading, channels can be:

- Flat fading channel.
- Frequency selective channel.
- Slow fading channel.
- Fast fading channel.

5.3 Channel estimation

Channel estimation plays an important part in an OFDM system. It is used for increasing the capacity of orthogonal frequency division multiple access (OFDMA) systems by improving the system performance in terms of bit error rate.

To facilitate the estimation of the channel characteristics, LTE uses cell-specific reference signals (pilot symbols) inserted in both time and frequency. These pilot symbols provide an estimate of the channel at given locations within a subframe. Through interpolation, it is possible to estimate the channel across an arbitrary number of subframes.

The pilot symbols in LTE are assigned positions within a sub-frame depending on the eNodeB cell identification number and which transmit antenna is being used, as shown in the following figure:

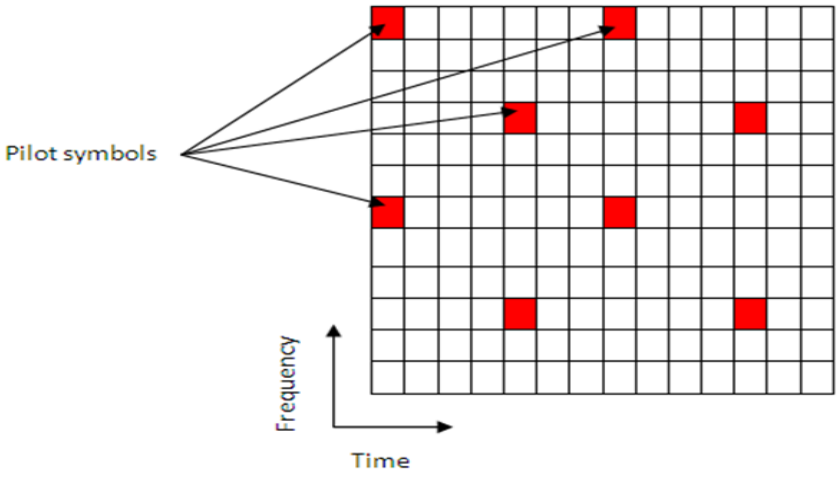


Figure 5.3: Pilot position

The unique positioning of the pilots ensures that they do not interfere with one another and can be used to provide a reliable estimate of the complex gains imparted onto each resource element within the transmitted grid by the propagation channel. [24]

Both transmit and receive chains and the propagation channel model are shown in the block diagram (Figure 5.4).

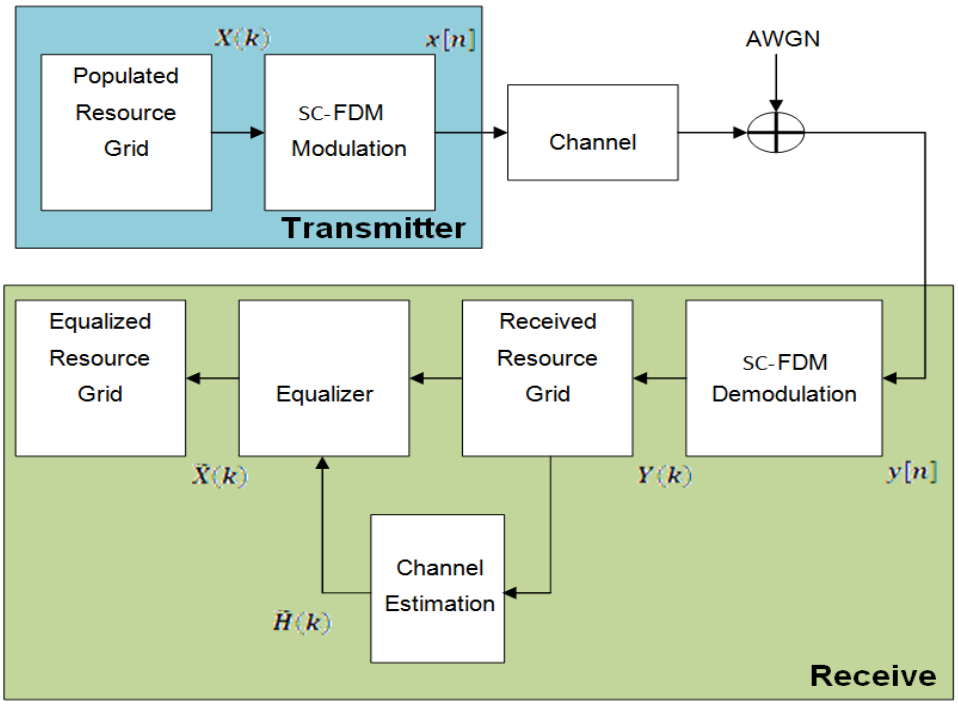


Figure 5.4: Both transmitter and receiver chain.

The populated resource grid represents several sub-frames containing data. This grid is then SC-FDM modulated and passed through the model of the propagation channel. Channel noise in the form of additive white Gaussian noise (AWGN) is added before the signal enters the receiver. Once inside the receiver the signal is SC-OFDM demodulated and a received resource grid can be constructed. The received resource grid contains the transmitted resource elements which have been affected by the complex channel gains and the channel noise. Using the known pilot symbols to estimate the channel, it is possible to equalize the effects of the channel and reduce the noise on the received resource grid.

LTE assigns each antenna port a unique set of locations within a subframe to which to map reference signals. Because no other antenna transmits data at these locations in time and frequency, channel estimation for multi-antenna configurations can be performed. The channel estimation algorithm extracts the reference signals for a transmit/receive antenna pair from the received grid. The least squares estimates of the channel frequency response at the pilot symbols are calculated as described in On Channel Estimation in SC-OFDM Systems.

The least squares estimates are then averaged to reduce any unwanted noise from the pilot symbols. Because it is possible that no pilots are located near the sub-frame edge, virtual pilot symbols are created to aid the interpolation process near the edge of the sub-frame. Using the averaged pilot symbol estimates and the calculated virtual pilot symbols, interpolation is then carried out to estimate the entire sub-frame. This process is demonstrated in the following block diagram.

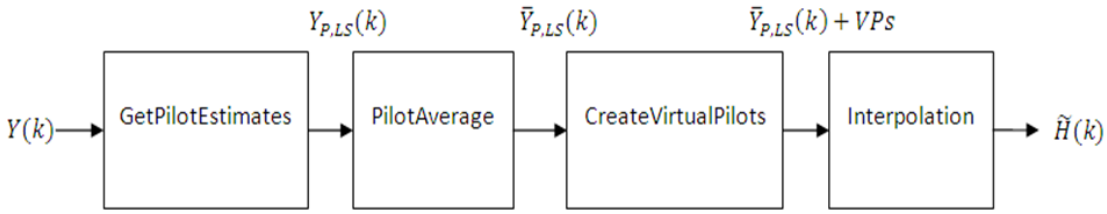


Figure 5.5: Channel estimation block diagram

5.3.1 Get Pilot Estimates subsystem

The first step in determining the least squares estimate is to extract the pilot symbols from their known location within the received sub-frame. Because the value

of these pilot symbols is known, the channel response at these locations can be determined using the least squares estimate. The least squares estimate is obtained by dividing the received pilot symbols by their expected value.

$$Y(k) = H(k).X(k) + noise \tag{5.4}$$

where

- $Y(k)$ is a received complex symbol value;
- $X(k)$ is a transmitted complex symbol value;
- $H(k)$ is a complex channel gain experienced by a symbol.

Known pilot symbols can be sent to estimate the channel for a subset of REs within a subframe. In particular, if pilot symbol $X_P(k)$ is sent in an RE, an instantaneous channel estimate $H_{PI}(k)$ for that RE can be computed using:

$$H_{pI}(k) = \frac{Y_p(k)}{X_p(k)} = H_p(k) + noise \tag{5.5}$$

where

- $Y_P(k)$ represents the received pilot symbol values;
- $X_P(k)$ represents the known transmitted pilot symbol values;
- $H_P(k)$ is the true channel response for the RE occupied by the pilot symbol.

5.3.2 Pilot Average subsystem

To minimize the effects of noise on the channel estimates, the least square estimates are averaged using an averaging window. This simple method produces a substantial reduction in the level of noise found on the pilot REs.

The Averaging method uses the approach described in TS 36.141 [25], Annex F.3.4. Time averaging is performed across each subcarrier that contains a pilot symbol, resulting in a column vector containing an average amplitude and phase for each subcarrier that is carrying a reference signal.

