TREBALL DE FI DE CARRERA

TÍTOL DEL TFC : Acceleració SDR de la cadena de processat de la capa física de LTE downlink mitjançant GPUs

TITULACIÓ: Doble titulació en Enginyeria Tècnica de Telecomunicació, especialitat Sistemes de Telecomunicació i Enginyeria Tècnica Aeronàutica, especialitat Aeron-avegació

AUTOR: Xavier Arteaga Martínez

DIRECTOR: Antoni Gelonch Bosch

DATA: July 7, 2012
Titol: Acceleració SDR de la cadena de processat de la capa física de LTE downlink mitjançant GPUs

Autor: Xavier Arteaga Martínez

Director: Antoni Gelonch Bosch

Data: July 7, 2012

Resum

La tecnologia avança molt ràpid, i cada vegada més en el desenvolupament de sistemes sense fils es tendeix al disseny de sistemes ràdio definits per software (SDR). Els nous estàndards de comunicacions sense fils incrementen la seva eficiència però també la seva complexitat, la qual demanda més processament.

El projecte FlexNets és una iniciativa per a la investigació en SDR. La missió de FlexNets és proporcionar les eines necessaries per al desenvolupament de SDR d’una forma senzilla i flexible.

Durant els darrers anys, els processadors gráfics (GPUs) han evolucionat, s’han estès en el mercat de les aplicacions de gran cost computacional. Les GPUs tenen un preu reduït i són de baix consum per a una alta capacitat computacional. Per tant, resulta força interessant estudiar si aquests processadors són aptes per a la aplicació en SDR.

Aquest treball està orientat a l’anàlisi de la viabilitat d’aquesta tecnologia per implementar sistemes ràdio basats en SDR sobre la plataforma de FlexNets.
Overview

The technology moves fast and the wireless systems tend to be software defined radio (SDR). The new wireless standards increase the efficiency of communications, also its complexity, which demand more processing.

The FlexNets is an open source project that explores the new possibilities of flexible radio communications. It provides some tools to develop SDR.

In recent years, graphical processors have evolved and expanded to the market of high performance computing. These processors (GPUs) are cheaper and consume less power per floating point operation than classical CPUs. It is therefore quite interesting to study whether these processors are suitable for application in the SDR.

This work is aimed at analyzing the feasibility of this technology to implement systems based on SDR radio.
To my family and friends
CONTENTS

INTRODUCTION .................................................. 1

CHAPTER 1. State of art ....................................... 3
  1.1. Introduction to Software Defined Radio ................. 3
  1.2. FlexNets project ........................................ 4
      1.2.1. ALOE processing layers and architecture .......... 4
      1.2.2. Taking advantage of multiprocessor platform ... 6
      1.2.3. ALOE module organization ....................... 6
      1.2.4. ALOE User Interface .............................. 7
  1.3. Long Term Evolution Downlink Physical Layer .......... 8
      1.3.1. LTE radio frame structure ....................... 8
      1.3.2. Spectrum configurations ......................... 9
      1.3.3. LTE processing chain ............................ 9
      1.3.4. Synchronization Signals ......................... 10
  1.4. Compute Unified Device Architecture .................... 11
  1.5. Working platform .................................... 12
  1.6. Scope of this work .................................. 13

CHAPTER 2. Case study ....................................... 15
  2.1. Bit level processing implementation .................... 15
      2.1.1. Turbo decoder implementation .................. 17
      2.1.2. Bit level configuration ......................... 18
  2.2. Symbol level processing implementation............... 18
      2.2.1. Digital Up and Down Converter .................. 19
      2.2.2. Time Synchronization module ................... 20
      2.2.3. Channel estimation implementation ............... 21
  2.3. Bottleneck test to improve real time .................. 22

CHAPTER 3. Processing chain improvements .................. 25
  3.1. Turbo decoder multiplexing, increasing throughput .... 25
3.2. CPU-GPU memory transfers .............................................. 25
3.3. Fast Fourier Transform ..................................................... 26
3.4. Primary synchronization signals detection ................................. 27
  3.4.1. Memory algorithm ..................................................... 27
  3.4.2. Reduction algorithm .................................................. 28
  3.4.3. Performance comparison .............................................. 29

CHAPTER 4. Real time execution ............................................... 31
  4.1. The Analog to Digital and Digital to Analog Converters ................. 31
    4.1.1. DUC output gain .................................................. 31
    4.1.2. LTE 128 Transmitter and receiver ................................ 31
  4.2. ALOE cluster platform .................................................. 32

CHAPTER 5. Future work ......................................................... 35
  5.1. Platform expansion ....................................................... 35
  5.2. Environment .............................................................. 35

CHAPTER 6. Conclusions ......................................................... 37

BIBLIOGRAPHY ........................................................................ 39

Appendix A. LTE configuration parameters .................................. 42

Appendix B. LTE Execution times .............................................. 43
  B.1. LTE 128, 72 subcarriers, only PDSCH, QPSK ......................... 43
  B.2. LTE 128/256/512/1024/2048, only PDSCH, QPSK ................. 44

Appendix C. C and CUDA code examples .................................... 45
  C.1. FFT3W sample code ..................................................... 45
  C.2. CUFFT sample code ..................................................... 46
  C.3. Memory algorithm sample code ....................................... 47
  C.4. Memory algorithm sample code ....................................... 48
# LIST OF FIGURES

1.1 SDR receiver schematic example .......................................................... 3
1.2 A DAS implementation example (Hermes Wireless Solutions)[4] ............... 4
1.3 ALOE processing layers [6] ....................................................................... 5
1.4 ALOE Architecture [6] ............................................................................. 5
1.5 Pipelining example .................................................................................... 6
1.6 ALOE’s module skeleton ........................................................................... 7
1.7 ALOE User Interface, waveform graph example ....................................... 7
1.8 Time-Frequency Resource Grid for LTE downlink (FDD, CP short) ........... 8
1.9 Radio Frame Structure for LTE downlink (FDD, CP short) ....................... 9
1.10 Schematic difference between CPU and GPU architecture ...................... 11
1.11 CUDA grids, blocks and threads hierarchy .......................................... 12
1.12 Platform computers connection ............................................................. 13

2.1 LTE downlink processing chain ............................................................... 15
2.2 Bit level processing standard definition schematic ................................... 16
2.3 Bit level processing implementation schematic ....................................... 16
2.4 Turbo decoder BER in function of SNR .................................................. 17
2.5 Turbo decoder number average iterations in function of SNR .................... 17
2.6 Turbo decoder throughput in function of SNR ......................................... 18
2.7 Discrete frequency DUC module input (ALOEUI screen shoot) .............. 19
2.8 Discrete frequency DUC module output (ALOEUI screen shoot) .......... 20
2.9 Efficient Digital Up Converter scheme .................................................... 20
2.10 PSS synchronization module, block scheme ......................................... 21
2.11 PSS decimated correlation results for different roots (ALOEUI screen shoots). 22
2.12 PSS synchronization module Probability Of Detection (POD) Vs. SNR ...... 23
2.13 Execution time for LTE128 application ................................................ 23

3.1 CUDA specific memory copy functions .................................................... 26
3.2 Bypass GPU Vs. CPU execution time ...................................................... 26
3.3 FFT execution time comparison between the CPU and the GPU ............... 27
3.4 Threat reduction algorithm example ....................................................... 29
3.5 Synchronization execution time for CPU and GPU ................................... 29

4.1 FFT algorithm performance over the CPU and the GPU ......................... 32
4.2 Cluster connection for DAS emulation with the ADC and the DAC ......... 33
4.3 Cluster serial connection ........................................................................ 34

5.1 Cluster connection for DAS implementation adding a server .................... 36

C.1 Basic code to call and use the FFTW3 library ......................................... 45
C.2 Basic code to call and use the CUFFT library ......................................... 46
C.3 Convolution code as example of memory managing ............................... 47
C.4 Detection kernel code, thread reduction algorithm ................................... 48
LIST OF TABLES

1  Processor comparison (May 2012) .................................................. 1
1.1 OFDM modulator parameters [7] .................................................. 9
1.2 Platform specifications ................................................................. 12
5.1 Processor power comparison ....................................................... 36
A.1 LTE128 application bit level parameters, 72 subcarriers, QPSK .......... 42
A.2 LTE256 application bit level parameters, 144 subcarriers, QPSK .......... 42
A.3 LTE512 application bit level parameters, 300 subcarriers, QPSK .......... 42
A.4 LTE1024 application bit level parameters, 600 subcarriers, QPSK ....... 42
A.5 LTE2048 application bit level parameters, 1200 subcarriers, QPSK ....... 42
B.1 LTE128 measured execution time, in $\mu$s .................................... 43
B.2 LTE measured execution times, in $\mu$s .......................................... 44
## ACRONYMS

