Guilherme Baltar Lopes Cardoso Lourenço

Sonda Espectral Distribuída de Tempo-Real

Real-Time Distributed Spectral Probe

Guilherme Baltar Lopes Cardoso Lourenço

Sonda Espectral Distribuída de Tempo-Real

Real-Time Distributed Spectral Probe

"*In theory, there is no difference between theory and practice. In practice, there is.*"

— Benjamin Brewster

Universidade de Aveiro
2023

Guilherme Baltar Lopes Cardoso Lourenço

Sonda Espectral Distribuída de Tempo-Real

Real-Time Distributed Spectral Probe

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia Eletrónica e Telecomunicações, realizada sob a orientação científica do Doutor Arnaldo Silva de Rodrigues Oliveira, Professor Associado do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro, e do Doutor Nuno Miguel Gonçalves Borges de Carvalho, Professor Catedrático do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro.

Este trabalho é financiado pela Fundação para a Ciência e Tecnologia (FCT) e Ministério da Ciência, Tecnologia e Ensino Superior (MCTES) através de fundos nacionais.

Este trabalho também é financiado por fundos comunitários no âmbito do projeto UIDB/50008/2020-UIDP/50008/2020, FEDER através do Programa COMPETE 2020 [Projetos IMMINENCE nº 112314 (POCI-01-0247-FEDER-112314) e POWER nº 070365 (POCI-01-0247-FEDER-070365)].

o júri / the jury

presidente / president

Prof. Doutor José Nuno Panelas Nunes Lau
Professor Associado da Universidade de Aveiro

vogais / examiners committee

Prof. Doutor Marco Alexandre Cravo Gomes
Professor Auxiliar da Faculdade de Ciências e Tecnologia da Universidade de Coimbra (Arguente)

Prof. Doutor Arnaldo Silva Rodrigues de Oliveira
Professor Associado da Universidade de Aveiro (Orientador)

agradecimentos / acknowledgements

Aos meus pais, um grande sentido de gratidão por me apoioarem incondicionalmente em todo o meu percurso de vida e fazerem-me sempre acreditar em mim. Obrigado, mãe, obrigado pai.

À minha namorada Beatriz, por todo o apoio nesta jornada.

Aos meus amigos e colegas que fizeram parte deste percurso em especial aos grandes amigos que fiz pelo caminho Gonçalo, Inês, Henrique, Diogo, Manuel e Leonardo, que tornaram este percurso muito mais alegre.

Um grande obrigado ao Luís e ao Francisco pela ajuda e orientação no projeto, em especial no desenvolvimento da plataforma. Aos meus restantes colegas do grupo de sistemas de rádio, Fábio, José e Samuel, pelo companheirismo e ajuda.

Ao Professor Doutor Hugerles Silva, pela oportunidade de iniciar a minha carreira na investigação científica e integração no grupo de Sistemas de Rádio. Obrigado por toda a aprendizagem, orientação e amizade.

Agradeço especialmente ao Professor Doutor Arnaldo Oliveira, meu orientador nestes últimos dois anos, pelo apoio e oportunidades que me deu, inclusivamente a de me incluir neste projeto.

Um agradecimento ao Doutor Ricardo Figueiredo pela ajuda, explicações e conversas sobre medições e recetores de Rádiofrequência.

Do grupo de Network Architectures and Protocols um agradecimento ao Doutor Pedro Rito pela ajuda ao longo do projeto, ao Eurico Dias pela ajuda e conversas sobre Network Sockets e ao Rodrigo Rosmaninho pela ajuda e conversas sobre Benchmarking e o Kernel de Linux.

Aos técnicos do Instituto de Telecomunições Paulo Gonçalves, Nuno Silva e António Correia pela ajuda e boa disposição.

À Universidade de Aveiro, ao Departamento de Eletrónica, Telecomunicações e Informática e ao Instituto de Telecomunicações por fornecerem as condições necessárias de trabalho e aprendizagem.

A todos, muito obrigado.

Palavras Chave

Sonda Espectral, Rádio Definido por Software, FPGA, Processamento Digital de Sinal em Tempo Real.