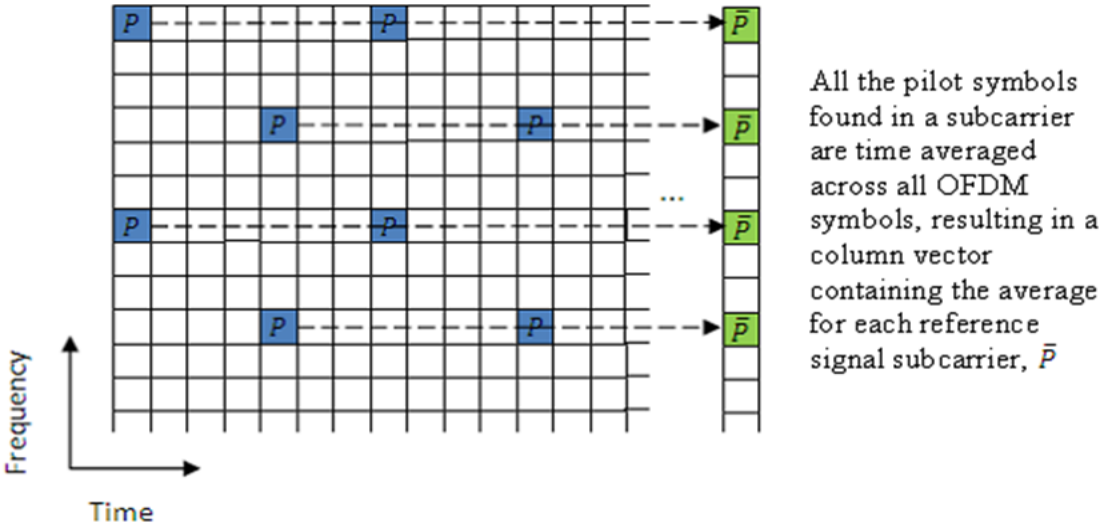


Figure 5.6: Time average of the received pilots.

The averages of the pilot symbol subcarriers are then frequency averaged using a moving window of maximum size 19.

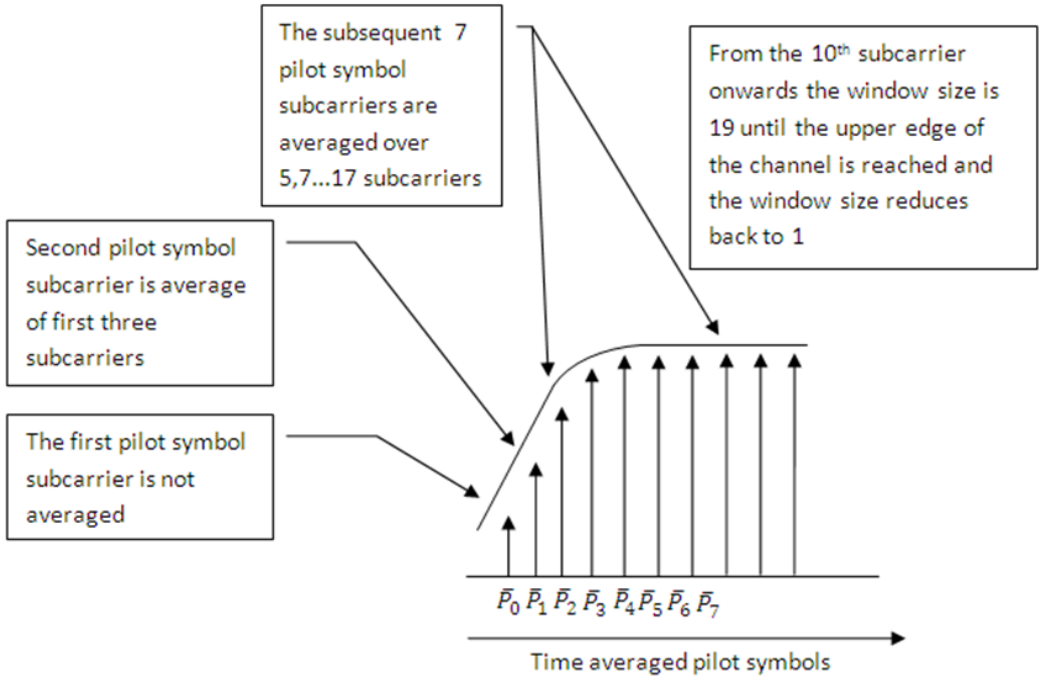


Figure 5.7: Frequency averaging of time averaged received pilots.

5.3.3 Create Virtual Pilot System

In many instances, edges of the resource grid do not contain any pilot symbols. This effect is shown in the resource grid in the following figure.

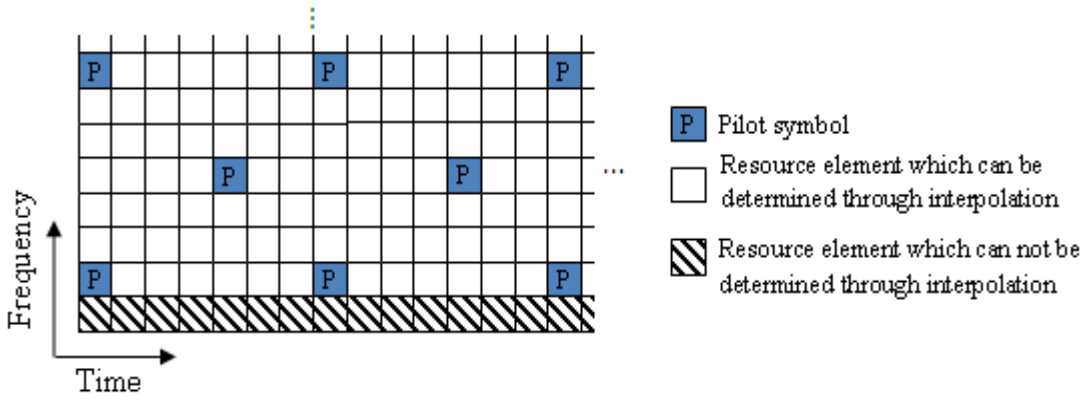


Figure 5.8: Symbol at Edges.

In this case, channel estimates at the edges cannot be interpolated from the pilot symbols. To overcome this problem, virtual pilot symbols are created. In our case no interpolation system is used so no virtual pilots are needed.

5.3.4 Interpolation Subsystem

Once the noise has been reduced or removed from the least squares pilot symbol averages and sufficient virtual pilots have been determined, it is possible to use interpolation to estimate the missing values from the channel estimation grid.

The pilot averaging method described in TS 36.141 [24], Annex F.3.4, requires the use of simple linear interpolation on the time-averaged and frequency-averaged column vector. The interpolation is one-dimensional, since it only estimates the values between the averaged pilot symbol subcarriers in the column vector. The resulting vector is then replicated and used as the channel estimate for the entire resource grid.

5.4 Noise Estimation

The performance of some receivers can be improved through knowledge of the noise power present on the received signal. The noise power can be determined by analyzing the noisy least squares estimates and the noise averaged estimates.

The noisy least-squares estimates from the Get Pilot Estimates Subsystem and the noise averaged pilot symbol estimates from the Pilot Average Subsystem provide an indication of the channel noise. The least-squares estimates and the averaged estimates contain the same data, apart from additive noise. Simply taking the

difference between the two estimates results in a noise level value for the least squares channel estimates at pilot symbol locations.

Averaging the instantaneous channel estimates over the smoothing window, we have

$$H_{pI}^{Avg}(k) = \frac{1}{|S|}\sum_{m\epsilon S} H_{PI}(m) \approx H_P(k) \tag{5.6}$$

Where *S* is the set of pilots in the smoothing window and |*S*| is the number of pilots in *S*. Thus, an estimate of the noise at a particular pilot RE can be formed using:

$$noise_{estimated} = H_{PI}(m) - H_{PI}^{Avg}(k) \tag{5.7}$$

In practice, it is not possible to remove all the noise using averaging. Because it is only possible to reduce the noise, only an estimate of the noise power can be made.

Using the value of the noise power found in the channel response at pilot symbol locations, the noise power per resource element (RE) can be calculated by taking the variance of the resulting noise vector. The noise power per RE for each transmit and receive antenna pair is calculated and stored. The mean of this matrix is returned as the estimate of the noise power per RE.

5.5 Channel Equalization

5.5.1 Zero forcing

In this method, the ISI component at the output of equalizer is forced to zero by using appropriate linear time invariant filter having suitable transfer function.

If transmitted symbol is represented by x_1 and x_2, h_{11} represent the channel from first transmitter to first receiver, h_{12} represent the channel from second transmitter to first receiver, h_{21} represent the channel from first transmitter to second receiver and h_{22} represent the channel from second transmitter to second receiver and n_1, n_2 represent noise on first and second receiver then the received symbol on first receiver is given by:

$$y1 = h_{11}x_1 + h_{12}x_2 + n_1 = (h_{11}\ h_{12})\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} + n_1 \tag{5.8}$$

And the received symbol on second receiver is given by:

$$y2 = h_{22}x_2 + h_{21}x_1 + n_2 = (h_{22}\ h_{21})\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} + n_2 \tag{5.9}$$

These two above equation can also be written as:

$$\begin{pmatrix} y_1 \\ y_2 \end{pmatrix} = \begin{pmatrix} h_{11} & h_{12} \\ h_{21} & h_{22} \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} + \begin{pmatrix} n_1 \\ n_2 \end{pmatrix} \tag{5.10}$$

From here it is clear that in order to find x from above equation, we need to find out the matrix which is inverse of matrix H.

