



University of Stuttgart
Germany



Implementation of an USRP-based software-defined radio platform for designing and evaluating artificial intel- ligent wireless transmission algorithms

*Implementierung einer USRP-basierten software-
definierten Funkplattform zur Entwurf und Auswertung
künstlicher intelligenter drahtloser Übertragungsalgorithmen*

Master's thesis of

Qiheng Zhou

Course of studies: Elektromobilität

Student ID: 3206403

under the direction of

Examiner: Prof. Dr.-Ing. H.-C. Reuss

Adviser: Dr.-Ing. D. Keilhoff

March 2020

Vorgelegt an der Universität Stuttgart

Institute for Internal Combustion Engines and Automotive Engineering,
Chair in Automotive Mechatronics

Institute for Internal Combustion Engines and Automotive Engineering
Chair in Automotive Mechatronics
Prof. Dr.-Ing. H.-C. Reuss
Pfaffenwaldring 12, 70569 Stuttgart, Germany

☎ +49 711 68 56 57 10

✉ info@ivk.uni-stuttgart.de

🔗 <http://www.ivk.uni-stuttgart.de>



**Deutsches
Forschungszentrum
für Künstliche
Intelligenz GmbH**

DFKI GmbH
Trippstadter Str. 122
67633, Kaiserslautern

Intelligent Networking Research Group
Dr.-Ing. Wei Jiang
Tel: +49 631 20575-3018
E-Mail: wei.jiang@dfki.de

Student/Intern Qiuheng ZHOU
Firmenseitige Betreuer Dr.-Ing. Wei JIANG
Bearbeitungszeitraum From: September 09, 2019 To: February 28, 2019

Arbeitstitel der Abschlussarbeit

Implementation of an USRP-based software-defined radio platform for designing and evaluating artificial intelligent-based next-generation wireless transmission algorithms

Beschreibung

Driven by the demand to cope with exponentially growing mobile data traffic and to support new traffic types from massive numbers of machine-type devices, academia and industry are thinking the potential of making full use of Artificial Intelligence technology to fuel the evolution of fifth generation(Beyond 5G) mobile networks. The aim of this work is to build a software defined radio module, for the design and evaluation of AI-based wireless algorithms.

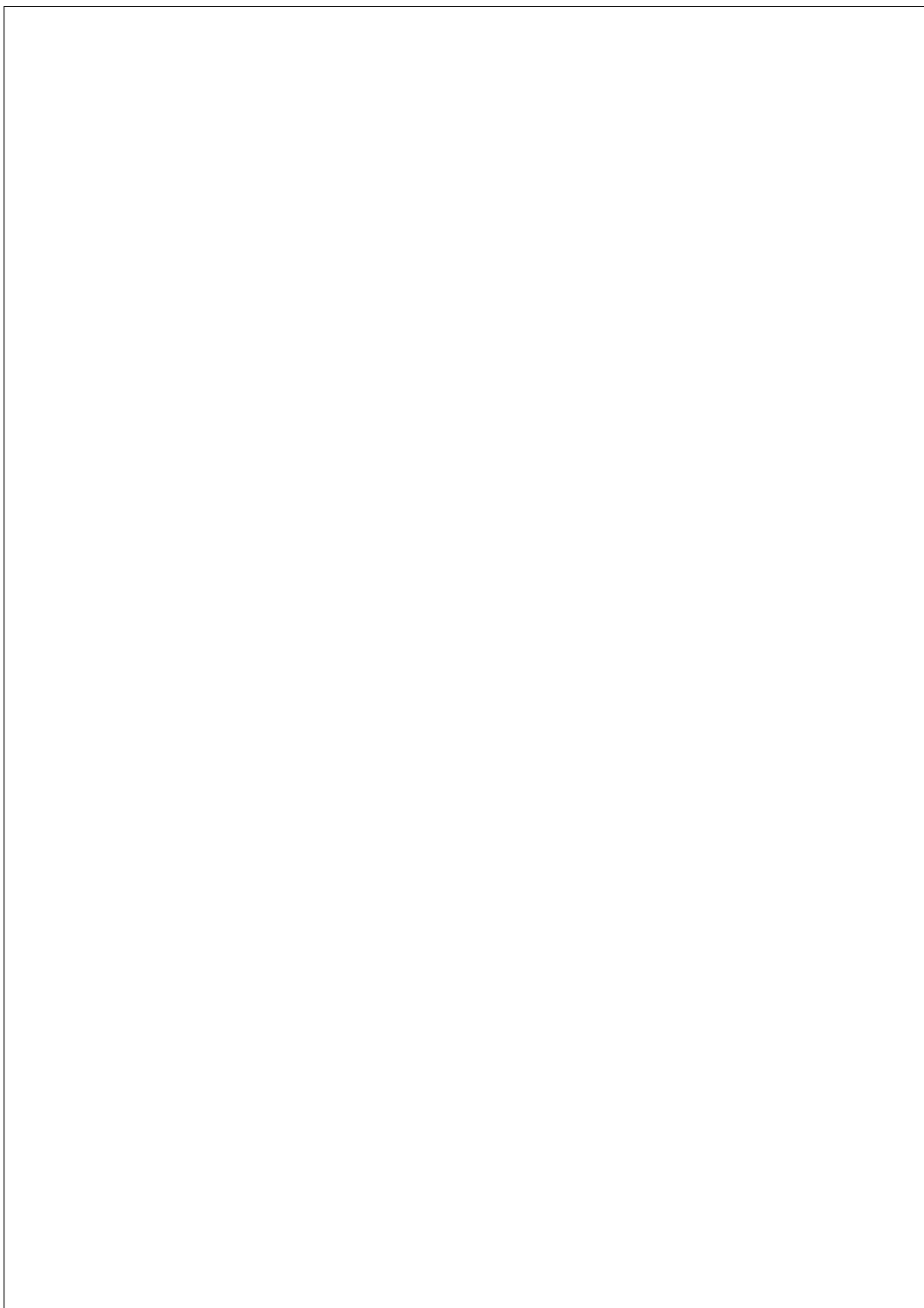
Folgende Teilaufgaben sind zu bearbeiten

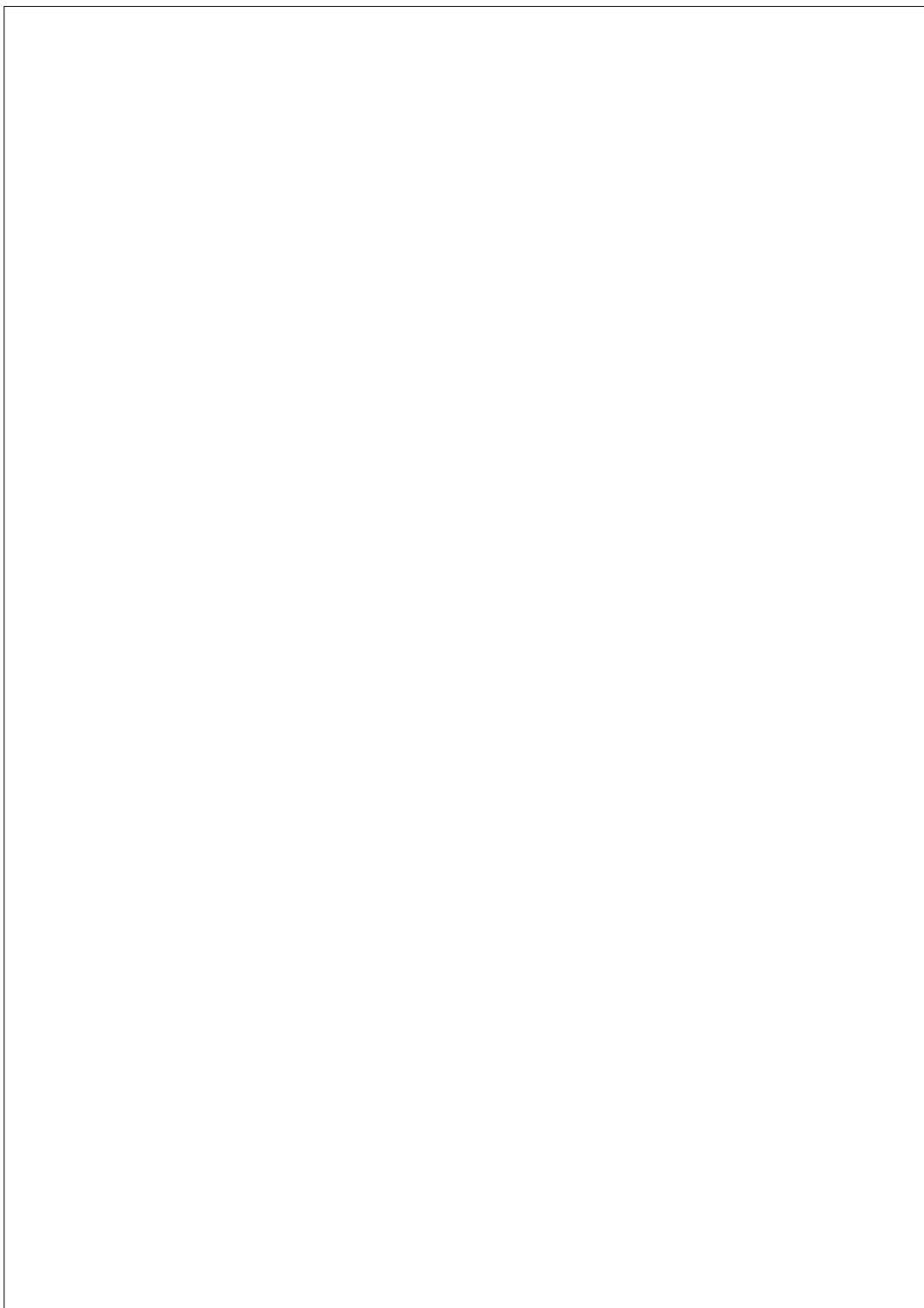
- Literature survey on the state of the art on wireless and AI technologies
- Set up an SDR module based on USRP platform
- Implement over-the-air LTE access through open-source software
- Perform real-time signal analysis and measurement according to the requirement of AI algorithms and acquire the measurement data
- Evaluate the performance of different algorithms
- Integrate AI algorithms and perform experiments on the platform
- Documentation of the hardware and software implementation, the design of AI-based algorithms, collected data sets, and experimental results

Hardware and software required for implementation are provided by DFKI.

All development and test steps must be carefully documented in electronic form, through test and test reports as well as accompanying text documentation.

The work is to be written in English or German and, if necessary, a short version in English.





Nondisclosure agreement

This master's thesis will be completed at the company *Deutsches Forschungszentrum für Künstliche Intelligenz GmbH*. Beginning with the closing date the results of this work have to be kept confidential for *five years*. Granting a third party access to this work is forbidden during this period of time. In case this work or parts of it are made publicly available by the author, the institute asks for a copy of this publication.

Examiner: Prof. Dr.-Ing. H.-C. Reuss

Adviser: Dr.-Ing. D. Keilhoff

External adviser: Dr.-Ing. W. Jiang

Starting date: September 09, 2019

Closing date: March 10, 2020

Qiheng Zhou

Prof. Dr.-Ing. H.-C. Reuss

Declaration of authorship

I hereby declare that I, Qiheng Zhou, completed this work on my own, did not make use of any unauthorized help, did not use any than the quoted resources and sufficiently highlighted the passages taken from the quoted resources.

The work has not been subject to any other examination procedure. Furthermore, it has not been already published in full or in parts. The electronic copy is identical to the other copies.

Kaiserslautern, March 10, 2020

Qiheng Zhou

Abstract

Long Term Evolution (LTE), the Fourth Generation Wireless Technology (4G) has been adopted by all major operators in the world and has already ruled the cellular landscape for around a decade. Driven by the need to cope with exponentially growing mobile data traffic and to support new traffic types from massive numbers of machine-type devices, currently, they are also formed as the starting point for further progress beyond the 4G mobile network towards the Fifth Generation Wireless Technology (5G). A lot of researches and new technologies are being considered as a potential element contributing to such a future mobile network, including the use of massive Multiple-Input and Multiple-Output (MIMO), exploiting unused spectrum, Cloud Radio Access Network (RAN)s. The lack of the realistic and flexible experimentation platforms has limited and slowed the landing of new approaches because the new technologies need a wide range of experimentation modes from real-world experimentation to controlled and scalable evaluations. Motivated by the outstanding performance of the approached channel prediction algorithms on the simulated MIMO channel, however, we have noticed that the simulated channel cannot capture the complex real-world environment well. This has resulted in the need for building a real wireless communication platform so that we can perform some real value channel measurements for further algorithm verification and development. Software Defined Radio (SDR) may provide flexible, upgradeable and longer lifetime radio equipment for the wireless communications infrastructure. SDR may also provide more flexible and possibly cheaper multi-standard-terminals for end users. Among all the General Purpose Processor (GPP)-based SDR systems, OpenAirInterface (OAI) is one of the most comprehensive and competitive open-source SDR systems. By altering the open-source code individually, we can freely perform the real value measurement.

This thesis provides a real LTE access based on the SDR OAI platform for verification of the channel prediction algorithm. Firstly, the experimentation platform will be established by using OAI, Universal Software Radio Peripheral (USRP) and commercial User Equipment (UE). Then, the author of this thesis has analyzed the source code of OAI and changed some parts of the codes so that the real-time over-the-air channel measurement can be achieved. The results from the measurement are then formed so that the channel prediction algorithm can be retrained and tested. The results of the test illustrate that the implemented experimentation platform can meet the need for algorithms' verification and can be further extended for more developments of algorithms.

Kurzzusammenfassung

LTE 4G wurde von allen großen Betreibern der Welt übernommen und beherrscht die zellulare Landschaft bereits seit rund einem Jahrzehnt. Aufgrund der Notwendigkeit, den exponentiell wachsenden mobilen Datenverkehr zu bewältigen und neue Verkehrstypen von einer großen Anzahl von Geräten vom Maschinentyp zu unterstützen, bilden sie derzeit auch den Ausgangspunkt für weitere Fortschritte über das 4G Mobilfunknetz hinaus in Richtung der 5G. Viele Forschungen und neue Technologien werden als potenzielles Element angesehen, das zu einem solchen zukünftigen Mobilfunknetz beiträgt, einschließlich der Verwendung von MIMO, der Erschließung der nicht genutzten Frequenzen, Cloud RAN. Das Fehlen realistischer und flexibler Experimentierplattformen hat die Landung neuer Ansätze der Technik begrenzt und verlangsamt, da die neuen Technologien eine breite Palette von Experimentiermodi erfordern, von realen Experimenten bis hin zu kontrollierten und skalierbaren Bewertungen. Motiviert durch die hervorragende Leistung der Algorithmen für die Vorhersage der Kanäle auf dem simulierten MIMO Kanal haben wir jedoch festgestellt, dass der simulierte Kanal die komplexe reale Umgebung nicht gut erfassen kann. Dies hat dazu geführt, dass eine echte drahtlose Kommunikationsplattform aufgebaut werden muss, damit wir einige echte Kanalmessungen zur weiteren Überprüfung und Entwicklung von Algorithmen durchführen können. SDR bietet möglicherweise flexible, aktualisierbare und lebenslange Funkgeräte für die drahtlose Kommunikationsinfrastruktur. SDR bietet Endbenutzern möglicherweise auch flexiblere und möglicherweise billigere Multi-Standard-Terminals. OAI ist eines der umfassendsten und wettbewerbsfähigsten Open-Source SDR Systeme unter allen GPP basierten SDR Systemen. Indem wir den Open-Source-Code individuell ändern, dann können wir die realwertige Kanalmessung frei durchführen.

Diese Arbeit bietet einen echten LTE-Zugang basierend auf der SDR OAI Plattform zur Überprüfung des Kanalvorhersagealgorithmus. Zunächst wird die Experimentierplattform unter Verwendung von OAI USRP und kommerzieller UE eingerichtet. Anschließend hat der Autor dieser Arbeit den Quellcode von OAI analysiert und einige Teile der Codes geändert, damit die echtzeit Kanalmessung über Kanal erreicht werden kann. Die Ergebnisse der Messung werden dann gebildet, so dass der Kanalvorhersagealgorithmus umgeschult und getestet werden kann. Die Testergebnisse zeigen, dass die implementierte Experimentierplattform die Notwendigkeit der Überprüfung von Algorithmen erfüllen und für die weitere Entwicklung von Algorithmen erweitert werden kann.

Contents

Acronyms and symbols	xi
List of Figures	xv
List of Tables	xvii
1 Introduction	1
1.1 Background	1
1.2 Motivation	2
1.3 Organization of Thesis	4
2 State Of The Art	5
2.1 Software Defined Radio (SDR)	5
2.2 System Architecture of OAI	10
2.3 Channel Estimation for Channel Prediction	15
2.3.1 Channel Estimation	15
2.3.2 Uplink frame structure	19
2.3.3 Uplink Data Transmission Scheduling	22
3 Implementation of LTE Access Platform	25
3.1 Software and IP Setup	25
3.2 Installation of Evolved Node B (eNB)	27
3.3 Installation of Evolved Packet Core (EPC)	30
3.4 User database configuration	36
3.5 Process OAI system startup and UE attachment	40
4 Implementation of Channel Measurement	49
4.1 OAI eNB source code analysis	49
4.2 Streaming SIMD Extensions (SSE) intrinsics and output data storage	54
4.3 Channel estimates formatting and visualizing	61
4.4 Changes of uplink Scheduling	65

5	Verification of Channel Prediction Algorithm	69
5.1	RNN	70
5.2	Recurrent Neural Network (RNN) for Channel Prediction	72
5.3	Numerical results and evaluations	74
6	Conclusion and Future Works	79
6.1	Conclusion	79
6.2	Future Works	80
	Bibliography	83

Acronyms and symbols

Acronyms

3GPP	3rd Generation Partnership Project
4G	the Fourth Generation Wireless Technology
5G	the Fifth Generation Wireless Technology
ACK	Acknowledgement
AI	Artificial Intelligent
APIs	Application Program Interfaces
AVX	Advanced Vector Extensions
CFI	Control Format Indicator
CPU	Central Processing Unit
CQI	Channel Quality Index
CSI	Channel State Information
DCI	Downlink Control Indicator
DDC	Digital Down Converter
DHCP	Dynamic Host Configuration Protocol
DRS	Demodulation Reference Signal
DSP	Digital Signal Processor
DUC	Digital Up Converter
eNB	Evolved Node B
EPC	Evolved Packet Core
FDD	Frequency Domain Division
FPGA	Field Programmable Gate Array
GPP	General Purpose Processor
GTP	GPRS Tunnelling Protocol
HARQ	Hybrid Automatic Repeat Request
HSS	Home Subscriber Server
I/Q	In-phase and Quadrature
IF	Intermediate Frequency

JTRS	The Joint Tactical Radio System project
KUAR	Kansas University Agile Radio
LNA	Low-Noise Amplifier
LO	Local Oscillator
LTE	Long Term Evolution
MAC	Media Access Control Layer
MIMO	Multiple-Input and Multiple-Output
MME	Mobility Management Entity
MSE	Mean Square Error
NAK	Negative Acknowledgement
NAS	Non-Access Stratum
NN	Neural Network
NR	New Radio
OAI	OpenAirInterface
PBCH	Physical Broadcast Channel
PCFICH	Physical Control Format Indicator Channel
PDCCH	Physical Downlink Control Channel
PDCP	Packet Data Convergence Protocol
PDSCH	Physical Downlink Shared Channel
PGW	Public Data Network (PDN) Gateway
PHICH	Physical channel HybridARQ Indicator Channel
PHY	Physical layer
PMCH	Physical Multicast Channel
PMI	Precoding Matrix Indicator
PRACH	Physical Random Access Channel
PSS	Primary Synchronization Signal
PUCCH	Physical Uplink Control Channel
PUSCH	Physical Uplink Shared Channel
RAN	Radio Access Network
RF	Radio Frequency
RI	Rank Indicator
RLC	Radio Link Control
RNN	Recurrent Neural Network
RRC	Radio Resource Control Protocol
RS	Reference Signal
SCA	Software Communications Architecture
SDR	Software Defined Radio
SGW	Serving Gateway

SIMD	Single Instruction Multiple Data
SISO	Single-Input and Single-Output
SNR	Signal to Noise Ratio
SRS	Sounding Reference Signal
SSE	Streaming SIMD Extensions
SSS	Secondary Synchronization Signal
TDD	Time Domain Division
UCI	Uplink Control Indicator
UE	User Equipment
UHD	USRP Hardware driver
USRP	Universal Software Radio Peripheral
WARP	Wireless open-Access Research Platform
ZC	Zadoff–Chu

Symbols

Symbol	Unit	Description
h		communication channel
\bar{h}		averaged channel estimate
f_n		reference signal
x		transmitted signal
y		received signal
n		estimated noise
T_S	<i>ms</i>	the time duration for one frame
T_s	<i>ms</i>	the time duration for one subframe
T_{slot}	<i>ms</i>	the time duration for one slot
k		the index of RE in the frequency domain
l		the index of SC-FDMA symbols in the time domain
N_{RB}^{UL}		the number of RB in the frequency domain
N_{SC}^{RB}		the number of subcarriers in a RB
N_{symp}^{UL}		the number of SC-FDMA symbols in an uplink slot
N		the number of input neurons
N_I		the number of external inputs
N_L		the number of hidden neurons
N_o		the number of output neurons
x_e		vector of external inputs

y_o	output vector
X	combination vector of x_e and delayed feedback
$w_{l,n}$	the weight connection between the n^{th} input and the l^{th} hidden neuron
$c_{m,l}$	the weight connection between the l^{th} hidden and the m^{th} output neuron
w_l	the l^{th} weight vector
$\gamma_l(t)$	the output of the l^{th} hidden neuron

List of Figures

2.1	General Architecture of Radio Transceiver	6
2.2	The First Evolution of SDR	7
2.3	The Second Evolution of SDR	8
2.4	The open-source USRP-based SDR system architecture	9
2.5	OpenAirInterface LTE software stack [26]	11
2.6	OpenAirInterface LTE RAN system directory	14
2.7	Block diagram of a simple MIMO system channel prediction	16
2.8	Channel estimation by comparing transmitted and received reference signal	17
2.9	Uplink frame structure	19
2.10	LTE uplink channel resource grid in a slot [3]	20
2.11	Reference signal distribution in a subframe	21
2.12	Uplink transmission under persistent scheduling	24
3.1	LTE platform hardware set up	26
3.2	LTE platform IP set up	27
3.3	CPU state setup result in i7z	29
3.4	IP configuration status of computer runs EPC	34
3.5	SIM card Info in database	38
3.6	Idmmeidentity of hostname	39
3.7	APN setting	39
3.8	pgw setting	40
3.9	pdn setting	40
3.10	The running status of Home Subscriber Server (HSS)	41
3.11	The running status of Mobility Management Entity (MME)	42
3.12	The additional status of HSS with connected MME	42
3.13	The running status of SPGW	43
3.14	LTE softmodem monitor and event selector without UE connected	45
3.15	LTE softmodem monitor with UE connected	46
3.16	The screenshots for displaying the LTE connection status	46

4.1	PHY folder catalogue structure	50
4.2	Single Instruction Single Data structure	54
4.3	Multiple Instruction Single Data structure	54
4.4	Single Instruction Multiple Data structure	55
4.5	Multiple Instruction Multiple Data structure	55
4.6	Storage format of 128 bit data stream	56
4.7	Outputs of channel estimates	63
4.8	Estimated channel spectrum	64
4.9	Estimated channel spectrum of a single subcarrier	64
4.10	Estimated channel spectrum after having changed the scheduling .	68
4.11	Estimated channel spectrum of a single subcarrier after having changed the scheduling	68
5.1	Schematic diagram of a recurrent neural network [19]	73
5.2	Channel prediction on the symbol continuous envelope of simulated channel estimates[19]	77
5.3	Channel prediction on the envelope of measured channel estimates	77
5.4	Channel prediction algorithm training progress	78
5.5	Channel prediction on the envelope of measured channel estimated using adam solver	78

List of Tables

3.1	Key parameters of SIM card	47
-----	--------------------------------------	----

Introduction

1.1 Background

As we all know, it's been a decade in the making, but 5G is finally becoming a reality. Carriers started rolling out fixed 5G to select cities a few years ago, and mobile 5G has already made an appearance in cities around the world. Rapidly growing mobile data traffic fuelled by the wide adoption of internet-connected mobile devices (smartphones, tablets, etc.) and expected great increase in the number of connected devices with different traffic patterns from personal communication devices has created a wide field for innovation in terms of mobile broadband protocols, algorithms, architecture, and services. Thus, while 4G LTE networks are globally popularized, research for the next-generation mobile networks has already begun with the examination and evaluation of candidate technologies and architectures [5, 4]. Paralleled to those contributions, A large number of researches are done, which utilize artificial intelligent methods, that process the outdated Channel State Information (CSI) to forecast the future CSI, so that the channel predictor can achieve a very high prediction accuracy in a simulated fast fading channel without any prior knowledge [18, 19, 20]. However, performance assessment of candidate innovations over wireless transmission technologies requires strict evaluation and real-world validation before deployment. While software for over-the-air simulation has evolved significantly over the years, it still cannot capture the complex real-world environment completely, and real-world experimentation is still considered essential, especially at the later stages of technology development. Real-world evaluation over platforms with commercial LTE equipment, however, restrict individual configuration capabilities, flexible deployment and extension to a certain extent due to constraints imposed by operators and large vendors, mainly because of commercial considerations. This has resulted in the need for an open and flexible

wireless (LTE, 5G) experimentation platform with a high degree of flexibility that the researchers can commonly use for understanding the complexities associated with real-world settings while at the same time obtain reproducible and verifiable results, all with the same ultimate aim of developing effective and future-oriented system architectures and technologies.

1.2 Motivation

All those needs above are satisfied until the SDR was well developed. SDR system has a great advantage in simulating and verifying the new communication technologies attributed to its striking advantage in flexibility and reconfigurability. The key idea of SDR is to construct a universal hardware platform, which is open, standardized and modularized. All sorts of communications modules are defined and realized by software, i.e., frequency band selection, modulation & demodulation, encoding & decoding, amplifier, filter, and communications protocol. To make the SDR system more flexible, the A/D and D/A converters must be close to the antenna to reduce the analog modules [1]. Thus, technology upgrading will only need to update the software rather than the hardware, which can save the research time and cost. The SDR systems can be developed on different hardware platforms, i.e., the Field Programmable Gate Array (FPGA), Digital Signal Processor (DSP), GPP [34]. Compared to FPGA and DSP solutions, the GPP-based SDR system has advantages in flexibility and reconfigurability, because it's easy to develop and upgrade on the GPP using high-level programming languages, rather than hardware description languages. Compared to DSP solutions, the GPP-based system is more powerful in processing wide-band signals. A typical GPP-based SDR system consists of a single piece of peripheral equipment connecting to the GPP. The peripheral equipment, e.g., USRP, is mainly responsible for frequency conversion and digitization and the GPP is responsible for baseband signal processing [37]. Building upon open-source software development, open-source SDR is being developed rapidly with the support of the open-source developer community.

With the evolution from 4G towards 5G wireless communications, some existing SDR-based LTE implementations can be utilized for the performance assessment of 5G candidate techniques, e.g., AmarisoftLTE, OpenLTE, OAI. Currently, AmarisoftLTE is the most complete SDR-based implementation of the LTE base station and core network, which can interoperate with commercial UE. However,

AmarisoftLTE is a closed-source commercial LTE system. OpenLTE is an open-source-based partial implementation of the 3rd Generation Partnership Project (3GPP) LTE specifications based on GNU Radio <https://www.gnuradio.org/>. By contrast, OAI is fully open-source and provides a complete software implementation of the entire LTE system architecture, which is more appealing and appropriate for developing and evaluating our proposed algorithm. OAI is a flexible platform created by the Mobile Communications Department at EURECOM, with the purpose of innovations in the area of wireless networking and communications [27]. As an excellent over-the-air experiment platform, OAI can verify various new communication technologies by changing some code but not the hardware.