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>3GPP</td>
<td>3rd Generation Partnership Project</td>
</tr>
<tr>
<td>ADC</td>
<td>Analog to Digital Converter</td>
</tr>
<tr>
<td>ALOE</td>
<td>Abstraction Layer and Operating Environment</td>
</tr>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>BTS</td>
<td>Base Transceiver Station</td>
</tr>
<tr>
<td>BW</td>
<td>Band Width</td>
</tr>
<tr>
<td>CP</td>
<td>Cyclic Prefix</td>
</tr>
<tr>
<td>CPU</td>
<td>Central Processing Unit</td>
</tr>
<tr>
<td>CRC</td>
<td>Cyclic Redundancy Check</td>
</tr>
<tr>
<td>CUDA</td>
<td>Compute Unified Device Architecture</td>
</tr>
<tr>
<td>DAC</td>
<td>Digital to Analog Converter</td>
</tr>
<tr>
<td>DAS</td>
<td>Distributed Antenna System</td>
</tr>
<tr>
<td>DC</td>
<td>Direct Current</td>
</tr>
<tr>
<td>DFT</td>
<td>Discrete Fourier Transform</td>
</tr>
<tr>
<td>DRAM</td>
<td>Dynamic Random Access Memory</td>
</tr>
<tr>
<td>FFT</td>
<td>Fast Fourier Transform</td>
</tr>
<tr>
<td>FLOPS</td>
<td>Floating-Point Operations per Second</td>
</tr>
<tr>
<td>FPGA</td>
<td>Field Programmable Gate Array</td>
</tr>
<tr>
<td>GPU</td>
<td>Graphical Processing Unit</td>
</tr>
<tr>
<td>HSPA</td>
<td>High Speed Packet Access</td>
</tr>
<tr>
<td>HPC</td>
<td>High Performance Computing</td>
</tr>
<tr>
<td>IFFT</td>
<td>Inverse Fast Fourier Transform</td>
</tr>
<tr>
<td>LTE</td>
<td>Long Term Evolution for UMTS</td>
</tr>
<tr>
<td>MAC</td>
<td>MultiplyAccumulate operation</td>
</tr>
<tr>
<td>OFDM</td>
<td>Orthogonal Frequency Division Multiplexing</td>
</tr>
<tr>
<td>PE</td>
<td>Processing Element</td>
</tr>
<tr>
<td>PSS</td>
<td>Primary Synchronization Signal</td>
</tr>
<tr>
<td>QAM</td>
<td>Quadrature Amplitude Modulation</td>
</tr>
<tr>
<td>RF</td>
<td>Radio Frequency</td>
</tr>
<tr>
<td>RS</td>
<td>Reference Signal</td>
</tr>
<tr>
<td>SBR</td>
<td>Software Based Radio</td>
</tr>
<tr>
<td>SDR</td>
<td>Software Defined Radio</td>
</tr>
<tr>
<td>SNR</td>
<td>Signal to Noise Ratio</td>
</tr>
<tr>
<td>SSS</td>
<td>Secondary Synchronization Signal</td>
</tr>
<tr>
<td>TS</td>
<td>Time Slot</td>
</tr>
<tr>
<td>UE</td>
<td>User Equipment</td>
</tr>
<tr>
<td>UMTS</td>
<td>Universal Mobile Telecommunications System</td>
</tr>
<tr>
<td>ZC</td>
<td>Zadoff-Chu</td>
</tr>
</tbody>
</table>
INTRODUCTION

Radio technology is evolving quickly. The wireless systems tend to be Software Radio Defined (SDR). The new communications standards are very efficient and too complex, that’s why it requires more computational resources.

FlexNets project provide some tools to develop SDR. Its facilitate execute processing chains in real time. The most important concepts of this project are the pipelining capacity and the processing chain integration in a computer cluster.

By the last years, the Graphical Processors Units (GPUs) have been evolving and expanding their market in high computational complexity applications. The GPUs have hundreds of lightweight processors specialized in floating point operations. The table 1 shows a comparison of different processors. There is a boundary between CPUs and GPUs. The processing capacity of CPUs are tens of Giga-Floating Point Operations Per Second (GFLOP) while GPUs have a capacity from hundreds to thousands of GFLOPS. The price of every processor depends of the application and the capacity. The Gaming GPUs are cheaper and have a high computational capacity, this feature makes them very interesting.

<table>
<thead>
<tr>
<th>Processor (Threats/Cores x Freq.)</th>
<th>Application</th>
<th>Price (€)</th>
<th>Capacity (GFLOPS)</th>
<th>Price/Capacity (€/GFLOP)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Xeon X5675 (12 x 3,1GHz)</td>
<td>Server</td>
<td>1260</td>
<td>37</td>
<td>34</td>
</tr>
<tr>
<td>Opteron 6282 (16 x 2,6GHz)</td>
<td>Server</td>
<td>938</td>
<td>42</td>
<td>22</td>
</tr>
<tr>
<td>i5 2500 (4 x 3,3GHz)</td>
<td>Regular desktop</td>
<td>190</td>
<td>13</td>
<td>15</td>
</tr>
<tr>
<td>i7 2700K (8 x 3,5GHz)</td>
<td>Regular desktop</td>
<td>296</td>
<td>28</td>
<td>11</td>
</tr>
<tr>
<td>Quadro 6000 (448 x 2,3GHz)</td>
<td>High-end CAD</td>
<td>4020</td>
<td>1030</td>
<td>3,9</td>
</tr>
<tr>
<td>Tesla C2075 (448 x 1,15GHz)</td>
<td>HPC</td>
<td>2007</td>
<td>1030</td>
<td>1,95</td>
</tr>
<tr>
<td>Quadro 4000 (256 x 1,9GHz)</td>
<td>High-end CAD</td>
<td>860</td>
<td>487</td>
<td>1,77</td>
</tr>
<tr>
<td>GF GTX 560 (336 x 1,9GHz)</td>
<td>Gaming</td>
<td>191</td>
<td>638</td>
<td>0,3</td>
</tr>
<tr>
<td>GF GTX 680 (1536 x 2GHz)</td>
<td>Gaming</td>
<td>520</td>
<td>3090</td>
<td>0,17</td>
</tr>
</tbody>
</table>

Table 1: Processor comparison (May 2012)

Long Term Evolution (LTE) is the newest standard for cellular mobile communication. The purpose of this work is implement a LTE downlink physical layer to test the FlexNets project tools and determine wether GPU acceleration is able to accelerate SDR.

The structure of this work is as follows. On the first chapter, some basic aspects will be discussed in order to explain the state of art of Software Defined Radio, FlexNets project, GPU computing and LTE. In second chapter, the most important skills of the LTE downlink implementation will be presented. The improvements solutions to achieve realtime will be presented in third chapter. In fourth chapter, the final resulting LTE processing chain working on the platform will be explained. And finally it is presented the future work and conclusions.
LTE downlink physical layer processing chain SDR application acceleration with GPUs
CHAPTER 1. STATE OF ART

1.1. Introduction to Software Defined Radio

In the radio’s world, the Software Based Radio (SBR) is an efficient technique to design wireless devices. The SBR replaces some traditional methods to design transmitters and receivers. The SBR technology have introduced some advantages like adaptation, reconfiguration and multiple functionalities with different operation modes, as well as, frequency bands, air interfaces and waveforms. The term ‘SDR’ (Software Defined Radio) is an implementation of SBR [1].

The Software Defined Radio (SDR) is a radio where some or all functionalities are software defined [2]. Radio is the technology used to transmit and receive wireless electromagnetic radiation, which enable the information transmission. A technology is Software Defined when some functionalities are implemented by some instructions that are executed by a programable device (for example CPU, FPGA, DSP).

![Figure 1.1: SDR receiver schematic example](image)

Normally, in SDR receivers, the signal digitalization is done after the intermediate frequency stage and the opposite is done for transmitters. The figure 1.1 shows a SDR receiver schematic example.

The SDR has benefits to radio manufacturers, systems integrators and users. The most important SDR benefit is the implementation of an entire radio family over the same architecture. It makes easier introduce more products into market quickly.

SDR enables infrastructure updates without extra big inversions and there are software remote updates. Users can get cheaper systems with less maintenance, wireless and the service that they need.

Nowadays, other concept has been introduced, it is Distributed Antenna System (DAS). It consists in divide the capacity of the cell among several low-power spatially separated antennas as is shown in figure 1.2. The DAS concept allows to have low power consumption Base Transceiver Station (BTS). Some BTS could be in a physical emplacement, it implies a energy and maintenance cost reduction.

The BTS could be formed by some Processing Elements (PE) [3] which generates the waveform taking advantage of the SDR technology.
1.2. FlexNets project

FlexNets is an open source project that explores the new possibilities of flexible radio communications.

FlexNets project makes easier the design and development of flexible radio communications systems. It provides the needed tools to share hardware and software resources in a processing infrastructure. In fact, FlexNets provides a realtime resource management for different SDR applications on one platform [5].

The main FlexNets application is the middleware called ALOE (Abstraction Layer and Operating Environment). This software provide some tools to make easier the use and portability of waveforms in multi processors platforms.

1.2.1. ALOE processing layers and architecture

ALOE is composed of four processing layers. All of these layers are shown in the figure 1.3.

The hardware layer is a group of processors (ex. computers) which are connected between them.

The ALOE layer is where the software ALOE is allocated. It runs separately in every computer which is connected in the network. All these connected computers form the ALOE platform.

The Application Layer is the layer where the programmed waveform is executed. It is composed of all the modules that the user have programmed and connected to create his
own processing chain.

The **Abstract Application Layer** is the layer where the different programmed modules are managed and mapped to the platform available resources.

![ALOE processing layers](image)

Figure 1.3: ALOE processing layers [6]

This organization makes easier execute a processing chain transparently over the platform.

Conditioned by the implementation ALOE uses some components and libraries, they are shown in the figure 1.4.

![ALOE Architecture](image)

Figure 1.4: ALOE Architecture [6]

The ALOE software daemons are a group of programs made for the correct execution of ALOE. The utility of these programs are resource managing, processor synchronization, parameters configuration, etc.

Also ALOE has an Application Programming Interface (API).
1.2.2. Taking advantage of multiprocessor platform

ALOE take advantage using some methods for performance improving in multiprocessor platforms. The most important is pipelining. It consists in divide an application in different process (modules) and every Processing Element (PE) perform a different process of the entire application. To implement this method all the PEs are time synchronized and process the data during a Time Slot (TS) as shown in figure 1.5.

The ALOE TS is the fraction of time which ALOE executes the modules. It fixes the execution cadence of each module.

Pipelining presents a disadvantage, the processing chain latency is proportional to the number of processes. In other words, the time which is spend for a data block during the processing chain is proportional to number modules.

![Figure 1.5: Pipelining example](image)

ALOE software has a scheduler which assigns the applications modules (processes) to the available PE. The mapping depends of the necessary computing resources of each module. The computing resources used for mapping are the necessary Multiply-Accumulate operations (MAC) and the necessary bandwidth between interfaces. The number of necessary MACs are used to know the PE occupancy by the module and the interfaces bandwidth is used to control the bandwidth between the PEs. However, the user can select which weight can have every parameter when the mapping is performed.