Resumo

Os desenvolvimentos tecnológicos nas comunicações sem fios, em particular a implementação das comunicações móveis da quinta geração (5G) e advento de posteriores, juntamente com a rápida expansão da Internet das Coisas (IoT) impulsionada pela adoção da norma IEEE 802.11ax (conhecida comercialmente como WiFi 6 e WiFi 6E), estão a conduzir a um aumento significativo do número de dispositivos sem fios interligados. Até 2025, prevê-se que 55,7 mil milhões de dispositivos estejam ligados em todo o mundo, 75% dos quais estarão ligados a uma plataforma IoT. Este crescimento significativo tornará cada vez mais difícil lidar com o já escasso espetro eletromagnético, à medida que as bandas existentes se tornam cada vez mais congestionadas. Neste contexto, as plataformas de sensorização de espetro tornaram-se ferramentas essenciais para a análise, monitorização e gestão do espetro.

Neste trabalho, é apresentada uma nova sonda espetral de rádio distribuída de tempo real. Tanto quanto é do conhecimento do autor, este é o primeiro trabalho em que é apresentada uma sonda espetral remotamente reconfigurável e baseada em Field Programmable Gate Array (FPGA) com uma interface multi-Gigabit aberta, escalável e implementável num cenário espacialmente distribuído. Foi configurada uma plataforma determinística de alto desempenho, capaz de executar algoritmos personalizados de processamento de sinais digitais em tempo real, optimizados para a arquitetura da Unidade Central de Processamento (CPU). O sistema foi testado e validado num ambiente controlado utilizando equipamento de laboratório de Rádio-Frequência (RF), bem como num cenário de deteção espetral multi-banda em ambientes interiores. A sonda desenvolvida apresenta um patamar de ruído de pelo menos -89 dBm numa gama de frequências de 0,8 a 3,5 GHz e é capaz de receber e processar sinais RF que atingem uma taxa de até 7,86 Gbit/s.

Keywords Spectrum Sensing, Software-Defined Radio, FPGA, Real-Time Digital Signal Processing.

Abstract Technological developments in wireless communications, particularly the deployment of Fifth Generation (5G) mobile communications and emergence of later ones, alongside the rapid expansion of the Internet of Things (IoT) driven by the adoption of IEEE 802.11ax (commercially known as WiFi 6 and WiFi 6E), are leading to a significant increase in the number of connected wireless devices. By 2025, 55.7 billion devices are expected to be connected worldwide, 75% of which will be connected to an IoT platform. This significant growth will make addressing the already scarce electromagnetic spectrum increasingly challenging as the existing bands become increasingly crowded. In this context, spectrum sensing platforms have become essential tools for spectrum analysis, monitoring, and management.
In this work, a novel distributed real-time radio spectral probe is presented. To the best of the author's knowledge, this is the first work in which a remotely reconfigurable and Field Programmable Gate Array (FPGA) wideband spectral probe with an open multi-Gigabit interface, scalable and deployable in a spatially distributed scenario, is presented. A high-performance, deterministic platform was configured, capable of running custom real-time digital signal processing algorithms optimized for the Central Processing Unit (CPU) architecture. The system was tested and validated in a controlled environment using Radio Frequency (RF) laboratory equipment, as well as in an indoor multi-band spectral sensing scenario. The developed probe features a noise floor of at least -89 dBm over a frequency range of 0.8 to 3.5 GHz and is capable of receiving and processing RF signals that reach a rate of up to 7.86 Gbit/s.

Contents

Contents i

List of Figures iii

List of Tables v

Acronyms vii

1 Introduction 1
- 1.1 Scope . . . 1
- 1.2 Motivation . . . 2
- 1.3 Objectives . . . 3
- 1.4 Publication . . . 5
- 1.5 Document Structure . . . 5

2 Spectral Probes 7
- 2.1 Introduction . . . 7
- 2.2 Software Defined-Radio Receivers . . . 7
  - 2.2.1 Direct-Conversion Receivers . . . 8
- 2.3 Spectrum Sensing . . . 9
  - 2.3.1 Spectrum Sensing Techniques . . . 9
  - 2.3.2 Spectrum Sensing Hardware . . . 10
- 2.4 RF Receiver Performance Characteristics . . . 10

3 System Architecture and Development Platform 17
- 3.1 Introduction . . . 17
- 3.2 General Architecture . . . 17
- 3.3 SDR Platform . . . 18
  - 3.3.1 RF Transceiver . . . 18
  - 3.3.2 FPGA . . . 19
- 3.4 Fronthaul . . . 19
  - 3.4.1 Radio over Ethernet . . . 19
- 3.5 Centralized Processing . . . 20
- 3.6 Analog RF Front-End and Antenna . . . 21