While up to now, the New Radio (NR)(5G) part in OAI hasn't provided a stable S1 interface for the internet connection. In consideration of the present way of 5G deployment, most of the approaches inherit the LTE network structure and combine with the multi-antenna technologies, such as MIMO [12, 22]. To verify and optimize the Artificial Intelligent (AI) algorithms proposed through the simulated wireless transmission channel [18, 19], a real LTE daily connection scenario between the LTE base station and UE (smartphone) is implemented, so that we can perform wireless channel measurement based on stable data transmission (Single-Input and Single-Output (SISO)) between the UE and base station. That is, we implement an SDR platform, which is based on the LTE part of OAI and USRP. Besides, a smartphone is configured to connect the based station. Successful platform construction is that the smartphone can connect to the radio signal from the base station and surf the internet stably. Because of this setup, we have only built the eNB and EPC of LTE using the OAI source code. So our work will focus on altering the source code of uplink channel processing, e.g., uplink channel estimation, uplink channel demodulation, uplink channel decoding, and uplink scheduling. By doing those works above, we can achieve a stable and constant channel measurement every 0.5 ms through the uplink procedure. Last but not least, the extracted complex-valued channel estimates from the channel measurement are then used for algorithm verification and optimization. The testing results prove the feasibility of the real-world channel measurement method based on SDR. By learning and testing on more complex channel state information from the real world, the proposed AI algorithms can be much more robust and convincing.

1.3 Organization of Thesis

The rest of this paper is organized as follows: in chapter 2, we will introduce some basic concepts and principles that are important for understanding most parts of our thesis, including the SDR technology, OAI architecture, channel estimation and channel/frame structure of LTE. Based on those principles, the implementation of the LTE access platform will be presented in chapter 3, where we will introduce the configuration and installation of OAI eNB, EPC, EPC database, USRP, and commercial UE. After we have implemented the experimentation platform of the LTE-access, we will then perform the real-time channel measurement motivated by the AI-based channel prediction algorithms, which is illustrated in chapter 4. For the final purpose of the channel measurement, we will then form and normalize the measured data for the testing of the algorithms, the RNN for channel prediction and the final results are depicted in chapter 5. At last, all our works in this thesis will be concluded in chapter 6, in addition, the future works are also conceived in this chapter.

State Of The Art

In this chapter, the relevant current technology and basic knowledge of this thesis will be introduced in detail. The goal of this thesis is to implement a software-defined radio platform by using OAI and USRP B210 for LTE access. Configured smartphone with SIM card can connect to this generated LTE network and surf the internet without any problem. By altering the source code of the eNB part of OAI with reference of the 3GPP TS 36.211 [3], TS 36.212 [2] and TS 36.213 [10], we can achieve a constant uplink channel measurement on every slot of the subframes, if the UE (smartphone) upload files with big data size. The output from channel measurement will be learned from the RNN-based channel predictor at the end. Those works involve SDR, OAI hardware network configuration, OAI software structure, channel demodulation, channel estimation, uplink channel scheduling, channel prediction and RNN.

This chapter will first introduce SDR technology in section 2.1. Section 2.2 presents the architecture of the OAI system. Some basic knowledge and details of channel estimation for channel prediction is explained in section 2.3, including channel estimation method, uplink frame structure, and uplink transmission scheduling.

2.1 Software Defined Radio (SDR)

SDR is a radio technology first formulated by Joseph Mitola III in 1991. The main idea of SDR is to use software applications, which can implement radio communication tasks on the computing platform, and to replace the traditional idea of designing a communication system by using hardware [34]. In other words, SDR uses a single hardware frontend but can flexibly change its frequency of operation, allocated bandwidth, and implementation of wireless transmission standards by

invoking software algorithms. The evolution towards SDR systems has been driven in part by the evolution of the enabling technologies, first and foremost the D/A and A/D converters and the DSPs, but also that of the GPPs and FPGAs. In [7], the hardware architectures of SDR are grouped into five categories, i.e., the GPP, co-processor, processor-centric, configurable units, and programmable blocks. Some hardware platforms in each category are introduced in detail, such as the USRP, Kansas University Agile Radio (KUAR), Wireless open-Access Research Platform (WARP), etc. A software framework provides an adaptive programming environment when developing SDR applications. Most existing SDR software frameworks are developed based on the Software Communications Architecture (SCA), which is a dataflow-based framework commissioned by the US Department of Defense via The Joint Tactical Radio System project (JTRS). Some typical software frameworks are introduced in [29, 7], such as GNU Radio, open-source SCA implementation, etc.

With various technical challenges, SDR has been evolved in multiple steps. Firstly, let's look into the general structure of a Radio Communication Device (RCD). It would be illustrated as shown in Figure 2.1. As we see, it is a pretty complicated hardware structure and relatively small portions of the software at the very end.

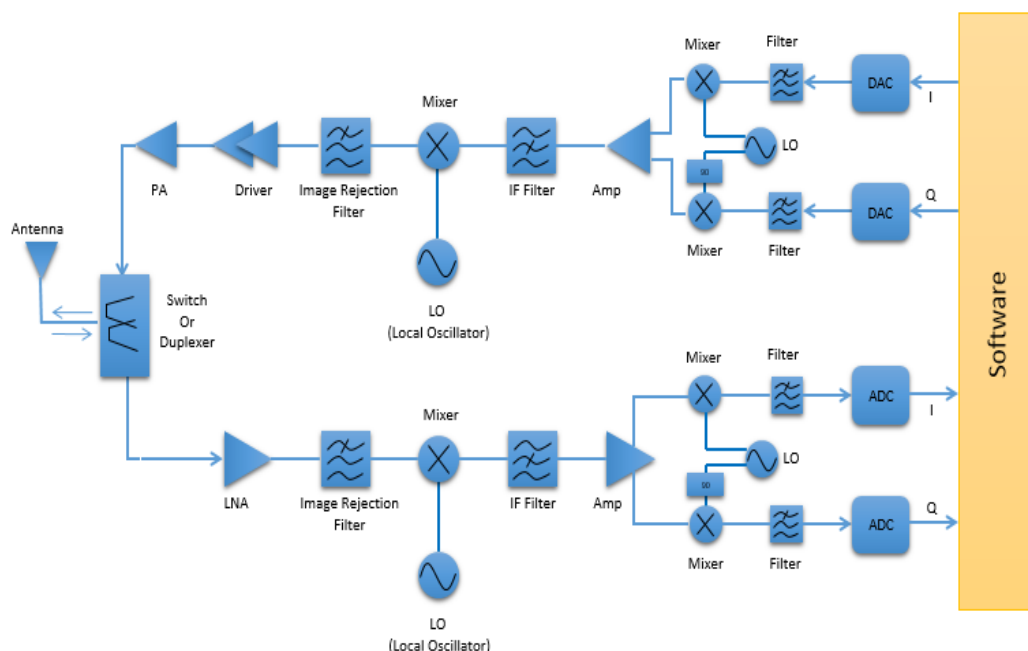


Figure 2.1: General Architecture of Radio Transceiver

The evolution of RCD has been done to simplify these structures. Which part of the hardware can be replaced by Software? The ideal solution that we may want to get would be as follows. Basically, this is for replacing almost all the hardware except the antenna, the A/D converter and the D/A converter. This would be the goal of the current SDR. Even though we have such a goal, as shown above, it will be tricky to implement this kind of SDR unless we have A/D and D/A converter with a super wide dynamic range and extremely high resolution. In addition, the A/D and D/A converter sampling rate should be extremely high if the radio frequency is very high.

The first stage of SDR has been as follows in Figure 2.2. At this stage, AD/DA converter is done at Intermediate Frequency (IF) stage and removes the most of hardware (downconvert/upconvert for IF to baseband and various filters) for baseband processing.

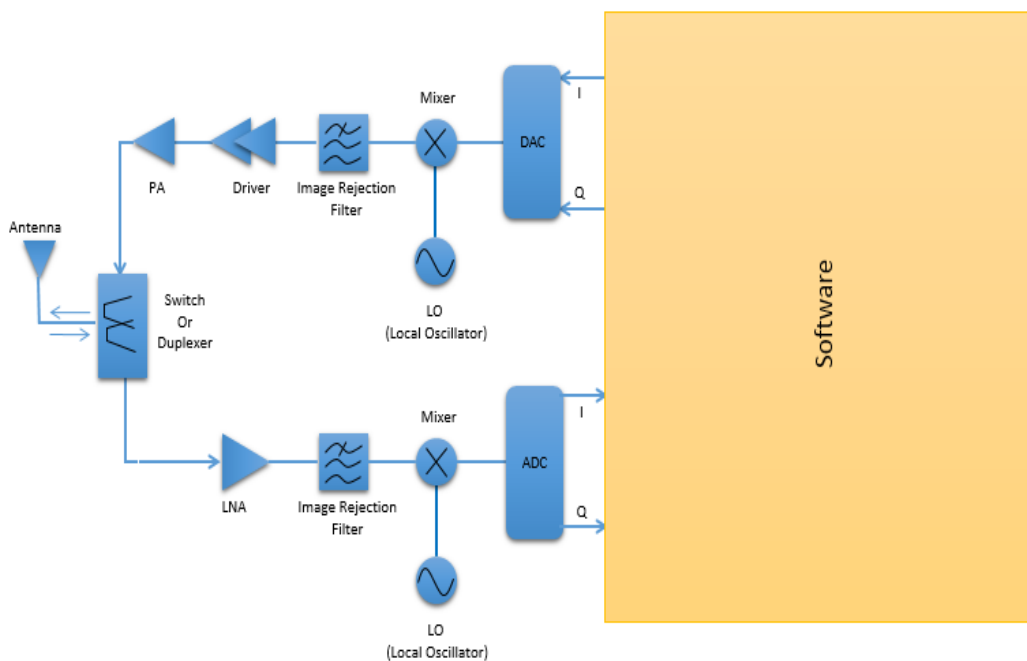


Figure 2.2: The First Evolution of SDR

The next stage of evolution would look as shown in Figure 2.3. The A/D and D/A converters are done directly at the Radio Frequency (RF) stage, but a couple of Amplifiers are still remaining at the hardware stage. As of now, we would see SDR in both as in Figure 2.2 and Figure 2.3 depending on the application.

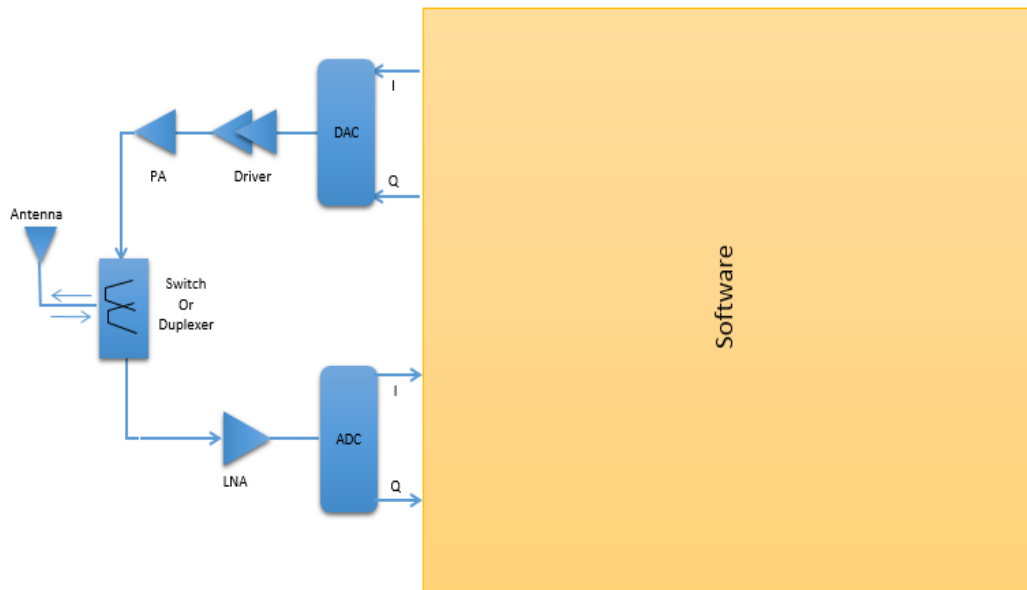


Figure 2.3: The Second Evolution of SDR

At present, most of the powerful products are at the first stage of SDR evolution. The most common hardware platform used by the GPP approach is the USRP family of devices from Ettus Research <https://www.ettus.com/>, which includes an inexpensive hardware frontend and an acquisition board with open designs and free schematics. Considering the compatibility with the LTE system and the practicability, we select USRP B210 as the peripheral equipment. With continuous frequency coverage from 70MHz to 6GHz, all LTE frequency bands can be used. Besides, its 61.44 MS/s sampling rate is a multiplier of all the sampling rates defined in 3GPP LTE specifications, which benefits the reduction of complexity of baseband signal processing in the GPP. The USRP consists of four key components as illustrated in Figure 2.4 in [37], i.e., the radio frequency frontend, A/D converter, D/A converter, FPGA, and an interface (USB 3.0).

The RF frontend is mainly responsible for frequency conversion and band selection, which operates in the analog domain. The RF signal received at the antenna is selected by a bandpass filter and amplified by a Low-Noise Amplifier (LNA). It is then down-converted to baseband or IF by a mixer and a Local Oscillator (LO). The A/D converter plays a significant role in digitization. The baseband or IF signal is converted to the digital domain, where it is further processed using digital signal processing methods. Since only a discrete set of frequencies can be generated by the LO synthesizer, the baseband signal cannot be tuned to the zero frequency

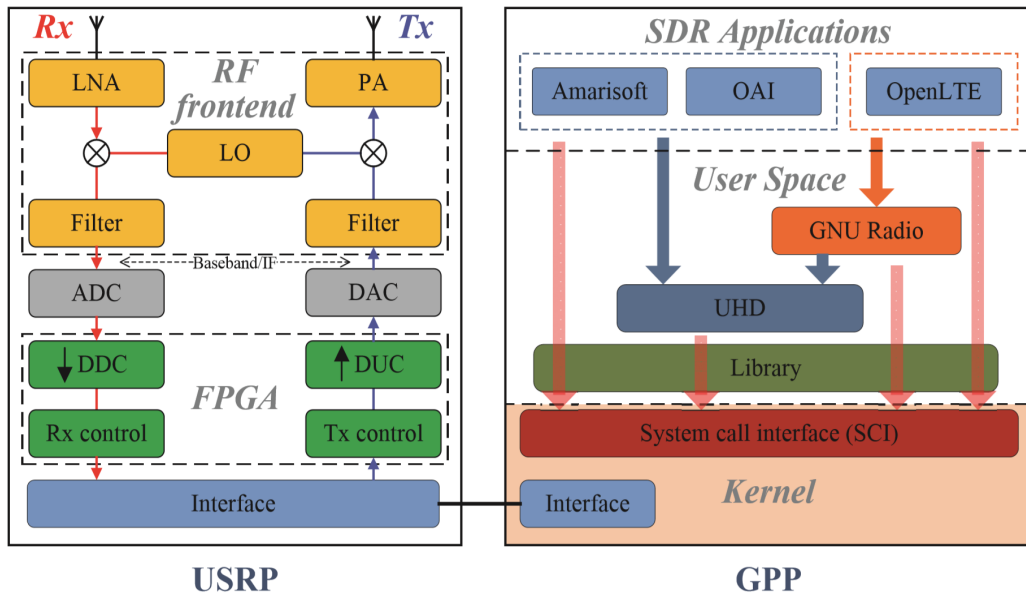


Figure 2.4: The open-source USRP-based SDR system architecture

precisely at the previous mixing stage. Considering the IF architecture of the RF frontend, a second mixing stage is necessary, which is realized with FPGA using a Digital Down Converter (DDC). The DDC removes any residual frequency offset and shifts the signal digitally. Moreover, decimation is conducted in the DDC to reduce the sample rate without violating the Nyquist criterion, which helps satisfy the data transfer capability of the interfaces, and meets easier further processing of the signal of interest. Finally, the digital baseband signal is transferred to the GPP via the interface. The transmitting path operates in a reverse manner as the receiving path. The GPP converts the sampled digital signal into a waveform, and then sends it to the USRP via the interface. After digital frequency conversion and interpolation using a Digital Up Converter (DUC), the digital signal is converted to an analog signal by the D/A converter, and then up-converted to the target RF signal.

The hardware part of the SDR is discussed above. After the baseband signal is transmitted via the interface to the GPP, all the baseband signal are processed only in the GPP. An operating system is necessary in GPP, which manages all the hardware resources, such as the Central Processing Unit (CPU), memory card, hard disk, interfaces, etc. The widest used open-source operating system is GNU/Linux, where a wide range of software libraries are available for free. This helps to accelerate the development speed as well as reducing development costs.

The USRP Hardware driver (UHD) is the hardware driver for all USRP devices, which provides a series of Application Program Interfaces (APIs) for programmers or engineers to control USRPs. These APIs provide numerous functions, such as frequency tuning, RF bandwidth setting, data transmit/receive, etc. This demonstrates the flexible and reconfigurable capabilities of SDR technology.

Actually, it is not easy to build an SDR application by using an independent UHD. Various programming techniques should be used for efficient signal processing. The open-source GNU radio is developed and compatible with the UHD, which is regarded as the most widespread real-time signal processing framework. GNU Radio has a set of signal processing tools and provides a library of signal processing blocks. These blocks are written in C and C++, while the signal flow graphs and visualization tools are mainly constructed using Python to connect the signal processing blocks. Aided by the signal processing framework, the development of SDR applications is streamlined. The GNU project is also strongly supported by the developer community all over the world. Based on those stable and efficient baseband signal processing, some applications are implemented for building 5G wireless transmission standard based networks. As is already introduced in chapter 1, compared to Amarisoft and OpenLTE, OAI is fully open-source and provides a complete software implementation of the entire LTE system architecture, which is more appealing and appropriate for developing and evaluating our proposed algorithm. So in this thesis, we will take the advantages of OAI, and build an USRP and OAI based LTE access. The system architecture of OAI will be introduced in the next section.

2.2 System Architecture of OAI

OAI is an open-source platform for software/hardware development for the core network (EPC) and access-network (EUTRAN) of the 3GPP cellular network, which implements the entire LTE protocol stack and provides all elements of the 4G LTE system architecture [26]. There are mainly two directions of usage of the OAI platform, real-time RAN experimentation and emulation [36]. In this thesis, we focus on the real-time RAN feature of OAI.

There are some features at the Physical layer provided by OAI in [26] at the time of the publication of this paper:

- LTE release 8.6 compliant, with a subset of release 10;
- Frequency Domain Division (FDD) and Time Domain Division (TDD) configuration in 5, 10, and 20 MHz bandwidth;
- Transmission mode: 1 (SISO), and 2, 4, 5, and 6 (MIMO);
- Channel Quality Index (CQI)/Precoding Matrix Indicator (PMI) reporting ;
- All Downlink channels are supported: Primary Synchronization Signal (PSS), Secondary Synchronization Signal (SSS), Physical Broadcast Channel (PBCH), Physical Control Format Indicator Channel (PCFICH), Physical channel HybridARQ Indicator Channel (PHICH), Physical Downlink Control Channel (PDCCH), Physical Downlink Shared Channel (PDSCH), Physical Multicast Channel (PMCH);
- All Uplink channels are supported: Physical Random Access Channel (PRACH), Physical Uplink Shared Channel (PUSCH), Physical Uplink Control Channel (PUCCH), Sounding Reference Signal (SRS), Demodulation Reference Signal (DRS);

At present, the OAI has followed the LTE release to the release 14 and provided up to 100MHz bandwidth. The new radio (5G) is also implemented in the OAI, but only for downlink transmission. In our case, we still focus on the LTE application structure. Figure 2.5 shows a schematic of the implemented LTE protocol stack in OAI.

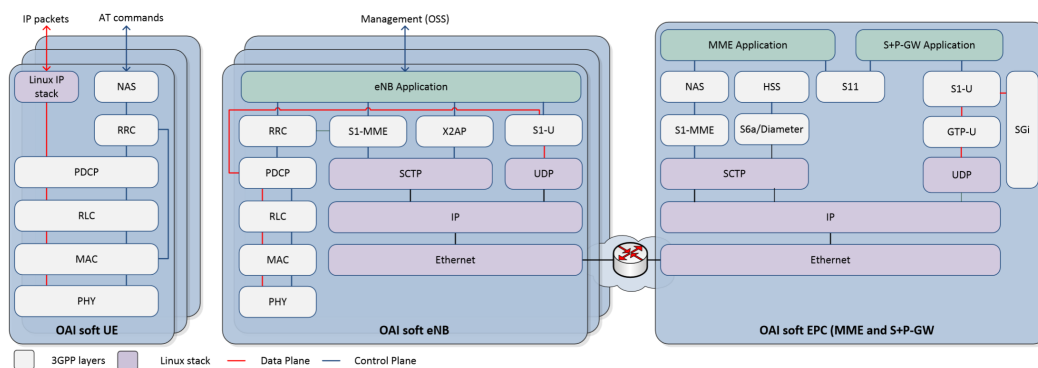


Figure 2.5: OpenAirInterface LTE software stack [26]

On the right of Figure 2.5 is the EPC. It consists of 4 main parts, respectively are MME, HSS, Serving Gateway (SGW), Public Data Network (PDN) Gateway

(PGW). MME can be seen as a center for all signaling message, which has following functionalities:

- Idle mode UE tracking Process
- Paging Process
- Bearer activation/deactivation process
- Choosing the SGW for a UE at the initial attach
- Core Network (CN) node relocation at time of intra-LTE handover
- Authenticating the user (by interacting with the HSS)
- Destination of Non-Access Stratum (NAS) message
- Generation and allocation of temporary identities to UEs
- Authorization of the UE to camp on the service provider's Public Land Mobile Network (PLMN)
- Enforces UE roaming restrictions
- Termination Point for Ciphering/Integrity for NAS signaling
- Security key management
- Lawful interception of signaling is also supported by the MME

HSS is a central database that contains user-related and subscription-related information. Some functionalities are listed below:

- Mobility management
- Call and session establishment support
- User authentication and access authorization

SGW is a center for all user data (packet data), which:

- Routes and forwards user data packets
- Act as the mobility anchor for the user plane during inter-eNB handovers
- Act as the anchor for mobility between LTE and other 3GPP technologies
- Terminates the downlink data path and triggers paging when downlink data arrives for the UE when UE is in Idle mode

- Manages and stores UE contexts, e.g. parameters of the IP bearer service, network internal routing information
- Performs replication of the user traffic in case of lawful interception

PGW:

- Provides connectivity from the UE to external packet data networks
- Performs policy enforcement, packet filtering for each user, charging support, lawful Interception and packet screening

In our system, the EPC serves only as an application tool for authentication and registration of commercial UE, so that the UE can be searched and connected to the LTE RAN and then surf the internet through allocated IP from EPC. Thus, we will only care about the configuration files of EPC for a successful Internet connection.

On the left of Figure 2.5 is the UE part, which consists of functions of Physical layer (PHY), Media Access Control Layer (MAC), Radio Link Control (RLC), Packet Data Convergence Protocol (PDCP), Radio Resource Control Protocol (RRC) and NAS. In this thesis, we will use commercial UE (smartphone) instead, so we will not look deep inside this UE part structure. We will focus on the eNB part in the middle of this image, which is also called the RAN part. The EPC part is also important for UE management and IP arrangement. When we look into the system directory structure of the OAI RAN part in Figure 2.6, we can find that OAI uses CMAKE tool to generate makefiles, and then compile the system. Because there are too many files in OAI, CMAKE is a very efficient tool for compiling such a big system. Some mathematics/data structure tools are defined in the folder called common. Based on 3GPP LTE standards TS 36.211 [3], TS 36.212 [2] and TS 36.213 [10], there are some functions, which are designed in the folder of openair1 (PHY), openair2 (L2) and openair3 (L3) respectively. Some main files and configuration files are stored in the folder named target. At present, OAI is managed by Git.

When we face such a big system, we should not pay attention to all the files in detail. Actually, only some parts of the source codes should be noticed. OAI platform can be used in several different configurations involving commercial components to varying degrees:

- OAI EPC + OAI eNB + OAI UE
- Commercial EPC + OAI eNB + OAI UE

```

nano@nano:~/openairinterface5g$ ls
ci-scripts      d2d_emulator_setup.txt  nfapi      openair2    svn2git
cmake_targets  doc                     NOTICE.txt openair3    targets
common         LICENSE                 oaienv     pre-commit
configuration  maketags                openair1   README.txt
nano@nano:~/openairinterface5g$
  
```

Figure 2.6: OpenAirInterface LTE RAN system directory

- OAI EPC + commercial eNB + OAI UE
- Commercial EPC + commercial eNB + OAI UE
- OAI EPC + commercial eNB + commercial UE
- Commercial EPC + OAI eNB + commercial UE
- OAI EPC + OAI eNB + commercial UE

For stable and real-time processing, OAI eNB and OAI UE should be individually installed on two powerful computers. OAI EPC doesn't have a high requirement of the GPP computation power. In order to build the system as simple as possible, we have chosen the last way of configuration from the list above. That is, we build the OAI EPC on a mini PC for an individual core network and a flexible IP configuration. In addition, OAI eNB is installed on a powerful computer for real-time signal processing and network management. We use an Android Smartphone as a commercial UE so that we can build a daily user scenario. Moreover, the use of commercial UE can save an extra USRP and a powerful GPP.

Based on this setup, we will only focus on the uplink channel information. Because only the code of OAI eNB part can be changed and the channel measurement will also be performed on eNB. To keep a real-time data processing, especially for the LTE, which requires time slot alignment, it is necessary, that OAI utilize the multi-thread processing method to improve the computation ability of the computer. There are mainly 3 threads included in the OAI eNB, respectively are eNB_thread, eNB_thread_tx and eNB_thread_rx. The eNB_thread is mainly responsible for managing the data interaction with the external hardware devices, such as USRP. That is, eNB_thread writes the sending data to the USRP and transforming the

received radio frequency signal to the GPP. Apparently, the eNB_thread_tx is a thread for sending data, ranging from RRC layer to PHY layer. Correspondingly, the eNB_thread_rx is designed for receiving signals.

A successful uplink channel measurement from our set up will then be used for the verification of the new proposed channel prediction algorithm empowered by artificial intelligence. In the next section, we will introduce the principle of channel estimation and channel prediction.

2.3 Channel Estimation for Channel Prediction

Nowadays, an increasing number of researches [18, 19, 20] are focusing on dealing with the outdated CSI in a fast fading channel to accurately forecast the future CSIs, which can optimize the performance of a wide variety of wireless techniques, such as antenna selection [38], MIMO [31], Massive MIMO [33], cooperative relaying [35], physical layer security [17] and mobility management [32], etc.

Channel prediction helps to improve the performance of an operation on both sides of the transceiver and receiver, such as adaptive coding and modulation, decoding, channel equalization, and beamforming. Here we take a basic CSI prediction block diagram of a simple MIMO system as an example, see Figure 2.7.

As we can see from the figure, the channel prediction acts as a middleware for processing the channel estimates, so that the channel can be better decoded. The core idea of channel prediction is to combine the present and past CSIs to predict future CSIs. The channel prediction relies on a model to represent the fading channel, so after receiving the channel estimates from the channel estimation, the channel predictor can predict future CSI with the help of the estimated channel parameters. Thus, firstly, we should know the principle of channel estimation for extracting channel estimates from the uplink channel.

2.3.1 Channel Estimation

In all communications, the signal goes through a medium (called channel) and the signal gets distorted or various noise is added to the signal while the signal goes through the channel. To properly decode the received signal without errors

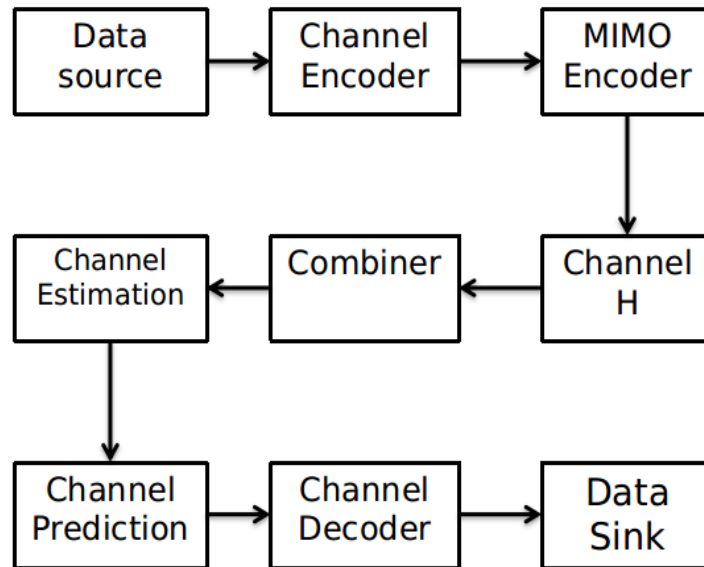


Figure 2.7: Block diagram of a simple MIMO system channel prediction

to remove the distortion and noise applied by the channel from the received signal. To do this, the first step is to figure out the characteristics of the channel that the signal has gone through. The process to characterize the channel is called channel estimation.