1.2.3. ALOE module organization

ALOE has some modules. The combination of these modules form the waveforms processing chains. Every ALOE module has a common skeleton. It includes the main processing code and some interfaces such as inputs and outputs. The figure 1.6 shows a schematic of the skeleton.

Every module has some configuration parameters called stats. Some of them are common in all modules such as data interfaces types and input gain. The others stats are specific for every module like the Fast Fourier Transform size or turbo decoder iterations. The processing main function is called from the skeleton. Also, the skeleton call the initialization
The modules check every TS whether the inputs interfaces have received the correct input data length. If the input data length is not correct, they log a message with the number of Time Slot (TS).

1.2.4. ALOE User Interface

The ALOE User Interface (ALOEUI) provides a graphical user-friendly control interface for loading and executing waveforms. It can show some data in real time such as platform configuration, waveforms stats and interfaces data. The figure 1.7 shows a screenshot of ALOEUI.
1.3. Long Term Evolution Downlink Physical Layer

One of the newest innovative communications standards is the UMTS Long Term Evolution (called LTE). It is a standard designed to satisfy the new mobile communications requirements.

LTE uses Orthogonal Frequency Division Multiplexing (OFDM), like Digital Video Broadcasting (DVB), Wi-Fi (IEEE 802.11) and WiMAX (IEEE 802.16). In OFDM systems the spectrum occupied by one carrier is divided in many narrow band subcarriers. It has some advantages such as radio spectrum flexibility and better channel equalization.

This work focuses the LTE downlink for Frequency Division Duplex (FDD) with short Cyclic Prefix (CP, seven OFDM symbols per slot).

1.3.1. LTE radio frame structure

The LTE downlink radio frame structure is organized in OFDM symbols. Every of them has groups of subcarriers. The Resource Blocks (RB) are groups of resource elements which are formed by 12 subcarriers and 7 OFDM symbols as is showed in figure 1.8. Some of that symbols are reference signals which are used for channel estimation in the receiver. The subcarrier spacing is 15 kHz.

![Figure 1.8: Time-Frequency Resource Grid for LTE downlink (FDD, CP short)](image)

The radio frame (10 ms duration) is composed by ten sub-frames (1 ms) and they are composed by two slots (0,5 ms). Every slot has seven (six for long CP) OFDM symbols. The figure 1.9 shows this structure.
1.3.2. Spectrum configurations

The bandwidth (BW) of the system is flexible, it means that every available bandwidth has a different configuration in the OFDM modulator. The table 1.1 shows all the parameters which change for every available bandwidth.

<table>
<thead>
<tr>
<th>BW (MHz)</th>
<th>1,25</th>
<th>2,5</th>
<th>5</th>
<th>10</th>
<th>15</th>
<th>20</th>
</tr>
</thead>
<tbody>
<tr>
<td>Slot Duration</td>
<td>0,5 ms</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Freq. between sub-carriers</td>
<td>15 kHz</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sampling Freq. (MHz)</td>
<td>1,92/3,84</td>
<td>3,84</td>
<td>7,68</td>
<td>15,36</td>
<td>23,04</td>
<td>30,72</td>
</tr>
<tr>
<td>FFT size</td>
<td>128</td>
<td>256</td>
<td>512</td>
<td>1024</td>
<td>1536</td>
<td>2048</td>
</tr>
<tr>
<td>Sub-carriers used</td>
<td>72</td>
<td>144</td>
<td>300</td>
<td>600</td>
<td>900</td>
<td>1200</td>
</tr>
<tr>
<td>Num. symbols OFDM per TS</td>
<td>7 (Short CP) / 6 (Long CP)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CP Short length (samples)</td>
<td>9×6 10×1</td>
<td>18×6 20×1</td>
<td>36×6 40×1</td>
<td>72×6 80×1</td>
<td>108×6 120×1</td>
<td>144×6 160×1</td>
</tr>
<tr>
<td>CP Long length (samples)</td>
<td>32×6</td>
<td>64×6</td>
<td>128×6</td>
<td>256×6</td>
<td>764×6</td>
<td>512×6</td>
</tr>
</tbody>
</table>

Table 1.1: OFDM modulator parameters [7]

1.3.3. LTE processing chain

The LTE downlink processing chain is based on an OFDM modulators. First, the data is coded, interleaved and rate matched. After that, the coded bits are converted to complex symbols (BPSK, QPSK, 16-QAM or 64-QAM) and they are allocated to spectrum. In the spectrum are allocated at same time the reference, synchronization signals and others physical channels (as broadcast, format indicator, etc) [8]. In this work, it will be only consider the Physical Downlink Shared Channel (PDSCH). After, the IFFT (Inverse Fast Fourier Transform) is computed to convert from discrete frequency to time domain. In time domain a cyclic prefix is added (some tail samples are copied at the beginning of the
OFDM symbol) and the frame is ready to be converted to a bandpass and send to the Analog to Digital Converter (ADC).

In the receiver side, the bandpass signal is captured by the Analog to Digital Converter (ADC) and digitally moved to baseband. The receiver must be synchronized with the transmitter using the time domain primary synchronization signal. When the receiver is completely synchronized the Cyclic Prefix is removed and the FFT is performed. Channel estimation and equalization are required to overcome the distortion channel effects. After the signal is equalized, the information is extracted and converted from symbols to bits (demapper). Finally, the data must be rate matched, deinterleaved and decoded.

### 1.3.4. Synchronization Signals

The LTE downlink uses two types of synchronization signals, the Primary Synchronization Signal (PSS) and the Secondary Synchronization Signal (SSS). The main purpose of these signals is synchronize the User Equipment (UE) in time and frequency domain. This subsection is target on the PSS.

The PSS and the SSS are calculated in function of the cell identification number. There are only 504 possible physical cell identifications and they are grouped in 168 groups with 3 possible identifications for every group. The cell identification ($N_{(cell)}^{(cell)}$) is defined as the expression 1.1 where $N_{ID}^{(2)}$ is in the range between 0 and 2. $N_{ID}^{(1)}$ is in the range between 0 and 167 [9].

$$N_{ID}^{(cell)} = 3N_{ID}^{(1)} + N_{ID}^{(2)}$$  \hspace{1cm} (1.1)

The UE first looks the PSS which is transmitted in the last OFDM symbol in the first slot (slot 0) and in 11th slot (slot 10). This enables the UE to get time synchronized every 5 ms. In frequency domain, six Resource Blocks (RB) around the DC subcarrier are reserved for transmission of the synchronization signals.

Both signals are generated one time and stored. The PSS is generated using a sequence called Zadoff-Chu (ZC). The equation 1.2 shows the function used to generate this sequence where the root of the ZC sequence $\mu$ depends of the identifier $N_{ID}^{(2)}$. The first 31 generated values are mapped in frequency domain under the DC and the other 31 above the DC. The ZC sequences are Constant Amplitude Zero Auto Correlation (CAZAC) sequences. This sequences achieve a constant amplitude with low peak-to-average power ratio and zero auto correlation remains in time domain [10].

$$d_{\mu}(n) = \begin{cases} e^{-j\frac{\pi n(n+1)}{63}} & \text{if } n = 0, 1, \ldots, 30 \\ e^{-j\frac{\pi(n+1)(n+2)}{63}} & \text{if } n = 31, 32, \ldots, 61 \end{cases}$$  \hspace{1cm} (1.2)
1.4. Compute Unified Device Architecture

The last years, moved by 3D real time applications as games, Computer-Aided Design (CAD), the Graphics Processors Units (GPU) have become very powerful using many core processors with hundreds of cores and high memory bandwidth.

Compute Unified Device Architecture (CUDA) is a parallel computing platform and programming model developed by NVIDIA. It enables dramatic increases in computing performance by harnessing the power of the GPUs [11].

Nowadays, the Central Processing Units (CPUs) are very fast and powerful but they have a problem: complexity. These processors usually have less than 10 cores. This number of cores is limited by the power requirements of each core. The GPUs have hundreds of simpler cores, the figure 1.10 shows the hardware architectural difference between the CPU and the GPU. Both of them have their memory (DRAM, Dynamic Random Access Memory) and Arithmetic Logic Units (ALU).

![Figure 1.10: Schematic difference between CPU and GPU architecture](image)

The main advantage of using GPUs against the CPUs is that GPUs are designed to compute high mathematical complexity algorithms which requires high computing resources and algorithms which could be parallelizable.

The GPU has some multiprocessors. They have an internal memory which is shared with their processors. The GPU is communicated with the CPU via PCI-Express. In CUDA jargon the device is referenced to the GPU while the host is referenced to the CPU.

CUDA enables the execution of some code in the device as kernels (functions). Every kernel is executed in a time fraction and some threads are executed in one kernel. The CUDA threads are lighter than CPU threads. Their creation is very fast and the thread commutation is instantaneous. Every thread of a kernel have the same code and has his identification. When a kernel is called a grid of threads is created. The threads are grouped in blocks, both of them are identified by using three dimension coordinates as it is shown in figure 1.11.

The blocks contribute between them using the global and the shared memory. The global memory is the physical memory which is not inside the GPU chip. The global memory size
is in order to units of gigabytes. On the other hand, the shared memory is inside the GPU chip. This memory is shared by all threads in a block. It is faster than global memory and its size is in order to tens of kilobytes.

1.5. Working platform

Defining the working platform is essential to have a reference point in signal processing. The studies, made in this project have been made by interconnecting multiple computers forming a platform. The computers specifications are in the table 1.2. Every computer has installed a Linux distribution (concretely Ubuntu 10.04) with a real time kernel patch.