3.7 Final Architecture 22

4 Implementation **23**
4.1 Introduction 23
4.2 SDR Platform 23
4.2.1 Loopback Design 24
4.2.2 Standard Design 25
4.2.3 Embedded Linux 25
4.2.4 Network Sockets 25
4.3 Centralized Processing Implementation 26
4.3.1 Real-Time Digital Signal Processing 27
Digital Filtering 28
4.3.2 Signal Power Calculation 30
4.3.3 Remote Access 30
4.4 Analog RF-front end 31
4.5 Physical Setup 32
4.5.1 Radio Frequency Instruments 33
4.5.2 Fronthaul 34
4.5.3 Final Setup 35

5 Tests and Results **37**
5.1 Introduction 37
5.2 Loopback Tests 37
5.3 Centralized Processing Tests 39
5.3.1 Spectrum Analysis 39
5.3.2 Signal Power Measurements 41
5.3.3 Signal Filtering Results 44
5.3.4 Execution Time Results 45
5.4 Spectrum Sensing Campaign 49

6 Conclusion **53**
6.1 Final Remarks 53
6.2 Future Work 53

A Real-Time Kernel Tunning and Low Jitter Computing Techniques **55**

B High-Performance DSP Software Module **63**

C Numerical Approximation and Optimized Implementation for Signal Power Calculation **69**

D System Implementation for On-Site Deployment **75**

Bibliography **77**

List of Figures

1.1 Wall box and Smart Lamp Post. 2
1.2 Aveiro Tech City Living Lab map in Aveiro. 2
1.3 Allocation of the electromagnetic spectrum in Portugal for the Fourth Generation Long Term Evolution bands and the Fifth Generation n78 band - retrieved from [4]. 4

2.1 Ideal SDR receiver block diagram. 8
2.2 DCR receiver block diagram with two IQ channels and without the RF front-end. 8
2.3 Cascaded amplifiers block diagram - adapted from [56]. . . . 13
2.4 Output power as a function of input of a RF receiver - retrieved from [55]. 15

3.1 High-level block diagram of distributed scenario. 17
3.2 eCPRI packet structure. 20
3.3 Block diagram of analog RF front-end and antenna. 21
3.4 Final system architecture block diagram. 22

4.1 Block diagram of Loopback Design implementation. 24
4.2 2100 MHz 4G LTE n1 FDD downlink band allocation in Portugal - retrieved from [4]. 28
4.3 Frequency response of a 12th order IIR filter implemented with single and double precision. 29
4.4 Frequency response of a 29th order FIR filter implemented with single and double precision. 29
4.5 Block diagram of the setup used to enable remote access. . . 31
4.6 QPL9057 evaluation board. 32
4.7 UWB antenna. 32
4.8 Zynq UltraScale+ MPSoC ZCU102 Evaluation Kit [50]. . . . 33
4.9 ADRV9371-WPCBZ radio card [51]. 34
4.10 Si570 Evaluation Board [52]. 34
4.11 Bidirectional SFP+. 35
4.12 Loopback SFP+. 35

4.13 Laboratorial setup of the spectral probe with the LNA and antenna. 36

5.1 Internal loopback results. 38
5.2 RoE loopback results. 38
5.3 Laboratory setup for the loopback tests. 39
5.4 FFT of the received signal (N = 2.028 ksample) on the server. 40
5.5 FFT of received signal (N = 20.28 sample) on the server. . . 40
5.6 FFT of the received signal (N = 202.8 ksample) on the server. 40
5.7 FFT of the received signal (N = 2028.0 ksample) on the server. 40
5.8 Spectrum of a 100 MHz signal before and after filtering using an FIR filter with 37 taps and 50 MHz cutoff frequency. . . . 45
5.9 Spectrum of a 100 MHz signal before and after filtering using an FIR filter with 100 taps and 50 MHz cutoff frequency. . . 46
5.10 Spectrum of a 100 MHz signal before and after filtering using an FIR filter with 37 taps and 20 MHz cutoff frequency. . . . 47
5.11 Spectrum of a 100 MHz signal before and after filtering using an FIR filter with 100 taps and 20 MHz cutoff frequency. . . 48
5.12 Spectral occupancy of 4G, and 2.4 GHz ISM bands. 50
5.13 Spectral occupancy of 5G bands. 51

B.1 Graph of the average execution time of the optimized and non-optimized FIR filter implementations as a function of the number of filter taps featuring linear regressions. 66