There are many different ways for channel estimation, but fundamental concepts are similar. The process is done as follows:

- Set a mathematical model to correlate transmitted signal and received signal using channel matrix.
- Transmit a known signal (we normally called this as reference signal or pilot signal) and detect the received signal.
- By comparing the transmitted signal and the received signal, we can figure out each elements of channel matrix.

As an example of this process, the channel estimation process in LTE is briefly described. A general algorithm of the channel estimation is illustrated as below in a very high level view (See Figure 2.8) [28]:

1. A set of predefined signal (This is called a reference signal) is embedded in both transmitted and received signals.

2. As the reference signal go through the channel h , it gets distorted (attenuated, phase-shifted, noised) along with other signals.
3. We detect/decode the received reference signal at the receiver.
4. Compare the transmitted reference signal and the received reference signal and find correlation between them.

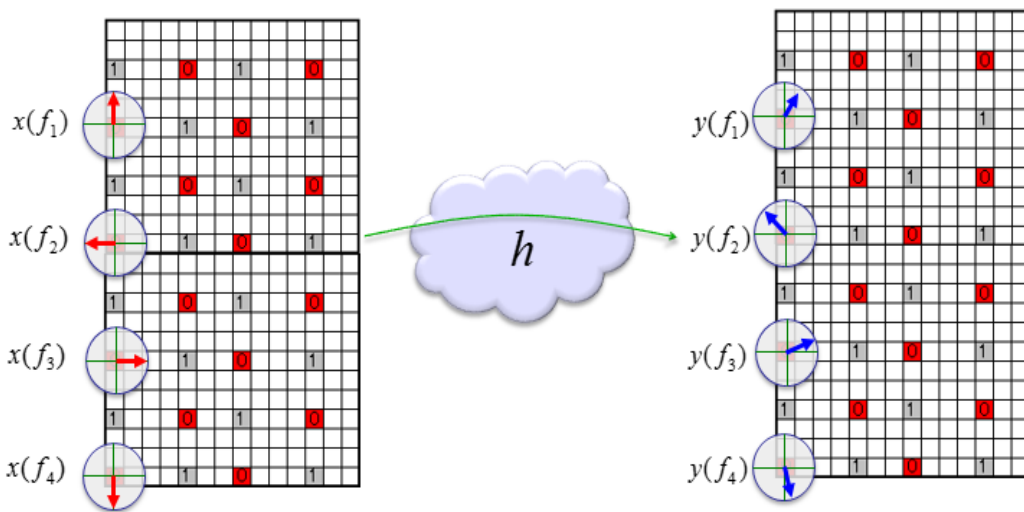


Figure 2.8: Channel estimation by comparing transmitted and received reference signal

Now let's think of the case of LTE SISO for the estimation of channel coefficient and noise. Reference signals are embedded onto only one antenna port. The vertical line in the downlink resource grid map in Figure 2.8 represents the frequency domain. The reference signals are indexed with $f_1, f_2, f_3 \dots f_n$. Each reference symbol can be a complex number (In-phase and Quadrature (I/Q) data) that can be plotted as shown in Figure 2.8 (red and blue arrows). Each complex number (Reference Symbol) on the left (transmission side) is modified (distorted) to each corresponding symbol on the right (received symbol). Channel Estimation is the process of finding a correlation between the array of complex numbers on the left and the array of complex numbers on the right. The detailed method of the channel estimation can vary depending on the implementation. Since our platform will firstly use SISO, system model for each transmitted reference signal and received reference signal can be represented as follows. $y()$ represents the array of received reference signal, $x()$ represents the array of the transmitted reference

signal, and $h()$ represents the array of channel coefficient. f_n is the indice.

$$y(f_n) = h(f_n) \cdot x(f_n) \quad (2.1)$$

We know what $x()$ is because it is given from the transmitted signals. And the $y()$ is also known, because it is measured/detected from the receiver. With this information, we can calculate the channel coefficient array as shown in equation 2.2, where H is a symbol, which represents Hermitian of a matrix.

$$h(f_n) = y(f_n) \cdot x^H(f_n) \quad (2.2)$$

Now we have all the channel coefficient for the location where reference signals are located. But we need channel coefficient at all the location including those points where there is no reference signal. It means that we need to figure out the channel coefficient for those location with no reference signal. The most common way to do this is to interpolate the measured coefficient array. Normally, we do a averaging of the channel coefficients first and then do interpolation over the averaged channel coefficient.

After getting the channel coefficients, we can estimate the noise properties. Theoretically, the noise can be calculated as below, where $\bar{h}()$ is the averaged channel estimate and $n()$ is the estimated noise.

$$n(f_n) = y(f_n) - \bar{h}(f_n) \cdot x(f_n) \quad (2.3)$$

However, what we need is the statistical properties of the noise, not the exact noise value. We can estimate the noise using only measured channel coefficient and averaged channel as shown below in equation 2.4. (Actually exact noise value does not have much meaning because the noise value keeps changing and it is of no use to use those specific noise values).

$$n(f_n) = h(f_n) - \bar{h}(f_n) \quad (2.4)$$

Through those methods above, we can preliminarily calculate the channel estimates and noise estimates. Correspondingly, the estimation of uplink channel coefficient utilizes the same method as above, but there are some differences, which will be introduced in the next subsection.

2.3.2 Uplink frame structure

While many of the requirements for the design of the LTE uplink physical layer and multiple-access scheme are similar to those of the downlink, the uplink also poses some unique characteristics. The multiple-access scheme selected for the LTE uplink so as to fulfill these principle characteristics is SC-FDMA, which achieves intra-cell orthogonality even in frequency-selective channels and avoids the high level of intra-cell interference. More details about SC-FDMA can be referred to the third part of the book [28]. As in the downlink, the LTE SC-FDMA uplink incorporates Reference Signal (RS) for data demodulation and channel sounding. In this subsection, the principles behind the RS are explained.

Firstly, the uplink frame structure looks as follows in Figure 2.9. In our case, we will use the FDD LTE. Figure 2.9 is an FDD frame structure that shows one frame in the time domain. It does not show any structure in the frequency domain. The time duration for one frame T_S is $10ms$. This means that we have 100 radio frames per second. The number of samples in one frame is 307200 samples. So the number of samples per second is $307200 \times 100 = 30.72M$. There are 10 subframes in one frame, and the number of slots in one subframe is 2. This means that we have 20 slots within one frame. Thus, the duration of one subframe T_s is $1ms$ and the duration of a slot T_{slot} is $0.5ms$.

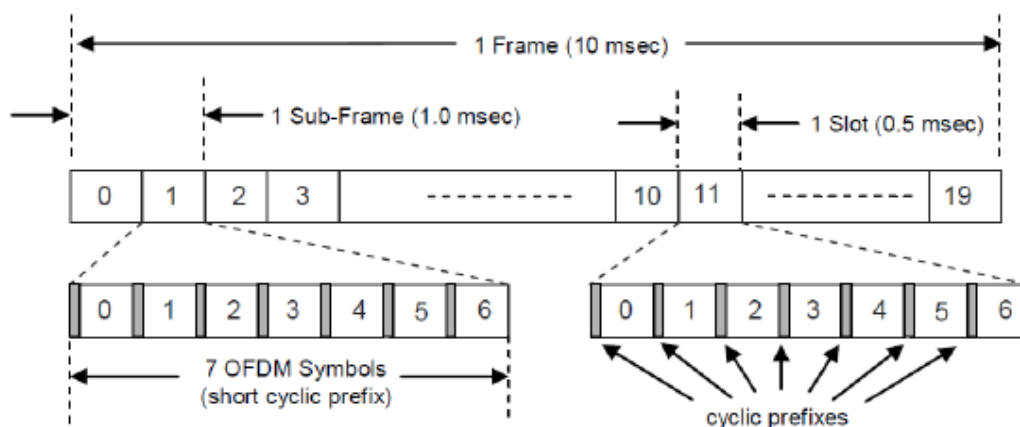


Figure 2.9: Uplink frame structure

We can see from Figure 2.9 that one slot is made up of 7 small blocks called a symbol. One symbol is a certain time span of signal that carries one spot in the I/Q constellation. And we see even smaller structures within a symbol. At the

beginning of a symbol, you see a very small span called 'Cyclic Prefix' and the remaining part is the real symbol data. There are two different type of Cyclic Prefix. One is 'normal Cyclic Prefix' and the other is 'Extended Cyclic Prefix' which is longer than the 'Normal Cyclic Prefix'. Since the length of one slot is fixed and cannot be changed, if we use 'Extended Cyclic Prefix', the number of symbols that can be accommodated within a slot should be decreased. So we can have only 6 symbols if we use 'Extended Cyclic Prefix'.

Following shows the overall subframe structure from "LTE Resource Grid" in Figure 2.10.

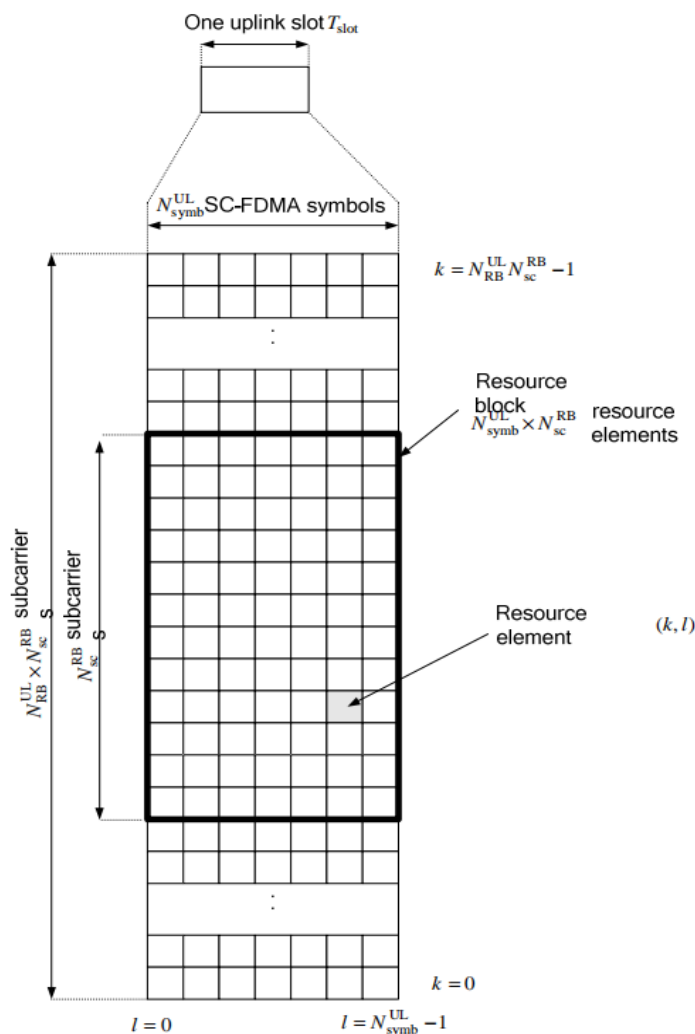


Figure 2.10: LTE uplink channel resource grid in a slot [3]

As shown in the Figure 2.10, if we expand a single slot in direction of vertical (frequency domain) and horizontal (time domain), we can get a source grid map, which consists of a lot of resource grids with the location index (k, l) , where k in the range of $0 \leq k \leq N_{RB}^{UL} N_{SC}^{RB} - 1$ is the index of RE in the frequency domain and l in the range of $0 \leq l \leq N_{symp}^{UL} - 1$ is the index of SC-FDMA symbols in the time domain. The N_{RB}^{UL} is the number of RB in the frequency domain, where $6 \leq N_{RB}^{UL} \leq 110$. The N_{SC}^{RB} is the number of subcarriers in a RB. And the N_{symp}^{UL} is the number of SC-FDMA symbols in an uplink slot in the time domain [3].

Now we will take a single subframe with a single RB as an example. As is shown in Figure 2.11, a few differences we would notice would be the location of each channel. Normally in the downlink case, a channel tends to lie across the whole bandwidth, but the channels in the uplink slot seem to be more localized. For example, PUCCH is located only at the lowest and highest end in the frequency domain and reference signals also localized in the time domain or both time domain and frequency domain.

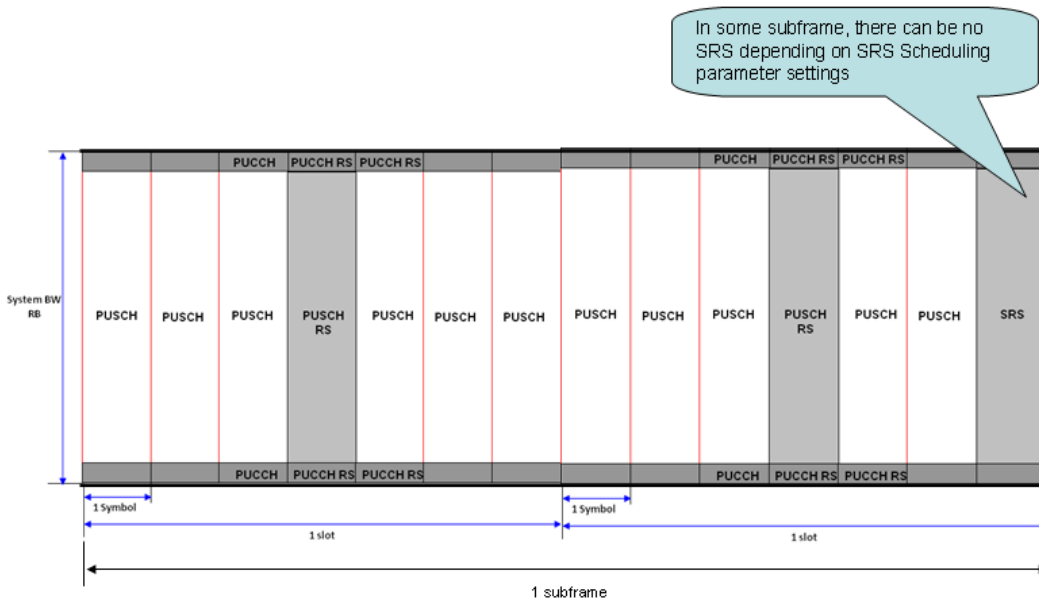


Figure 2.11: Reference signal distribution in a subframe

In this figure, we can see the RSs that are located in the region marked with grey. The roles of the uplink RSs include enabling channel estimation to aid coherent demodulation, channel quality estimation for uplink scheduling, power

control, timing estimation and direction-of-arrival estimation to support downlink beam-forming. Two types of RS are supported on the uplink:

- **DeModulation RS** (DRS), associated with transmissions of uplink data on the PUSCH and control signalling on the PUCCH. These RSs are primarily used for channel estimation for coherent demodulation.
- **Sounding RS** (SRS), not associated with uplink data and control transmissions, and primarily used for channel quality determination to enable frequency-selective scheduling on the uplink.

The uplink RSs are time-multiplexed with the data symbols. The DRSs of a given UE occupy the same bandwidth (i.e. the same RBs) as its PUSCH/PUCCH data transmission. Thus, the allocation of orthogonal (in frequency) sets of RBs to different UEs for data transmission automatically ensures that their DRSs are also orthogonal to each other. The uplink reference signals in LTE are mostly based on Zadoff–Chu (ZC) sequences [6, 30]. These sequences satisfy the desirable properties for RS, such as ideal cyclic autocorrelation, and optimal cross-correlation. The cross-correlation property results in the impact of an interfering signal being spread evenly in the time domain after time-domain correlation of the received signal with the desired sequence, this results in more reliable detection of the significant channel taps. By comparing the received reference signal and transmitted reference signal as in equation 2.2, we can get the estimated channel coefficients. However, for a channel estimation constant in every slot, that is, forcing the uplink channel demodulation to happen every 0.5 ms , the uplink scheduling needs to be configured.

2.3.3 Uplink Data Transmission Scheduling

There are a couple of Data Transmission Scheduling Schemes in LTE. The most simple in terms of algorithm would be the persistent scheduling. According to Figure 2.12, in this scheduling mode, the network sends 'Grant' in Downlink Control Indicator (DCI) Format 0 for every subframe. DCI carries the information about uplink resource allocation and descriptions about downlink data transmitted to the UE. The network sends the first data on downlink PDSCH and PDCCH which has DCI format 1 for Downlink Data Decoding and DCI format 0 for uplink Grant. If there is no downlink data to be transmitted, the network transmits only PDCCH with DCI format 0 without any PDSCH data. UE will decode the PCFICH to figure

Control Format Indicator (CFI) value, which tells how many OFDM symbols are used for carrying control channels at each subframe. In addition, UE will decode the PDCCH and gets the information on DCI format 1. Based on DCI format 1, UE can decode the downlink data. Then, UE sends Uplink Control Indicator (UCI) to the network, which is carried by PUCCH or PUSCH. The information carried out by UCI is mainly Scheduling Request (SR), Hybrid Automatic Repeat Request (HARQ) Acknowledgement (ACK)/Negative Acknowledgement (NAK) and CQI/PMI/Rank Indicator (RI). UE transmits a certain combination of this three information depending on the situation. Sometimes it carries only Scheduling Request, or Scheduling Request and HARQ ACK/NAK together, etc. There are two channels that can carry the UCI. When UE transmits the user data and it has to use PUSCH. In this case, PUCCH is not allowed to be transmitted, in this case, PUSCH carries UCI. When there is no user data to be transmitted, PUCCH is transmitted carrying UCI in it. Look back to the uplink schedule, If Grant is allowed by the UE, the UE then transmits the uplink data through PUSCH. The Network will decode the PUSCH data and send ACK/NAK back via PHICH.

In the case of SISO or MIMO communication, to transmit data always through the channel with the best quality, the uplink channels are estimated and ranked by the receiver eNB. The CSI in uplink plays an important role. It is a kind of indicator of how good or bad the channel is at a specific time. The CSI has three major components, respectively are CQI, PMI, RI. Not all of these indicators are measured for every CSI report. In many applications, including OAI, the uplink CQI is calculated from the Signal to Noise Ratio (SNR), which is the estimated value from channel estimation in section 2.3.1. In our case, we only have a single UE and single eNB, after the uplink channel is estimated, the CQI is the information that eNB sends to the UE. In an LTE cell, the uplink channels will be estimated and compared by the eNB through random access, and then the UEs will be informed with the CSI to transmit data in order. The final goal of designing the concept of CQI and implementing it in such a complicated way is to achieve the least amount of error and the best possible rate of throughput. Many factors are influencing the throughput and each of the factors would have some kind of correlations with other factors. In Lab test, it is relatively easy to figure out those correlations since we can control those factors as fitting the best for analysis, but in a live network, it is not always that easy to figure out those correlations because most of those factors change dynamically. The general rule of thumb for the correlation between CQI and throughput can be summarized as follows [13].

- High throughput does not necessarily mean high CQI. High throughput de-

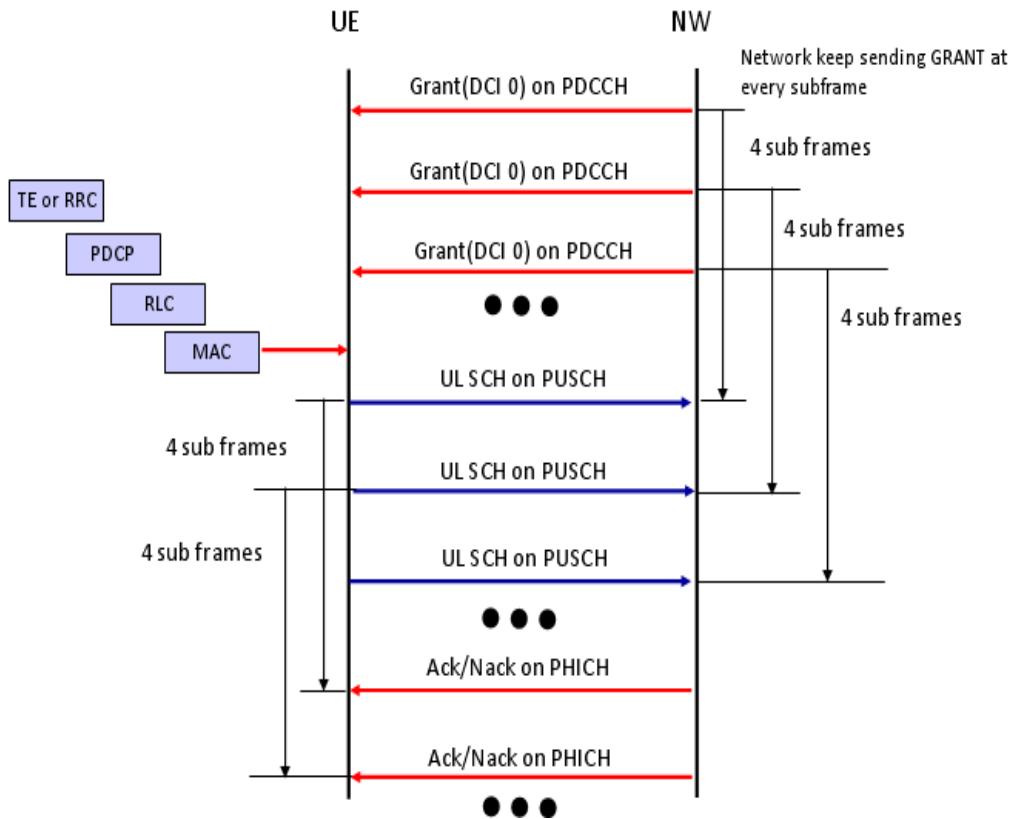


Figure 2.12: Uplink transmission under persistent scheduling

depends not only on CQI, but also on transport block size. For example, number of RB. Even when CQI is high, eNB may assign small resources due to various other factors.

- Low throughput does not necessarily mean low CQI.
- With low CQI, it is for sure that we cannot achieve the maximum throughput. So, it is very likely that we would see low CQI when we see throughput drop in live network test.

Our goal is to extract the uplink channel estimates with constant Bandwidth assigned from the eNB. Also, the uplink channel estimation must exist in every uplink subframe from a single eNB. So, more works need to be down according to the OAI system procedure, which will be introduced in the implementation later.

Implementation of LTE Access Platform

In this chapter, the process of building a mini LTE base station based on the USRP and OAI on two GPPs will be introduced. As is presented in chapter 2, we have an overview of an SDR system. In our case, we take the advantage of OAI, which is open source and easy configurable, to build an OAI LTE eNB connected with OAI EPC using S1 interface. Firstly, section 3.1 introduces the hardware setup and IP setup of the platform. In section 3.2, the detailed process of eNB installation will be presented. And then, the process of EPC installation will be explained in section 3.3. The configuration of SIMCard and the EPC database are depicted later in section 3.4. Finally, the instructions of running the system by using the eNB configuration file, the UE attach procedure and the system running status are shown in section 3.5

3.1 Software and IP Setup

When we get to know the OAI for the first time, it is difficult to know how to set it up with some concrete objects. For example, the number of GPP should be used, the connection between the GPPs and the connection between the GPP and USRP. After some research on the official OAI tutorial, we have found a suggested hardware setup. That is, there are two computers for running OAI eNB and OAI EPC respectively, the two computers are connected through an Ethernet cable and can ping to each other. The computer, which runs OAI EPC, should have another Ethernet port for connecting to the Ethernet switch or router. The computer, which runs OAI eNB, should have an USB 3.0 port for the communication with USRP. In addition, when there is a need for the computer with eNB installed to access the Internet, it should also have another Ethernet port available.

According to our working environment, we have simplified the suggested set up a little bit. In our case, we connect the computers to the switch, so that, the computer needs only one Ethernet port for each of them. One thing should be paid attention, that is we should not use the Dynamic Host Configuration Protocol (DHCP) from the switch to allocate IP addresses, we assigned static IP for the computers instead. The hardware set up can be seen in Figure 3.1. The computers with static IP addresses can ping to each other through the switch, at the same time have access to the Internet through the switch, which connects to a router. The USRP receives the signal from the eNB through a USB 3.0 interface and broadcast the signal to the air. The configured smartphone can then access the signal and connect to this network.

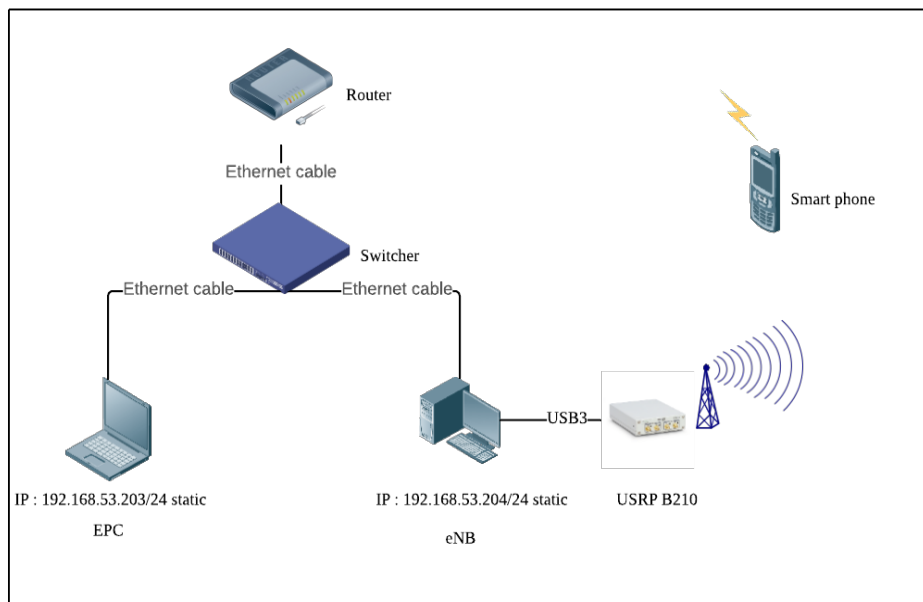


Figure 3.1: LTE platform hardware set up

Referred to the OAI architecture, which is introduced in the last chapter, we will combine the hardware setup and the OAI interior architecture and interfaces to assign the IP address to each interface, which will then be used in the configuration file from installation. In Figure 3.2, the IP assignments for the interfaces are depicted. There are two planes of the S1 interface between the OAI eNB and OAI EPC, respectively are S1-U and S1-C. The S1 interface in LTE is used between eNBs and the EPC: specifically, the MME and SGW. In the user plane

(S1-U), this interface will be based on GPRS Tunneling Protocol (GTP) User Data Tunneling, where GPRS is the General Packet Radio Service, for the transfer of user data. In the control plane (S1-C), the interface is more similar to Radio Access Network Application Part, with some simplifications and changes due to the different functional split and mobility within Evolved Packet System (EPS). So the MME and SGW will share the IP address of EPC to connect to the eNB and then connect with each other through the S11 interface, which is based on GTP-control with some additional functions for paging coordinating. This interface is designed from the OAI internal default. We can also find an interface called S6a between the MME and HSS. The S6a used for the transfer of subscription, authentication, and authorization of users, so that the UE can legally connect to the EPC after a successful authentication by the HSS. There is an interface called SGi, which is used between PGW and the internet (also external PDN). This interface is based on the IP packet, also enabling the exchange of signaling. Both S6a and SGi interfaces are also designed in the OAI EPC as default.