<table>
<thead>
<tr>
<th>Name</th>
<th>CPU (Cores × Freq.)</th>
<th>GPU (Cores × Freq.)</th>
<th>Expansion</th>
</tr>
</thead>
<tbody>
<tr>
<td>Odissey071</td>
<td>i7-920 (4×2.67 GHz)</td>
<td>GF 9600GT (64×1.2 GHz)</td>
<td>-</td>
</tr>
<tr>
<td>Odissey072</td>
<td>i7-920 (4×2.67 GHz)</td>
<td>GF 9600GT (64×1.2 GHz)</td>
<td>-</td>
</tr>
<tr>
<td>host SDR</td>
<td>i7-920 (4×2.67 GHz)</td>
<td>GF 260GTX (192×1.2 GHz)</td>
<td>ADC+DAC</td>
</tr>
</tbody>
</table>

Table 1.2: Platform specifications

The platform Computers are connected between them as showed in figure 1.12. The computer Odissey-desktop and the Odissey72 are connected between them using two 1 Gbps ethernet connections to ensure higher bit rates. The LAN interfaces connected to the switch (net identifier 192.168.100.XXX) are connected for control and synchronization between them.

In order to test CUDA performance, it is used other computer which is not in the table 1.2. This computer has some improvements like higher CPU and DRAM frequencies. Also has a GPU which can compute some kernels at the same time.
1.6. Scope of this work

The objective of this work is program the LTE downlink processing chain (OFDM modem) for FDD and short CP. After programming the processing chain, a performance test have been done. The measures have been done using the entire 10 ms radio frame.

The first step is integrate the processing chain in the platform and determines if it is possible to run the processing chain in the available platform. If possible then the objective is to identify the system bottlenecks. The main goal of this project is to run the processing chain in that kind of platform, identify the troubles when dealing with high computing requirements applications running in real time and propose solutions.

The tools that will be used to reduce the bottlenecks effects are combine modules, force platform mapping, CUDA acceleration and modules parallelization.

The implementation is based on LTE Release 8 of the 3GPP specification. It is considered 1 base station and one user equipment with 1 antenna each one. The focus is mainly on the physical layer. Only PDSCH (Physical Downlink Shared Channel) is considered and no control channels are considered. FDD (Frequency-Division Duplexing) is supported. The implemented synchronization and the channel equalizer will be simple as possible. Achieving the 3GPP LTE standard latency is not an objective. Every ALOE TS will be precessed the 10 ms radio frame without be divided.
LTE downlink physical layer processing chain SDR application acceleration with GPUs
CHAPTER 2. CASE STUDY

This chapter explains with detail the LTE processing chain running on ALOE. The LTE processing chain could be divided in two level: Bit Level Processing and Symbol Level processing. Every one of both levels are composed of their modules. The processing chain is showed in the figure 2.1.

![LTE processing chain diagram](image)

2.1. Bit level processing implementation

The main objective of bit level processing is to add redundancy to detect and correct bit errors. Also incorporate those elements identified as interleaving, scrambling and rate matching. The scheme of this module is shown in figure 2.2.

The first step is add a Cyclic Redundancy Check (CRC). The purpose of this CRC ($g_{CRC24A}$) is detect the received code blocks contains some errors. If the code block is longer than 6144 bits, segmentation must be performed to ensure that data size is not bigger than maximum code block size. When the segmentation is done using other CRC ($g_{CRC24B}$).

After adding CRCs the data is coded with a turbo encoder. The turbo encoder rate is 1/3, which means two redundancy bits per one data bit. After encoding data, it is interleaved with a 32 columns interleaving matrix which writes by rows and read by columns using a defined permutation pattern. The mission of interleaving is distribute a burst of errors between different rows.

The interleaved data are collected, scrambled with the others bit streams and saved in
a virtual circular buffer. A bit selection and pruning is performed over the virtual circular buffer. This is the rate matching. The objective of Rate Matching (RM) is adjust the bit stream to the resource block resources [12].

The implemented processing chain is showed in the figure 2.3 where the data blocks are serialized to reduce the number of modules and interfaces. The modules are programmed to disjoin every data block before being processed.
2.1.1. Turbo decoder implementation

The channel coding for PDSCH in LTE downlink adopts turbo coding which is a kind of robust channel coding. The coding rate of the turbo encoder is 1/3. According to Section 5.1.3.2 of [12] (3GPP LTE Standard), the scheme of the Turbo encoder is a Parallel Concatenated Convolutional Code (PCCC) with two 8-state constituent encoders and one internal interleaver. The Bit Error Rate of this turbo code is showed in the figure 2.4 for three different code blocks sizes. The decoding uses a LOGMAX algorithm with a factor equal to 0.5.

![Figure 2.4: Turbo decoder BER in function of SNR](attachment:ber.png)

Coding and decoding modules have been implemented by using the IT++ library. It must be compiled as C++ (g++) and the module must be compiled as C (gcc). To add the C++ code to the module is used a Dynamic Link Library (DLL). The DLL (in Linux OS sufix are *.so and not *.dll) is loaded in initialization function. This turbo code has a dynamic halt, it stops decoding when the two last decoded data blocks are the same.

![Figure 2.5: Turbo decoder number average iterations in function of SNR](attachment:iterations.png)

The throughput is related with the SNR and the number of iterations needed to decode. In other words, lowest is the SNR more iterations are needed. The figure 2.5 shows the
number of iteration in function of the SNR. Then, the throughput increases with the SNR as is shown in figure 2.6.

This turbo code implementation can encode and decode using any polynomial. Although, the encoder and decoder are flexible, the throughput is low (< 1 Mbps, one iteration).

![Turbo decoder throughput vs. SNR](image)

Figure 2.6: Turbo decoder throughput in function of SNR

2.1.2. Bit level configuration

Finding the bit level parameters is the most tedious task when the LTE downlink waveform is being configured. All the parameters must be calculated from the turbo decoder processing time and the number of RBs (resource Blocks) used.

The time spend by the turbo decoder is very important. It fixes the minimum ALOE TS duration. The code blocks size depends of the number of the RBs transmitted in the ALOE TS.

In this work, the ALOE time slot is fixed to 10 ms and the encoded codewords corresponds to PDSCH resource elements in every subframe (in exception of LTE2048). In other words, 10 codewords are encoded in every ALOE TS and every codeword is mapped in a subframe. All the implemented configurations are in the appendix A.

For the application LTE2048 (1200), the codewords are encoded for every slot to avoid codewords bigger than 6144 bits.

2.2. Symbol level processing implementation

The symbol level processing stage is simpler than the bit level processing stage. It is composed of mapper module which converts bits to complex symbols. The pilot module takes the data symbols from the mapper and put it in the LTE downlink resource grid with the reference signals and synchronization signals. The next module is the inverse fast
fourier transform (IFFT) which use the fft3w library which will be explained in the next chapter.

After compute IFFT must be added the cyclic prefix to every data processed block at IFFT output. After add the Cyclic Prefix, the complex data in baseband frequency must be converted to real baseband to be send to the Digital to Analogical Converter (DAC).

In the receiver side, the received signal in the Analog to Digital Converter (ADC) is digitally moved from bandpass to baseband frequency. The receiver synchronize the radio frame to ensure the correct Fast Fourier Transform (FFT) window. In case the FFT window is shifted a lineal phase error will be detected in the demodulator. After, the cyclic prefix is removed and the FFT is performed. When the data is in frequency domain the channel is estimated and equalized to compensate fades and phase shifts. After equalize channel the data symbols are extracted from the frequency domain to be converted to bits in the demapper module.

2.2.1. Digital Up and Down Converter

A Digital Up Converter (DUC) is used to convert signal from baseband to bandpass. The input discrete frequency domain is shown in figure 2.7.

![Figure 2.7: Discrete frequency DUC module input (ALOUEI screen shoot)](image)

The DUC is designed to send the waveform thought the DAC to analog RF module. The output discrete frequency domain data of this module is showed in the figure 2.8. It is centered in the discrete frequency $f_s/4$.

This module doubles the necessary sampling frequency ($F_s$) doing an interpolation using two low pass polyphase filters to avoid spectrum overlap when the signal is mixed. The interpolated signal is mixed to $F_s/4$ in order to make it more efficient. The figure 2.9 shows the DUC scheme.
In the receiver is used the opposite, the Digital Down Converter (DDC). The DDC module schematic is very similar to the DUC module but its functionality is the opposite. It converts from real bandpass to IQ (inphase and quadrature) to baseband frequency.

### 2.2.2. Time Synchronization module

The time synchronization is such important that in case the FFT windows is shifted one or more samples there will be a linear error in the sub carriers phases. To perform this function some methods could be performed. The easier and efficiency enough is used in this work.

As said in previous chapter, one cell can transmit three possible PSS depending of the $N_{ID}^{(2)}$ value. The generated ZC sequence is 64 samples length (62 subcarriers plus DC). In the transmitter, is performed a 128/256/512/1024/1536/2048 IFFT to get the time domain samples. On the other hand, it could be performed using a 64 length IFFT.

Using ZC sequences time domain zero auto correlation property. The synchronization
could be performed doing a time domain correlation. Due the PSS could be one of three, the correlation must be done with all three possible sequences. To reduce computing requirements, the correlation is performed with a 64 sample sequence. Then, the received data must be decimated with a decimation factor equal to the IFFT size in the transmitter divided by 64. Although, the detected window using this correlation could be between some values because the received data has been decimated. Then, after detect the decimated window, a fine correlation is performed in order to detect the correct FFT window. The scheme of this module is showed in the figure 2.10.

The decimated correlation for different roots is shown in the figure 2.11. The first graph is the correlation result from two ZC generated sequences with the same roots and the second is generated from two different roots. In order to detect the correlation peak, it is computed the correlation mean and the maximum. When the maximum divided by the mean is above a threshold it means that the module is synchronized.

The configuration parameters of this module are the decimation factor and the threshold. This module has a probability of detection which depends of the received noise. The figure 2.12 shows this Probability Of Detection (POD) in function of the Signal to Noise Ratio (SNR).