If W represents the inverse of H then it must satisfy the property

$$WH = I \tag{5.11}$$

Where I is the identity matrix.

The matrix W which satisfies the above mentioned property is known as the zero forcing linear detector, and is computed by following equation:

$$W = (H^H H)^{-1} H^H \tag{5.12}$$

Where $H^H H$ is calculated using the following:

$$H^H H = \begin{pmatrix} h_{11}^* & h_{21}^* \\ h_{12}^* & h_{22}^* \end{pmatrix}\begin{pmatrix} h_{11} & h_{12} \\ h_{21} & h_{22} \end{pmatrix} \tag{5.13}$$

From this matrix it is clear that the off diagonal term is non-zero and hence zero forcing equalizer cancel out the interference signal. It is reasonably simple and easy to implement but its main drawback is that it tends to amplify the noise and hence gives noisy output.

5.5.2 MMSE Equalizer (Minimum Mean Square Error)

This type of equalizer uses the squared error as performance measurement. The receiver filter is designed to fulfill the minimum mean square error criterion. Main objective of this method is to minimize the error between target signal and output obtained by filter. The computation for this method is as follows.

- Computes the coefficient of matrix W using MMSE algorithm which minimize the condition:

$$E\{[W_y - x][W_y - x]^H\} \tag{5.14}$$

- Solving the above equation gives

$$W = (H^H H + N_0 I)^{-1} H^H \quad (5.15)$$

- From that equation, it is clear that this equation is different from the equation of zero forcing equalizer by the term $N_o I$. If we put $N_o I$=0 in this equation, then MMSE equalizer becomes zero forcing equalizer.

5.6 Channel models

The multipath fading channel model specifies the following three delay profiles.

- Extended Pedestrian A model (EPA)
- Extended Vehicular A model (EVA)
- Extended Typical Urban model (ETU)

These three delay profiles represent a low, medium, and high delay spread environment, respectively. The multipath delay profiles for these channels are shown in the following tables.

Table 5.2: EPA Delay Profile.

Excess Tap delay (ns)	Relative Power (dB)
0	0.0
30	-1.0
70	-2.0
90	-3.0
110	-8.0
190	-17.2
410	-20.8

Table 5.3: EVA Delay Profile

Excess Tap delay (ns)	Relative Power (dB)
0	0.0
30	-1.5
150	-1.4
310	-3.6
370	-0.6

710	-9.1
1090	-7.0

Table 5.4: ETU Delay Profile

Excess Tap delay (ns)	Relative Power (dB)
0	-1.0
50	-1.0
120	-1.0
200	-0.0
230	-0.0
500	-0.0
1600	-3.0
2300	-5.0
5000	-7.0

All the taps in the preceding tables have a classical *Doppler* spectrum. In addition to multipath delay profile, a maximum Doppler frequency is specified for each multipath fading propagation condition, as shown in the following table.

Table 5.5: Maximum Doppler frequency for different fading propagation.

Channel model	Maximum Doppler frequency
EPA 5Hz	5Hz
EVA 5Hz	5Hz
EVA 70Hz	70Hz
ETU 70Hz	70Hz
ETU 300Hz	300Hz

Chapter 6: Project Flow

6.1 Project milestones

1. Learning Steps:
   - LTE Fundamentals through Rohde and Schwarz documents.
   - 3GPP standards (releases 8, 9 and 10) for LTE (PUSCH).
   - Parallel Programming Course using CUDA Language (Udacity online course).
   - AVX instruction (Supported in Intel® Core™ i7 Processors).
2. Implementing PUSCH using MATLAB LTE System Toolbox.
3. Implementing PUSCH chain on MATLAB with the help of the openLTE [26] open source implementation and testing its results versus the MATLAB LTE System Toolbox results.
4. Implementing the PUSCH on Intel® Core™ i7 processors (ANSI-C code) and test it versus MATLAB results. Make use of Intel MKL and AVX instructions for parallelizing the LTE chain.
5. Implementing the PUSCH with CUDA (C code) and test it versus MATLAB results.
6. Optimizing the C code implementation for SISO chain using CUDA.
7. Implementing 2x2, 4x4 and 64x64 MIMO chains.
8. Obtaining time profiling results for both SISO and MIMO PUSCH chains.

6.2 Used Tools

To achieve better results in both testing and implementation, faster, more productive, and well-known tools were used. The used tools can be divided into:

1. System design/test tools
   - MATLAB R2016b
2. Integrated Development Environments
   - Microsoft Visual Studio Community 2015
   - Eclipse
3. Other Tools
   - Intel C++ Compiler

- CUDA 8.0 (C/C++)
- Nvidia Visual Profiler

6.3 Version Control Software

GitHub Repository was used which offers multiple benefits:

- Easy to get last updated version.
- Full Local History.
- Restoring Previous Versions.
- Easier teamwork.

Link of the Repository: https://github.com/AhmedMoustafaHosni/LTE_PHY_GP

6.4 Attempted optimization techniques

Not only the mentioned techniques in chapters (2) and (3) were experimented, but also other techniques were examined on both Intel and Nvidia architectures.

Intel techniques:

- Cilk spawn
- Intel internal graphics off loading
- Intel openMP

Nvidia techniques:

- Parallel Streams
- Unified Memory

All of the above techniques were tested on the chain. One example to see the effect of the cilk spawn operation on the performance of Intel CPU will be seen in Figure 6.1. It shows the experiment of the Fibonacci series algorithm over $n = 40$, where n is the argument of the recursive function. Each recursive call is spawned using cilk_spawn. Finally cilk_sync is used at the end of the function body to end threads together. All CPUs reach their max performance at the same time.

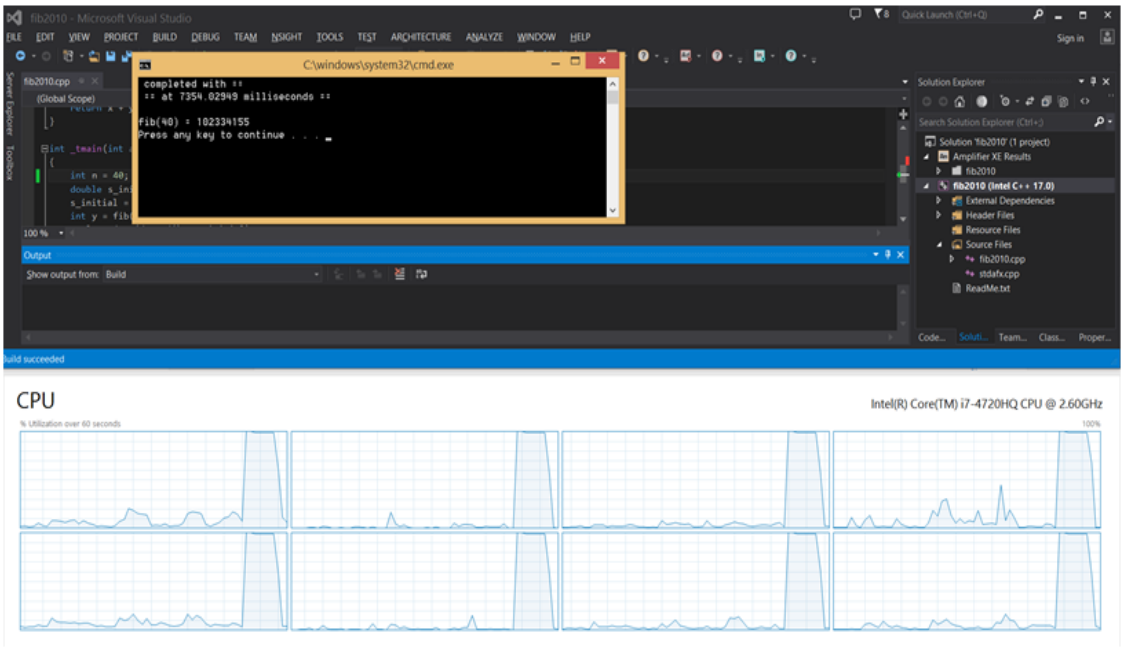


Figure 6.1: Performance of Cilk for operation on Fibonacci series example.