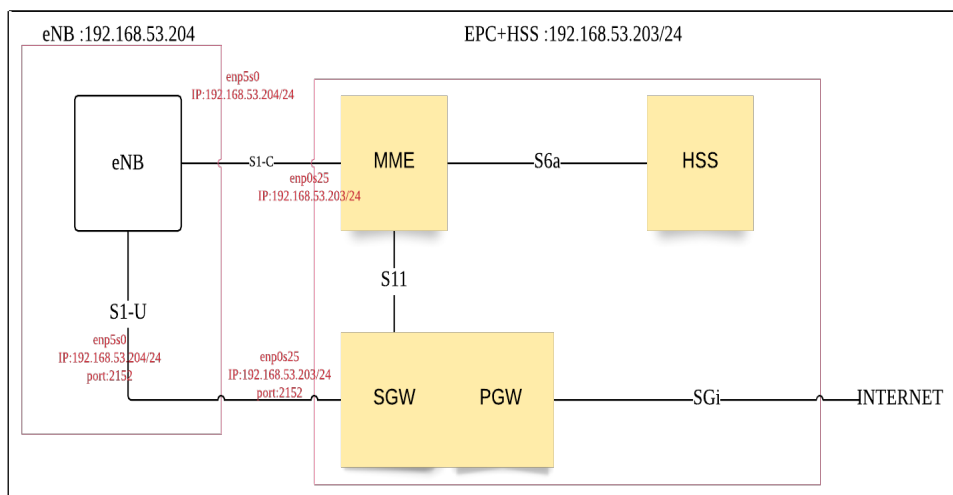


Figure 3.2: LTE platform IP set up

3.2 Installation of eNB

Current OAI software requires Intel architecture based PCs for the eNB or UE targets. This requirement is due to optimized DSP functions which make heavy use

of integer Single Instruction Multiple Data (SIMD) instructions (SSE, SSE2, SSE3, SSE4, and Advanced Vector Extensions (AVX)2). The software is suggested to be tested on the following processor families:

- Generation 3/4/5/6 Intel Core i5, i7
- Generation 2/3/4 Intel Xeon
- Intel Atom Rangeley, E38xx, x5-z8300

Based on the hardware requirements, we use the PC equipped with Intel(R) Xeon(R) CPU E3-1245 v6 @ 3.70GHz. The operating system is also suggested Ubuntu 14.04 LTS or higher (64-bit). In our case, we use the Ubuntu 16.04 LTS, which is the most widely used by the researcher and developer. Before installing the eNB, it is also necessary to care about the Kernel Setup. Firstly, we should disable the C-states from the BIOS. Secondly, make sure that the CPU frequency scaling is deactivated. Then, we can restart the operating system and install the low-latency kernel. Some Ubuntu Terminal commands are listed below.

```
1 ~$ sudo apt-get update
2 ~$ sudo apt-get install linux-image-4.4.0-166-lowlatency
3 ~$ sudo apt-get install linux-headers-4.4.0-166-lowlatency
4 ~$ shutdown -r now
```

After the system restart, check the kernel version by typing **uname -a** in the terminal. We can also use the Grub-Customizer to set the low-latency kernel as default. To ensure that the CPU runs with the highest performance, we also need to change the power management settings.

```
1 # open the grub configuration file by vim
2 ~$ sudo vim /etc/default/grub
3 # Add two lines in this file, then save and exit
4 GRUB_CMDLINE_LINUX_DEFAULT="quiet intel_pstate=disable";
5 GRUB_CMDLINE_LINUX_DEFAULT="processor.max_cstate=1 intel_idle.
   ↳max_cstate=0 idle=poll";
6
7 # Open the blacklist configuration file by vim
8 ~$ sudo vim /etc/modprobe.d/blacklist.conf
9 # Add a line at the end of this file, then save and exit
10 blacklist intel_powerclamp
11 # Restart
12 ~$ shutdown -r now
```


After finishing all these settings, we can type **sudo i7z** to check the C0, which has 100% occupancy on all the 4 cores. We should also check, that Hyper-Threading is OFF. A correct setup is shown by i7z can be referred to Figure 3.3. After we have

```

Cpu speed from cpufreq 3695.00MHz
cpufreq might be wrong if cpufreq is enabled. To guess correctly try estimating via tsc
Linux's inbuilt cpu_khz code emulated now
True Frequency (without accounting Turbo) 3695 MHz
CPU Multiplier 37x || Bus clock frequency (BCLK) 99.86 MHz

Socket [0] - [physical cores=4, logical cores=4, max online cores ever=4]
TURBO ENABLED on 4 Cores, Hyper Threading OFF
Max Frequency without considering Turbo 3794.86 MHz (99.86 x [38])
Max TURBO Multiplier (if Enabled) with 1/2/3/4 Cores is 41x/40x/40x/39x
Real Current Frequency 3891.14 MHz [99.86 x 38.96] (Max of below)

```

Core [core-id]	Actual Freq (Mult.)	C0%	Halt(C1)%	C3 %	C6 %	Temp	VCore
Core 1 [0]:	3891.13 (38.96x)	100	0	0	0	43	1.1702
Core 2 [1]:	3891.13 (38.96x)	100	0	0	0	41	1.1702
Core 3 [2]:	3891.13 (38.96x)	100	0	0	0	41	1.1702
Core 4 [3]:	3891.14 (38.96x)	100	0	0	0	40	1.1702

```

C0 = Processor running without halting
C1 = Processor running with halts (States >C0 are power saver modes with cores idling)
C3 = Cores running with PLL turned off and core cache turned off
C6, C7 = Everything in C3 + core state saved to last level cache, C7 is deeper than C6
Above values in table are in percentage over the last 1 sec
[core-id] refers to core-id number in /proc/cpuinfo
'Garbage Values' message printed when garbage values are read
Ctrl+C to exit

```

Figure 3.3: CPU state setup result in i7z

correctly set the operating system and CPU, we have then downloaded the source code of OAI eNB part by git. Before we compile the eNB, the drive and interface of USRP should be done. In our case, we use the USRP B210, which is powered by a USB 3.0 cable. The USB 3.0 cable is also responsible for the baseband signal transmission. On Linux, **udev** handles USB plug and unplug events. The following commands install a **udev** rule so that non-root users may access the device. This step is only necessary for devices that use USB to connect to the host computer, such as the B200, B210, and B200mini. This setting should take effect immediately and does not require a reboot or logout/login.

```

1 ~$ cd $HOME/workarea/uhd/host/utils
2 ~$ sudo cp uhd-usrp.rules /etc/udev/rules.d/
3 ~$ sudo udevadm control --reload-rules
4 ~$ sudo udevadm trigger

```

Be sure that no USRP device is connected via USB when running these commands. Sometimes, if we only use the UHD and the USRP for doing development and prototyping, it is always suggested to build the UHD from source code. However, the OAI is developed and released only based on some specific version of the UHD. There is already some installation API, which installs the correspond UHD version. Then we can compile the eNB application using CMAKE. Some relevant commands are listed below.

```
1 # Download/clone source file
2 ~$ sudo apt-get install git
3 ~$ git clone https://gitlab.eurecom.fr/oai/openairinterface5g.
   ↳git
4
5 # Compile eNB
6 ~$ cd openairinterface5g
7 ~$ source oaienv
8 ~$ cd cmake_targets
9 ~$ sudo ./build_oai -I -w USRP
10 ~$ ./build_oai --eNB -c -w USRP
```

The **-I** option will automatically download and install all the relevant or essential software packages to build the UHD for USRP. There are also a lot of other options for compiling the eNB project. For example, when we have other SDR frontend hardware, we can give the corresponding hardware after the **-w** option. Now the installation of UHD and OAI eNB. At this point, connect the USRP to the host computer. Because the interface is USB. We can open a terminal window and run **lsusb**. We would see the USRP listed on the USB bus with the VID of 2500 and PID of 0021 for B210. For testing the UHD drive, we can run **uhd_find_devices** and **uhd_usrp_probe**. If there aren't some warnings that occur, we have a correct setup.

3.3 Installation of EPC

In this section, we will introduce the process of installing OAI EPC. We have built the OAI EPC and OAI eNB on different hosts. The OAI EPC doesn't need a powerful processor. So we have chosen a mini PC equipped with Intel(R) i5-5300U 2.30GHz. In the same way, we have installed the OAI eNB on the Ubuntu 16.04 LTS. Because of a real-time requirement, we also need to change the kernel to be

a low-latency kernel. Besides, the configuration of CPU state, power management, and hyper-threading are not necessary.

First, we have downloaded the source code using git. The OAI EPC has an independent repository as the eNB. In the terminal of ubuntu, we typed:

```
1 # Install git
2 ~$ sudo apt-get update
3 ~$ sudo apt-get install subversion git
4
5 # Download source code of EPC
6 ~$ git clone https://gitlab.eurecom.fr/oai/openair-cn.git
```

Before we compile the OAI EPC, we should do some configurations for server and account management. We configured first the Fully Qualified Domain Name (FQDN), which is a domain name that specifies its exact location in the tree hierarchy of the Domain Name System (DNS) [25]. It specifies all domain levels, including the top-level domain and the root zone. A fully qualified domain name is distinguished by its lack of ambiguity: it can be interpreted only in one way. It usually consists of a hostname and at least one higher-level domain (label) separated by the symbol "." and always ends in the top-level domain. The correct way to change the hostname is that, we edit the host file with the defined hostname in the root environment. Commands are listed below.

```
1 # change the host name in the root environment
2 ~$ su root
3 hostname
4 vim /etc/hosts
5 # Type information like below:
6 127.0.0.1    localhost
7 127.0.1.1    edge.openair4G.eur  edge  #edge is your defined
8             host name
9 127.0.1.1    hss.openair4G.eur    hss
10 # After editing, save and exit
```

We ran **hostname -f** to check whether we have changed the hostname. In our case, the terminal showed us **edge.openair4G.eur**.

The next step is to install the mysql-server, which is responsible for starting and closing the mysql service. Correspondingly, we also need to install the mysql-client.

The mysql server package will install the mysql database server which we can interact with using a mysql client. We can use the mysql client to send commands to any mysql server on a remote computer or our own. The mysql server is used to persist the data and provide a query interface for it (SQL). The mysql-client's purpose is to allow us to use that query interface. The client package also comes with utilities that allow us to easily backup/restore data and administer the server.

```
1 ~$ sudo apt-get install mysql-server
2 # By installing, remember to set the user name and password, and
   ↳keep the password in mind. Here we set the 'root' as user
   ↳name and set the password the same as the ubuntu system
   ↳password.
3 ~$ sudo apt-get install mysql-client
```

Then, we install the apache2, which is a web server with two main functionalities. The first function is parsing the web language like HTML, PHP, JSP, etc. The second is receiving the request from web clients and give some response. The command is given below. After the installation, for testing the apache2, we can type **localhost** or **127.0.0.1** into the address bar of a web explorer. The explorer will show us the main page of the apache2.

```
1 ~$ sudo apt-get install apache2
```

Next, we need to install the PHP. The PHP is a general-purpose programming language originally designed for web development. Here, the PHP will be called by the Apache2 to parse the received PHP program and return a string to the client. The commands are listed below.

```
1 ~$ sudo apt-get install php7.0
2 ~$ sudo apt-get install libapache2-mod-php7.0
3 ~$ sudo service apache2 restart
4 # We can test the php done as below
5 # We can create a new file
6 ~$ sudo vim /var/www/html/info.php
7 # In this file we can type:
8 <?php
9 echo "<P>Hello World!</P> "?>"
10 # Save and exit.
11 # Then in the web explorer visit localhost/info.php
12 # We can see a web page with a sentence "Hello World!"
```

The last essential package is the phpmyadmin, which is an administration tool for mysql's web hosting service. Some commands are listed below. After the installation, if we visit the **localhost/phpmyadmin** or **127.0.0.1/phpmyadmin** in a web browser, we can see a login page of the database.

```
1 ~$ sudo apt-get install phpmyadmin
2 ~$ sudo ln -s /usr/share/phpmyadmin /var/www/html
```

All those works above are done for serving the OAI EPC user database management, which stores the SIM card information and operator keys for authentication. Then, we started to build the OAI EPC.

```
1 # firstly, We open the EPC source file path
2 ~$ cd openair-cn
3 # Secondly, we switch the EPC release version to 'develop'
4 ~$ git checkout develop
5 ~$ git pull
6 # Thirdly, we open the folder called scripts
7 ~$ cd scripts
8 # By running these three commands with option -i below, the
   ↳system will automatically download the essential packages
   ↳from building the three part of EPC
9 ~$ ./build_mme -i
10 ~$ ./build_hss -i
11 ~$ ./build_spgw -i
12 # When no error occurs, means everything ok.
```

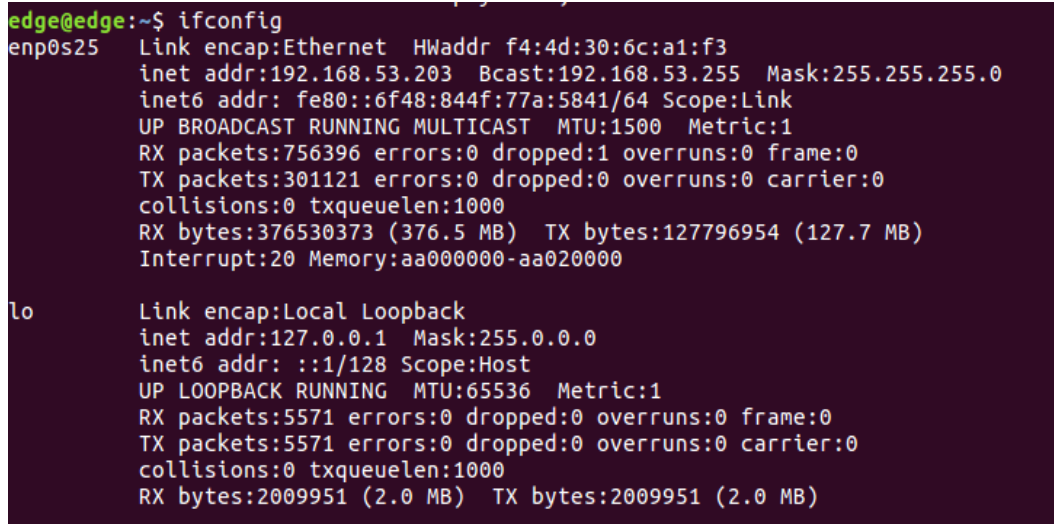
Then, like the process of building the OAI eNB, we also need to configure the system files before compiling. First of all, we need to create a new folder called freeDiameter. After that, we need to copy all the configuration files of MME, HSS, SGW and PGW to the new created local user path for OAI. Here the SGW and PGW are combined together into SPGW. Moreover, the EPC system free diameter configuration files of MME, HSS, SGW and PGW will also be copied to the new created local user path for freeDiameter. The detailed commands for the copy of the files are listed below.

```

1  ~$ sudo mkdir -p /usr/local/etc/oai/freeDiameter
2  ~$ sudo cp ~/openair-cn/etc/mme.conf /usr/local/etc/oai
3  ~$ sudo cp ~/openair-cn/etc/hss.conf /usr/local/etc/oai
4  ~$ sudo cp ~/openair-cn/etc/spgw.conf /usr/local/etc/oai
5  ~$ sudo cp ~/openair-cn/etc/acl.conf /usr/local/etc/oai/
   ↳freeDiameter
6  ~$ sudo cp ~/openair-cn/etc/mme_fd.conf /usr/local/etc/oai/
   ↳freeDiameter
7  ~$ sudo cp ~/openair-cn/etc/hss_fd.conf /usr/local/etc/oai/
   ↳freeDiameter

```

Before we altering the parameter in the configuration file, we have typed the command **ifconfig** in the terminal to check the IP configurations, which is already done and introduced in section 3.1. As is shown in Figure 3.4, we mentioned the Ethernet card name, IP address, mask and DNS, which will be used later in the configuration file.



```

edge@edge:~$ ifconfig
enp0s25  Link encap:Ethernet  HWaddr f4:4d:30:6c:a1:f3
         inet addr:192.168.53.203  Bcast:192.168.53.255  Mask:255.255.255.0
         inet6 addr: fe80::6f48:844f:77a:5841/64  Scope:Link
         UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
         RX packets:756396  errors:0  dropped:1  overruns:0  frame:0
         TX packets:301121  errors:0  dropped:0  overruns:0  carrier:0
         collisions:0  txqueuelen:1000
         RX bytes:376530373 (376.5 MB)  TX bytes:127796954 (127.7 MB)
         Interrupt:20  Memory:aa000000-aa020000

lo       Link encap:Local Loopback
         inet addr:127.0.0.1  Mask:255.0.0.0
         inet6 addr: ::1/128  Scope:Host
         UP LOOPBACK RUNNING  MTU:65536  Metric:1
         RX packets:5571  errors:0  dropped:0  overruns:0  frame:0
         TX packets:5571  errors:0  dropped:0  overruns:0  carrier:0
         collisions:0  txqueuelen:1000
         RX bytes:2009951 (2.0 MB)  TX bytes:2009951 (2.0 MB)

```

Figure 3.4: IP configuration status of computer runs EPC

Now let's configure MME, SPGW and HSS one by one, which is listed through the terminal operation commands below in detail.

```

1  # Configure MME
2  ~$ sudo vim /usr/local/etc/oai/mme.conf
3  # Check and change following information using our own IP
   ↳configuration information in this file
4  MME_INTERFACE_NAME_FOR_S1_MME = "enp0s25";
5  MME_IPV4_ADDRESS_FOR_S1_MME = "192.168.53.203/24";

```

```

6
7 # Configure SPGW
8 ~$ sudo vim /usr/local/etc/oai/spgw.conf
9 # Pay attention to these lines in this file
10 SGW_INTERFACE_NAME_FOR_S1U_S12_S4_UP      = "enp0s25";
11 SGW_IPV4_ADDRESS_FOR_S1U_S12_S4_UP       = "192.168.53.203/24";
12
13 PGW_INTERFACE_NAME_FOR_SGI                  = "enp0s25";
14 PGW_MASQUERADE_SGI = "yes"; #add this line after the line
    ↳above
15
16 # Configure HSS freeDiameter
17 ~$ sudo vim /usr/local/etc/oai/freeDiameter/hss_fd.conf
18 # Check these information:
19 Identity = "hss.openair4G.eur";
20 Realm = "openair4G.eur";
21
22 # Configure MME freeDiameter
23 ~$ sudo vim /usr/local/etc/oai/freeDiameter/mme_fd.conf
24 # Check the following information:
25 Identity = "edge.openair4G.eur"; #same as our hostname
26 Realm = "openair4G.eur";
27 ConnectPeer= "hss.openair4G.eur" { ConnectTo = "127.0.0.1";
    ↳No_SCTP ; No_IPv6; Prefer_TCP; No_TLS; port = 3868; realm
    ↳= "openair4G.eur";};
28
29 # Configure HSS
30 ~$ sudo vim /usr/local/etc/oai/hss.conf
31 # Check the informations below:
32 mysql_user = "root"; # Set by the installation of mysql
33 mysql_pass = "*****"; # Put here the root password of mysql
    ↳ database that was provided during installation
34 OPERATOR_key = "11111111111111111111111111111111"; # OP key
    ↳for oai_db.sql, Must match to that of UE Sim card, OP_Key.
35
36 # Register certification
37 ~$ cd ~/openair-cn/scripts
38 ~$ ./check_hss_s6a_certificate /usr/local/etc/oai/freeDiameter/
    ↳ hss.openair4G.eur
39 ~$ ./check_mme_s6a_certificate /usr/local/etc/oai/freeDiameter/
    ↳ edge.openair4G.eur #our hostname setup

```

Great attention should be paid to that information, even a small detail that we didn't notice will lead to a failure by compiling. Then, we continued to build the HSS, MME and SPGW one by one separately.

```

1 # Build HSS
2 ~$ cd ~/openair-cn/scripts
3 ~$ sudo ./build_hss -c # here the -c delete the previous
    ↳generated files from compiling
4 # run the hss for the first time and load the oai_db.sql to the
    ↳database server

```

```
5  ~$ ./run_hss -i ~/openair-cn/src/oai_hss/db/oai_db.sql #only
    ↳at the first time, otherwise just run ./run_hss -i
6
7  # Build MME, in a new terminal
8  ~$ cd ~/openair-cn/scripts
9  ~$ sudo ./build_mme -c
10
11 # Build SPGW
12 ~$ cd ~/openair-cn/scripts
13 ~$ sudo ./build_spgw -c
```

Here we didn't have any problem by compiling, then we continued to configure the EPC database with the programmed SIM card data, which is introduced in the next section.

3.4 User database configuration

Now, the eNB, EPC and HSS can work properly. We may move to the next stage. The SIM card that we used is a programmed SIM card with the original data in Table 3.1, which is programmed with the OPERATOR KEY from manufacturer Sysmocom (<http://shop.sysmocom.de/products/sysmousim-sjs1>). So, we can not use all the default information of the SIM card directly. Since we don't know the OPERATOR KEY from the manufacturer, we should use the OPERATOR KEY in the configuration file of HSS in the last section to re-encrypt the SIM card with the default Ki value of the SIM card. We have programmed the SIM card with Mileage support using open-cells.com reader/writer. We have first compiled the Open Cells program, which can be downloaded from <https://open-cells.com/d5138782a8739209ec5760865b1e53b0/uicc-v1.6.tgz>. Then, we have inserted the SIM card into the reader/writer and used the Open Cells program to encrypt the SIM card. Here is the information we needed to program the SIM card for OAI.

- Algorithm: Milenage
- Ki: 1854073054080694900C418F06A169DA (from Table 3.1 at the end of this chapter)
- OP: 11111111111111111111111111111111 (from hss.conf)

- C1:00, C2: 01, C3: 02, C4: 04, C5: 08, R1: 40, R2: 00, R3: 20, R4: 40, R5: 60 all in hex.
- SPN (service provider Name): OpenAirInterface
- Mobile Country Code (MCC): 262 (DE)
- Mobile Network Code (MNC): 70 (research)
- IMSI: 262700000008960 (MCC|MNC|id), from Table 3.1, where id is incremented. Note that here the PLMN (26270) is 5 digit, which is why the id 10 digits. Otherwise, for 6 digit PLMN, you only have 9 digit for id.

After finishing the SIM card programming, we turned to the OAI EPC host computer. We visited the **localhost/phpadmin** in a web browser, sign in with the username "root" and password. So, we accessed the HSS database. We can check out the existing users in the database via phpmyadmin. However, we are not able to insert a user record on phpmyadmin, because the Key (as well as OPc) is stored as binary in the database. We used the following commands as a guide. These commands are executed in the root environment.

```

1  shell > mysql -u root -p
2  # The password is the phpmyadmin login password
3  mysql > use oai_db;
4  # show all tables
5  mysql > show tables;
6  # show all entries in mmeidentity
7  mysql > select * from mmeidentity;
8  mysql > INSERT INTO users ('imsi', 'msisdn', 'imei', 'imei_sv',
    ↳ 'ms_ps_status', 'rau_tau_timer', 'ue_ambr_ul', 'ue_ambr_dl
    ↳ ', 'access_restriction', 'mme_cap', '
    ↳ mmeidentity_idmmeidentity', 'key', 'RFSP-Index', 'urrrp\_mme
    ↳ ', 'sqn', 'rand', 'OPc') VALUES ('262700000008960',
    ↳ '88211008960', '354198065175626', '02', 'PURGED', '120',
    ↳ '50000000', '10000000', '47', '000000000', '1',
    ↳ 1854073054080694900C418F06A169DA, '1', '0', '', 0
    ↳ x00000000000000000000000000000000, '');

```

There are a couple of things to note:

- msisdn: even though it is optional, we have not leaved it blank since OAI will check its presence.
- OPc: OPc is computed from OP_KEY and Ki. We have leaved it NULL, since HSS will compute it for us.

We can check the result on the phpmyadmin. Select the path **oai_db->users**, we can find and open the correspond imsi in the user list. See Figure 3.5. The

Column	Type	Function	Null	Value
imsi	varchar(15)		<input type="checkbox"/>	262700000008960
msisdn	varchar(46)		<input type="checkbox"/>	88211008960
imei	varchar(15)		<input type="checkbox"/>	354198065175626
imei_sv	varchar(2)		<input type="checkbox"/>	02
ms_ps_status	enum	--	<input type="checkbox"/>	NOT_PURGED
rau_tau_timer	int(10) unsigned		<input type="checkbox"/>	120
ue_ambr_ul	bigint(20) unsigned		<input type="checkbox"/>	50000000
ue_ambr_dl	bigint(20) unsigned		<input type="checkbox"/>	100000000
access_restriction	int(10) unsigned		<input type="checkbox"/>	47
mme_cap	int(10) unsigned zerofill		<input type="checkbox"/>	0000000000
mmeidentity_idmmeidentity	int(11)		<input type="checkbox"/>	1
key	varbinary(16)		<input type="checkbox"/>	1854073054080694900c418fc
RFSP-Index	smallint(5) unsigned		<input type="checkbox"/>	0
urrrp_mme	tinyint(1)		<input type="checkbox"/>	0
sqn	bigint(20) unsigned zerofill		<input type="checkbox"/>	00000000000000016846
rand	varbinary(16)		<input type="checkbox"/>	e3059d69b2d00f1869e08501
OPc	varbinary(16)		<input type="checkbox"/>	49689cc702d56b3065dcb05c

Figure 3.5: SIM card Info in database

EPC hostname is edge.openair4G.eur, so we have changed the mmehost in the mmeidentity to the hostname of EPC, and remember the idmmeidentity. See in Figure 3.6 , here the idmmeidentity is 1.

In addition, we have added a new APN under the oai_db option list, which will be later used in the smartphone APN setting. See in Figure 3.7, where the id is also set to 1 with the APN name oai.ipv4.

The next step is to set the pgw in the oai_db option list. Here we set the ipv4 value to be the IP of EPC for id=3. The ipv6 is not important. As is shown in Figure 3.8. It is to be noticed, that this id is the id of pgw, not the id of mmeidentity.

The last one is the pdn in the oai_db option list. We located the id=1 of mmeidentity and pgw_id=3, and then changed the users_imsi with the imsi of the programmed SIM card. It can be seen in figure 3.9. After finishing the database settings, we can

`SELECT * FROM `mmeidentity``

Show all | Number of rows: 25 | Filter rows: Search this table

Sort by key: None

+ Options

	id	mmehost	mmerealm	UE-Reachability
<input type="checkbox"/>	2	mme2.openair4G.eur	openair4G.eur	0
<input type="checkbox"/>	1	edge.openair4G.eur	openair4G.eur	0
<input type="checkbox"/>	5	abeille.openair4G.eur	openair4G.eur	0
<input type="checkbox"/>	4	yang.openair4G.eur	openair4G.eur	0
<input type="checkbox"/>	3	mme3.openair4G.eur	openair4G.eur	0
<input type="checkbox"/>	6	calisson.openair4G.eur	openair4G.eur	0

Check all | With selected: Edit Copy Delete Export

Figure 3.6: Idmmeidentity of hostname

`SELECT * FROM `apn``

Show all | Number of rows: 25 | Filter rows: Search this table

+ Options

	id	apn-name	pdn-type
<input type="checkbox"/>	1	oai.ipv4	IPv4

Check all | With selected: Edit Copy Delete Export

Show all | Number of rows: 25 | Filter rows: Search this table

Figure 3.7: APN setting

start the OAI EPC and OAI eNB separately and attach the smartphone to the OAI LTE network.