C.1 Absolute error of the signal power approximation as a function of the signal approximation power for signals with 2028 and 20280 samples. 72

D.1 Field-ready system setup. 76

List of Tables

2.1 Comparison of spectral probes based on FPGA with transceiver, USRP, and SA. 11

4.1 RF chain cascaded NF, and noise floor calculations for different frequencies. 31

4.2 UWB antenna gain at different frequencies. 32

5.1 Measurements of received signal power and absolute error for different signal powers without digital and RF chain gains. . . 42

5.2 Measurements of received signal power and absolute error for different signal powers with 30 dB digital reconfigurable gain only. 42

5.3 Measurements of received signal power and absolute error for different signal powers with 30 dB digital reconfigurable gain and 21.2 dB gain from the RF chain. 43

5.4 Measured RF chain gain and noise floor values at different frequencies. 44

5.5 Execution times for the *recvfrom* C function. Each of the 5 trials received 1000 packets. 46

5.6 Execution times for the byte shifting operation. Each of the 5 trials processed 10 million packets. 47

5.7 Execution times for the whole software pipeline: receiving, byte shifting, filtering (with 37 filter taps), and signal power calculation. Each of the 5 trials processed 1000 packets . . . 49

A.1 Execution performance time using the *clock_gettime()* function from the C Time library. Each of the five trials made 100 million function calls. 60

A.2 Execution performance in cycles and time using the custom function based on the *rdtscp()* instruction. Each of the five trials made 100 million function calls. 61

B.1 Execution performance time of the custom FIR filter function for different filter tap counts. Each of the five trials made 1 million function calls. 65

C.1 Execution performance time for different signal power calculation implementations considering a signal with 2028 IQ samples. Each of the five trials made 10 million function calls. 73

Acronyms

3GPP 3rd Generation Partnership Project. 32, 49

4G Fourth Generation. iii, iv, 1, 4, 28, 41, 43, 49, 50, 53

5G Fifth Generation. iii, iv, 1, 2, 4, 18, 19, 32, 41, 43, 49, 51, 53

ADC Analog-to-Digital Converter. 1, 7, 18, 24, 25, 37, 69, 71

API Application Programming Interface. 26

APU Accelerated Processing Unit. 30, 75

ATCLL Aveiro Tech City Living Lab. 1, 35, 43, 75

AVX Advanced Vector Extensions. 6, 20, 27, 47, 54, 63, 64, 69, 72

BBU BaseBand Unit. 19

BIOS Basic Input/Output System. 55

CLI Command Line Interface. 25, 30

CPRI Common Public Radio Interface. 19

CPU Central Processing Unit. 1, 20, 26, 27, 55–59, 61, 63

CR Cognitive Radio. 9

CU Centralized Unit. 24

DAC Digital-to-Analog Converter. 24, 37, 38

DCR Direct-Conversion Receiver. iii, 7–9, 18

DDR4 Double Data Rate 4. 20

DSA Dynamic Spectrum Access. 9

DSP Digital Signal Processing. 6, 20, 26, 27, 40, 54

DTFT Discrete Time Fourier Transform. 27, 28

eCPRI Evolved Common Public Radio Interface. iii, 19, 20, 24, 26, 27, 38, 39, 71

ED Energy Detection. 9, 30

EVM Error Vector Magnitude. 37, 38

FDD Frequency Division Duplex. iii, 18, 28

FFT Fast Fourier Transform. iv, 28, 39–41, 48, 63

FIFO First In First Out. 24

FIR Finite Impulse Response. iii, iv, vi, 6, 24, 27–29, 45–48, 54, 63–66

FMC FPGA Mezzanine Card. 33

FPGA Field Programmable Gate Array. v, 1, 5, 10, 11, 18–20, 23, 24, 26, 32, 33, 37, 47, 75

GPIO General Purpose Input/Output. 75

GRUB GRand Unified Bootloader. 56

IF Intermediate Frequency. 8

IIR Infinite Impulse Response. iii, 28, 29

IoT Internet of Things. 1–3

IP Intellectual Property. 23–25, 32

IQ In-Phase and Quadrature. iii, vi, 8, 20, 23, 24, 27, 30, 37–39, 47, 48, 50, 69, 70, 72, 73