6.5 Time Profiling

The time profiling for the chain is performed to make sure that the software meets the timing requirements of the LTE standard. The LTE standard specifies the maximum timing for the physical layer chain to be 1 ms. The following figures show the implemented uplink transmitter chains (SISO/MIMO) defined by the LTE standard. The time profiling results are presented in section 7.2.

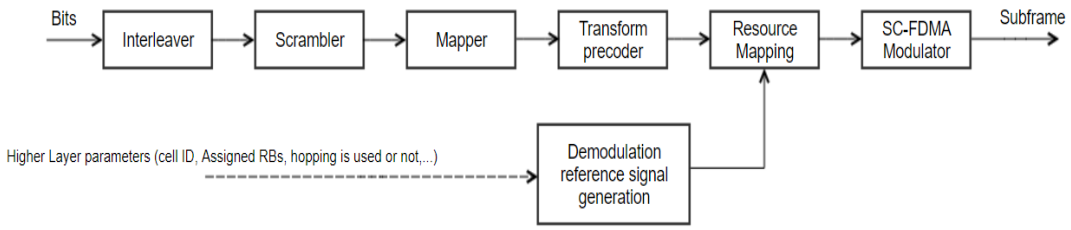


Figure 6.2: Uplink transmitter SISO chain.

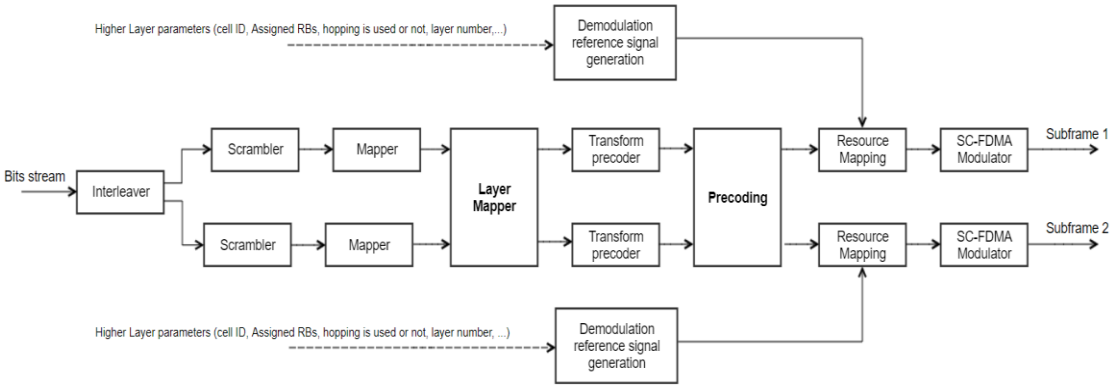


Figure 6.3: Uplink transmitter MIMO chain.

The timing is measured using two methods:

- Chrono time library supported from C++11.
- Nvidia visual profiler.

Both methods get the same timing for the chain. The figure below shows an example of profiling the transmitter chain using Nvidia visual profiler. The timing of three blocks of SISO chain: Interleaver, Scrambler, and Mapper (five kernels) is shown in the timeline in Figure 6.4.

The overhead of memory allocation and plan creation for FFT is neglected as it is done only one time at startup (see Figure 6.5).

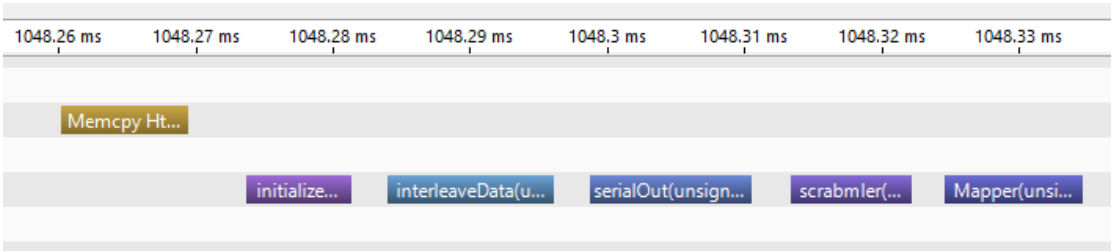


Figure 6.4: Nvidia visual profiler for interleaver, scrambler and Mapper.

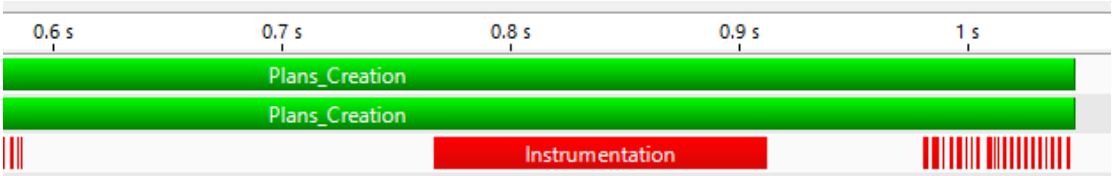


Figure 6.5: Overhead of Plans Creation.

Chapter 7: Results

In this chapter we are going to conclude our achievements and results through a year full of team work, enthusiasm, hard work and research.

7.1 BER plots on MATLAB

7.1.1 SISO chain plots for AWGN channel

BER of the SISO chain in Figure 6.2 is plotted for QPSK and 64QAM modulation schemes as shown in Figure 7.1 and Figure 7.2 respectively.

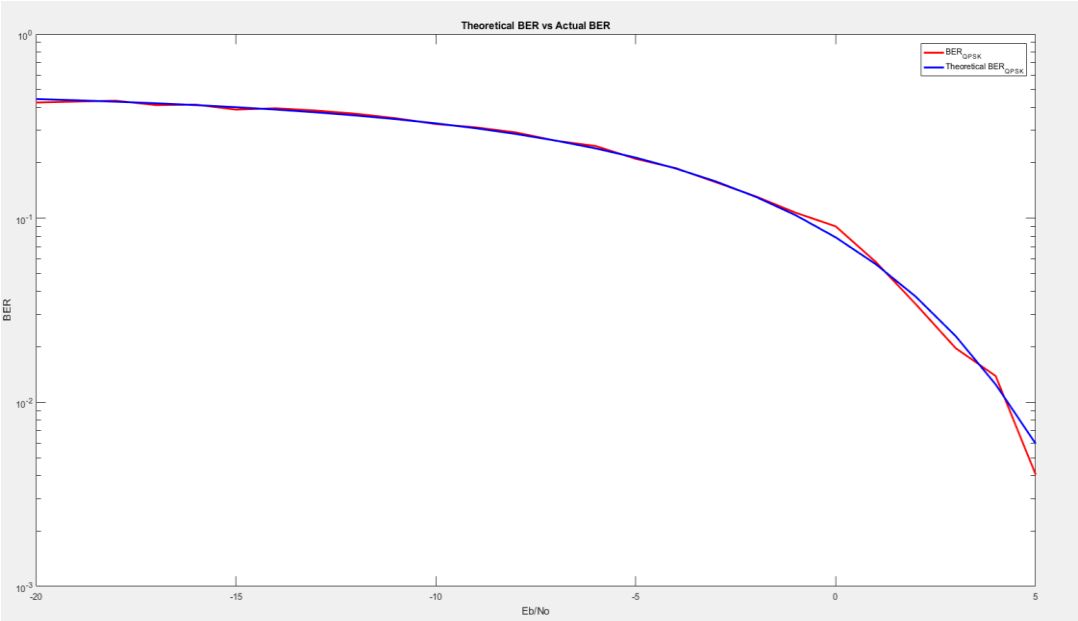


Figure 7.1: BER plot of SISO chain with QPSK modulation.