### 2.2.3. Channel estimation implementation

The implemented channel estimation algorithm is simpler as possible. It takes the magnitude and phase errors from comparison of the received and desired Reference Signals (RS). These errors are set in the position where are located the RS in the Resource Grid (RG) and the others symbols are set to 0. An interpolation filter is used to interpolate the corrections in order to fill the gaps between RS positions. The interpolation filter is a linear 2D filter which makes the mean of the nearest pilots in frequency and time. It is a $11 \times 13$ mesh.
2.3. Bottleneck test to improve real time

After programming the LTE downlink processing chain a performance test must be done. The graph showed in the figure 2.13 shows the execution time of the most expensive modules. The execution information is available in the table B.1.

The bottleneck is the module "gen9SK_turboDec" (for 1 iteration). In fact, this module is consuming the 77% of one PE. If a PE is loaded more than 80% can have real time execution fails. Then this module is in the limit. In case the number of iterations increases, the application can not achieve real time. Then the turbo decoder throughput must be improved. The minimum noise can provoke a real time-fail. Other module which might be improved is the "gen9SK_PSSdet" which performs the time synchronization.
Figure 2.12: PSS synchronization module Probability Of Detection (POD) Vs. SNR

Figure 2.13: Execution time for LTE128 application
CHAPTER 3. PROCESSING CHAIN IMPROVEMENTS

As said in the last section of the previous chapter, some modules makes difficult the real time simulation. This chapter focuses in some hypothesis to improve the performance of the processing chain.

3.1. Turbo decoder multiplexing, increasing throughput

As said in previous chapter, the turbo decoder module has a low throughput (<1.5 Mbps) with only one iteration. Generally speaking, the decoder is such slow that it hinders real time execution. Try to reprogram the code is not a solution. Then, multiplexing decoders is the best solution. If the decoders are parallelized they can take more time to decode their code words and the real time could be reached. It implies add two new modules to the processing chain, a multiplexer and the demultiplexer. The new throughput is multiplied by the number of parallelized decoders.

The key of this method is having enough processors to distribute these turbo decoders. It works if the number of PE is enough bigger to allocate the necessary decoders.

This solution has a big disadvantage, the multiplexor add more complexity to ALOE module mapping. The times spend by the turbo decoder using different LTE applications are found in the table B.2. Although, multiplexing resolved the throughput problem for LTE128 and LTE256, the others applications need more throughput to achieve real time.

Finally, this method will be used for LTE128 and LTE256, the turbo decoder must be changed in order to be executed in real time in the other configurations.

3.2. CPU-GPU memory transfers

The purpose of the implementation of this module is test how fast are the memory in two transfers, from host to device and device to device. Generally speaking, it is the basic module to test the CUDA module interfaces throughput. The interfaces of CUDA enabled modules are modified in the module skeleton. The input interfaces copy all received data to the device. And the output interfaces copy the processed data from device to host.

As said in the section "1.4. Compute Unified Device Architecture" the device and the host do not share their memory. Then the memory must be copied from host to device using the function cudaMemcpy() which syntax is showed in the figure 3.1. This function has three possible types of copies, from host to device, from device to host and finally from device to device.

The results of this test are shown in the figure 3.2. The data transfers between the host
26 LTE downlink physical layer processing chain SDR application acceleration with GPUs

```c
// Host -> Device
cudaMemcpy(device_destination, host_source, size, cudaMemcpyHostToDevice);
// Device -> Host
cudaMemcpy(host_destination, device_source, size, cudaMemcpyDeviceToHost);
// Device -> Device
cudaMemcpy(device_destination, device_source, size, cudaMemcpyDeviceToDevice);
```

Figure 3.1: CUDA specific memory copy functions

and the device are slower than host to host. The time spend in transfers is such high that it is not depreciable.

![Transfer time graph](image)

Figure 3.2: Bypass GPU Vs. CPU execution time

### 3.3. Fast Fourier Transform

To compute the FFT and IFFT algorithms is used an external library called fft3w. This library incorporates the Discrete Fourier Transform (DFT), the installation is very easy and it is the most efficient open source DFT algorithm.

To use this library the desired data must be translated to the library data types. In ALOE, the modules interfaces uses two arrays of floating point values for inputs (one for real and the other for imaginary values). At this point, the data must be translated to a structure as is shown in the figure C.1. Also, a plan must be created to initialize the FFT and IFFT algorithms. Once time it is initialized, it could be called all times are needed [13].

Although this library is fast, it could be accelerated using CUDA (GPU). An existing library can do it, the CUFFT library. This library is based in the fftw3 algorithm. The difference between the two libraries is where the code is executed. It implies that to perform CUFFT the data must be send to the device using the interfaces generated in the previous section.

As said for the fft3w library, the CUFFT library needs other data format conversion. The algorithm is shown in the figure C.2. The data translation is done calling a CUDA kernel.

The performance comparison between the two libraries is showed in the figure 3.3. The fft3w is faster than CUFFT library for low sizes but CUFFT tends to be faster for big sizes.
The GPU needs a minimum data to be faster than the CPU.

![FFT execution time comparison between the CPU and the GPU](image.png)

**Figure 3.3: FFT execution time comparison between the CPU and the GPU**

### 3.4. Primary synchronization signals detection

The PSS synchronization is other module which can be accelerated. In order to reduce the CPU load it can be accelerated using CUDA. It implies convert the host (CPU) functions to device (GPU) kernels.

In order to convert these functions two important algorithms will be explained: the memory algorithm and the reduction algorithm. Both algorithms are designed to take advantage of parallelization capability. The kernels decimation, decimated correlation and fine correlation need the optimized memory algorithm. The kernel detection which detects the decimated correlation peak need a reduction algorithm.

#### 3.4.1. Memory algorithm

Taking advantage of CUDA means manage well the memory. It implies use an efficient algorithm to reduce the access to global memory. In other words, all the executing threads are not allowed to access at the same time in the global memory so it decreases the memory throughput. In order to reduce global memory accesses every thread in a block access to a different pointer in global memory to save it in the shared memory which is faster.

The kernels are divided functionally in three stages. The first one is copy data from global memory to shared. This stage consist in copy the data which is common needed by some threats in the block. For the convolution example, the needed data are the filter which is used by all threats and the input data. Every thread in the block only read one sample of the filter and store it in shared. The others threats in the block will store the other samples.
The second stage is process the data which is located in the shared memory. For convolution example, the input samples and the filter which are stored in shared memory are convolved. Thanks to the memory copy from the first stage the access to the memory is such fast that the total execution time is shorter.

Finally, every thread left their result in the global memory. It is very important do not access to global memory during the processing algorithm (stage 2). It is recommended that save results in the thread registers or shared memory. In the third stage these results are returned to the global memory.

The figure C.3 contains the code of an example of convolution algorithm. Although, the function looks longer than classical C code, the execution is faster.

The code showed in figure C.3 is not valid for filters bigger than the block size. In case the filter is longer than the block size, the code must divide the filter in some segments and compute the first and second stages for every filter segment. In this case the algorithm becomes more difficult and it is very easy commit errors in the memory managing.

In order to detect these errors exist some tools developed by NVIDIA which detects forbidden memory access. Also, is not easy fix all these bugs.

### 3.4.2. Reduction algorithm

As said in the section "2.2.2. Time Synchronization module", it must find the maximum and compute the mean of decimated correlation in order to detect the correlation peak. Although, these algorithms (calculate maximum and mean) are very simple for CPU code, it is not for GPU. The CUDA algorithm involves a conceptual change from serial to parallel execution.

The implemented algorithm is called reduction algorithm. This algorithm start with a number of threats which iterates the compare and sum operations. Each iteration the number of active threads is reduced by a reducing factor.

The figure 3.4 show an example of this algorithm which compute the sum of an input array. In the example, every threat add two values and they are reduced by 2 every iteration. At final the first thread has the final value of the sum. This parallelized algorithm is faster than copy data from device to host and compute it in the CPU. In fact, the maximum threat operations are 3, wether it was serialized (CPU execution) there was 7 operations. For 8192 input values implies a maximum of 13 threat operations while serialized were 8191 operations. Although, the number of operations is reduced, the number of GPU processor is limited and the reduction algorithm is slower. However, it keeps faster than CPU algorithm.

The code of the figure C.4 shows the complete code for a kernel reduction. It is the simplest code which performs this algorithm. The implemented code performs a reduction for the three possible sequences at the same time.
3.4.3. Performance comparison

The synchronization execution time for every LTE application is showed in the figure 3.5. The GPU execution time doubles with input data increasing. It happens because the kernel which perform the decimation has not enough capability to parallelize when the decimation factor increases. However the other kernels time remains practically constant. In the CPU the effect is the same. The decimation function doubles their time while the other remains constant. However the CPU clock is higher and the decimation is faster in the CPU algorithm. However, the total time is lower in GPU algorithm. Although the CUDA algorithm is faster, it is more complex.
LTE downlink physical layer processing chain SDR application acceleration with GPUs
CHAPTER 4. REAL TIME EXECUTION

4.1. The Analog to Digital and Digital to Analog Converters

The host SDR is equipped with a mixed signal board (Innovative Integration X5-400m) which includes a FPGA, two 16-bit DAC and two 14-bit ADC. ALOE has an interface which can send and receive data through these interfaces.

The DAC and ADC sampling frequency is set using an external clock. The minimum sampling frequency is fixed by the manufacturer and it is 20 MHz. Then the minimum LTE sampling frequency which is above 20 MHz is 30.72 MHz (for 1024 point FFT, after DUC).

ALOE incorporates some parameters which configure the board such as decimation, interpolation, clock divider, packet size, etc. The sampling frequency \( F_s \) and the packet size fixes the ALOE TS duration. The board diver is executed in the PE number 0. It takes the 80% of time. It means that in this PE no one module can be executed. It is a restriction for modules mapping when the waveform is loading.

To convert the \( F_s \) which is in the standard to the fixed frequency (30.72 MHz), the DUC (ALOE module) repeats the output samples like it was a Zero Order Hold (ZOH, simplest Low Pass Filter).