ISM Industrial, Scientific, and Medical. iv, 3, 43, 49, 50, 53

JTAG Joint Test Action Group. 32, 75

LiDAR Light Detection and Ranging. 1

LNA Low-Noise Amplifier. iv, 13, 21, 31, 35, 36, 41–43, 45

LO Local Oscillator. 8

LPF Low-Pass Filter. 28

LTE Long Term Evolution. iii, 1, 4, 28, 32, 41, 43, 49, 53

MDS Minimum Discernible Signal. 13, 14

MPSoC Multiprocessor System on Chip. iii, 3, 23, 32, 33, 75

Msps Megasamples per second. 1, 18, 23, 41

NF Noise Figure. v, 12, 13, 21, 31

NR New Radio. 18, 32, 41, 43, 49

OFDM Orthogonal Frequency-Division Multiplexing. 41

OSI Open System Interconnection. 25

PAPR Peak to Average Power Ratio. 41

PC ID Protocol Control Identifier. 19

PRBS Pseudorandom Binary Sequence. 40, 41

PU Primary User. 9

QAM Quadrature Amplitude Modulation. 8, 37, 40, 41, 44, 71

RF Radio Frequency. iii, v, 1, 5, 7–11, 13, 15, 17, 18, 21, 23, 26, 31, 37, 38, 42–44, 49, 50, 53

RoE Radio over Ethernet. 5, 19, 23–25, 32, 49

RRU Remote Radio Head. 20

SA Spectrum Analyzer. v, 10, 11, 33, 37

SDR Software-Defined Radio. iii, 1, 3, 5, 7–10, 17, 20, 21, 23, 30, 39, 53

SDRAM Synchronous Dynamic Random-Access Memory. 20

SEQ ID Sequence Identifier. 19

SFP Small Form-factor Pluggable. iii, 20, 24, 33–35, 37, 38

SIMD Single Instruction Multiple Data. 6, 20, 27, 29, 48, 54, 63, 64, 69, 72

SMA SubMiniature version A. 33, 35

SNR Signal-To-Noise Ratio. 9, 12, 13

SSH Secure Shell. 30

SU Secondary User. 9

TCC Time Coordinated Computing. 56

TDD Time Division Duplex. 18

TSC Time Stamp Counter. 60, 61

TSN Time Sensitive Networking. 56

UART Universal Asynchronous Receiver/Transmitter. 30, 32, 75

UDP User Datagram Protocol. 25

USB Universal Serial Bus. 1, 32, 75

USRP Universal Software Radio Peripheral. v, 10, 11

UWB Ultra-Wideband. iii, v, 31, 32

VSA Vector Signal Analyzer. 33, 37, 38, 40

VSG Vector Signal Generator. 33, 41

WLAN Wireless Local-Area Network. 14, 49

Chapter 1

Introduction

1.1 Scope

This work is inserted in the IMMINENCE project, which carries on the work of the Aveiro Tech City Living Lab (ATCLL) project. The ATCLL project developed an advanced communications infrastructure, an urban data management and innovation platform. A significant part of the infrastructure is composed of 44 lamp posts or wall boxes, called nodes, spread throughout the city of Aveiro and connected by 16 km of fiber optics that converge on a data center [1]. Each of these nodes is equipped with sensors such as spectral probes, traffic radars, Light Detection and Ranging (LiDAR), and video cameras. The spectral probes are connected to low-cost commercial Software-Defined Radio (SDR) ADALM-PLUTO connected to a Raspberry Pi computer to process and collect data [1]. Figure 1.2 depicts the ATCLL in the map of Aveiro, its fiber connections, and the different nodes placed in strategic locations in the city.

While the ADALM PLUTO is a versatile tool, it was designed as a learning platform and has some limitations. Based on the AD9396 transceiver with 12-bit Analog-to-Digital Converter (ADC)s, it offers a tuning range of 325 MHz to 3800 MHz and a configurable receiver bandwidth of 200 kHz to 20 MHz. These specifications, especially the bandwidth, are somewhat limited; for example, it is insufficient to cover an entire Fourth Generation (4G) Long Term Evolution (LTE) band. In addition, the device's reliance on a Universal Serial Bus (USB) 2.0 interface imposes additional constraints, particularly when it comes to efficient data streaming, with a maximum of 4 Megasamples per second (Msps), hindering real-time data acquisition [2].

In this context, the aim of this project is to develop a spectrum sensing platform with higher bandwidth, resolution, and real-time data transfer and processing.