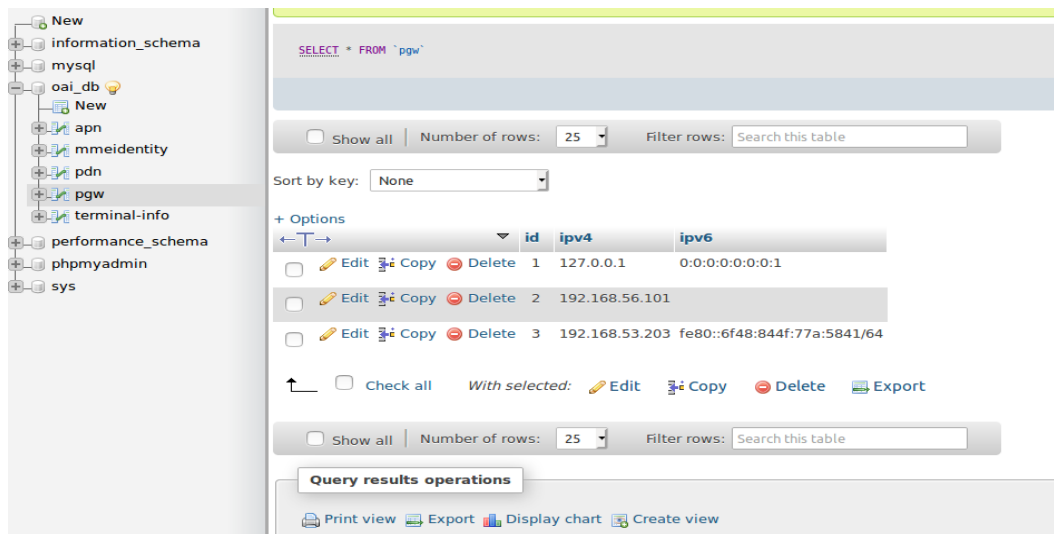


Figure 3.8: pgw setting

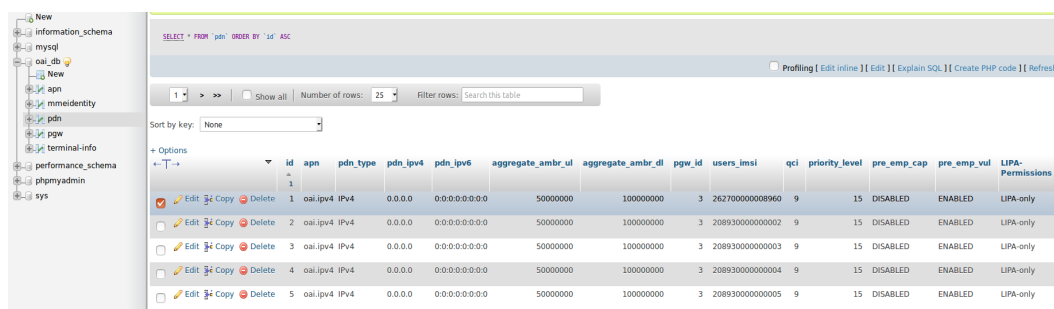


Figure 3.9: pdn setting

3.5 Process OAI system startup and UE attachment

In this section, the startup of the OAI EPC and OAI eNB are first introduced. After the OAI LTE successfully works, we will configure the smartphone and attach to the OAI LTE network. Although we have built the MME in section 3.4, before we run the MME in EPC, we have to update some information in the mme.conf like below:

```

1 # We should set the MCC, MNC and TAC the same as the programmed
  ↳SIM card
2 # Check these two lines below
3 GUMMEI_LIST = ({MCC="262" ; MNC="70" ; MME_GID="4" ; MME_CODE
  ↳="1" ; });
4 TAI_LIST = ({MCC="262" ; MNC="70" ; TAC = "1" ; });

```

Then, we build the MME again. We should start the OAI EPC in this order, that we first run the HSS in the first Ubuntu Terminal window, we can see the output of the HSS running status in Figure 3.10, where we can see that the S6A interface and the oai_db are successfully initialized.

```

Initializing s6a layer
02/10/20,10:17:19.559520 NOTI libfdproto '1.2.0' initialized.
02/10/20,10:17:19.559609 NOTI libgnutls '3.4.10' initialized.
02/10/20,10:17:19.560241 DBG Core state: 0 -> 1
02/10/20,10:17:19.560257 NOTI libfdcore '1.2.0' initialized.
02/10/20,10:17:19.563053 DBG Generating fresh Diffie-Hellman parameters of size 1024 (this takes some time)...
02/10/20,10:17:19.621491 DBG Loading : /usr/local/lib/freeDiameter/acl_wl.fdx
02/10/20,10:17:19.621683 NOTI Extension ACL_wl initialized with configuration: '/usr/local/etc/oai/freeDiameter/acl.conf'
02/10/20,10:17:19.621697 DBG Loading : /usr/local/lib/freeDiameter/dict_nas_nipdv6.fdx
02/10/20,10:17:19.621797 DBG Dictionary Extension 'MIPv6 NAS-to-HAAA Interaction' initialized
02/10/20,10:17:19.621805 DBG Loading : /usr/local/lib/freeDiameter/dict_s6a.fdx
02/10/20,10:17:19.622896 NOTI Dictionary Extension 'S6A from 3GPP standard v.10.5' initialized
02/10/20,10:17:19.622906 NOTI All extensions loaded.
02/10/20,10:17:19.622931 NOTI freeDiameter configuration:
02/10/20,10:17:19.622937 NOTI Default trace level .... : +1
02/10/20,10:17:19.622943 NOTI Configuration file .... : /usr/local/etc/oai/freeDiameter/hss_fd.conf
02/10/20,10:17:19.622948 NOTI Diameter Identity ..... : hss.openair4G.eur (1:17)
02/10/20,10:17:19.622953 NOTI Diameter Realm ..... : openair4G.eur (1:13)
02/10/20,10:17:19.622958 NOTI Tc Timer ..... : 30
02/10/20,10:17:19.622962 NOTI Tw Timer ..... : 30
02/10/20,10:17:19.622967 NOTI Local port ..... : 3868
02/10/20,10:17:19.622971 NOTI Local secure port ..... : 5868
02/10/20,10:17:19.622975 NOTI Number of SCTP streams .. : 3
02/10/20,10:17:19.622980 NOTI Number of clients thr .. : 5
02/10/20,10:17:19.622984 NOTI Number of app threads .. : 4
02/10/20,10:17:19.622988 NOTI Local endpoints ..... : Default (use all available)
02/10/20,10:17:19.622993 NOTI Local applications ..... : (none)
02/10/20,10:17:19.622997 NOTI Flags : - IP ..... : Enabled
02/10/20,10:17:19.623002 NOTI - IPv6 ..... : DISABLED
02/10/20,10:17:19.623006 NOTI - Relay app .... : DISABLED
02/10/20,10:17:19.623010 NOTI - TCP ..... : Enabled
02/10/20,10:17:19.623015 NOTI - SCTP ..... : DISABLED
02/10/20,10:17:19.623019 NOTI - Pref. proto .. : TCP
02/10/20,10:17:19.623023 NOTI - TLS method .. : Separate port
02/10/20,10:17:19.623028 NOTI TLS : - Certificate .. : /usr/local/etc/oai/freeDiameter/hss.cert.pem
02/10/20,10:17:19.623032 NOTI - Private key .. : /usr/local/etc/oai/freeDiameter/hss.key.pem
02/10/20,10:17:19.623037 NOTI - CA (trust) .. : /usr/local/etc/oai/freeDiameter/hss.cacert.pem (1 certs)
02/10/20,10:17:19.623041 NOTI - CRL ..... : (none)
02/10/20,10:17:19.623045 NOTI - Priority ..... : (default: 'NORMAL')
02/10/20,10:17:19.623050 NOTI - DH bits ..... : 1024
02/10/20,10:17:19.623054 NOTI Origin-State-Id ..... : 1581326239
02/10/20,10:17:19.623062 NOTI Loaded extensions: '/usr/local/lib/freeDiameter/acl_wl.fdx' [/usr/local/etc/oai/freeDiameter/acl.conf], loaded
02/10/20,10:17:19.623067 NOTI Loaded extensions: '/usr/local/lib/freeDiameter/dict_nas_nipdv6.fdx' [(no config file)], loaded
02/10/20,10:17:19.623071 NOTI Loaded extensions: '/usr/local/lib/freeDiameter/dict_s6a.fdx' [(no config file)], loaded
02/10/20,10:17:19.623086 DBG Core state: 1 -> 2
02/10/20,10:17:19.623505 NOTI Local server address(es): 192.168.0.220[---L]
02/10/20,10:17:19.623533 DBG Core state: 2 -> 3
Initializing s6a layer: DONE
  
```

Figure 3.10: The running status of HSS

Then we run the MME in another Terminal window. The status of MME is shown in Figure 3.11, in which we can see a Table, which shows the number of attached/-connected UEs and connected eNBs. Moreover, the status of the HSS database service is opened. And in the first Terminal window, it shows an additional status for the database service in Figure 3.12.

Last but not least, we open a Terminal to run the SPGW, which will build a bridge for the HSS networking and allocate IP for UE by using the GTP kernel. The S11

```
000288 00090:742038 7FAB0AFFD700 DEBUG MME-AP src/mne_app/mne_app_statistics.c:0045 ===== STATISTICS =====
===
000289 00090:432738 7FA7FBFFF700 ALERT S6A ge/openair-cn/src/s6a/s6a_task.c:0080 'STATE_OPEN' <- 'FDEVP_PSH_TIMEOUT' ((nil),0) 'hss.openair4G.eur'
000290 00090:432737 7FA7FBFFF700 ALERT S6A ge/openair-cn/src/s6a/s6a_task.c:0080 Peer timeout reset to 30 seconds
000291 00090:432811 7FA7FBFFF700 ALERT S6A ge/openair-cn/src/s6a/s6a_task.c:0080 'hss.openair4G.eur' in state 'STATE_OPEN' waiting for next event.
000292 00090:432858 7FA797FFF700 ALERT S6A ge/openair-cn/src/s6a/s6a_task.c:0080 SENT to 'hss.openair4G.eur': 'Device-Watchdog-Request'0/280 f:R--- src:'(nil)' len:84 [C:264
/l:26,C:296/l:21,C:278/l:12]
000293 00090:432881 7FA797FFF700 ALERT S6A ge/openair-cn/src/s6a/s6a_task.c:0080 Sending 84b data on connection {----} TCP,#38->127.0.0.1(3868)
000294 00090:433400 7FA7FBFFF700 ALERT S6A ge/openair-cn/src/s6a/s6a_task.c:0080 'STATE_OPEN' <- 'FDEVP_CNX_MSG_RECV' (0x7fa798000910,96) 'hss.openair4G.eur'
000295 00090:433485 7FA7FBFFF700 ALERT S6A ge/openair-cn/src/s6a/s6a_task.c:0080 RCV from 'hss.openair4G.eur': (no model)0/280 f:---- src:'hss.openair4G.eur' len:96 [C:268/l
:12,C:264/l:25,C:296/l:21,C:278/l:12]
000296 00090:433517 7FA7FBFFF700 ALERT S6A ge/openair-cn/src/s6a/s6a_task.c:0080 Iterating on rules of COMMAND: 'Device-Watchdog-Answer'.
000297 00090:433543 7FA7FBFFF700 ALERT S6A ge/openair-cn/src/s6a/s6a_task.c:0080 Peer timeout reset to 30 seconds (+/- 2)
000298 00090:433555 7FA7FBFFF700 ALERT S6A ge/openair-cn/src/s6a/s6a_task.c:0080 'hss.openair4G.eur' in state 'STATE_OPEN' waiting for next event.
000299 00090:742028 7FAB0AFFD700 DEBUG MME-AP src/mne_app/mne_app_statistics.c:0033 ===== STATISTICS =====
===
000300 00090:742045 7FAB0AFFD700 DEBUG MME-AP src/mne_app/mne_app_statistics.c:0034
000301 00090:742050 7FAB0AFFD700 DEBUG MME-AP src/mne_app/mne_app_statistics.c:0036
000302 00090:742056 7FAB0AFFD700 DEBUG MME-AP src/mne_app/mne_app_statistics.c:0038
000303 00090:742062 7FAB0AFFD700 DEBUG MME-AP src/mne_app/mne_app_statistics.c:0040
000304 00090:742067 7FAB0AFFD700 DEBUG MME-AP src/mne_app/mne_app_statistics.c:0042
000305 00090:742072 7FAB0AFFD700 DEBUG MME-AP src/mne_app/mne_app_statistics.c:0044
000306 00090:742078 7FAB0AFFD700 DEBUG MME-AP src/mne_app/mne_app_statistics.c:0045 ===== STATISTICS =====
===

```

	Current Status	Added since last display	Removed since last display
Connected eNBs	0	0	0
Attached UEs	0	0	0
Connected UEs	0	0	0
Default Bearers	0	0	0
S1-U Bearers	0	0	0

Figure 3.11: The running status of MME

```
Query: SELECT idmmeidentity FROM mmeidentity WHERE mmeidentity.mmehost='edge.openair4G.eur'
Accepting edge.openair4G.eur peer
02/10/20,10:25:34.766408 NOTI Connected to 'edge.openair4G.eur' (TCP,#oc7), remote capabilities:
02/10/20,10:25:34.766430 NOTI 'Capabilities-Exchange-Request'
02/10/20,10:25:34.766438 NOTI Version: 0x01
02/10/20,10:25:34.766444 NOTI Length: 200
02/10/20,10:25:34.766450 NOTI Flags: 0x80 (R---)
02/10/20,10:25:34.766455 NOTI Command Code: 257
02/10/20,10:25:34.766461 NOTI ApplicationId: 0
02/10/20,10:25:34.766466 NOTI Hop-by-Hop Identifier: 0x309FF15F
02/10/20,10:25:34.766472 NOTI End-to-End Identifiers: 0x10E22029
02/10/20,10:25:34.766479 NOTI {internal data}: src:(nil)(0) rmb:(nil) rt:0 cb:(nil),(nil)((nil) qry:(nil) asso:1 sess:(nil)
02/10/20,10:25:34.766486 NOTI AVP: 'Origin-Host'(264) l=26 f=M val="edge.openair4G.eur"
02/10/20,10:25:34.766493 NOTI AVP: 'Origin-Realm'(296) l=21 f=M val="openair4G.eur"
02/10/20,10:25:34.766499 NOTI AVP: 'Origin-State-Id'(278) l=12 f=M val=1581326734 (0x5e41218e)
02/10/20,10:25:34.766505 NOTI AVP: 'Host-IP-Address'(257) l=14 f=M val=192.168.0.220
02/10/20,10:25:34.766510 NOTI AVP: 'Vendor-Id'(266) l=12 f=M val=0 (0x0)
02/10/20,10:25:34.766516 NOTI AVP: 'Product-Name'(269) l=20 f=- val="FreeDiameter"
02/10/20,10:25:34.766521 NOTI AVP: 'Firmware-Revision'(267) l=12 f=- val=10200 (0x27d8)
02/10/20,10:25:34.766527 NOTI AVP: 'Inband-Security-Id'(299) l=12 f=M val="NO_INBAND_SECURITY" (0 (0x0))
02/10/20,10:25:34.766533 NOTI AVP: 'Vendor-Specific-Application-Id'(260) l=32 f=M val=(grouped)
02/10/20,10:25:34.766541 NOTI AVP: 'Auth-Application-Id'(258) l=12 f=M val=16777251 (0x1000023)
02/10/20,10:25:34.766547 NOTI AVP: 'Vendor-Id'(266) l=12 f=M val=10415 (0x28af)
02/10/20,10:25:34.766552 NOTI AVP: 'Supported-Vendor-Id'(265) l=12 f=M val=10415 (0x28af)
02/10/20,10:25:34.766584 DBG SENT to 'edge.openair4G.eur': 'Capabilities-Exchange-Answer'0/257 f:---- src:'(nil)' len:200 [C:268/l:12,C:264/l:25,C
:296/l:21,C:278/l:12,C:257/l:14,C:266/l:12,C:267/l:20,C:267/l:12,C:260/l:32,C:265/l:12]
02/10/20,10:25:34.766645 NOTI No TLS protection negotiated with peer 'edge.openair4G.eur'.
02/10/20,10:25:34.766782 NOTI 'STATE_CLOSED' -> 'STATE_OPEN' 'edge.openair4G.eur'
```

Figure 3.12: The additional status of HSS with connected MME

interface is also initialized for connecting to the MME. The output is shown in Figure 3.13. All the commands are listed below.

```
1 # First open a terminal
2 # Then locate to the HSS build file
3 ~$ cd ~/openair-cn/scripts
4 ~$ ./run_hss #this is the second time that we run the hss.
5
6 # Open another terminal
7 # Then locate to the MME build file
8 ~$ cd ~/openair-cn/scripts
9 # Build the MME again, because we have updated the mme.conf
10 ~$ sudo ./build_mme -c
11 ~$ ./run_mme
12
```

```

13 # Open another terminal
14 # Then locate to the MME build file
15 ~$ cd ~/openair-cn/scripts
16 ~$ ./run_spgw

```

```

000134 00000:177067 7F85C7708700 DEBUG S11 dge/openair-cn/src/s11/s11_sgw.c:0244 Tx UDP INIT IP addr 127.0.11.2
000135 00000:177080 7F85C7708700 DEBUG S11 dge/openair-cn/src/s11/s11_sgw.c:0301 Initializing S11 interface: DONE
000136 00000:177086 7F85C7708700 DEBUG SPGW-A ge/openair-cn/src/sgw/sgw_task.c:0148 Initializing SPGW-APP task interface
000137 00000:177091 7F85C7708700 DEBUG GTPV1- air-cn/src/gtpv1-u/gtpv1u_task.c:0096 Initializing GTPV1U interface
000138 00000:177092 7F85C7708700 DEBUG UDP /src/udp/udp_primitives_server.c:0126 Creating new listen socket on address 127.0.11.2 and port 2123
000139 00000:178381 7F85C37D7700 DEBUG UDP /src/udp/udp_primitives_server.c:0171 Inserting new descriptor for task 6, sd 31
000140 00000:178431 7F85C37D7700 DEBUG UDP /src/udp/udp_primitives_server.c:0187 Received 1 events
000141 00000:207959 7F85C7708700 NOTIC GTPV1- -cn/src/gtpv1-u/gtp_mod_kernel.c:0081 Using the GTP kernel mode (genl ID is 28)
000142 00000:228528 7F85C7708700 DEBUG GTPV1- -cn/src/gtpv1-u/gtp_mod_kernel.c:0104 Setting route to reach UE net 172.16.0.0 via gtp0
000143 00000:228879 7F85C7708700 NOTIC GTPV1- -cn/src/gtpv1-u/gtp_mod_kernel.c:0111 GTP kernel configured
000144 00000:229392 7F85C7708700 DEBUG GTPV1- air-cn/src/gtpv1-u/gtpv1u_task.c:0124 Initializing GTPV1U interface: DONE
000145 00000:311106 7F85C7708700 DEBUG SPGW-A ge/openair-cn/src/sgw/sgw_task.c:0208 Initializing SPGW-APP task interface: DONE

```

Figure 3.13: The running status of SPGW

Now the OAI EPC is successfully booted. Then, we turn to the computer, which has OAI eNB installed. In section 3.2, we have installed the driver of the USRP. For initializing the eNB, OAI has already provided some configuration files for different hardware platforms for software radio, different wireless communication bands and various bandwidths. In our implementation, we take the configuration file of the USRP b210 with 25 RBs in the frequency domain of band 7. The configuration file can be found in this path: /openairinterface5g/targets/PROJECTS/GENERIC-LTE-EPC/CONF/enb.band7.tm1.25PRB.usrpb210.conf. We have changed some information in the configuration referred to our system's setup, which is listed below:

```

1 # Open the configuration file
2 ~$ sudo vim ~/openairinterface5g/targets/PROJECTS/GENERIC-LTE-
   ↳EPC/CONF/enb.band7.tm1.50PRB.usrpb210.conf
3 # Change some information from these lines below
4 tracking_area_code = "1";
5 mobile_country_code = "262";
6 mobile_network_code = "70";
7 mme_ip_address = ( { ipv4 = "192.168.53.203";
8
9
10 ENB_INTERFACE_NAME_FOR_S1_MME = "enp5s0";
11 ENB_IPV4_ADDRESS_FOR_S1_MME = "192.168.53.204/24";
12
13
14 ENB_INTERFACE_NAME_FOR_S1U = "enp5s0";
15 ENB_IPV4_ADDRESS_FOR_S1U = "192.168.53.204/24";
16
17 parallel_config = "PARALLEL_SINGLE_THREAD";

```

where `mme_ip_address` is the IP address of our EPC, and the last four lines about `ENB_INTERFACE` is our eNB related network interface information. We have also built the OAI eNB in section 3.2. The OAI eNB will generate a application binary file called `lte-softmodem`, which is a main function for the eNB. There are also many tools that can be compiled at the same time when the eNB is built. For example, the T tracer, which is sort of a framework to debug and monitor the softmodem and made of two main parts. One part is an events' collector integrated into real-time processing. Another part is a separate set of programs to receive, record, display and analyze the events sent by the collector. In order to visualize the softmodem, we should start the `lte-softmodem` with additional command **T_stdout 0**, and after the eNB is initialized, it will wait for a tracer to connect to it before processing, which brings convenience for the work of channel measurement later. Then, we can go into the directory of the T tracer to start it. All those processes are listed below.

```

1 # Open a Terminal for running the eNB
2 # Go into the directory of lte-softmodem, which is generated
   ↳after the compilation
3 ~$ cd openairinterface5g/cmake_targets/lte_build_lte/build
4 # run the lte-softmodem with visualization option
5 ~$ sudo -E ./lte-softmodem -O $OPENAIR_DIR/targets/PROJECTS/
   ↳GENERIC-LTE-EPC/CONF/nb.band7.tm1.25PRB.usrpb210.conf --
   ↳T_stdout 0
6 # The system will start to initialization and waiting for the T
   ↳tracer
7
8 # Open another Terminal for running T tracer
9 # Go into the directory of T tracer
10 ~$ cd openairinterface5g/common/utils/T/tracer
11 ~$ ./enb -d ../T_messages.txt

```

After having executed those command above, two GUI windows will pop up. One is the main softmodem monitor, the other is the selector of the event options. We can see the softmodem monitor like in Figure 3.14, means that, we have successfully started the OAI eNB, which broadcasts the LTE network signal.

In the next step, we will insert the programmed SIM card to the smartphone. In our experiment, we use the LG-D802 with the Android operating system. We type the hidden code `*##4636##` on the Dial Pad of the phone, we can open the interface of the phone information setting. Here we find the option of network and select `LTE_ONLY`. Actually, the different smartphone has a different way of settings.

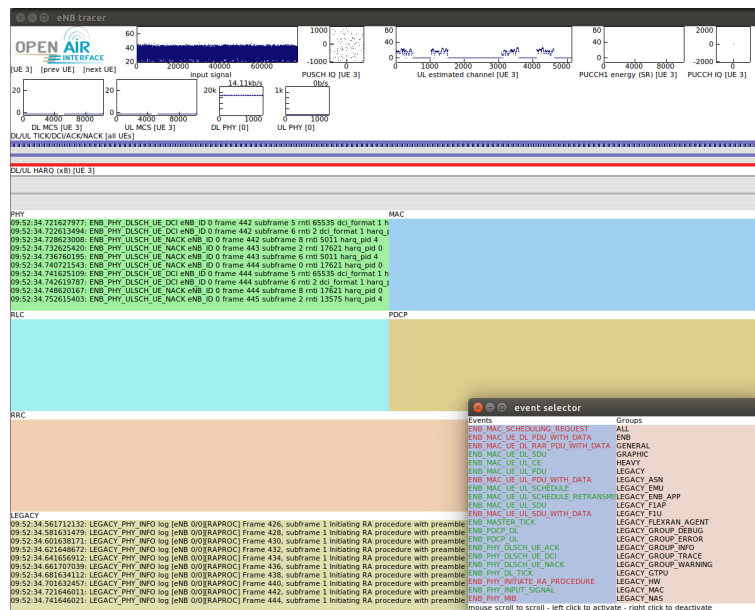


Figure 3.14: LTE softmodem monitor and event selector without UE connected

Then, we need to reboot the phone and find the Network settings page and do something like the following steps:

- Add a new APN, name is at will. And the APN is oai.ipv4, the same as in the database. Others can be blank.
- Open data roaming
- Change the option of automatic selecting in the Carrier option to manual selecting. After few seconds, we can see a network called 26270, then we can select and register onto this network.

After we have connected the smartphone to the network, we can see more information shown on the monitor of the lte-softmodem, see Figure 3.15. So we have successfully built an LTE network by using the OAI EPC and eNB. Moreover, we have connected the smartphone to this LTE network, which can surf the internet for uploading/downloading without any problem 3.16. Based on the stable wireless communication network, we can perform further channel measurements with the help of the eNB source code, which will be introduced in the next section.

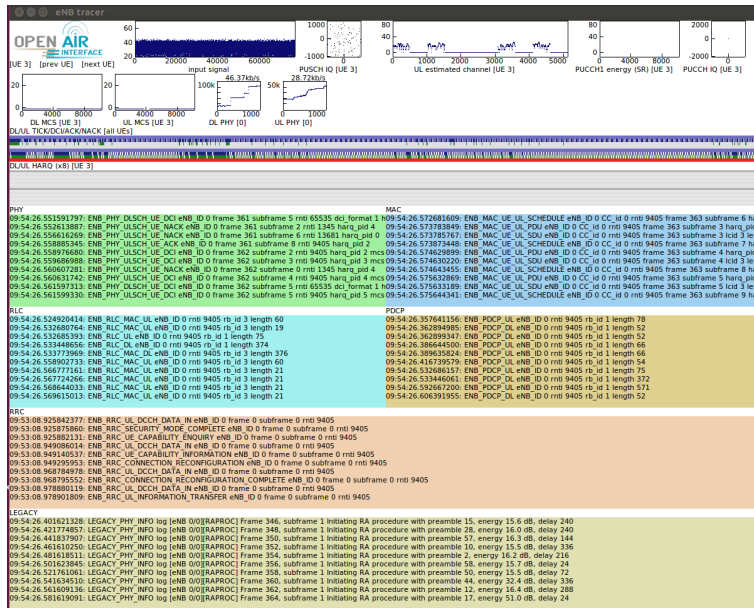


Figure 3.15: LTE softmodem monitor with UE connected

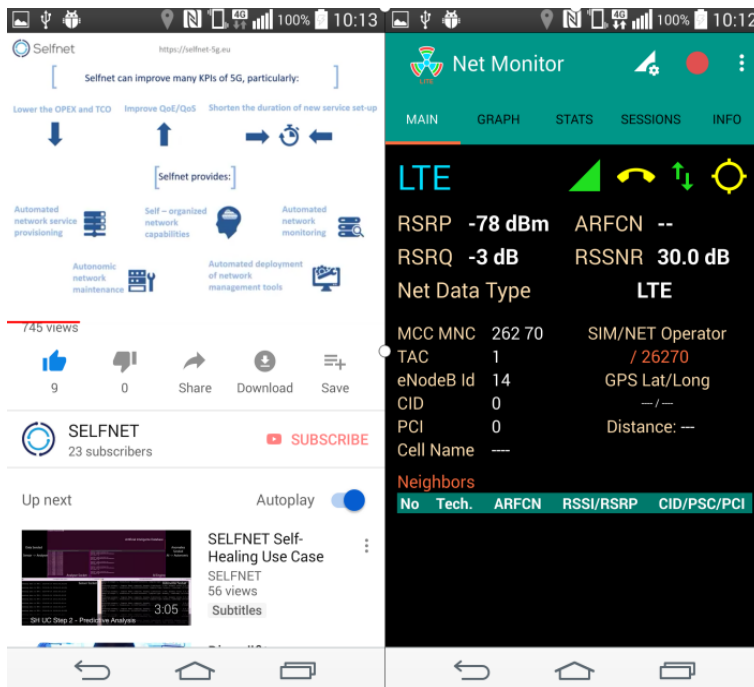


Figure 3.16: The screenshots for displaying the LTE connection status



Parameter	Value
imsi	262700000008960
iccid	8988211000000089600
pin1	9937
puk1	16302637
pin2	1358
puk2	87946054
ki	1854073054080694900C418F06A169DA
adm1	87544677
kic1	395A419A450293D548FE667A34B4B26B
kid1	B76DE94EDBDCE589B5E0C5302A8E0AD4
kik1	1E151D814886590AAACC574EE85C4978
msisdn	88211008960
acc	0001
opc	68ac1dc9320c5bf91a83d56e19eec0eb

Table 3.1: Key parameters of SIM card

Implementation of Channel Measurement

In the previous sections, we have introduced the fundamental concepts of our SDR platform, the channel estimation and the scheduling of the OAI LTE uplink channel. We have also implemented an SDR platform based LTE access and successfully connected the smartphone to this network with a programmed SIM card. The network is proved to be very stable by watching some online videos and taking video phone calls. Based on the steady LTE communication between the OAI eNB and smartphone, we can perform uplink channel measurement on the uplink channel receiving side. In our case is the eNB side. With a specific purpose, we mainly focus on the part of uplink physical transmission and procedure.

In this chapter, we will present the method we have taken to achieve a bandwidth-stable channel estimation in every time slots of every uplink subframes. In the section 4.1, we will first analyse the source codes of OAI eNB. Some special methods for computing in the channel estimation part are depicted in section 4.2. Methods for rapid data read and output is also introduced in this section. The implemented C code for data formatting and the deficient channel estimates are shown in section 4.3. For solving the problem of the deficient channel estimates, we will change the uplink channel scheduling, so that we can get the right output as we expect. Those works and better output of channel estimates will be presented in section 4.4.