4.1.1. DUC output gain

The digital OFDM systems as LTE has a important disadvantage, it is the Peak to Average Power ratio (PAPR). The PAPR is the ratio between the peak power and the average power. These kind of systems apply some PAPR reduction methods such as the channel coding and modulation types. However the amplitude is not always constant.

The samples which are send to the DAC must be short integers (16 bit). The DUC output is in floating point. The output data from the DUC is converted to short using a gain. The value of this gain must be as low as possible to avoid that amplitude peaks distort the signal. Also, this gain must be high as possible to ensure the highest output SNR. The criteria was ensure that one per million frames have a sample which is over the scale.

4.1.2. LTE 128 Transmitter and receiver

After configured well the the DAC was the time to run the application LTE128Tx. It is the LTE128 transmitter processing chain. It transmits 72 subcarriers modulated using a QPSK modulation. The figure 4.1 shows the LTE128Tx application ouput spectrum. The signal is centered in 960 kHz and the bandwidth is 1.1 MHz.
The application LTE128Rx is the receiver. The computational costs of this application makes impossible their mapping in the host SDR. It is because as said at the fist section of this chapter, the DAC driver consumes the 80% of the PE 0. Then ALOE can’t map well these applications because the platform needs more resources.

4.2. ALOE cluster platform

This section describes how ALOE is tested in the cluster. The used hardware in this section was described in the section "1.5. Working platform". The instructions to configure the ALOE platforms for multiprocessing are available in the FlexNets project webpage [5].

In the ALOE platform one computer must be configured as synchronization and control master and the others connected computers must be configured as slaves. All the connected computers must have the same configured TS. Then the master will detects the platform with all the available PE.

The first test was connecting two computers as it is explained in the tutorial [14]. After connect and configure both computers some application had been run. The applications were the LTE128 bit level and after the LTE128. Both application run smoothly in real time.

The next step was add other computer. The ALOE interfaces were all connected between them and the total declared processing elements was 10. The total number of processors is 12 but ALOE can not work with more than 10 PE.
Although, ALOE map the modules in the PE, some modules are not correctly load in the PE. ALOE sends an error saying that it did not find an interface. In order to avoid this problem, the mapping can be forced. In this case the application was mapped, loaded and ran successfully. The application LTE128 runs with this configuration in real time without any errors. The LTE256 application was mapped with the same mapping than LTE128. It ran but it showed some errors in the Bit Error Rate (BER). However, ALOE did not show any real time error. After examine module logs, the problem was that ALOE lost some data between some computer interfaces. Although, find where the data were lost was easy, finding the bug was not.

When the application was debugged step by step all packets were received because ALOE runs without the real time restrictions. However, it is difficult fix the errors.

Other configuration is the emulation of the DAS concept where every computer is a different element. The figure 4.2 shows this schematic. The main objective of this configuration is transmit data using the DAC and receive the transmitted data with the ADC. The computer Odissey071 role was being the transmitter and the Odissey072 the receiver.

![Figure 4.2: Cluster connection for DAS emulation with the ADC and the DAC](image)

It involves force modules to be mapped in a specific computer. Although the modules are mapped correctly, the application LTE128 runs with errors. It loses packets between interfaces.

This configuration is not efficient. The transmitter had more resources than it needed. In the SDR host was loaded with the board drivers, the DUC and DDC modules. The data bandwidth coming from the odissey071 was 123 Mbps. The data bandwidth which was send to the Odissey072 was 123 Mbps. All the connections between computers were connected to a 1Gbps Switch. Probably the packets collisions made that this configuration did not run.

The next purposed configuration was connect the computers serially. As is showed in the figure 4.3. Two extra 1Gbps ethernet interfaces were added to the computers Odisseyy072 and SDR host to increase the available bandwidth. These interfaces were configured in the ALOE platform as unidirectional in order to avoid packet collisions. Using this configuration the transmitter and the receiver were distributed in Odissey071 and Odissey072. The connection between these computers was used only for bit level transfers which requires
low bandwidth.

The mapping and loading were done successfully. The modules DUC and DDC were forced to be mapped in the SDR host. Although, it ran in real time, sometimes, after a while starts to show errors. Some frames were lost between computer interfaces.

It depended of the load of the computers. The modules which were executing had a big variance in their execution time. It means that they consume more resources in determined TS and it made ALOE unstable.
CHAPTER 5. FUTURE WORK

Solving the problems which do not allow the correct ALOE execution in the computer cluster platform are essential to continue. After solve these errors and the LTE applications run well in the platform some modules must be optimized to ensure real time. One of needed improvement is change the turbo decoder. Other important improvement is implement the modules DUC and DDC in the FPGA of the ADC and DAC board. With these improvements ALOE would be able to execute the LTE applications in real time.

When the modules are enough optimized the ALOE Time Slot (TS) must be decreased (by dividing the LTE radio frame) in order to decrease the latency. Then the modules must be modified to support the radio frame division.

The LTE symbol level modules in the receiver could be implemented all in one to be executed by a GPU.

5.1. Platform expansion

The platform could be expanded adding other computer. The purpose of this new computer is increment the computing capacity. It will be integrated as the receiver processing chain in the DAS system implementation as is showed in the figure 5.1. It is expected that this computer could decode the most LTE higher bit rate configuration. It is composed of 24 PE (Two processors at 2.67GHz with 12 core) and one of the latest GPUs.

5.2. Environment

The Distributed Antenna System (DAS) concept is designed to save energy. The distance between the user and the antennas is less than conventional systems, then the transmitted power can be reduced. Also, having some BTS in a physical emplacement implies less consumption because it involves only one computing infrastructure. The SDR implementation can reduce the BTS power consumption. Some processing elements can be switched off if they are not necessary.

ALOE tool mapping is designed to map the processing modules in order to save energy and maximize the capacity of the platform.

The GPUs consume less power per Giga-Flop than CPUs. The table 5.1 shows a comparison between some processors. The showed power is the maximum Thermal Design Power (TDP) which is given by all the manufacturers.

To sum up, accelerate modules using graphics processors is more efficient than add more computers in the cluster.
Figure 5.1: Cluster connection for DAS implementation adding a server

<table>
<thead>
<tr>
<th>Processor</th>
<th>Application</th>
<th>Max TDP (W)</th>
<th>Capacity (GFLOPS)</th>
<th>TDP/Capacity (W/GFLOP)</th>
</tr>
</thead>
<tbody>
<tr>
<td>i5 2500 (4 × 3.3GHz)</td>
<td>Regular desktop</td>
<td>95</td>
<td>13</td>
<td>7.3</td>
</tr>
<tr>
<td>i7 2700K (8 × 3.5GHz)</td>
<td>Regular desktop</td>
<td>95</td>
<td>28</td>
<td>3.4</td>
</tr>
<tr>
<td>Opteron 6282 (16 × 2.6GHz)</td>
<td>Server</td>
<td>140</td>
<td>42</td>
<td>3.3</td>
</tr>
<tr>
<td>Xeon X5675 (12 × 3.1GHz)</td>
<td>Server</td>
<td>95</td>
<td>37</td>
<td>2.6</td>
</tr>
<tr>
<td>Quadro 4000 (256 × 1.9GHz)</td>
<td>High-end CAD</td>
<td>142</td>
<td>487</td>
<td>0.3</td>
</tr>
<tr>
<td>Quadro 6000 (448 × 2.3GHz)</td>
<td>High-end CAD</td>
<td>204</td>
<td>1030</td>
<td>0.2</td>
</tr>
<tr>
<td>Tesla C2075 (448 × 1.15GHz)</td>
<td>HPC</td>
<td>225</td>
<td>1030</td>
<td>0.2</td>
</tr>
<tr>
<td>GF GTX 560 (336 × 1.9GHz)</td>
<td>Gaming</td>
<td>150</td>
<td>638</td>
<td>0.2</td>
</tr>
<tr>
<td>GF GTX 680 (1536 × 2GHz)</td>
<td>Gaming</td>
<td>195</td>
<td>3090</td>
<td>0.1</td>
</tr>
</tbody>
</table>

Table 5.1: Processor power comparison
CHAPTER 6. CONCLUSIONS

SDR enables the implementation of flexible radios which could be updated easily. For SDR applications can be used personals computers as processing elements.

ALOE is a powerful tool. Programming the signal processing modules is easy. In spite of the fact that ALOE runs without any errors when the applications and the platform are light, some errors happens when the application and the platform are bigger. ALOE needs more tools in order to debug and find problems in big applications such as LTE.

LTE is the one of the most newest standards. The implementation of the downlink processing chain looks easy and simpler, however, it is not. The programmer must have specific and advanced knowledge of signal processing so as to implement advanced functionalities.

Reach a real time simulations is relatively easy in the theory. In practice, the software and hardware have some bugs (errors) and limitations. The most real time problems usually come because the execution is done at high level of the operating system.

The GPU execution time is lower than CPU when the parallelization capability of the algorithms is higher. However, the CUDA implemented accelerators have a bottleneck: host-device memory transfers. Integrating modules reduces the memory transfers, the it increases CUDA efficiency. Although the C code is quite similar to CUDA code, program CUDA involves a conceptual change from a serial to parallel execution. The main algorithms are quite similar in both processors though GPU algorithms need some improvements such as memory management and reduction algorithms.

CUDA model is very powerful in terms of computing. The actual high performance computing tends use GPU computing. On the other hand, the people who program CUDA algorithms must be very good at programming algorithms in order to take advantage of the GPU parallelization capability. It involves that the SDR programmers must be good in signal processing and in parallel algorithms.

The CUDA accelerated modules are not able to execute by all the possible ALOE processing elements such as ARM architecture processors, DSPs and FPGAs. It makes that ALOE portability decreases. Also, CUDA acceleration supposes a conceptual problem for ALOE mapping which does not contemplate the possibility to add hardware accelerators.