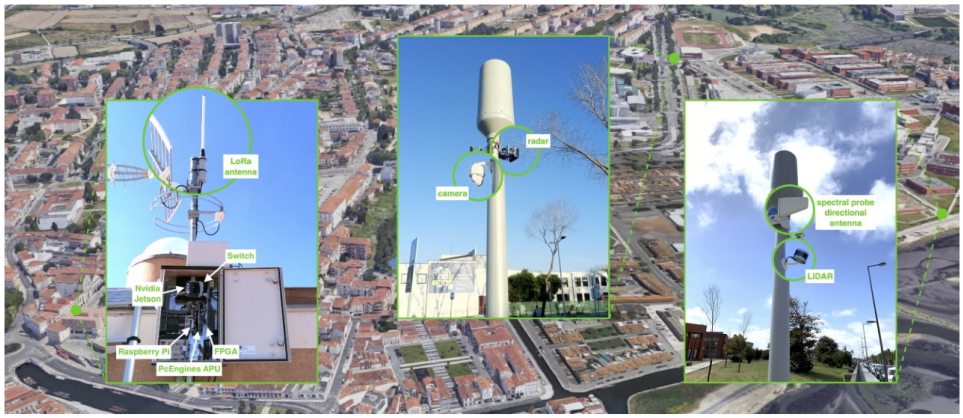

Figure 1.1: Wall box and Smart Lamp Post.

Figure 1.2: Aveiro Tech City Living Lab map in Aveiro.

1.2 Motivation

Technological developments in wireless communications, particularly the emergence of Fifth Generation (5G) mobile communications and beyond, alongside the rapid expansion of the Internet of Things Internet of Things (IoT) driven by the adoption of IEEE 802.11ax (commercially known as

WiFi 6 and WiFi 6E), are leading to a significant increase in the number of connected wireless devices. By 2025, 55.7 billion devices are expected to be connected worldwide, 75% of which will be connected to an IoT platform [3]. This significant growth will make addressing the already scarce electromagnetic spectrum increasingly challenging as the existing bands become increasingly crowded.

The emergence of new licensed mobile communications technologies narrows the bandwidth available for unlicensed communications. This leads to the congestion of unlicensed bands.

The congestion of the unlicensed 2.4 GHz Industrial, Scientific, and Medical (ISM) band is an example of this overload and can be noted by the technologies operating in it, namely: IEEE 802.11 (better known as WiFi), IEEE 802.15.4 (the basis of the physical and media access control layer for low-rate wireless personal area and industrial networks such as ZigBee, ISA100.11a, and WirelessHART), IEEE 802.15.1 (commercialized as Bluetooth) and its evolution Bluetooth Low Energy [5]. In addition to these technologies, the expansion of the LoRa communication protocol to the 2.4 GHz band was announced in 2020 [6].

Spectrum monitoring is therefore fundamental to the evolution of wireless communications, especially in environments with a high density of connected devices, such as cities and smart industries.

In addition to spectrum analysis, monitoring, and management, the following use cases also arise:

- Experimenting with various spectrum sensing algorithms with real data [23].
- Capture datasets with different climatic conditions, such as temperature, humidity, and weather, in order to study their impact on the communication channel [7].
- Capture large datasets to feed emerging artificial intelligence techniques [7].

1.3 Objectives

The main objective of this work is to implement a flexible real-time spectral probe. To this end, a set of smaller milestones were defined. These milestones are listed below:

1. Familiarization with the current spectral probe platform.
2. Requirements assessment and architecture delineation.
3. Familiarization with the devices and tools for developing and validating Multiprocessor System on Chip (MPSoC) and SDR-based systems.