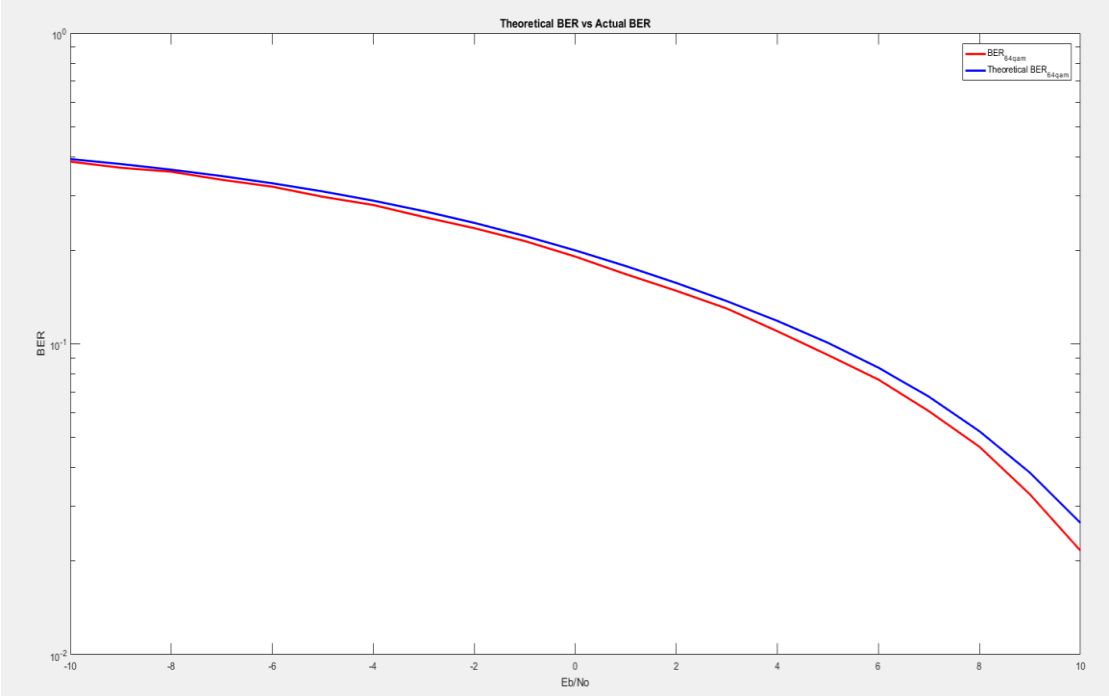


Figure 7.2: BER plot of SISO chain with 64-QAM modulation.

7.1.2 SISO chain plots for noisy fading channel

BER of full SISO chain without channel encoding and using channel estimation and zero forcing equalization in Extended Pedestrian A channel is shown in Figure 7.3:

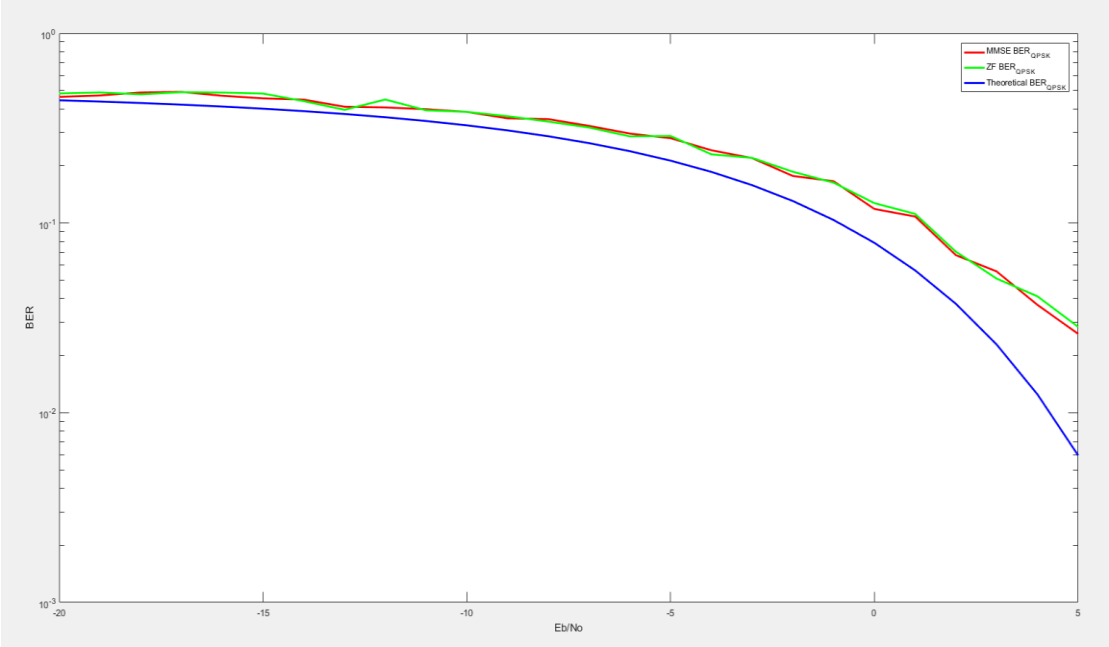


Figure 7.3: BER for fading channel

7.1.3 SISO chain with tail biting convolutional channel encoding plots for AWGN channel

Tail biting convolutional encoding and Viterbi decoder were implemented, and the BER of the chain for AWGN channel is plotted in Figure 7.4.

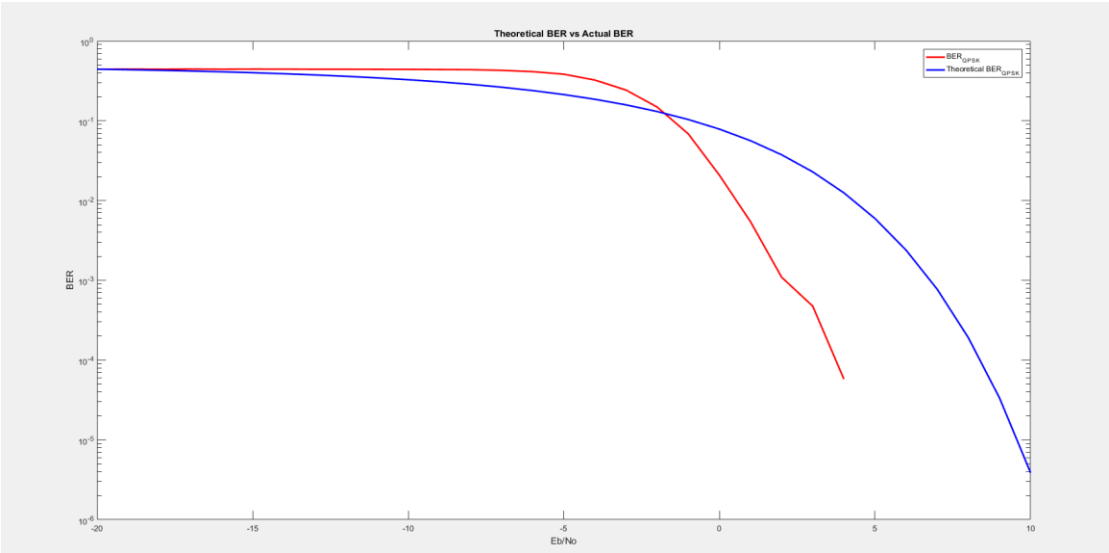


Figure 7.4: BER of SISO chain with channel encoding for AWGN channel.

7.1.4 MIMO chain plots for AWGN channel

BER of the MIMO chain in Figure 6.3 is plotted for QPSK as shown in Figure 7.5.

Figure 7.5: BER plot of MIMO chain with QPSK modulation.

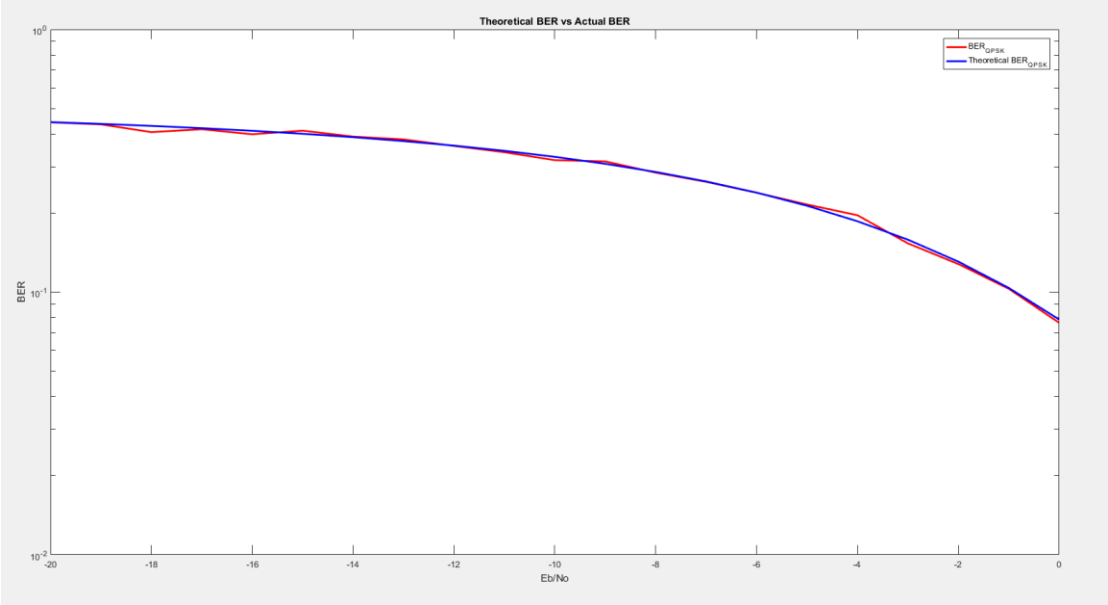


Figure 7.5: BER plot of MIMO chain with QPSK modulation.