4.1 OAI eNB source code analysis

As is introduced in section 2.2, the functionalities are hierarchically designed in the three folders of OAI referred to the LTE standards. We have directly located the physical layer functions in the folder called OpenAir1. In this folder, the functions

for baseband signal processing and some physical layer functional modules, like OFDM, modulation/demodulation, coding/decoding, channel estimation are fully implemented for both uplink and downlink channel in the folder called PHY. Besides, some physical layer scheduling functions for controlling the procedures are also provided in folders named with SCHED. When we look into the PHY folder, we can see the files defined as in Figure 4.1. In the folder called CODING, the turbo decoder, the convolution coding/decoding and the cyclic redundancy check (cyc) code generations are designed. The parameter initialization and memory allocation are included in the INIT. Most of our concerns are the LTE_ESTIMATION, LTE_REFSIG, LTE_TRANSPORT, and MODULATION, where the timing, reference signals, channel estimation, channel compensation, and channel modulation/demodulation are implemented.

```

nano@nano:~/openairInterface5g/openair1$ cd PHY
nano@nano:~/openairInterface5g/openair1/PHY$ tree -L 1
├── CODING
├── defs_common.h
├── defs_eNB.h
├── defs_L1_NB_IoT.h
├── defs_UE.h
├── impl_defs_lte.h
├── impl_defs_lte_NB_IoT.h
├── impl_defs_top.h
├── impl_defs_top_NB_IoT.h
├── INIT
├── LTE_ESTIMATION
├── LTE_REFSIG
├── LTE_TRANSPORT
├── LTE_UE_TRANSPORT
├── MODULATION
├── NBioT_TRANSPORT
├── NR_REFSIG
├── NR_TRANSPORT
├── NR_UE_TRANSPORT
├── phy_extern.h
├── phy_extern_ue.h
├── phy_vars.h
├── phy_vars_ue.h
├── sse_intrin.h
├── TOOLS
├── types.h
└── types_NB_IoT.h
12 directories, 15 files
  
```

Figure 4.1: PHY folder catalogue structure

We have already introduced the principle of uplink channel estimation with the help of embedded demodulation reference signals and sounding reference signals. We can find the C code program file `lte_ul_ref.c` for the generation of the uplink reference signal for both transceiver and receiver. Here we focus on the code snippet for the receiver, because the eNB in the uplink channel works as a receiver.

```

1 void generate_ul_ref_sigs_rx(void) {
  
```



```

2  .....
3  // These are the complex conjugated Zadoff-Chu sequences
   ↳quantized to QPSK stored in repeated format (for RB 3-100)
4  for (Msc_RS=2; Msc_RS<34; Msc_RS++) {
5      for (u=0; u<30; u++) {
6          for (v=0; v<2; v++) {
7              qbar = ref_primes[Msc_RS] * (u+1)/(double)31;
8              ul_ref_sigs_rx[u][v][Msc_RS] = (int16_t *)malloc16(2*
   ↳sizeof(int16_t)*dftsizes[Msc_RS]);
9
10             if (((int)floor(2*qbar))&1) == 0)
11                 q = (int)(floor(qbar+.5)) - v;
12             else
13                 q = (int)(floor(qbar+.5)) + v;
14
15             .....
16             for (n=0; n<dftsizes[Msc_RS]; n++) {
17                 m=n%ref_primes[Msc_RS];
18                 phase = (double)q*m*(m+1)/ref_primes[Msc_RS];
19                 ul_ref_sigs_rx[u][v][Msc_RS][n<<1] = (int16_t)(floor
   ↳(32767*cos(M_PI*phase)));
20                 ul_ref_sigs_rx[u][v][Msc_RS][1+(n<<1)] = -(int16_t)(
   ↳floor(32767*sin(M_PI*phase))); }
21             .....
22         }
23     }
24 }
25 }
26 }

```

In this code, the complex conjugated ZC sequence are mentioned, which has the following properties [28]:

- A ZC sequence has constant amplitude, and its root index point DFT also has constant amplitude. The constant amplitude property limits the Peak-to-Average Power Ratio (PAPR) and generates bounded and time-flat interference to other users. It also simplifies the implementation as only phases need to be computed and stored, not amplitudes.
- ZC sequence of any length have ideal cyclic autocorrelation. This property is of major interest when the received signal is correlated with a reference sequence and the received reference sequences are misaligned.
- The absolute value of the cyclic cross-correlation function between any two ZC sequences is constant and equal to $1/\sqrt{N_{ZC}}$, if $|q_1 - q_2|$ (where q_1 and q_2 are the sequence indices and N_{ZC} is the sequence length) is relatively prime with respect to N_{ZC} (a condition that can be easily guaranteed if N_{ZC}

is a prime number). The cross-correlation of $1/\sqrt{N_{ZC}}$ at all lags achieves the theoretical minimum cross-correlation value for any two sequences that have ideal autocorrelation.

Referred to the 3GPP TS 36.211 [3], the reference signal sequences are divided in groups, where $u \in \{0, 1, \dots, 29\}$ is the group number and v is the base sequence number within the group. The space of memory is firstly reserved through `malloc16`. Some reference tables are already defined in this file according to the TS 36.211 so that the root index q can be assigned and calculated with the number of RB in a reference sequence m and the `ref_primes` value at the index of sequence length M_{sc}^{RS} so that we can return the phase of ZC. A phase can then be split into a real part and an imaginary part through trigonometric function. The developer of this code has additionally multiply `32767` for the real and imaginary part to avoid loss of data.

The variable of reference signal defined in the C code above is then called in the C code file for uplink shared channel demodulation and estimation. We can find the demodulation file for the uplink shared channel in the folder `LTE_TRANSPORT`. The main function of the `ulsch_demodulation.c` is called `rx_ulsch`, which has also invoked the function for extracting the data in the RBs in the received signal. The function for uplink channel estimation is also invoked after the received data extraction.

```

1
2 void rx_ulsch(PHY_VARS_eNB *eNB,
3             L1_rxtx_proc_t *proc,
4             uint8_t UE_id) {
5
6     .....
7     for (l=0; l<(frame_parms->symbols_per_tti-ulsch[UE_id]->
8         ↳harq_processes[harq_pid]->srs_active); l++) {
9         if(LOG_DEBUGFLAG(DEBUG_ULSCH)) {
10            LOG_I(PHY,"rx_ulsch : symbol %d (first_rb %d,nb_rb %d),
11                ↳rxdataF %p, rxdataF_ext %p\n",l,
12                ulsch[UE_id]->harq_processes[harq_pid]->first_rb,
13                ulsch[UE_id]->harq_processes[harq_pid]->nb_rb,
14                common_vars->rxdataF,
15                pusch_vars->rxdataF_ext);
16        }
17
18        ulsch_extract_rbs_single(common_vars->rxdataF,
19                                pusch_vars->rxdataF_ext,
20                                ulsch[UE_id]->harq_processes[harq_pid]
21                                ↳->first_rb,
22                                ulsch[UE_id]->harq_processes[harq_pid]
23                                ↳->nb_rb,
24                                1%(frame_parms->symbols_per_tti/2),

```



```

21         l/(frame_parms->symbols_per_tti/2),
22         frame_parms);
23
24     lte_ul_channel_estimation(eNB,proc,
25                             UE_id,
26                             l%(frame_parms->symbols_per_tti/2),
27                             l/(frame_parms->symbols_per_tti/2));
28
29     .....
30 }
31 }

```

The input of the demodulation function is the pointer of the eNB variables, the pointer of the time process variables and the ID of UE. As time goes forward, the data on every symbol of the time shaft are extracted. The extracted data on the location of the reference signal will be further calculated with the locally generated reference signal for the channel estimation. The location of the reference signal is already shown in Figure 2.11, normally the reference signal of uplink shared channel exists at the fourth symbol of a slot. But if there are cyclic prefixes in the slots, then we can find the reference signal at the third symbol position. We can notice that the two invoked functions have two same input parameters, which are related to the l . l is the index of the a symbol. So we can deduce that the division operation returns the index of time slots in a subframe, and the modulus operation returns the index of symbol in one of the time slot in a subframe. Moreover, the HARQ process will also decide the span in the frequency domain, here is instructed by the number of RBs.

With the knowledge of how the reference signal is generated and how we calculated the channel estimates using the generated reference signal and received a reference signal, we can now analyze further the source code for the channel estimation, which is implemented in the `lte_ul_channel_estimation.c` in the folder `LTE_ESTIMATION`. Now in the program of channel estimation, the parameter l represents the index of a symbol in the slot, ranges from 0 to 6. By using this if sentence `if (l == (3 - frame_parms->Ncp))`, the symbol with reference signal will be then calculated, where the N_{cp} is 0 or 1 represents without or with cyclic prefix respectively.

The calculation channel estimates is implemented for x86/64 and the arm platform with the SSE method, which has obviously speed up the calculation and meet the real-time requirement. We will introduce the SSE in the next section.

4.2 SSE intrinsics and output data storage

Micheal Flynn has firstly classified the computer system structure according to the controller instructions and the data stream. Respectively are SISD, SIMD, MISD and MIMD.

SISD, is Single Instruction Single Data, the traditional von Neumann architecture is included in this structure. The architecture is shown in Figure 4.2.

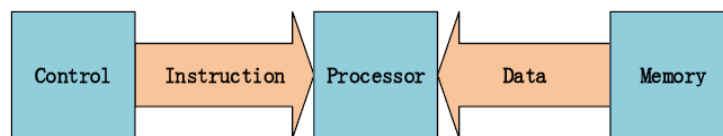


Figure 4.2: Single Instruction Single Data structure

MISD, is Mutiple Instruction Single Data, which process a single data stream with multiple different instructions. It is suitable for classification tasks. The structure can be seen in Figure 4.3.

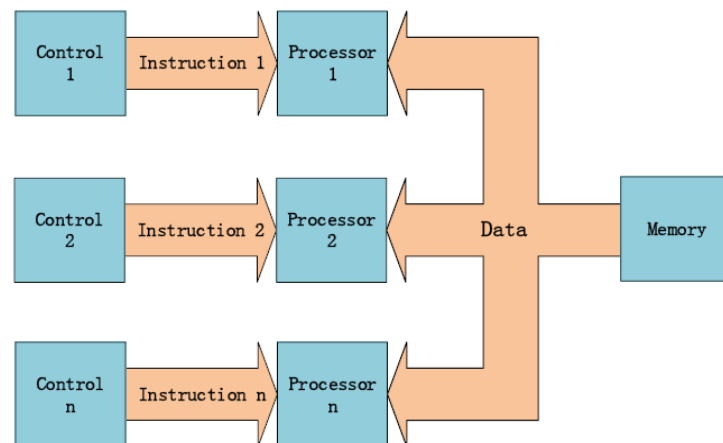


Figure 4.3: Mutiple Instruction Single Data structure

SIMD, is Single Instruction Multiple Data. In this structure, multiple different groups of data stream are processed with the same instruction. It is suitable for implementation of parallel algorithm, and is also widely used in the voice and image signal processing. The structure is depicted in Figure 4.4 below.

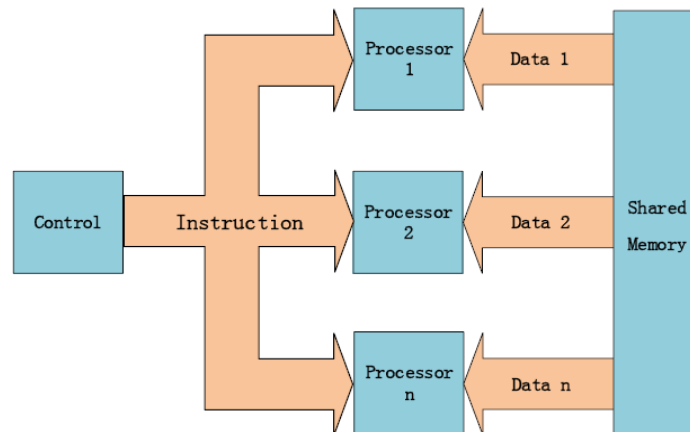


Figure 4.4: Single Instruction Multiple Data structure

MIMD, is Multiple Instruction Multiple Data, which process multiple different data streams from shared memory with different instructions. It can asynchronously execute different operations. MIMD is the most powerful structure, which is also difficult to be designed and implemented. The architecture is shown in Figure 4.5.

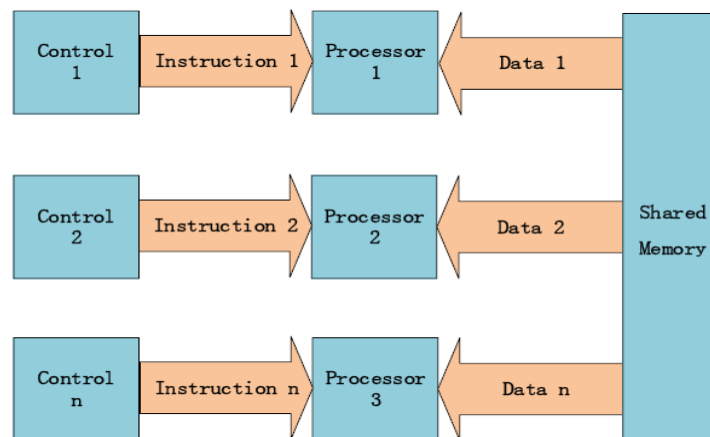


Figure 4.5: Multiple Instruction Multiple Data structure

SSE instruction is an extended instruction for SIMD provided by Intel. SSE has some following advantages:

- Functions for image browsing and processing with higher resolution.
- Occupy less CPU resources.

- Higher accuracy and quicker response speed.

To my understanding, the SIMD will enable the CPU to get an instruction from the memory to the instruction register, then read the data stream from the memory to the SIMD register. And the CPU will use this instruction parallel process this group of structure. The new version after SSE2 can process 128-bit data every time of operation. The possible storage format may be 16*8 bit, 8*16 bit, 4*32 bit, 2*64 bit and 1*128 bit. This can be referred to Figure 4.6

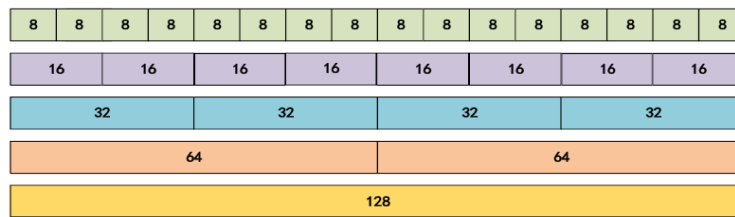


Figure 4.6: Storage format of 128 bit data stream

The intrinsic functions of SSE can be referred in <https://software.intel.com/sites/landingpage/IntrinsicsGuide/#expand=0>, which is an interactive tool for intrinsic functions. A common usage of the SSE is the complex value multiplication. In the c file of channel estimation, the channel estimates are so calculated that the complex conjugate of the received signal at the symbol with reference signals will be multiplied by the from the receiver generated reference signal. The related C codes are listed here.

```

1   for (i=0; i<Msc_RS/12; i++) {
2       // multiply by conjugated channel
3       mmtmpU0 = _mm_madd_epi16(ul_ref128[0], rxddataF128[0]);
4       // mmtmpU0 contains real part of 4 consecutive outputs
5       // (32-bit)
6       mmtmpU1 = _mm_shufflelo_epi16(ul_ref128[0], _MM_SHUFFLE
7       // (2,3,0,1));
8       mmtmpU1 = _mm_shufflehi_epi16(mmtmpU1, _MM_SHUFFLE
9       // (2,3,0,1));
10      mmtmpU1 = _mm_sign_epi16(mmtmpU1, *(_m128i*)&conjugate
11      // [0]);
12      mmtmpU1 = _mm_madd_epi16(mmtmpU1, rxddataF128[0]);
13      // mmtmpU1 contains imag part of 4 consecutive outputs
14      // (32-bit)
15      mmtmpU0 = _mm_srai_epi32(mmtmpU0, 15);
16      mmtmpU1 = _mm_srai_epi32(mmtmpU1, 15);
17      mmtmpU2 = _mm_unpacklo_epi32(mmtmpU0, mmtmpU1);
18      mmtmpU3 = _mm_unpackhi_epi32(mmtmpU0, mmtmpU1);
19
20      ul_ch128[0] = _mm_packs_epi32(mmtmpU2, mmtmpU3);

```

```

16 // multiply by conjugated channel
17 mmtmpU0 = _mm_madd_epi16(ul_ref128[1], rxdataF128[1]);
18 // mmtmpU0 contains real part of 4 consecutive outputs
   ↳ (32-bit)
19 mmtmpU1 = _mm_shufflelo_epi16(ul_ref128[1], _MM_SHUFFLE
   ↳ (2,3,0,1));
20 mmtmpU1 = _mm_shufflehi_epi16(mmtmpU1, _MM_SHUFFLE
   ↳ (2,3,0,1));
21 mmtmpU1 = _mm_sign_epi16(mmtmpU1, *(_m128i*) conjugate);
22 mmtmpU1 = _mm_madd_epi16(mmtmpU1, rxdataF128[1]);
23 // mmtmpU1 contains imag part of 4 consecutive outputs
   ↳ (32-bit)
24 mmtmpU0 = _mm_srai_epi32(mmtmpU0, 15);
25 mmtmpU1 = _mm_srai_epi32(mmtmpU1, 15);
26 mmtmpU2 = _mm_unpacklo_epi32(mmtmpU0, mmtmpU1);
27 mmtmpU3 = _mm_unpackhi_epi32(mmtmpU0, mmtmpU1);
28
29 ul_ch128[1] = _mm_packs_epi32(mmtmpU2, mmtmpU3);
30
31 mmtmpU0 = _mm_madd_epi16(ul_ref128[2], rxdataF128[2]);
32 // mmtmpU0 contains real part of 4 consecutive outputs
   ↳ (32-bit)
33 mmtmpU1 = _mm_shufflelo_epi16(ul_ref128[2], _MM_SHUFFLE
   ↳ (2,3,0,1));
34 mmtmpU1 = _mm_shufflehi_epi16(mmtmpU1, _MM_SHUFFLE
   ↳ (2,3,0,1));
35 mmtmpU1 = _mm_sign_epi16(mmtmpU1, *(_m128i*) conjugate);
36 mmtmpU1 = _mm_madd_epi16(mmtmpU1, rxdataF128[2]);
37 // mmtmpU1 contains imag part of 4 consecutive outputs
   ↳ (32-bit)
38 mmtmpU0 = _mm_srai_epi32(mmtmpU0, 15);
39 mmtmpU1 = _mm_srai_epi32(mmtmpU1, 15);
40 mmtmpU2 = _mm_unpacklo_epi32(mmtmpU0, mmtmpU1);
41 mmtmpU3 = _mm_unpackhi_epi32(mmtmpU0, mmtmpU1);
42
43 ul_ch128[2] = _mm_packs_epi32(mmtmpU2, mmtmpU3);
44
45 ul_ch128+=3;
46 ul_ref128+=3;
47 rxdataF128+=3;
48 }

```

As is introduced in the frame structure, there are 12 resource elements (RE) in a resource block RB, so the for-loop is indexed with the RB, where the total number of resource elements in a symbol is represented with Msc_RS. The data of reference signals are stored in the format of 16*8 bit, where each real or imaginary part of the complex value are stored in 16-bit containers. So, each 128-bit data stream contains 4 complex values with the real and imaginary parts. Thus, for every time computation of the channel estimates in a RB, it needs 3 parts of complex value multiplication. Here the outputs `ul_ch128[0]`, `ul_ch128[1]`,

`u1_ch128[2]` have 128-bit data size, which are the results of original channel estimates at the position of reference signal without any channel compensation. Each output contains 8 values with 16-bit data size, respectively are the 4 complex values' real and imaginary parts. Along with the for-loop of RBs, the for-loop of symbols and the input frame /subframe indices, we can calculate and read the value of channel estimates, as long as the channel demodulation happens on a subframe.

After we have located the output of the complex-valued channel estimates. We can start to read and store the data for further researches. Because of the initialization of the eNB configuration, that we have set the `parallel_single_thread`, which means that all the TX and RX frames are transmitted in a single thread. This initialization will alleviate the pressure of the processor for operation but have a high restriction on real-time processing. For getting the channel estimates continuously on every resource element of every symbol of every subframe, there is a huge data size of output channel estimates within every 0.5 ms .

Here we have first tried a method that directly prints the data to a local text file. That is, after each for-loop of RB, we will read the 12 complex values (24 integers of real and imaginary value) and print it to a line of the text file. The problems are that:

1. The estimation functions are called every slot (0.5 ms). So the duration of the file open and close on a hard disk will have a negative impact on the real-time processing. Then the eNB will get crashed.
2. Attempts are also taken, that we have defined a new variable in the variable header file of eNB `def_eNB.h` for the file name pointer and moved the file open/close function into the demodulation function or function in higher level. So that the file open/close will not be called so frequently. However, because of the exists of the buffer zone of the `fprintf` in the memory, if the buffer size reaches the default size (1024 bytes), then the data in the buffer will be flashed to the TXT file, which is stored in the hard disk. The duration of writing the data onto the hard disk will lead to a crash of the system.
3. Then the method is further improved, that we have manually set the buffer size to be 0 so that we can actively flush the buffer zone by using `fflush` ↪ (`stdout`). This method helps a lot to some extent if the UE only has some download tasks and the uplink channel serves only for downlink requests and slight uplink transmissions. Because, under this situation,

the channel demodulation and estimation on the side of eNB don't happen in every subframe, and the resource elements are not occupied completely on each symbol. However, for a continuous uplink channel estimation, we should upload files with big data size, then the quantities of generated channel estimates will be much more than before. In our case of SISO communication, the uplink channel will be fully reserved for the single UE. Due to the eNB initialization, we have set a bandwidth with 25 RBs. Thus, the PUSCH reserves 20 RBs for the demodulation reference signal. So, after channel estimation on a symbol, we should write $12 \times 2 \times 20 = 480$ 16 bit integers in $0.5ms \times 6/7 \approx 0.4ms$ before the next channel estimation executes. There are still some symbols of latency warnings displayed on the Terminal window. Because of those warnings, there are also errors that appear on the corresponding uplink shared channel subframes reported from the LEGACY.

We would say that the I/O between the hard disk and processor is not quick enough for the read of real-time generated channel estimates. Faced to those problems, we have reviewed the relevant codes of the OAI eNB, and have found that the OAI has also used some `printf` in the functions like below in the code of channel estimation:

```
1 LOG_E(PHY, "lte_ul_channel_estimation: index for Msc_RS=%d not
  ↳found\n", Msc_RS);
2 LOG_D(PHY, "subframe %d, Ns %d, l %d, Msc_RS = %d, Msc_RS_idx =
  ↳%d, u %d, v %d, cyclic_shift %d\n", subframe, Ns, l, Msc_RS,
  ↳Msc_RS_idx, u, v, cyclic_shift);
3 LOG_M("drs_seq0.m", "drsseq0", ul_ref_sigs_rx[u][v][Msc_RS_idx
  ↳], 2*Msc_RS, 2, 1);
```

In those LOG functions, the inputs will then be printed to the monitor of the softmodem. All those data transmissions are between the processor and memory, which has a nano-second level I/O speed so that it can display various system status information without any impact on the real-time computation.

We have then tried to directly display the channel estimates onto the Terminal window since the `printf` function will flash the data at the end of every line because of the line breaks. All those operations are executed on memory, which doesn't have any problem with the speed. In order to keep a clean Terminal

Window for purely displaying the output of channel estimates, all the other system statuses, which are shown additionally on the Terminal are deactivated.

According to the experiments done in the previous steps, we have concluded, that the occupied RBs will be up to 2 if the uplink channel doesn't transmit much data. If there is a mass of data transmission, then there will be 20 RBs occupied. In order to filter some useless output, we have also added a `if` function to get the channel estimates only if the occupied RBs is more than 18, and only take 100 frames for testing. Some codes are listed below.

```

1  if (1 == (3 - frame_parms->Ncp)) {
2      symbol_offset = frame_parms->N_RB_UL*12*(1+((7-frame_parms->
        ↳Ncp)*(Ns&1)));
3      for (aa=0; aa<nb_antennas_rx; aa++) {
4          .....
5
6          for(i=symbol_offset; i<symbol_offset+Msc_RS; i++) {
7              ul_ch_estimates_re = ((int16_t*) ul_ch_estimates[aa])[i
        ↳<<1];
8              ul_ch_estimates_im = ((int16_t*) ul_ch_estimates[aa])[i
        ↳<<1+1];
9              if(N_rb_alloc>18 && frame>900 && frame<1000 && i<
        ↳symbol_offset+12){
10                 printf("frame %d subframe %d, slot %d, re %d, im %d\n",
        ↳ frame, subframe, Ns, ul_ch_estimates_re,
        ↳ ul_ch_estimates_im);
11             }
12         }
13     }
14 }

```

Where the `N_rb_alloc` in this code is the allocated RBs for the symbol of the reference signal. The variable `aa` is the index of antenna for receiving the uplink data. The `symbol_offset` represents the start position of the reference signal on the uplink frame structure. Because there are always 24 values in a RB, so we can convert the output format by brutally simplifying the code like below. The code is executed directly after each loop of the channel estimates computation in a RB. So that we don't need other several loops for the RB and symbol indices. We can also display all the 24 values in a line on the Terminal window, which will take benefits for the data formatting later. After having investigated some efforts, we can finally read the channel estimates on continuous subframes.

```

1
2  .....

```



```

3
4   mmtmpU0 = _mm_srari_epi32(mmtmpU0,15);
5   mmtmpU1 = _mm_srari_epi32(mmtmpU1,15);
6   mmtmpU2 = _mm_unpacklo_epi32(mmtmpU0,mmtmpU1);
7   mmtmpU3 = _mm_unpackhi_epi32(mmtmpU0,mmtmpU1);
8
9   ul_ch128[2] = _mm_packs_epi32(mmtmpU2,mmtmpU3);
10
11  if(N_rb_alloc>18 && frame>700 && frame<801){
12      printf("f %d s %d i %d ch:
13          %d %d %d
14          %d %d %d %d
15          %d %d %d %d
16          %d %d %d %d
17          %d %d %d %d
18          %d %d %d %d %d\n",
19          frame, subframe, i,
20          ((int16_t*)ul_ch128)[0],((int16_t*)ul_ch128)[1],((int16_t*)
21          ↳ul_ch128)[2],((int16_t*)ul_ch128)[3],
22          ((int16_t*)ul_ch128)[4],((int16_t*)ul_ch128)[5],((int16_t*)
23          ↳ul_ch128)[6],((int16_t*)ul_ch128)[7], ((int16_t*)ul_ch128
24          ↳)[8],((int16_t*)ul_ch128)[9],((int16_t*)ul_ch128)[10],((
25          ↳int16_t*)ul_ch128)[11],
26          ((int16_t*)ul_ch128)[12],((int16_t*)ul_ch128)[13],((int16_t*)
27          ↳ul_ch128)[14],((int16_t*)ul_ch128)[15], ((int16_t*)
28          ↳ul_ch128)[16],((int16_t*)ul_ch128)[17],((int16_t*)
29          ↳ul_ch128)[18],((int16_t*)ul_ch128)[19],
30          ((int16_t*)ul_ch128)[20],((int16_t*)ul_ch128)[21],((int16_t*)
31          ↳ul_ch128)[22],((int16_t*)ul_ch128)[23]);
32  }

```

4.3 Channel estimates formatting and visualizing

In order to format the data to be readable from MATLAB, we need to convert the data of massive integers to be like the complex values that are stored in the frame structure. The output data stored in a TXT file is formed like in Figure 4.7. The measurement for 100 frames will generate $100 \times 10 \times 2 \times 20 = 40000$ lines of channel estimates outputs. The channel estimates in a single RB are displayed in a line.