To sum up, adding some CUDA modules in ALOE open a new way
LTE downlink physical layer processing chain SDR application acceleration with GPUs
BIBLIOGRAPHY


[16] Dahlman, Erik; Parkvall, Stefan; Sköld, Johan; Beming, Per. 3G Evolution, HSPA and LTE for Mobile Broadband. 2nd Ed. Amsterdam: Elsevier, 2008.

[17] Ruetsch, Greg; Oster, Brent. nVISION 08 The world of visual computing[PDF]: Getting Started with CUDA. Santa Clara (CA): NVIDIA Corporation, 208.


APÈNDIXS
## APPENDIX A. LTE CONFIGURATION PARAMETERS

<table>
<thead>
<tr>
<th>Subframes</th>
<th>CRC block</th>
<th>Turbo code block size</th>
<th>Subblock Interleaver size</th>
<th>Rate matching Symbols</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>200</td>
<td>224</td>
<td>256</td>
<td>336</td>
</tr>
<tr>
<td>5</td>
<td>384</td>
<td>408</td>
<td>416</td>
<td>612</td>
</tr>
<tr>
<td>1/2/3/4/6/7/8/9</td>
<td>480</td>
<td>504</td>
<td>512</td>
<td>756</td>
</tr>
</tbody>
</table>

Table A.1: LTE128 application bit level parameters, 72 subcarriers, QPSK

<table>
<thead>
<tr>
<th>Subframe</th>
<th>CRC block</th>
<th>Turbo code Block size</th>
<th>Subblock Interleaver Block size</th>
<th>Rate matching Symbols</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>616</td>
<td>640</td>
<td>672</td>
<td>948</td>
</tr>
<tr>
<td>5</td>
<td>792</td>
<td>816</td>
<td>832</td>
<td>2448</td>
</tr>
<tr>
<td>1/2/3/4/6/7/8/9</td>
<td>984</td>
<td>1008</td>
<td>1024</td>
<td>3024</td>
</tr>
</tbody>
</table>

Table A.2: LTE256 application bit level parameters, 144 subcarriers, QPSK

<table>
<thead>
<tr>
<th>Subframe</th>
<th>CRC block</th>
<th>Turbo code Block size</th>
<th>Subblock Interleaver Block size</th>
<th>Rate matching Symbols</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1512</td>
<td>1536</td>
<td>1568</td>
<td>2274</td>
</tr>
<tr>
<td>5</td>
<td>1704</td>
<td>1704</td>
<td>1760</td>
<td>2550</td>
</tr>
<tr>
<td>1/2/3/4/6/7/8/9</td>
<td>2088</td>
<td>2112</td>
<td>2144</td>
<td>3150</td>
</tr>
</tbody>
</table>

Table A.3: LTE512 application bit level parameters, 300 subcarriers, QPSK

<table>
<thead>
<tr>
<th>Subframe</th>
<th>CRC block</th>
<th>Turbo code Block size</th>
<th>Subblock Interleaver Block size</th>
<th>Rate matching Symbols</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>3240</td>
<td>3264</td>
<td>3296</td>
<td>4824</td>
</tr>
<tr>
<td>5</td>
<td>3432</td>
<td>3456</td>
<td>3488</td>
<td>5100</td>
</tr>
<tr>
<td>1/2/3/4/6/7/8/9</td>
<td>4200</td>
<td>4224</td>
<td>4256</td>
<td>6300</td>
</tr>
</tbody>
</table>

Table A.4: LTE1024 application bit level parameters, 600 subcarriers, QPSK

<table>
<thead>
<tr>
<th>Slot</th>
<th>CRC block</th>
<th>Turbo code Block size</th>
<th>Subblock Interleaver Block size</th>
<th>Rate matching Symbols</th>
</tr>
</thead>
<tbody>
<tr>
<td>0/10</td>
<td>1448</td>
<td>1472</td>
<td>1504</td>
<td>2200</td>
</tr>
<tr>
<td>1</td>
<td>5160</td>
<td>5184</td>
<td>5216</td>
<td>7724</td>
</tr>
<tr>
<td>2/4/6/8/12/14/16/18</td>
<td>3048</td>
<td>3072</td>
<td>3104</td>
<td>4600</td>
</tr>
<tr>
<td>3/5/7/9/11/13/15/17/19</td>
<td>5352</td>
<td>5376</td>
<td>5408</td>
<td>8000</td>
</tr>
</tbody>
</table>

Table A.5: LTE2048 application bit level parameters, 1200 subcarriers, QPSK
APPENDIX B. LTE EXECUTION TIMES

B.1. LTE 128, 72 subcarriers, only PDSCH, QPSK

<table>
<thead>
<tr>
<th>PE</th>
<th>Name</th>
<th>Relinq</th>
<th>Mean</th>
<th>Var</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 1</td>
<td>gen9SK_SINK</td>
<td>71</td>
<td>37</td>
<td>7.04</td>
</tr>
<tr>
<td>2 0</td>
<td>gen9SK_UNCRC</td>
<td>198</td>
<td>89</td>
<td>13.87</td>
</tr>
<tr>
<td>2 3</td>
<td>gen9SK_turboDec</td>
<td>7769</td>
<td>7703</td>
<td>102.45</td>
</tr>
<tr>
<td>2 2</td>
<td>LTE_UNRM</td>
<td>525</td>
<td>489</td>
<td>14.98</td>
</tr>
<tr>
<td>2 1</td>
<td>gen9SK_sftDemap</td>
<td>206</td>
<td>126</td>
<td>11.06</td>
</tr>
<tr>
<td>2 1</td>
<td>gen9SK_depilot</td>
<td>1051</td>
<td>840</td>
<td>21.22</td>
</tr>
<tr>
<td>2 1</td>
<td>gen9SK_fft</td>
<td>1388</td>
<td>328</td>
<td>14.93</td>
</tr>
<tr>
<td>2 1</td>
<td>gen9SK_decyclic</td>
<td>1578</td>
<td>179</td>
<td>5.66</td>
</tr>
<tr>
<td>2 0</td>
<td>gen9SK_PSSdet</td>
<td>4008</td>
<td>3810</td>
<td>65.05</td>
</tr>
<tr>
<td>2 2</td>
<td>gen9SK_DDC</td>
<td>2445</td>
<td>1913</td>
<td>33.92</td>
</tr>
<tr>
<td>2 1</td>
<td>gen9SK_DUC</td>
<td>2428</td>
<td>841</td>
<td>14.49</td>
</tr>
<tr>
<td>2 2</td>
<td>gen9SK_cyclic</td>
<td>2669</td>
<td>219</td>
<td>20.83</td>
</tr>
<tr>
<td>2 1</td>
<td>gen9SK_ifft</td>
<td>2798</td>
<td>367</td>
<td>16.56</td>
</tr>
<tr>
<td>2 2</td>
<td>gen9SK_pilot</td>
<td>2971</td>
<td>297</td>
<td>24.57</td>
</tr>
<tr>
<td>2 2</td>
<td>gen9SK_mapper</td>
<td>3064</td>
<td>87</td>
<td>5.39</td>
</tr>
<tr>
<td>2 1</td>
<td>LTE_RM</td>
<td>3207</td>
<td>402</td>
<td>12.38</td>
</tr>
<tr>
<td>2 2</td>
<td>gen9SK_turboEnc</td>
<td>4071</td>
<td>1002</td>
<td>35.93</td>
</tr>
<tr>
<td>2 1</td>
<td>gen9SK_CRC</td>
<td>3304</td>
<td>87</td>
<td>5.56</td>
</tr>
<tr>
<td>2 1</td>
<td>gen9SK_SOURCE</td>
<td>3330</td>
<td>20</td>
<td>6.14</td>
</tr>
<tr>
<td></td>
<td>TOTAL:</td>
<td></td>
<td></td>
<td>18836</td>
</tr>
</tbody>
</table>

Table B.1: LTE128 measured execution time, in $\mu$s
B.2. LTE 128/256/512/1024/2048, only PDSCH, QPSK