7.1.5 MIMO chain with tail biting convolutional channel encoding plots for AWGN channel

BER rate plot of MIMO chain with channel encoding and viterbi decoding is plotted in Figure 7.6.

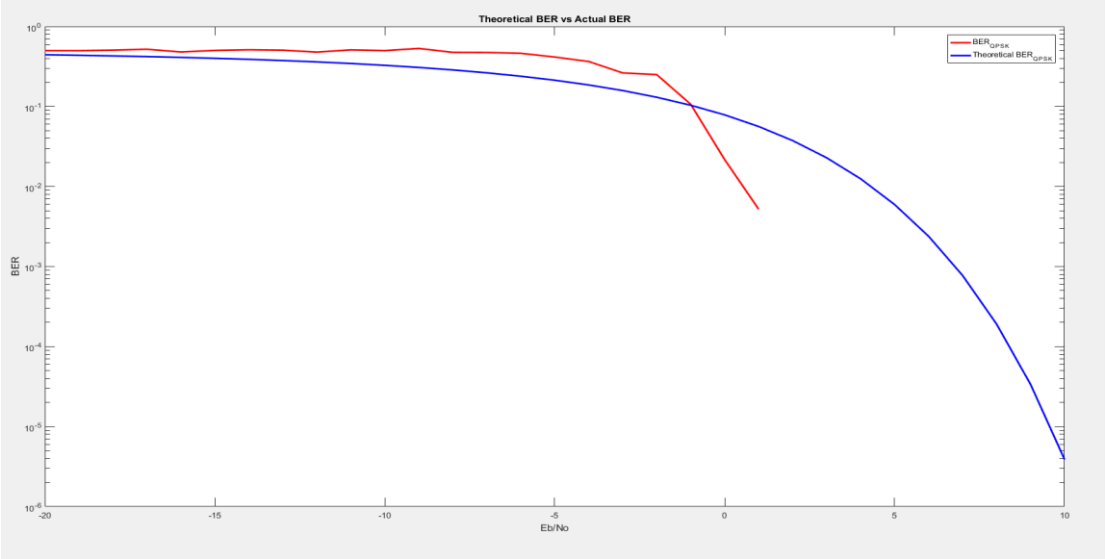


Figure 7.6: BER of MIMO chain with channel encoding for AWGN channel.

7.1.6 MIMO chain plots for noisy fading channel

BER of full MIMO chain without channel encoding and using channel estimation and equalization in Extended Pedestrian A model channel with no channel correlation is shown in Figure 7.7:

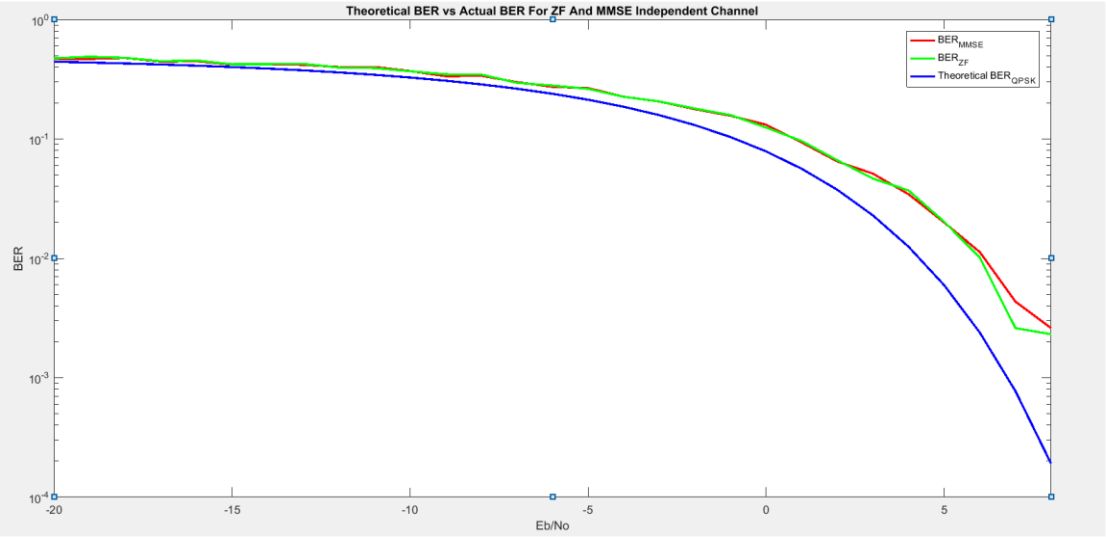


Figure 7.7: BER of MIMO chain without channel correlation for AWGN channel

BER of full MIMO chain without channel encoding and using channel estimation and equalization in Extended Pedestrian A model channel with channel correlation is shown in Figure 7.8:

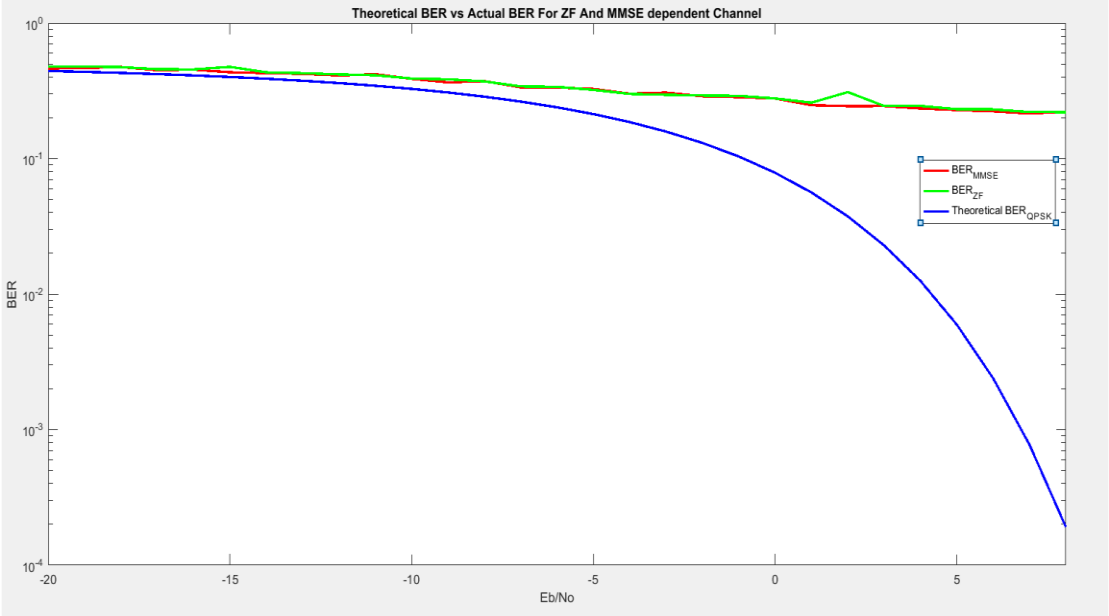


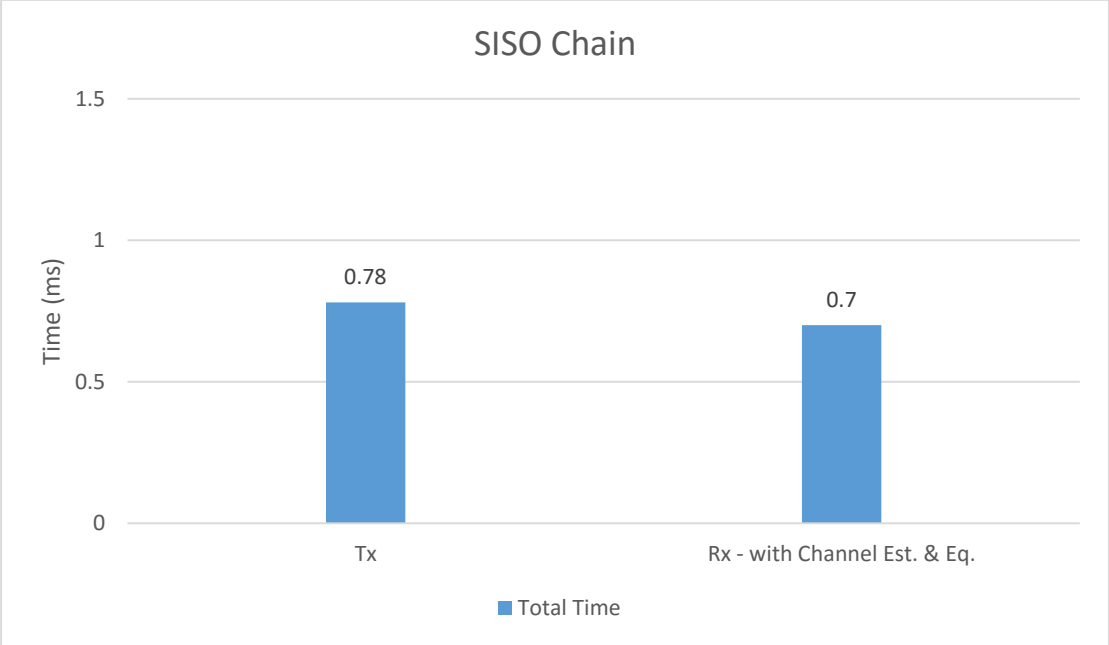
Figure 7.8: BER of MIMO chain with channel correlation for AWGN channel.