A C code program is written for reading and converting the outputs to our expected form. As listed below, we have first defined an array to be the final expected form, which has the data format of *complexint*, that is `complex int ↳subframe_ch_est [] []`, where the first index of the array is the total number



of resource elements in the frequency domain and the second index is the number of slots in the time domain. We have used the `fgets()`, and `sscanf()` to get and read the data line by line. Those for data processing useless symbols will be discarded by `"*s"`, all the useful data are read by the `"s"` as a string and later must be converted into integers through `atoi()`. The correspondent integers of real and imaginary parts combined to be a complex value, in which the integers of the imaginary part are multiplied by an imaginary symbol `i`.

```
1  while (fgets(line, sizeof(line), file)){
2      sscanf(line,
3          "%*s %*s %*s %*s %*s %*s %*s
4          %s %s %s %s %s %s %s %s %s %s %s %s
5          %s %s %s %s %s %s %s %s %s %s",
6          re0, im0, re1, im1, re2, im2, re3, im3, re4, im4, re5, im5,
7          re6, im6, re7, im7, re8, im8, re9, im9, re10, im10, re11,
8          im11);
9      //string to int
10     re_int0 = atoi(re0); re_int1 = atoi(re1);
11     re_int2 = atoi(re2); re_int3 = atoi(re3);
12     re_int4 = atoi(re4); re_int5 = atoi(re5);
13     re_int6 = atoi(re6); re_int7 = atoi(re7);
14     re_int8 = atoi(re8); re_int9 = atoi(re9);
15     re_int10 = atoi(re10); re_int11 = atoi(re11);
16     im_int0 = atoi(im0); im_int1 = atoi(im1);
17     im_int2 = atoi(im2); im_int3 = atoi(im3);
18     im_int4 = atoi(im4); im_int5 = atoi(im5);
19     im_int6 = atoi(im6); im_int7 = atoi(im7);
20     im_int8 = atoi(im8); im_int9 = atoi(im9);
21     im_int10 = atoi(im10); im_int11 = atoi(im11);
22
23     if(re==36)
24     {
25         re=re-36;
26         coloumn++;
27     }
28     subframe_ch_est[re+0][coloumn]=re_int0 + im_int0*_Complex_I;
29     subframe_ch_est[re+1][coloumn]=re_int1 + im_int1*_Complex_I;
30     subframe_ch_est[re+2][coloumn]=re_int2 + im_int2*_Complex_I;
31     subframe_ch_est[re+3][coloumn]=re_int3 + im_int3*_Complex_I;
32     subframe_ch_est[re+4][coloumn]=re_int4 + im_int4*_Complex_I;
33     subframe_ch_est[re+5][coloumn]=re_int5 + im_int5*_Complex_I;
34     subframe_ch_est[re+6][coloumn]=re_int6 + im_int6*_Complex_I;
35     subframe_ch_est[re+7][coloumn]=re_int7 + im_int7*_Complex_I;
36     subframe_ch_est[re+8][coloumn]=re_int8 + im_int8*_Complex_I;
37     subframe_ch_est[re+9][coloumn]=re_int9 + im_int9*_Complex_I;
38     subframe_ch_est[re+10][coloumn]=re_int10 + im_int10*
39     _Complex_I;
40     subframe_ch_est[re+11][coloumn]=re_int11 + im_int11*
41     _Complex_I;
42     //printf("%d %d %d\n",row, re_int, im_int); //check the value
43     re=re+12;
44 }
```

```

f 1021 s 7 i 0 ch: -70 1763 -264 -1722 479 1611 -729 -1511 864 1356 -1073 -1242 1387 1058 -1472 -843 1581 504 -1572 -288 1798 46 -1647 170
f 1021 s 7 i 1 ch: 1633 -477 -1463 868 1406 -920 -1138 1161 1036 -1279 -761 1584 517 -1469 -217 1720 -125 -1697 310 1734 -505 -1665 752 1463
f 1021 s 7 i 2 ch: -952 -1422 1142 1135 -1353 -990 1516 769 -1680 -524 1630 340 -1639 82 1744 -161 -1645 474 1503 -752 -1418 922 1281 -1112
f 1021 s 7 i 3 ch: -968 1471 687 -1457 -620 1700 160 -1574 -87 1595 -214 -1722 560 1609 -840 -1439 1141 1264 -1351 -1347 1359 980 -1523 -868
f 1021 s 7 i 4 ch: 1590 564 -1737 -144 1737 38 -1772 363 1600 -583 -1553 846 1364 -1042 -1270 1330 1083 -1390 -778 1512 517 -1649 -293 1678
f 1021 s 7 i 5 ch: -63 -1722 225 1665 -586 -1626 814 1474 -1016 -1418 1202 1338 -1367 -911 1456 793 -1670 -455 1684 297 -1723 54 1738 -282
f 1021 s 7 i 6 ch: -1714 519 1541 -831 -1290 923 1146 -1188 -940 1379 787 -1416 -413 1783 278 -1826 10 1722 -276 -1714 650 1657 -796 -1646
f 1021 s 7 i 7 ch: 1185 1405 -1185 -1199 1354 1097 -1576 -711 1716 470 -1668 -107 1795 -70 -1675 374 1567 -462 -1430 825 1402 -1086 -1188 1260
f 1021 s 7 i 8 ch: 953 -1525 -750 1463 514 -1635 -155 1631 -66 -1574 196 1611 -466 -1466 859 1484 -1106 -1246 1238 1026 -1323 -975 1458 716
f 1021 s 7 i 9 ch: -1602 -359 1634 188 -1651 94 1487 -332 -1521 628 1395 -899 -1344 1041 1041 -1168 -988 1398 647 -1699 -409 1615 218 -1614
f 1021 s 7 i 10 ch: 144 1633 -308 -1762 714 1596 -784 -1438 1084 1388 -1255 -1022 1381 786 -1472 -704 1620 413 -1522 -172 1713 -159 -1603 382
f 1021 s 7 i 11 ch: 1601 -470 -1482 825 1296 -1028 -1124 1301 816 -1328 -212 1501 208 -1310 -263 1529 -73 -1596 303 1540 -664 -1600 795 1459
f 1021 s 7 i 12 ch: -1124 -1355 1224 1149 -1417 -839 1487 542 -1672 -329 1723 52 -1608 226 1606 -413 -1600 542 1469 -863 -1306 961 1108 -1329
f 1021 s 7 i 13 ch: -907 1305 566 -1548 -312 1756 96 -1767 278 1680 -554 -1695 721 1551 -874 -1554 1056 1299 -1405 -1013 1465 990 -1610 -471
f 1021 s 7 i 14 ch: 1668 380 -1788 -42 1706 -166 -1586 416 1608 -631 -1460 737 1306 -1117 -931 1320 785 -1325 -565 1562 400 -1645 18 1720
f 1021 s 7 i 15 ch: -109 -1775 514 1612 -603 -1540 785 1482 -1142 -1275 1394 1073 -1464 -770 1671 562 -1708 -432 1666 99 -1680 1 1688 -500
f 1021 s 7 i 16 ch: -1536 715 1422 -1083 -1393 1091 1099 -1342 -753 1389 578 -1555 -382 1586 107 -1658 68 1610 -565 -1651 744 1424 -871 -1349
f 1021 s 7 i 17 ch: 1037 1262 -1327 -1047 1453 808 -1645 -593 1582 198 -1535 -53 1705 -145 -1685 422 1551 -700 -1482 861 1184 -1195 -945 1337
f 1021 s 7 i 18 ch: 746 -1452 -519 1506 378 -1608 -50 1646 -333 -1555 486 1556 -667 -1543 921 1376 -1179 -1134 1378 1063 -1577 -736 1580 604
f 1021 s 7 i 19 ch: -1700 -402 1632 80 -1563 250 1521 -523 -1495 755 1394 -915 -1200 1215 963 -1318 -688 1437 536 -1524 -196 1579 -14 -1632
f 1021 s 7 i 0 ch: -106 1707 70 1734 512 1635 632 1561 893 1582 1048 1192 1340 1105 1420 765 1581 620 1577 369 1777 108 1774 -318
f 1021 s 7 i 1 ch: 1637 -363 1575 -691 1407 -913 1242 -1009 1101 -1331 861 -1372 624 -1540 389 -1610 130 -1675 -127 -1667 -440 -1574 -651 -1528
  
```

Figure 4.7: Outputs of channel estimates

The converted data will be read and visualized in MATLAB so that we can macroscopically observe the time-frequency spectrum and check whether we have extracted the channel estimates correctly. Figure 4.8 has shown some samples of the spectrum from our measurement. We can notice that the channel on the same symbol of a different frequency is at the same magnitude level. So we can randomly pick one subcarrier and plot it on the X-Z axis view.

Like in Figure 4.9, we have picked the first subcarrier, normalized and plotted the estimated channel magnitude of the first 200 time samples. We can find an apparent problem from this plot. Between the red ellipse marked waveform and the green ellipse marked waveform, there should not be cliff-like changes like these that exist. As expected, all the estimated channel magnitude should have a thread like the green ellipse marked waveform on all the time samples. We notice that, there are many envelopes that are continuous near zero, which won't exist in a normal continuous wireless transmission environment. Assumptions are that, that the channel estimation is executed continuously on every time slot, but only the noises are received instead of information. The uplink channel will automatically allocate the bandwidth for an upload task, but because of the fairness of the UE uplink channel quality competition, there might be some missing or "waiting for retransmission" instructions of the uplink signal, which will lead to the receiver falsely estimate only the noise in the channel. Another reason might be the high bandwidth on a single time sample. The computer should compute massive data in around 0.07 ms , thus, there might be some latency for continuous demodulation,

which causes the false transmission of channel quality information. So we have further analyzed the OAI source code and focused on the uplink scheduling part.

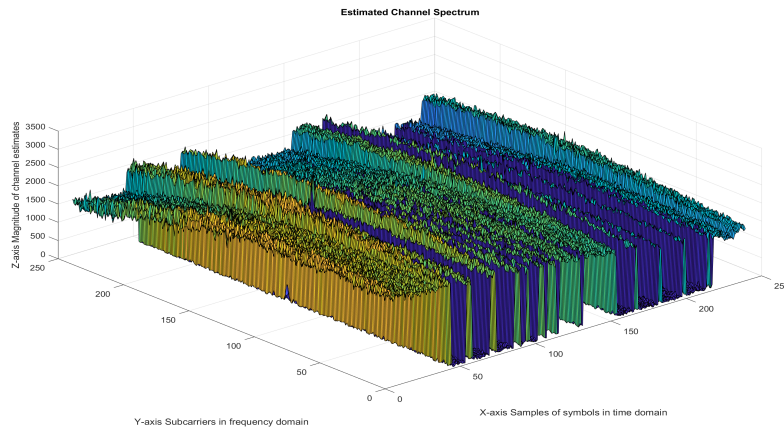


Figure 4.8: Estimated channel spectrum

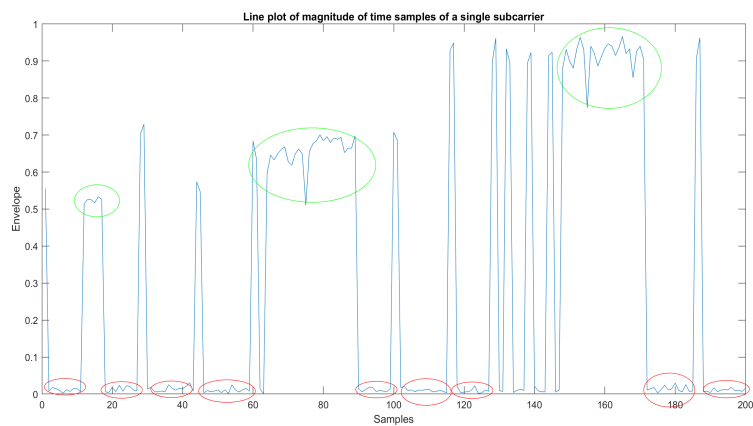


Figure 4.9: Estimated channel spectrum of a single subcarrier

4.4 Changes of uplink Scheduling

We have already introduced some concepts and procedures of downlink and uplink transmission in section 2.3.3. In a random access network, in order to dynamically and flexibly arrange the best channel for active UEs, the uplink channels are estimated and ranked by the receiver eNB. Commonly, the signal-noise ratio is converted to be the channel quality index, and the ranked channel qualities are referenced from the scheduler so that the bandwidth in the frequency domain and the subframes in the time domain can be allocated for a certain uplink channel demodulation. OAI has implemented the procedure in the `phy_procedures_lte_eNB` `.c` in the file path `openairinterface5g/openair1/SCHED`. As the codes of the `pusch_procedures()` are listed below, where some conditions are pre-judged before the execution of channel demodulation function `rx_ulsch(eNB, proc, i)`.

```

1 void pusch_procedures(PHY_VARS_eNB *eNB, L1_rxtx_proc_t *proc){
2     .....
3     .....
4     if ((ulsch) &&
5         (ulsch->rnti>0) &&
6         (ulsch_harq->status == ACTIVE) &&
7         (ulsch_harq->frame == frame) &&
8         (ulsch_harq->subframe == subframe) &&
9         (ulsch_harq->handled == 0)) {
10        .....
11        .....
12        start_meas(&eNB->ulsch_demodulation_stats);
13        rx_ulsch(eNB, proc, i);
14        stop_meas(&eNB->ulsch_demodulation_stats);
15        start_meas(&eNB->ulsch_decoding_stats);
16        ret = ulsch_decoding(eNB, proc,
17                            i,
18                            0, // control_only_flag
19                            ulsch_harq->V_UL_DAI,
20                            ulsch_harq->nb_rb>20 ? 1 : 0);
21        stop_meas(&eNB->ulsch_decoding_stats);
22        .....
23        .....
24    }
25 }
```

For example here the conditions `ulsch` and `ulsch->rnti>0`. When there is an uplink shared channel, that is allowed to work, then the `ulsch` has the value 1, otherwise, it will be assigned to be 0. The `rnti` stands for the Radio Network

Temporary Identifier, which is a kind of UE ID for the traffic between UE and eNB lower layer. It would be more accurate to think of it as [UE ID] + UCI Type ID since each of UCI message is scrambled by a specific RNTI value. As is introduced in section 2.3.3, the CQI is an important part of the UCI, which is calculated by the value of SNR. In this file `phy_procedures_lte_eNB.c`, the CQI is calculated under different application environment. Some related codes are listed below.

```

1  .....
2  void fill_rx_indication(PHY_VARS_eNB *eNB,
3                          int UE_id,
4                          int frame,
5                          int subframe) {
6  .....
7      // estimate UL_CQI for MAC (from antenna port 0 only)
8      int SNRtimes10 = dB_fixed_times10(eNB->pusch_vars[UE_id]->
9          ↳ulsch_power[0]) - 10 * eNB->measurements.
10         ↳n0_subband_power_dB[0][0];
11
12     if (SNRtimes10 < -640)
13         pdu->rx_indication_rel8.ul_cqi = 0;
14     else if (SNRtimes10 > 635)
15         pdu->rx_indication_rel8.ul_cqi = 255;
16     else
17         pdu->rx_indication_rel8.ul_cqi = (640 + SNRtimes10) / 5;
18 }
19 .....
20 void fill_uci_harq_indication (PHY_VARS_eNB *eNB, LTE_eNB_UCI *
21     ↳uci, int frame, int subframe, uint8_t *harq_ack, uint8_t
22     ↳tdd_mapping_mode, uint16_t tdd_multiplexing_mask)
23 {
24     .....
25     // estimate UL_CQI for MAC (from antenna port 0 only)
26     pdu->ul_cqi_information.tl.tag = NFAPI_UL_CQI_INFORMATION_TAG;
27     int SNRtimes10 = dB_fixed_times10(uci->stat) - 10 * eNB->
28         ↳measurements.n0_subband_power_dB[0][0];
29
30     if (SNRtimes10 < -100)
31         LOG_I (PHY, "uci->stat %d \n", uci->stat);
32
33     if (SNRtimes10 < -640)
34         pdu->ul_cqi_information.ul_cqi = 0;
35     else if (SNRtimes10 > 635)
36         pdu->ul_cqi_information.ul_cqi = 255;
37     else
38         pdu->ul_cqi_information.ul_cqi = (640 + SNRtimes10) / 5;
39
40     pdu->ul_cqi_information.channel = 0;
41 }

```

The SNR is so converted, that ten times the SNR will be restraint in the `ul_cqi` range from 0 to 255. The value, which is smaller than -640 db , will be assigned to be 0 and is considered as poor channel quality. The value, which is bigger than 635 db will be assigned to be 255, and is considered as transmission dominated and high energy consumption. The other values will be then linearly converted in the range from 0 to 255. The `ul_cqi` is then transferred to the MAC layer in the `eNB_scheduler_ulsch.cin` in the file path `openairinterface5g ↪ /openair2/LAYER2/MAC` for assuming the accumulated transmission power control value (tpc). The `ul_cqi`, which is bigger than 200, will also be considered as too high energy pattern here for assuming the tpc. For a SISO channel measurement, we actually don't need the ability of scheduling, on the contrary, we need a constant channel quality index for every subframe of the single uplink channel. So, we have assigned all the values, which in the range from 0 to 255 to be 150, like this `pdu->ul_cqi_information.ul_cqi = 150` in the listed codes above. Actually, there is also a function called `fill_sr_indication ↪()`, which contains the `ul_cqi` computation operations like the listed codes above. However, we didn't activate the sounding reference signal by using the eNB configuration file. So, this function can be ignored here.

Then we have rebuilt the eNB, and reboot the system. We have used the same method for extracting the uplink channel estimates and visualized it in MATLAB, which can be found in Figure 4.10. In this measurement, the allocated bandwidth of the uplink channel is constrained to 3 RBs, which are 36 subcarriers. So, there is much less pressure for the processor to calculate channel estimates in a time symbol. We have chosen 240 symbols of the hole measurement and visualized, we can see that the channel is now much better than the previous one in Figure 4.8. With a more narrower bandwidth, for achieving the same transmission speed as the wide-bandwidth channel, the powers or envelopes on each resource element will have higher magnitude level.

Identically, we have chosen a random subcarrier of the frequency domain and plot the magnitude of channel estimates from 200 symbols in the time domain, we can see the output in Figure 4.11. Now the magnitude of each channel estimates are not continuously near to zero and the cliff-like changes in Figure 4.9 has disappeared. Empirically, we have extracted the correct channel estimates initially. More tests should still be performed. Here we will check these measurements in a channel prediction algorithm, to check whether the algorithms can well perform on the real data from channel measurement. These works will be illustrated in the next chapter.

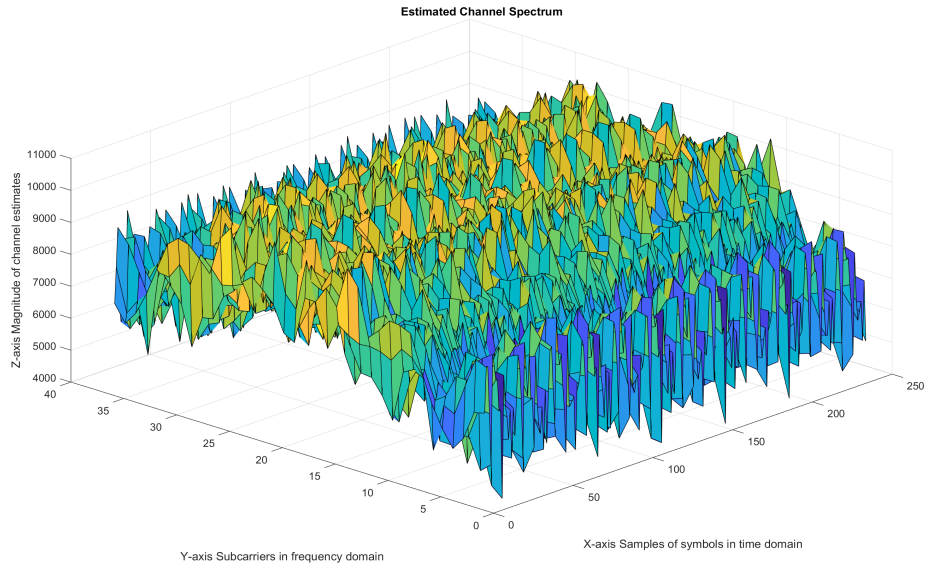


Figure 4.10: Estimated channel spectrum after having changed the scheduling

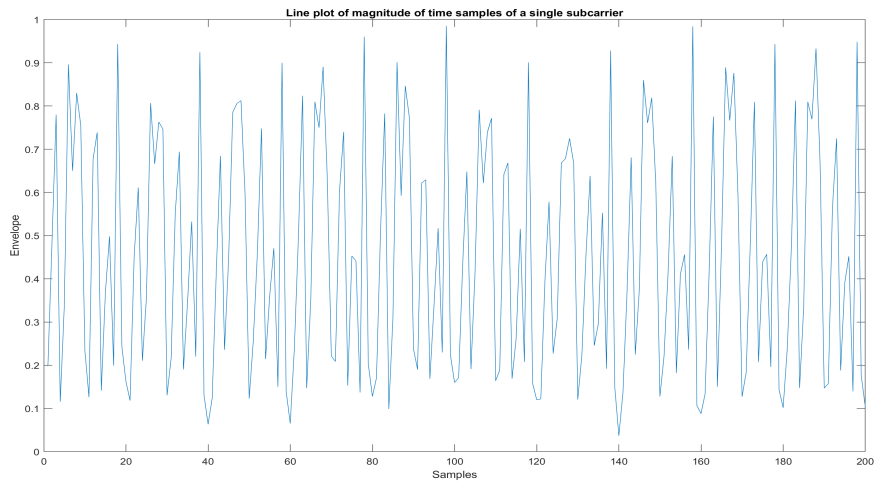


Figure 4.11: Estimated channel spectrum of a single subcarrier after having changed the scheduling

Verification of Channel Prediction Algorithm

The main motivation of this thesis is to provide a real LTE communication platform so that some new proposed artificial intelligent next-generation wireless algorithms can be tested on the real channel parameters. One of the algorithms is RNN based channel prediction [19, 18, 20]. Outdated CSI between the transmitter and receiver can badly deteriorate the performance of a wireless communication system. The channel prediction provides an efficient approach by improving the quality of CSI directly without spending extra wireless resources, and therefore attracts much attention from researchers [9]. The wireless community started to apply the techniques of AI to solve communication problems a long time ago. Taking advantage of the functionality of time-series prediction of Neural Network (NN), a predictor for narrow-band channel [24] and its extensions for the MIMO channel [8] have been proposed. The main reason for using the NN structure for channel prediction is that it is analogous to a time-varying infinite impulse response (IIR) filter, which well suits the nonlinearity of the fading channel. Without any prior knowledge or assumptions, only a number of past channel states can train a NN-based predictor, which in turn eliminates the gap between modeling and reality.

In the rest of this chapter, we will first discuss the RNN in section 5.1 to find the superiority of RNN on the channel prediction. In section 5.2 we will briefly introduce the RNN structure for channel prediction. In section 5.3 we will finally test the algorithm on the data from the measurement on our implemented platform, and some comparisons and evaluations will then depicted.

5.1 RNN

A recurrent neural network (RNN) is a class of artificial intelligent neural networks where connections between nodes form a directed graph along a temporal sequence. This allows it to exhibit temporal dynamic behavior. Derived from feed-forward neural networks, RNNs can use their internal state (memory) to process variable-length sequences of inputs. This makes them applicable to tasks such as unsegmented, connected handwriting recognition or speech recognition. The term "recurrent neural network" is used indiscriminately to refer to two broad classes of networks with a similar general structure, where one is finite impulse and the other is infinite impulse. Both classes of networks exhibit temporal dynamic behavior. A finite impulse recurrent network is a directed acyclic graph that can be unrolled and replaced with a strictly feedforward neural network, while an infinite impulse recurrent network is a directed cyclic graph that can not be unrolled. Both finite impulse and infinite impulse recurrent networks can have additional stored states, and the storage can be under direct control by the neural network. The storage can also be replaced by another network or graph if that incorporates time delays or has feedback loops. Such controlled states are referred to as gated state or gated memory and are part of long short-term memory networks and gated recurrent units. This is also called Feedback Neural Network.

RNNs come in many variants. Basic RNNs are a network of neuron-like nodes organized successive "layers." Each node in a given layer is connected with a directed (one-way) connection to every other node in the next successive layer. Each node (neuron) has a time-varying real-valued activation. Each connection (synapse) has a modifiable real-valued weight. Nodes are either input nodes (receiving data from outside the network), output nodes (yielding results), or hidden nodes (that modify the data en route from input to output).

For supervised learning in discrete time settings, sequences of real-valued input vectors arrive at the input nodes, one vector at a time. At any given time step, each non-input unit computes its current activation (result) as a nonlinear function of the weighted sum of the activations of all units that connect to it. Supervisor-given target activations can be supplied for some output units at certain time steps. For example, if the input sequence is a speech signal corresponding to a spoken digit, the final target output at the end of the sequence may be a label classifying the digit.

In reinforcement learning settings, no teacher provides target signals. Instead, a fitness function or reward function is occasionally used to evaluate the RNN's performance, which influences its input stream through output units connected to actuators that affect the environment. This might be used to play a game in which progress is measured with the number of points won.

Each sequence produces an error as the sum of the deviations of all target signals from the corresponding activations computed by the network. For a training set of numerous sequences, the total error is the sum of the errors of all individual sequences.

The most important part of building an efficient deep learning architecture is the parameter tuning in the training process. Gradient descent is a first-order iterative optimization algorithm for finding the minimum of a function. In neural networks, it can be used to minimize the error term by changing each weight in proportion to the derivative of the error with respect to that weight, provided the non-linear activation functions are differentiable. A major problem with gradient descent for standard RNN architectures is that error gradients vanish exponentially quickly with the size of the time lag between important events.[14, 15]. Long short-term memory (LSTM) combined with backpropagation through time (BPTT)/real-time recurrent learning (RTRL) hybrid learning method attempts to overcome these problems. [16] This problem is also solved in the independently recurrent neural network (IndRNN)[23] by reducing the context of a neuron to its own past state and the cross-neuron information can then be explored in the following layers. Memories of the different range including long-term memory can be learned without the gradient vanishing and exploding problem.

Training the weights in a neural network can be modeled as a non-linear global optimization problem. A target function can be formed to evaluate the fitness or error of a particular weight vector as follows: First, the weights in the network are set according to the weight vector. Next, the network is evaluated against the training sequence. Typically, the sum-squared-difference between the predictions and the target values specified in the training sequence is used to represent the error of the current weight vector. Arbitrary global optimization techniques may then be used to minimize this target function.

5.2 RNN for Channel Prediction

RNN is a popular machine learning technique that has shown great potential in time-series prediction tasks. The authors of these papers [24, 8] have proposed to use a RNN to build a predictor for narrow-band and wide-band wireless communication channels. In [19, 18, 20], the authors of this paper proposed to apply a real-valued RNN to implement a multi-step MIMO channel predictor and further illustrated its achievable performance in a multi-antenna system. All those papers has proved the outstanding performance of the proposed algorithms on simulated communication data. However, none of them has tested the algorithms on a real communication platform.

We will verify the algorithm proposed in [19] first as an ending of this thesis. As is shown in Figure 5.1, the RNN consists of three layers: a hidden layer with N_L neurons, an output layer having N_o outputs, which is represented by a vector $y_o = [y_1, \dots, y_{N_o}]^T$, and an input layer with N neurons including N_I external inputs $x_e = [x_1, \dots, x_{N_I}]^T$ and N_o feedback inputs $y(t+1) = [y_1(t+1), \dots, y_{N_o}(t+1)]$. At the time instant of t , the external inputs are represented by the vector $x_e = [x_1(t), \dots, x_{N_I}(t)]^T$. So the total input vector can be combined by the external input vector and delayed feedback vector $y(t)$, which is written as $X = [x_1(t), \dots, x_{N_I}(t), y_1(t), \dots, y_{N_o}(t)]$. Every connection between the neurons are assigned with weights, which can determine the behaviour of a RNN. Here the $w_{l,n}$ denotes the real-valued weight connecting the l^{th} hidden neuron and the n^{th} input, while $c_{m,l}$ is the weight for connecting output m and hidden neuron l .