<table>
<thead>
<tr>
<th>Modules</th>
<th>LTE 128</th>
<th>LTE 256</th>
<th>LTE 512</th>
<th>LTE 1024</th>
<th>LTE 2048</th>
</tr>
</thead>
<tbody>
<tr>
<td>gen9SK_SINK</td>
<td>21</td>
<td>28</td>
<td>45</td>
<td>81</td>
<td>135</td>
</tr>
<tr>
<td>gen9SK_UNCRC</td>
<td>124</td>
<td>188</td>
<td>343</td>
<td>647</td>
<td>1221</td>
</tr>
<tr>
<td>gen9SK_demuxDL</td>
<td>88</td>
<td>93</td>
<td>104</td>
<td>124</td>
<td>155</td>
</tr>
<tr>
<td>gen9SK_turboDe9</td>
<td>879</td>
<td>1620</td>
<td>3352</td>
<td>6551</td>
<td>11402</td>
</tr>
<tr>
<td>gen9SK_turboDe8</td>
<td>908</td>
<td>1637</td>
<td>3371</td>
<td>6574</td>
<td>12352</td>
</tr>
<tr>
<td>gen9SK_turboDe7</td>
<td>881</td>
<td>1599</td>
<td>3333</td>
<td>6517</td>
<td>12264</td>
</tr>
<tr>
<td>gen9SK_turboDe6</td>
<td>869</td>
<td>1594</td>
<td>3293</td>
<td>6474</td>
<td>12313</td>
</tr>
<tr>
<td>gen9SK_turboDe5</td>
<td>723</td>
<td>1330</td>
<td>2681</td>
<td>5392</td>
<td>12349</td>
</tr>
<tr>
<td>gen9SK_turboDe4</td>
<td>870</td>
<td>1606</td>
<td>3351</td>
<td>6544</td>
<td>12449</td>
</tr>
<tr>
<td>gen9SK_turboDe3</td>
<td>877</td>
<td>1602</td>
<td>3322</td>
<td>6561</td>
<td>12360</td>
</tr>
<tr>
<td>gen9SK_turboDe2</td>
<td>871</td>
<td>1612</td>
<td>3306</td>
<td>6544</td>
<td>12319</td>
</tr>
<tr>
<td>gen9SK_turboDe1</td>
<td>874</td>
<td>1613</td>
<td>3408</td>
<td>6708</td>
<td>12918</td>
</tr>
<tr>
<td>gen9SK_turboDe0</td>
<td>475</td>
<td>1106</td>
<td>2466</td>
<td>5262</td>
<td>12726</td>
</tr>
<tr>
<td>gen9SK_mux</td>
<td>145</td>
<td>262</td>
<td>455</td>
<td>819</td>
<td>1525</td>
</tr>
<tr>
<td>LTE_UNRM</td>
<td>449</td>
<td>945</td>
<td>1999</td>
<td>3815</td>
<td>7598</td>
</tr>
<tr>
<td>gen9SK_sftDemap</td>
<td>106</td>
<td>299</td>
<td>885</td>
<td>721</td>
<td>1695</td>
</tr>
<tr>
<td>gen9SK_depilot</td>
<td>898</td>
<td>1761</td>
<td>3964</td>
<td>8168</td>
<td>18567</td>
</tr>
<tr>
<td>gen9SK_fft</td>
<td>456</td>
<td>977</td>
<td>2472</td>
<td>4306</td>
<td>7801</td>
</tr>
<tr>
<td>gen9SK_deccyclic</td>
<td>256</td>
<td>501</td>
<td>1524</td>
<td>2374</td>
<td>5919</td>
</tr>
<tr>
<td>gen9SK_PSSdet</td>
<td>3885</td>
<td>4429</td>
<td>5548</td>
<td>7994</td>
<td>13364</td>
</tr>
<tr>
<td>gen9SK_DDC</td>
<td>1980</td>
<td>4131</td>
<td>7915</td>
<td>15865</td>
<td>32049</td>
</tr>
<tr>
<td>gen9SK_DUC</td>
<td>898</td>
<td>1941</td>
<td>3948</td>
<td>6718</td>
<td>13460</td>
</tr>
<tr>
<td>gen9SK_cyclic</td>
<td>211</td>
<td>416</td>
<td>781</td>
<td>2202</td>
<td>5282</td>
</tr>
<tr>
<td>gen9SK_ifft</td>
<td>386</td>
<td>779</td>
<td>1660</td>
<td>4164</td>
<td>8854</td>
</tr>
<tr>
<td>gen9SK_pilot</td>
<td>292</td>
<td>713</td>
<td>1277</td>
<td>2345</td>
<td>5570</td>
</tr>
<tr>
<td>gen9SK_mapper</td>
<td>88</td>
<td>158</td>
<td>319</td>
<td>627</td>
<td>1462</td>
</tr>
<tr>
<td>LTE_RM</td>
<td>399</td>
<td>797</td>
<td>1651</td>
<td>3284</td>
<td>6575</td>
</tr>
<tr>
<td>gen9SK_turboEnc</td>
<td>1011</td>
<td>1377</td>
<td>2187</td>
<td>3714</td>
<td>7378</td>
</tr>
<tr>
<td>gen9SK_CRC</td>
<td>89</td>
<td>163</td>
<td>312</td>
<td>608</td>
<td>1199</td>
</tr>
<tr>
<td>gen9SK_SOURCE</td>
<td>22</td>
<td>34</td>
<td>46</td>
<td>75</td>
<td>138</td>
</tr>
<tr>
<td><strong>TOTAL</strong></td>
<td><strong>20031</strong></td>
<td><strong>35311</strong></td>
<td><strong>69318</strong></td>
<td><strong>131778</strong></td>
<td><strong>263399</strong></td>
</tr>
</tbody>
</table>

Table B.2: LTE measured execution times, in $\mu$s
APPENDIX C. C AND CUDA CODE EXAMPLES

C.1. FFT3W sample code

```c
#include <fftw3.h>

[...]
// Define needed arrays and variables
fftw_complex in[length];
fftw_complex out[length];
fftw_plan plan;
[...]
// Create FFT plan
plan = fftw_plan_dft_1d(FFT_SIZE, in, out, FFTW_FORWARD, FFTW_ESTIMATE);
[...]
// Translate input data to specific data type
for (i = 0; i < FFT_SIZE; i++) {
    in[i][0] = in_i[i];  // Real part
    in[i][1] = in_q[i];  // Imaginary part
}
[...]
// can call many times fft plan without modify the plan
fftw_execute(plan);
[...]
// Translate the output data
for (i = 0; i < FFT_SIZE; i++) {
    out_i[i] = out[i][0];  // Real part
    out_q[i] = out[i][1];  // Real part
}
```

Figure C.1: Basic code to call and use the FFTW3 library
C.2. CUFFT sample code

```c
#include <cufft.h>
#include <cuda_runtime.h>
...
{
    // Define needed arrays and variables
    cufftHandle plan;
    cufftComplex *complex_in;  // Device complex data pointer
    cufftComplex *complex_out;

    // Device memory allocation functions
    cudaMalloc((void**)&complex_in, sizeof(cufftComplex)*FFT_num*FFT_size);
    cudaMalloc((void**)&complex_out, sizeof(cufftComplex)*FFT_num*FFT_size);
    ...  
    // Plan creation
    cufftPlan1d(&plan, FFT_size, CUFFT_C2C, FFT_num);
    ...  
    // Translation in data kernel call (not included in CUFFT library)
    setComplex<<<blocksPerGrid, threadsPerBlock>>>(in_buffer_i, in_buffer_q, complex_in, len);
    ...  
    // FFT function call
    cufftExecC2C(plan, complex_in, complex_out, CUFFT_FORWARD);
    ...  
    // Translation out data kernel call (not included in CUFFT library)
    getComplex<<<blocksPerGrid, threadsPerBlock>>>(out_buffer_i, out_buffer_q, complex_out, len);
    ...  
    // Free memory
    cufftDestroy(plan);
    cudaFree(complex_in);
    cudaFree(complex_out);
    ...
}
```

Figure C.2: Basic code to call and use the CUFFT library
C.3. Memory algorithm sample code

```c
// Definition shared memory size
#define MAXTXB 512  // Maximum threads per block
#define MAXFILTERLEN 32

__global__ void conv (float* inSamples, float* outSamples, 
                        float* stat, float* filter, int filterlen, int inputlen)
{
    // Kernel thread identification
    int idx = blockIdx.x * blockDim.x + threadIdx.x;

    // Block threat identification
    int i = threadIdx.x;
    int j;
    float result = 0.0;

    // Shared memory definitions
    __shared__ float shared_filter[MAXFILTERLEN];
    __shared__ float shared_inSamples[MAXTXB + MAXFILTERLEN];

    // First Stage: Copy data from global to memory
    if (i<filterlen) // Copy filter
        shared_filter[i]=filter[i];
    if (idx<filterlen) // Copy convolution stat
        shared_inSamples[i]=stat[idx];
    else if (idx<inputlen) // Copy input data
        shared_inSamples[i]=inSamples[idx-filterlen];
    else if (i<filterlen) // Copy the tail
        shared_inSamples[i+blockDim.x]=inSamples[idx+blockDim.x];

    // Synchronize the block threads
    __syncthreads();

    // Second Stage: Process data (convolution )
    for (j=0; j<FILTERLEN; j++){
        result += shared_inSamples[i+j]*shared_filter[filterlen-1-j];
    }

    // Third stage: Return data to the global memory
    if (idx<filterlen) // Update stat
        stat[idx]=inSamples[inputlen-filterlen+idx];
    outSamples[idx] = result;  // Return Result
}
```

Figure C.3: Convolution code as example of memory managing
C.4. Memory algorithm sample code

```c
#define MAXTXB 512 // Maximum threads per block

__global__ void detection ( int len, int * output, float * input, 
                           float threshold)
{
    // Thread index definition
    int index = blockDim.x * blockIdx.x + threadIdx.x;

    // Shared memory definitions
    __shared__ float sum [MAXTXB]; // Array for save the sum
    __shared__ float max [MAXTXB]; // Array for maximum Values
    __shared__ int imax [MAXTXB]; // Array for maximum index

    // Local variables to support the algorithms
    float local_sum = 0;
    float local_max = 0;
    int local_imax = 0;
    int reduction;
    float k = (float)len; // Number of samples
    float value;

    // Set input pointer to global memory
    float * inptr = input + reduction*index;

    // Perform 1st reduction reading from global memory
    // And save the 1st reduced data in shared mem.
    reduction = len/blockDim.x;
    for (int i=0; i<reduction; i++)
    {
        // Perform find maximum and sum algorithm
        value = inptr[i];
        local_sum += value;
        value *= (value<0)? -1:1; // make absolute
        if (local_max < value){
            local_max = value;
            local_imax = reduction*index + i;
        }
    }

    // Save data in shared memory
    sum[index] = local_sum;
    max[index] = local_max;
    imax[index]= local_imax;
    len /= reduction;

    // Reduction Algorithm
    while (len != 1){
        // Set the new reduction factor
        reduction = (len%2==0)?2:(len%3==0)?3:(len%5==0)?5:1;
        len /= reduction; // Reduce threats
        __syncthreads(); // Synchronize threads

        // Process if the thread is not reduced
        if (index<len){
            // Perform find maximum and sum algorithm
            local_max = max[index];
            for (int i = 1; i<reduction; i++)
            {
                value = max[index+len*i];
                sum[index] += sum[index+len*i];
                if (local_max < value){
                    imax[index] = imax[index+len*i];
                    local_max = value;
                }
            }
            max[index] = local_max;
        }
    }

    // Save the detected delay in the delay array
    if (index==0 && max[0]/(sum[0]/k))>threshold )
        delay_dd[0] = imax[0];
}
```

Figure C.4: Detection kernel code, thread reduction algorithm