7.2 Timing Results

Used platform:

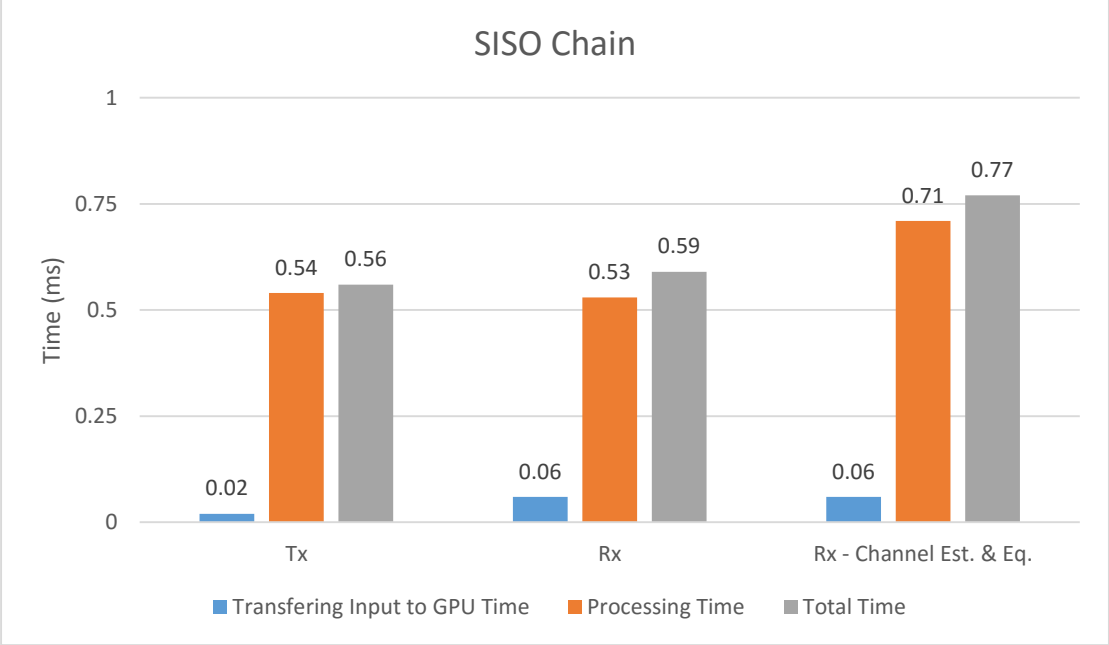
- **CPU:** Intel® Core™ i7-4720HQ (6M Cache, up to 3.60 GHz)
- **GPU:** NVIDIA GeForce GTX 960M (CUDA Cores: 640, Base Clock: 1096 MHz)
- **Memory:** 16.0 GB DDR3L (Dual-Channel)
- **Hard Drive:** Samsung SSD 850 EVO 1TB
- **Operating System:** Windows 10 Pro (Build 10240)