The transfer function typically falls into one of three categories: linear, threshold and sigmoid. To deal with the nonlinearity of a fading channel, the sigmoid function is chosen in each hidden neuron, which is given by:

$$S(x) = \frac{1}{1 + e^{-x}} \quad (5.1)$$

Then we can get the output of the l^{th} hidden neurons:

$$\gamma_l(t) = S(w_l \cdot X) = S\left(\sum_{n=1}^{N_I} w_{l,n}x_n(t) + \sum_{n=N_I+1}^N w_{l,n}y_{(n-N_I)}(t)\right), \quad (5.2)$$

where $w_l \cdot X$ is the dot product of the input vector X and the l^{th} weight vector $w_l = [w_{l,1}, w_{l,2}, \dots, w_{l,N}]$. Given the output neuron with a linear transfer function,

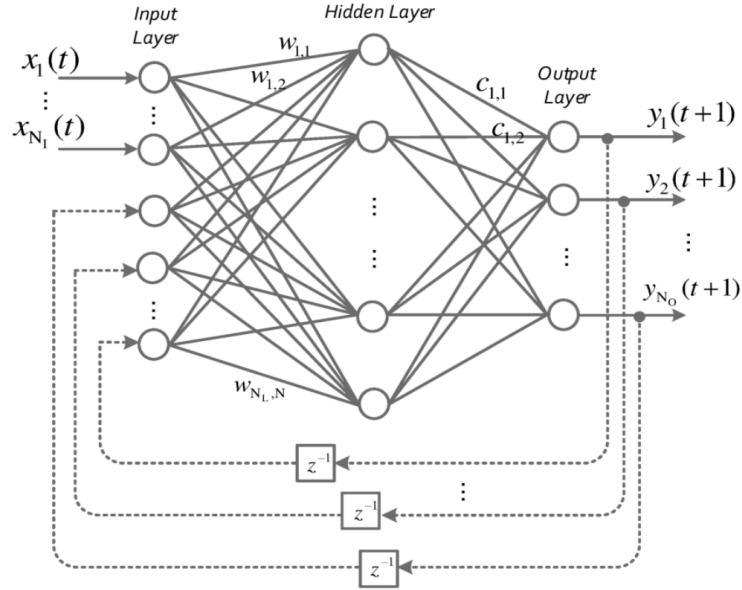


Figure 5.1: Schematic diagram of a recurrent neural network [19]

the m^{th} predicted output can be expressed as

$$y_m(t+1) = \sum_{l=1}^{N_L} c_{n,l} \gamma_l(t). \quad (5.3)$$

Based on the RNN structure, the channel prediction can be applied. In a MIMO channel with N_t transmit antennas and N_r receiver antennas, the channel can be modeled as

$$\mathbf{r}(t) = \mathbf{H}(t)\mathbf{s}(t) + \mathbf{n}(t), \quad (5.4)$$

where $\mathbf{r}(t)$ represents the $N_r \times 1$ received vector at time t , $\mathbf{s}(t)$ is the $N_t \times 1$ transmit symbol vector, $\mathbf{n}(t)$ stands for the vector of noise and \mathbf{H} is the matrix of time-varying channel coefficients. Here we take the channel estimates as the coefficients, which is represented by a complex-valued variable h_{n_r, n_t} . Owing to the processing and feedback delay, the CSI at the time of selecting transmission parameters may substantially differ from the CSI at the instant of using the selected parameters to transmit, i.e., $\mathbf{H}(t) \neq \mathbf{H}(t + \tau)$ where τ denotes the delay. The outdated CSI imposes a severely negative impact on a wide variety of wireless techniques. The task of channel prediction is to predict a channel state $\hat{\mathbf{H}}(t + \tau)$ that is as close as possible to the actual upcoming state $\mathbf{H}(t + \tau)$.

In the paper [19], the author has firstly used the single-antenna case to shed

light on how the RNN predicts envelope and real/imaginary parts of the channel estimates, and at last, has extended to forecast the channel state multiple steps ahead. In our implementation, we can also provide the same form of data, so that the algorithm can first be trained and then tested on the rest data.

If we predict the envelope of a channel, in the time t , the channel envelope is already known, while a number of d past values $|h(t-1)|, |h(t-2)|, \dots, |h(t-d)|$ can be kept through a tapped delay line. So the external input $x_e(t)$ will be the vector $|h(t)|, |h(t-1)|, |h(t-2)|, \dots, |h(t-d)|$. Additionally, the delayed feedback $|\hat{h}(t)|$ will help the prediction of channel envelope $|\hat{h}(t+1)|$.

If we predict the complex-valued channel coefficients, the channel coefficient $h(t)$ can be written as

$$h(t) = h^r(t) + ih^i(t), \quad (5.5)$$

where h^r and h^i are the real and imaginary parts. In this case, the external inputs can be written as $x_e(t) = [h^r(t), h^i(t), \dots, h^r(t-d), h^i(t-d)]$. By combining the predicted real and imaginary parts, the channel coefficient for the next time instant can be obtained as $\hat{h}(t+1) = \hat{h}^r(t+1) + i\hat{h}^i(t+1)$.

For a multi-step prediction, the output of the channel prediction at time t is changed from $\hat{h}(t+1)$ to $\hat{h}(t+D)$, where D stands for the number of steps being predicted ahead.

5.3 Numerical results and evaluations

Now we can test the algorithm on our measured channel estimates. First of all, the data should be normalized. In the field of machine learning, different evaluation indicators (the different features in the feature vector are the different evaluation indicators described) often have different dimensions and dimensional units. This situation will affect the results of data analysis. To eliminate indicators' dimensional impact, we need to standardize the data to solve the comparability between data indicators. After the standardization of the original data, all indicators are in the same order of magnitude, which is suitable for comprehensive comparative evaluation. Among them, the most typical is the normalization of data. In short, the purpose of normalization is to limit the preprocessed data to a certain range, thereby eliminating the adverse effects caused by the singular sample data.

Some common normalization methods are

- **Min-Max Normalization:** maps the value in the range of $[0, 1]$, the transformation function is $x' = \frac{x - \min(x)}{\max(x) - \min(x)}$. This normalization method is more suitable for situations where the values are relatively concentrated. If max and min are unstable, it is easy to make the normalization result unstable and make the subsequent use effect unstable. In practice, empirical constants can be used instead of max and min.
- **Z-score:** Processed data conforms to standard normal distribution. The transformation function is $x^* = \frac{x - \mu}{\sigma}$, where μ is the mean value of the data and σ is the standard deviation. This method requires that the distribution of the original data can be approximated as a Gaussian distribution, otherwise the effect of normalization will become very bad.
- **Nonlinear normalization:** The method includes logarithmic, tangent, etc., and needs to determine the curve of the non-linear function according to the data distribution. This normalization method is often used in scenarios where data differentiation is relatively large. It maps the original values with some mathematical functions.

In [19], the author has normalized the envelope and the real part of the complex value of the channel estimates using Z-score normalization because the channel is assumed to be a flat-fading Rayleigh channel with an average power gain of $0dB$, where the channel coefficient h is zero-mean circularly-symmetric complex Gaussian random variable with the variance of 1, i.e., $h \sim \mathcal{CN}(0, 1)$. The results of this paper have shown a great performance of the channel prediction algorithm. For example in Figure 5.2, the channel prediction algorithm has predicted the channel envelope marked by red dots quite match the actual values denoted by a curve with high accuracy of Mean Square Error (MSE) $\sigma^2 = 4.97 \times 10^{-6}$. Some other extensions like the real/imaginary part prediction, multi-antenna channel prediction, multi-step channel prediction also have shown great results. In our case, we have extracted the complex-valued channel estimates not on every symbol, but on every symbol position of the reference signal. So we can predict the channel estimates with an interval of 7 symbols. The channel estimates are normalized into the range of $[0, 1]$. After we have performed the channel prediction algorithm, the outputs can be seen in Figure 5.3, which has a accuracy of Mean Square Error (MSE) $\sigma^2 = 0.0023$. Although the result is much worse than the prediction accuracy on simulated data, the accuracy of the prediction has verified that the

real channel estimates can also be predicted by this algorithm. Some efforts of optimizations are still essential for higher prediction accuracy.

Since the author of [19] has utilized the Levenberg-Marquardt [11] backpropagation algorithm, the number of neurons in the hidden layer can be increased from 10 to 100, which increase the prediction accuracy to $\sigma^2 = 0.0131$. The new release of the MATLAB deep learning toolbox has provided various useful solver of neural networks.

In this thesis, we have additionally tried the 'adam' solver in the deep learning toolbox, which helps the neural network training progress by flexibly setting the number of learning epochs, iteration numbers, learning rate and the number of neurons. Referred to the monitoring of the training progress like in Figure 5.4, we can learn how the training is progressing. For example, we can determine if and how quickly the network accuracy is improving, and whether the network is starting to overfit the training data. By analyzing the curve of training progress, we can alter the parameters of the neural network, mainly the number of epochs and the learning rate.

After some tries, we have achieved an accuracy of the channel prediction with MSE $\sigma^2 = 0.0023$, by setting 500 learning epochs, the first 450 epochs with learning rate $\eta = 0.005$ and the last 50 epochs with learning rate $\eta = 0.001$. The channel prediction result is shown in Figure 5.5. In order to better visualize the fluctuation trend, we have drawn the line-dot-line for depicting the predicted value. So, we can see that the marked predicted value can already match the actual measured value well with acceptable MSE. This sample prediction is equivalent to the 7 symbols ahead prediction, which has a considerable accuracy compared to the 5 symbols ahead prediction in [19]. All those outputs have verified that our experimentation platform can be used for channel measurements and the measured channel estimates can be predicted by this channel prediction algorithm. Now, we have finally finished all the tasks of this thesis.

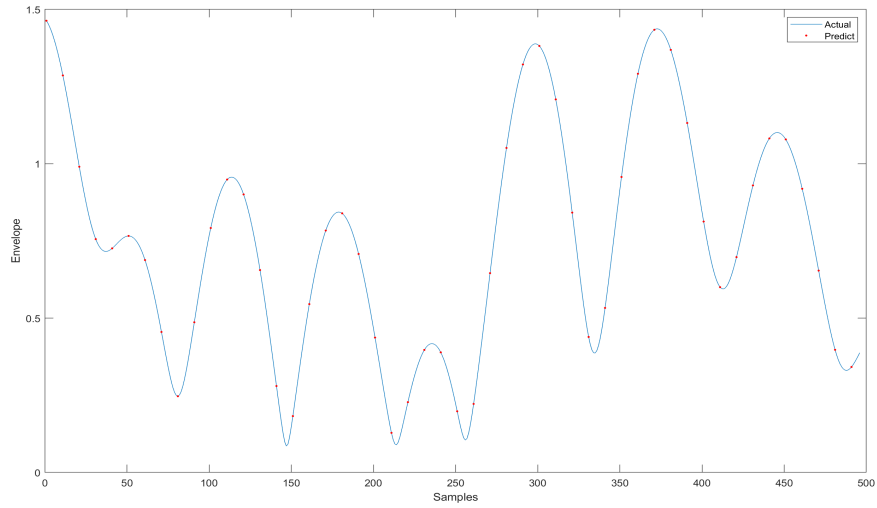


Figure 5.2: Channel prediction on the symbol continuous envelope of simulated channel estimates[19]

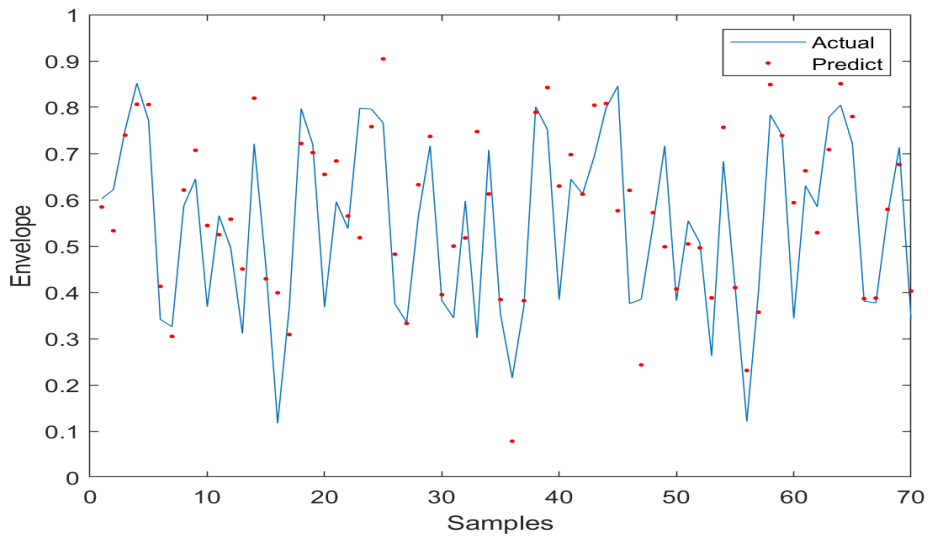


Figure 5.3: Channel prediction on the envelope of measured channel estimates

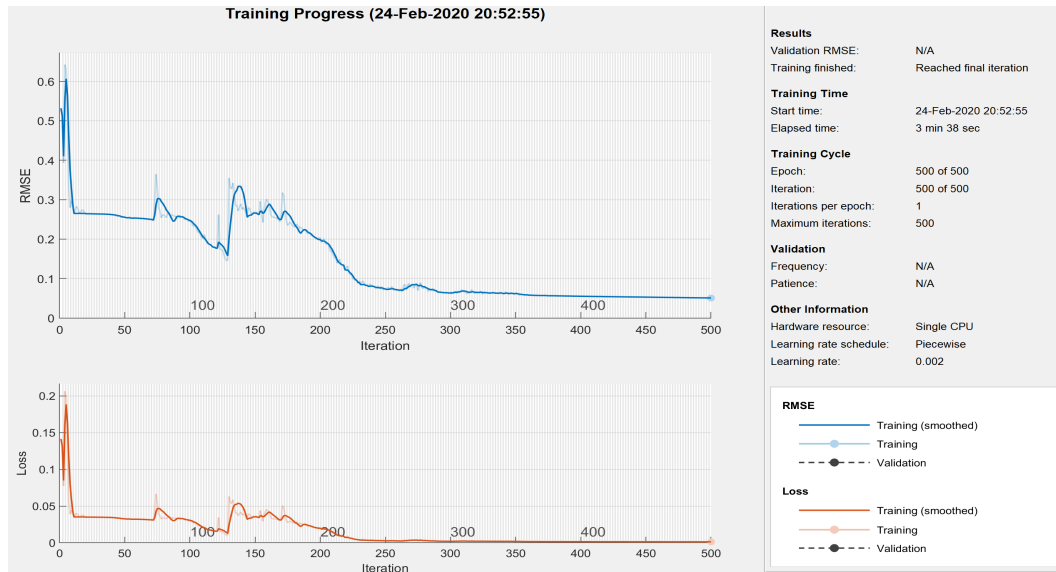


Figure 5.4: Channel prediction algorithm training progress

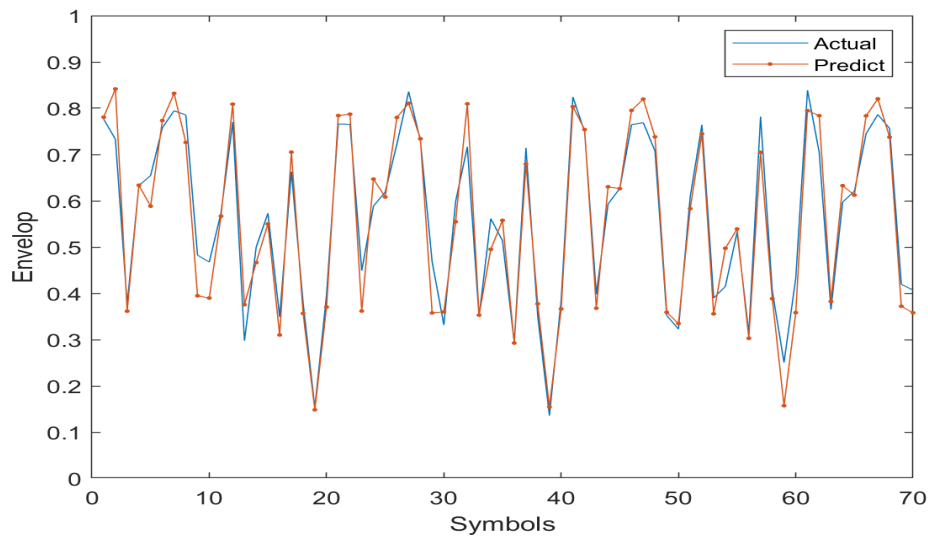


Figure 5.5: Channel prediction on the envelope of measured channel estimated using adam solver

Conclusion and Future Works

6.1 Conclusion

Our thesis has aimed to build a real LTE-access experimentation platform for performing the channel measurement, so that we can verify the AI-based next-generation wireless transmission algorithms. All the set tasks have been perfectly met. The OAI EPC and eNB are successfully installed on two independent GPPs. The driver of USRP is configured on the GPP, which runs the OAI eNB. With a correct configuration file, the eNB can successfully build the communication with USRP and transfer the baseband signal to it for broadcasting. A for research purpose configured SIM card is new encrypted for the commercial UE (smartphone). After some configuration works of the EPC serving database and smartphone network, the smartphone ID can be successfully authenticated by the EPC serving database and can get the LTE access, which is broadcast by the USRP-based eNB. Employing the routing bridge implemented in EPC, the smartphone can surf the internet without any problem.

Motivated by the performance of the channel prediction algorithms on the channel response collected on the simulated wireless channel, we have firstly oriented our direction on real-timely measuring the channel estimates based on the LTE signal transmission. In consideration of our platform setup, we have chosen the uplink channel for research. That is, we got the channel estimates on the receiver side of the uplink channel, in our case is the eNB side. After some theoretical researches on the uplink frame structure, uplink shared channel and uplink channel scheduling by having referred the 3GPP standards and the LTE principles, we have known the functionalities of the reference signal, which can be used for channel response estimation and signal demodulation. By changing the uplink channel scheduling,

the bandwidth of the uplink channel can be restrained and made constant. In our implementation, channel estimation on a relative narrower bandwidth will relieve the computation pressure of the processor, which computes the channel estimates real-timely on the symbol positions of reference signals. Because the reference signals only exist on one symbol of each time slot, so we can real-timely get the complex values of channel estimated of all the allocated subcarriers in the frequency domain on every 7 symbols in the time domain without any latency problem.

The extracted channel estimates will be formed as the time-frequency format. Then we have normalized the channel estimates into the range of $[0, 1]$ for solving the comparability between data indicators, which is appropriate for comprehensive comparative evaluation. The normalized data was been directly tested on the algorithm in [19], which has shown an acceptable accuracy on channel prediction, but the performance has weakened itself a lot compared to the performance on the simulated data. So we have further optimized the algorithm with more hidden neurons and tried other solvers for the network training progress. Moreover, the learning rate and the number of epochs are set regarding the MSE curve of training progress for algorithm optimization. Through our efforts, we have improved the prediction accuracy by the factor of 10 on the real measured channel estimates. More importantly, the proposed predictor has the flexibility of conducting multi-step prediction and is more robust against interpolation errors.

In conclusion, the SDR platform is a really powerful solution for wireless transmission algorithm verification, which offers extensive advantages and features for individual research purposes. With the aid of open-source code, we can easily change the code for a certain task. Our implemented OAI and USRP-based experimentation platform has been proved to be powerful enough for some research and verification purposes, which can also be extended for other tasks. That will be introduced in the next section.

6.2 Future Works

Concerning this work, some following approaches are possible in the future. Because the time is limited, we have only build a SISO communication for testing the algorithm. We have utilized this SDR platform for the verification of SISO channel

prediction. In [19], the channel prediction algorithm has also shown great performance on the MIMO communication channel. OAI provides also the multi-antenna ability, which needs another USRP connected to another GPP installed with OAI UE instead of commercial UE.

Later, we will build a USRP-based OAI UE for connecting the USRP-based OAI eNB. A single USRP device can provide 2 input and 2 output antenna ports. So two USRPs on the eNB and UE sides can consist of a 2×2 MIMO communication. If more USRPs are put into use, the OAI at present can support up to 16×16 MIMO communication. Besides, OAI NR has been developed, which can provide the 5G full network connection. Faced with such a wide-band channel, we have to take deep researches on how to measure the channel estimates on a wide-band channel, since we have only achieved a narrow-band channel measurement.

The improvements of the channel prediction accuracy on the real data have motivated us to continue the research of this algorithm for a much preciser prediction result. Other advanced AI techniques may provide a greater potential in channel prediction, which is worth taking efforts to explore in the next step. It would be also meaningful, that we transplant and embed this prediction algorithm in the source code of the OAI platform to enhance communication efficiency, which is also a contribution to the OAI developer community.

Another aspect is given, that we will not only focus on the proposed research achievement on the physical layer but also look beyond on other algorithms like the network management [21], time-sensitive networking on other communication layers in the future works. Most of our works in this thesis are irrelevant to the other OAI implemented communication layers out of the physical layer. The test-bed experimented with in this thesis should make it possible to extend the solution to those. Different types of setup can also be tried in the future.

Bibliography

- [1] ABIDI, Asad A.: The path to the software-defined radio receiver. In: *IEEE Journal of solid-state circuits* 42 (2007), Nr. 5, S. 954–966
- [2] ACCESS, Evolved Universal Terrestrial R.: *Multiplexing and Channel Coding (Release 10)*, 3GPP TS 36.212 V10. 1.0. 2009
- [3] ACCESS, Evolved Universal Terrestrial R.: Physical channels and modulation, 3GPP TS 36.211. In: *V10 2* (2009)
- [4] BHUSHAN, Naga ; LI, Junyi ; MALLADI, Durga ; GILMORE, Rob ; BRENNER, Dean ; DAMNJANOVIC, Aleksandar ; SUKHAVASI, Ravi T. ; PATEL, Chirag ; GEIRHOFER, Stefan: Network densification: the dominant theme for wireless evolution into 5G. In: *IEEE Communications Magazine* 52 (2014), Nr. 2, S. 82–89
- [5] CHIN, Woon H. ; FAN, Zhong ; HAINES, Russell: Emerging technologies and research challenges for 5G wireless networks. In: *IEEE Wireless Communications* 21 (2014), Nr. 2, S. 106–112
- [6] CHU, David: Polyphase codes with good periodic correlation properties (corresp.). In: *IEEE Transactions on information theory* 18 (1972), Nr. 4, S. 531–532
- [7] DARDAILLON, Mickaël ; MARQUET, Kevin ; MARTIN, Jérôme ; RISSET, Tanguy ; CHARLES, Henri-Pierre: Cognitive radio programming: Existing solutions and open issues. (2013)
- [8] DING, Tianben ; HIROSE, Akira: Fading channel prediction based on combination of complex-valued neural networks and chirp Z-transform. In: *IEEE Transactions on Neural Networks and Learning Systems* 25 (2014), Nr. 9, S. 1686–1695

- [9] DUEL-HALLEN, Alexandra: Fading channel prediction for mobile radio adaptive transmission systems. In: *Proceedings of the IEEE 95* (2007), Nr. 12, S. 2299–2313
- [10] ETSI, TS: Physical layer procedures, 3GPP TS 36.213 version 11.3. 0 Release 11. In: *LTE.,(Jul. 2013)* 178 (2013)
- [11] FU, Xingang ; LI, Shuhui ; FAIRBANK, Michael ; WUNSCH, Donald C. ; ALONSO, Eduardo: Training recurrent neural networks with the Levenberg–Marquardt algorithm for optimal control of a grid-connected converter. In: *IEEE transactions on neural networks and learning systems* 26 (2014), Nr. 9, S. 1900–1912
- [12] HAMPTON, Jerry R.: *Introduction to MIMO communications*. Cambridge university press, 2013
- [13] HANDBOOK, LTE: Share Technote. In: *first published on or about Jul 13* (2012)
- [14] HOCHREITER, Sepp: Untersuchungen zu dynamischen neuronalen Netzen. In: *Diploma, Technische Universität München* 91 (1991), Nr. 1
- [15] HOCHREITER, Sepp ; BENGIO, Yoshua ; FRASCONI, Paolo ; SCHMIDHUBER, Jürgen u. a.: *Gradient flow in recurrent nets: the difficulty of learning long-term dependencies*. 2001
- [16] HOCHREITER, Sepp ; SCHMIDHUBER, Jürgen: Long short-term memory. In: *Neural computation* 9 (1997), Nr. 8, S. 1735–1780
- [17] HYADI, Amal ; REZKI, Zouheir ; ALOUINI, Mohamed-Slim: An overview of physical layer security in wireless communication systems with CSIT uncertainty. In: *IEEE Access* 4 (2016), S. 6121–6132
- [18] JIANG, Wei ; SCHOTTEN, Hans D.: Recurrent neural network-based frequency-domain channel prediction for wideband communications. In: *2019 IEEE 89th Vehicular Technology Conference (VTC2019-Spring)* IEEE, 2019, S. 1–6
- [19] JIANG, Wei ; SCHOTTEN, Hans D.: Multi-antenna fading channel prediction empowered by artificial intelligence. In: *2018 IEEE 88th Vehicular Technology Conference (VTC-Fall)* IEEE, 2018, S. 1–6
- [20] JIANG, Wei ; SCHOTTEN, Hans D.: A comparison of wireless channel predictors: Artificial Intelligence versus Kalman filter. In: *ICC 2019-2019 IEEE International Conference on Communications (ICC)* IEEE, 2019, S. 1–6



- [21] JIANG, Wei ; STRUFE, Mathias ; SCHOTTEN, Hans D.: Intelligent network management for 5G systems: The SELFNET approach. In: *2017 European Conference on Networks and Communications (EuCNC)* IEEE, 2017, S. 1–5
- [22] LARSSON, Erik G. ; EDFORS, Ove ; TUFVESSON, Fredrik ; MARZETTA, Thomas L.: Massive MIMO for next generation wireless systems. In: *IEEE communications magazine* 52 (2014), Nr. 2, S. 186–195
- [23] LI, Shuai ; LI, Wanqing ; COOK, Chris ; ZHU, Ce ; GAO, Yanbo: Independently recurrent neural network (indrnn): Building a longer and deeper rnn. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, S. 5457–5466
- [24] LIU, Wei ; YANG, Lie-Liang ; HANZO, Lajos: Recurrent neural network based narrowband channel prediction. In: *2006 IEEE 63rd Vehicular Technology Conference* Bd. 5 IEEE, 2006, S. 2173–2177
- [25] MOCKAPETRIS, Paul u. a.: Domain names-implementation and specification. (1987)
- [26] NIKAEIN, Navid ; KNOPP, Raymond ; KALTENBERGER, Florian ; GAUTHIER, Lionel ; BONNET, Christian ; NUSSBAUM, Dominique ; GHADDAB, Riadh: OpenAirInterface: an open LTE network in a PC. In: *Proceedings of the 20th annual international conference on Mobile computing and networking*, 2014, S. 305–308
- [27] NIKAEIN, Navid ; MARINA, Mahesh K. ; MANICKAM, Saravana ; DAWSON, Alex ; KNOPP, Raymond ; BONNET, Christian: OpenAirInterface: A flexible platform for 5G research. In: *ACM SIGCOMM Computer Communication Review* 44 (2014), Nr. 5, S. 33–38
- [28] PARADISI, Alberto ; YACCOUB, Michel D. ; FIGUEIREDO, Fabrício Lira ; TRONCO, Tania: *Long Term Evolution: 4G and Beyond*. Springer, 2015
- [29] PAWELCZAK, Przemyslaw ; NOLAN, Keith ; DOYLE, Linda ; OH, Ser W. ; CABRIC, Danijela: Cognitive radio: Ten years of experimentation and development. In: *IEEE Communications Magazine* 49 (2011), Nr. 3, S. 90–100
- [30] POPOVIC, Branislav M.: Generalized chirp-like polyphase sequences with optimum correlation properties. In: *IEEE Transactions on Information Theory* 38 (1992), Nr. 4, S. 1406–1409



- [31] RAMYA, TR ; BHASHYAM, Srikrishna: Using delayed feedback for antenna selection in MIMO systems. In: *IEEE Transactions on Wireless Communications* 8 (2009), Nr. 12, S. 6059–6067
- [32] TENG, Yinglei ; LIU, Mengting ; SONG, Mei: Effect of outdated CSI on handover decisions in dense networks. In: *IEEE Communications Letters* 21 (2017), Nr. 10, S. 2238–2241
- [33] TRUONG, Kien T. ; HEATH, Robert W.: Effects of channel aging in massive MIMO systems. In: *Journal of Communications and Networks* 15 (2013), Nr. 4, S. 338–351
- [34] ULVERSOY, Tore: Software defined radio: Challenges and opportunities. In: *IEEE Communications Surveys & Tutorials* 12 (2010), Nr. 4, S. 531–550
- [35] VICARIO, Jose L. ; BEL, Albert ; LOPEZ-SALCEDO, Jose A. ; SECO, Gonzalo: Opportunistic relay selection with outdated CSI: outage probability and diversity analysis. In: *IEEE Transactions on Wireless Communications* 8 (2009), Nr. 6, S. 2872–2876
- [36] WANG, Renyuan ; PENG, Yuexing ; QU, Huan ; LI, Wenfang ; ZHAO, Hui ; WU, Bin: OpenAirInterface-An effective emulation platform for LTE and LTE-Advanced. In: *2014 Sixth International Conference on Ubiquitous and Future Networks (ICUFN)* IEEE, 2014, S. 127–132
- [37] XIONG, Xiong ; XIANG, Wei ; ZHENG, Kan ; SHEN, Hengyang ; WEI, Xingguang: An open source SDR-based NOMA system for 5G networks. In: *IEEE Wireless Communications* 22 (2015), Nr. 6, S. 24–32
- [38] YU, Xiangbin ; XU, Weiye ; LEUNG, Shu-Hung ; WANG, Jiang: Unified performance analysis of transmit antenna selection with OSTBC and imperfect CSI over Nakagami-m fading channels. In: *IEEE Transactions on Vehicular Technology* 67 (2017), Nr. 1, S. 494–508