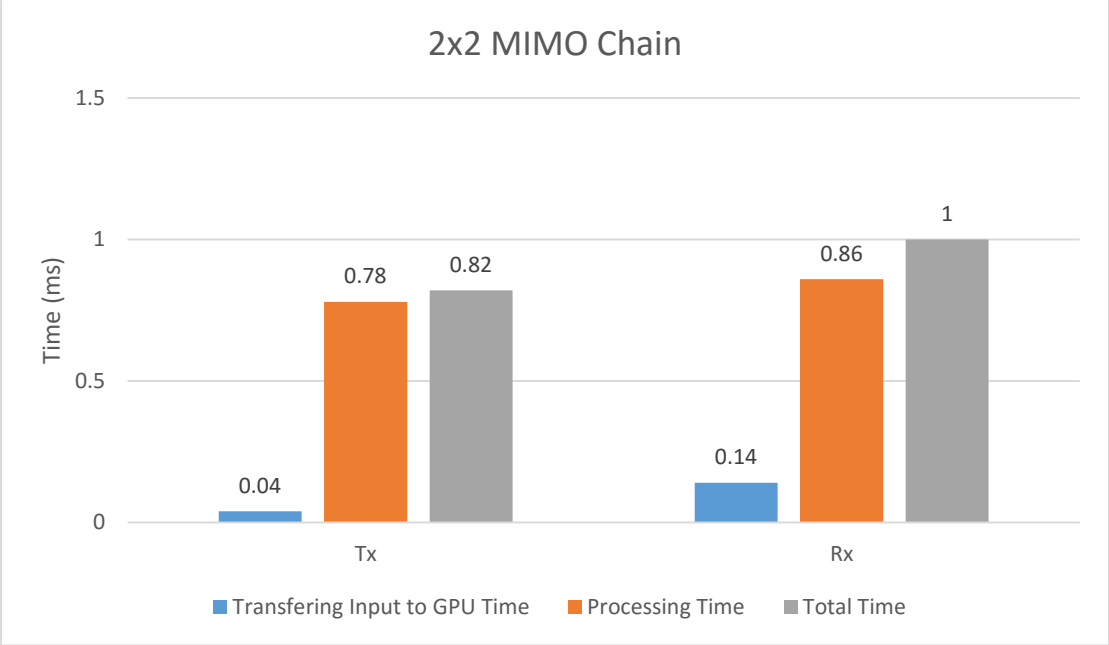
7.2.1 Intel AVX & MKL Implementation

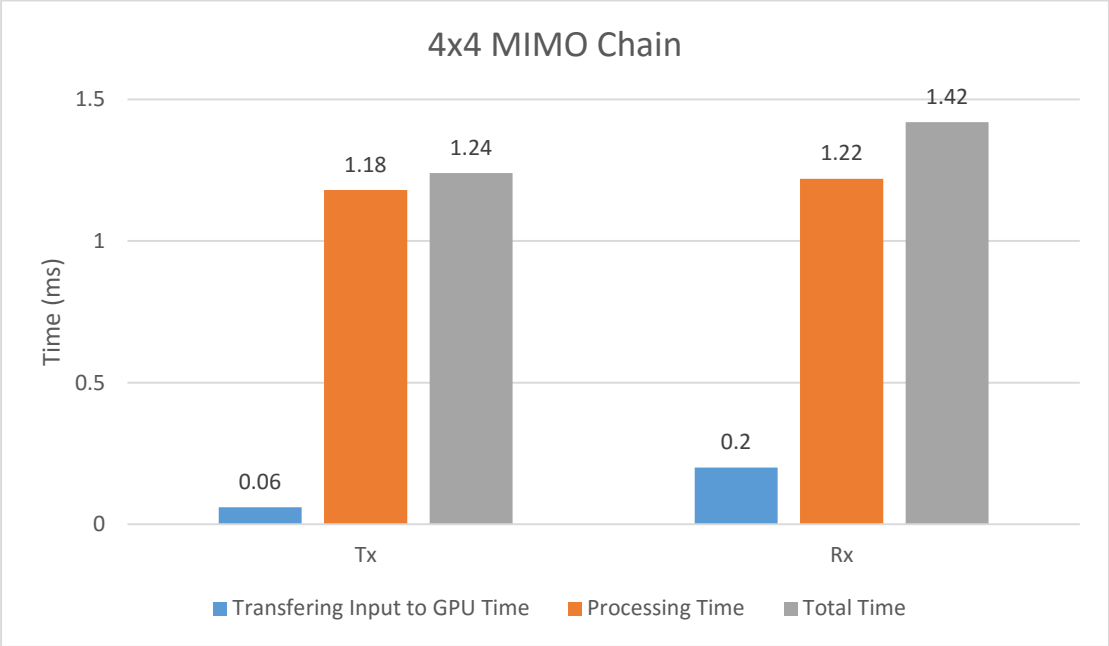


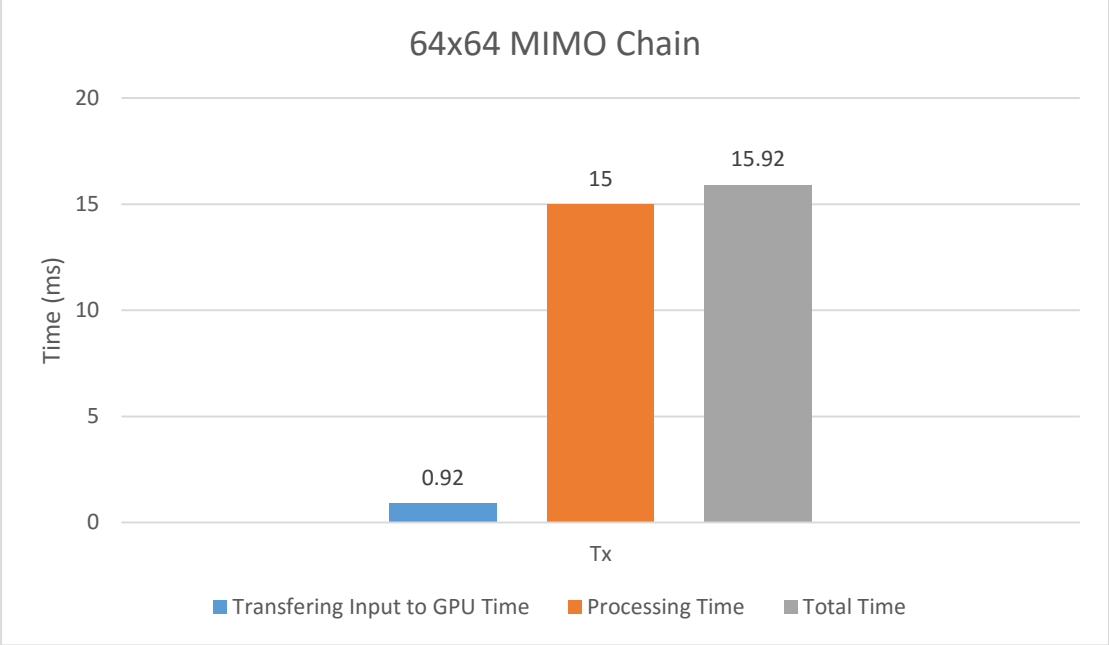
7.2.2 CUDA Implementation

The following timing results are the average of 1000 iterations.




2x2 MIMO Chain
1.5
1
0.5
0
Time (ms)
0.04
0.78
0.82
0.14
0.86
1
Tx
Rx
Transfering Input to GPU Time
Processing Time
Total Time


4x4 MIMO Chain
1.5
1
0.5
0
Time (ms)
0.06
1.18
1.24
0.2
1.22
1.42
Tx
Rx
Transfering Input to GPU Time
Processing Time
Total Time


64x64 MIMO Chain
20
15
10
5
0
Time (ms)
0.92
15
15.92
Tx
Transfering Input to GPU Time
Processing Time
Total Time

Conclusion

For Intel implementation, subframe time of the **SISO** bit processing for the Tx and Rx are $\mathbf{0.78}\ \boldsymbol{ms}$ and $\mathbf{0.70}\ \boldsymbol{ms}$ respectively which satisfies the 3GPP standard too. MIMO wasn't implemented as no further parallelism could be achieved unlike CUDA which has **Parallel Streams** which allow parallel execution of MIMO layers.

For CUDA implementation, subframe time of the **SISO** bit processing for the Tx and Rx are $\mathbf{0.56}\ \boldsymbol{ms}$ and $\mathbf{0.77}\ \boldsymbol{ms}$ respectively. Also, for **2x2 MIMO Rx** implementation, subframe time is $\mathbf{1}\ \boldsymbol{ms}$**,** while for the **2x2 MIMO Tx**, subframe time is $\mathbf{0.82}\ \boldsymbol{ms}$ which satisfies the 3GPP standard stating that the processing time of the subframe should be less than 1 ms. Massive MIMO processing time is investigated to give indication of the required improvements of the entire solution in order to serve large number of antennas.

Future work

After the implementation of the PUSCH chain using MATLAB and C, investigating the processing time of the SISO and MIMO configurations, and trying multiple optimization techniques, there is still work to be done to finish the processing of the physical layer completely.

The work that needs to be done is:

- Trying other optimization techniques like kernel merging to reduce the time needed by the CPU to invoke multiple kernels.
- Add **parallel streams** to MIMO Rx.
- Support control channels.
- Trying more complex channel estimation and equalization techniques which can mitigate the effect of the correlation of the channels for the MIMO system.
- Trying Intel AVX2 and AVX512.
- Trying architecture optimization by merging between Intel AVX and CUDA.
- Use High-End Nvidia GPU.
- Implementing parallel algorithm for Pseudo-Random Sequence Generation.

References

[1] Intel Embedded, Design Center, "Introduction to Intel® Architecture," Intel, [Online]. Available: https://www.intel.com/content/www/us/en/intelligent-systems/embedded-systems-training/ia-introduction-basics-paper.html. [Accessed 10 July 2017].

[2] R. Rajwar, M. Dixon and R. Singhal, "Specialized Evolution of the General-Purpose CPU," cidrdb.org, Hillsboro, USA, 2015.

[3] Intel, "Intel Intrinsics Guide," Intel, [Online]. Available: https://software.intel.com/sites/landingpage/IntrinsicsGuide/#techs=AVX,AVX2. [Accessed 10 July 2017].

[4] I. Corporation. [Online]. Available: https://software.intel.com/en-us/node/522582.

[5] C. developers. [Online]. Available: https://www.cilkplus.org/cilk-plus-tutorial.

[6] B. Barney and L. Livermore, "Introduction to Parallel Computing," [Online]. Available: https://computing.llnl.gov/tutorials/parallel_comp/. [Accessed 10 July 2017].

[7] "Graphics Processing Unit (GPU)," [Online]. Available: http://www.nvidia.com/object/gpu.html. [Accessed 11 7 2017].

[8] "What’s the Difference Between a CPU and a GPU?," [Online]. Available: https://blogs.nvidia.com/blog/2009/12/16/whats-the-difference-between-a-cpu-and-a-gpu/. [Accessed 11 7 2017].

[9] E. Dahlman, S. Parkvall and J. Skold, 4G: LTE/LTE-Advanced for Mobile Broadband, Oxford: Academic Press, 2014.

[10] Rohde and Schwarz, Rohde & Schwarz: LTE Technology Introduction, Munich: http://www.academia.edu, 2012, pp. 15-16.

[11] "OFDM Orthogonal Frequency Division Multiplexing Tutorial," [Online].

Available: http://www.radio-electronics.com/info/rf-technology-design/ofdm/ofdm-basics-tutorial.php. [Accessed 7 July 2017].

[12] "Cyclic prefix," [Online]. Available: https://en.wikipedia.org/wiki/Cyclic_prefix. [Accessed 7 July 2017].

[13] "LTE OFDM, OFDMA SC-FDMA & Modulation," [Online]. Available: http://www.radio-electronics.com/info/cellulartelecomms/lte-long-term-evolution/lte-ofdm-ofdma-scfdma.php. [Accessed 7 July 2017].

[14] "An Introduction To SC-FDMA Used By LTE In Uplink Direction," [Online]. Available: http://mobilesociety.typepad.com/mobile_life/2007/05/an_introduction.html. [Accessed 7 July 2017].

[15] Keysight Technologies, Inc., "LTE Physical Layer Overview," Keysight Technologies, Inc., [Online]. Available: http://rfmw.em.keysight.com/wireless/helpfiles/89600B/webhelp/subsystems/lte/content/lte_overview.htm. [Accessed 7 July 2017].

[16] Telesystem Innovations Inc., LTE in a Nutshell: The Physical Layer, Telesystem Innovations Inc., 2010, pp. 4-7.

[17] 3GPP, 3GPP TS 36.101; User Equipment (UE) radio transmission and reception (Release 8), France: 3GPP, 2008.

[18] "wirelessdictionary," [Online]. Available: http://www.wirelessdictionary.com/Wireless-Dictionary-Interleaving-Definition.html. [Accessed 11 July 2017].

[19] 3GPP TS 36.212 version 8.8.0 Release 8, NICE: 3GPP, 2010.

[20] 3GPP TS 36.211 version 8.7.0 Release 8, Nice: 3GPP, 2009.

[21] 3GPP TS 36.213 version 8.8.0 Release 8, Nice: 3GPP, 2009.

[22] N. Instruments, "National Instruments," [Online]. Available:

http://www.ni.com/white-paper/14919/en/.

[23] Wikipedia , "Wikipedia," 30 June 2017. [Online]. Available: https://en.wikipedia.org/wiki/Fading.

[24] MathWorks, "MathWorks," [Online]. Available: https://www.mathworks.com/help/lte/ug/channel-estimation.html.

[25] 3GPP, 3GPP TS 36.141 version 10.1.0 Release 10, Nice: 3GPP, 2011.

[26] "openLTE, an open source 3GPP LTE implementation.," [Online]. Available: https://sourceforge.net/projects/openlte/. [Accessed 12 July 2017].