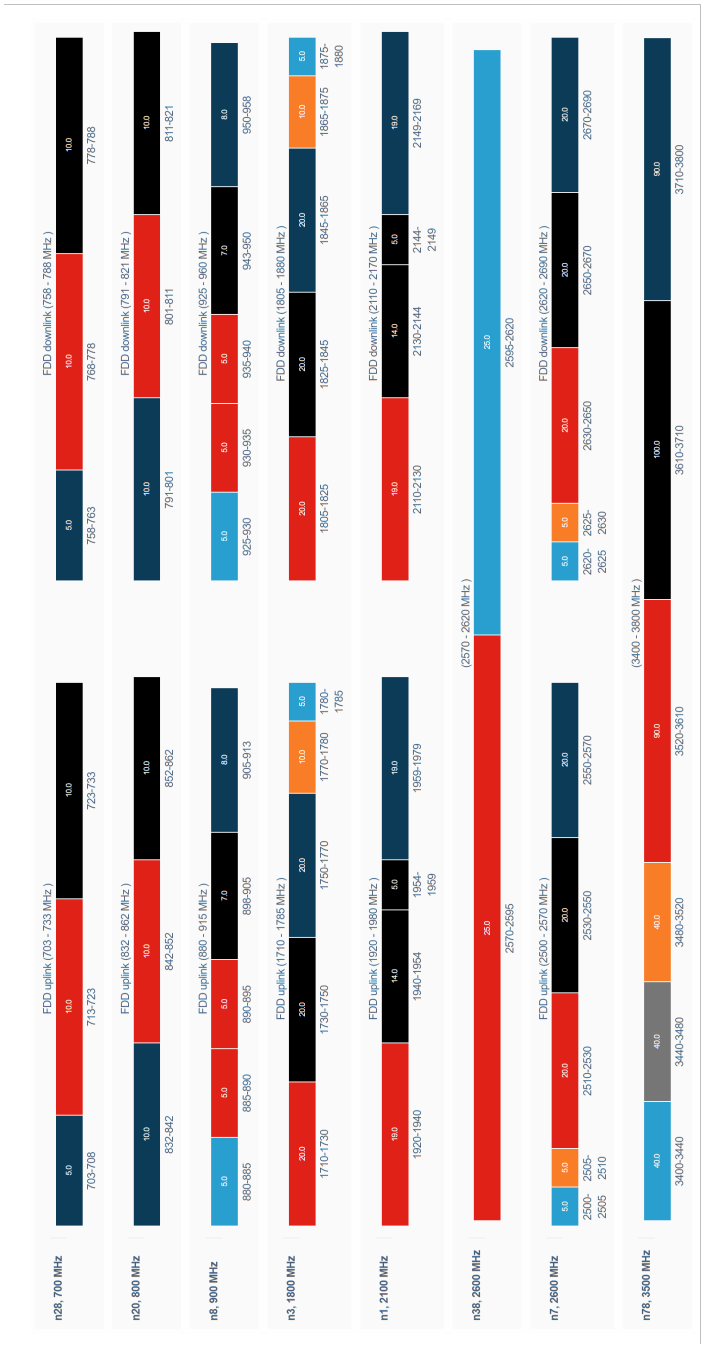

Figure 1.3: Allocation of the electromagnetic spectrum in Portugal for the Fourth Generation Long Term Evolution bands and the Fifth Generation n78 band - retrieved from [4].

4. Setup of a high-performance real-time central processing unit capable of receiving and processing radio signals.
5. Familiarization with RF laboratory instrumentation and measurement techniques.
6. Validation of the spectral probe by conducting a multi-band spectral occupancy campaign.
7. Migration of the lab-based system to a field-ready version.

1.4 Publication

The work done in the scope of this dissertation resulted in the acceptance of one paper focused on the implementation of a flexible real-time spectral probe in a spectrum sensing scenario. The accepted paper is listed below:

An RoE-based Real-Time Radio Spectral Probe

G. B. L. C. Lourenço, F. A. Serôdio, L. F. Almeida, H. S. Silva, and A. S. R. Oliveira in *IEEE Radio Wireless Week (RWW)*, 2024

This paper presents a novel, RoE based, real-time, flexible radio spectral probe designed to address the growing demands for diverse wireless communication systems. The proposed system architecture encompasses a remotely reconfigurable design with a (SDR) platform, integrating Field Programmable Gate Array (FPGA), radio frequency (RF) front-end, and an open fronthaul. A simultaneous multi-band spectrum occupancy measurement campaign in an indoor environment using the proposed spectral probe is conducted. To the best of the author's knowledge, this is the first work in which a real-time, open, and remotely reconfigurable FPGA-based spectral probe is presented.

1.5 Document Structure

The remainder of this document is organized as follows:

- **Chapter 2, Spectral Probes** - An overview of spectrum sensing techniques, the architecture of RF SDR receivers and some key concepts and figures of merit of RF receivers are presented.
- **Chapter 3, System Architecture and Development Platform** - A high-level architecture of the system is presented, followed by a breakdown of the various subsystems, and ending with a lower-level architectural representation.

- **Chapter 4, Implementation** - The details of the implementation and integration of the various subsystems that make up the entire spectral probe are detailed.
- **Chapter 5, Tests and Results** - The testing procedures utilized to validate the system are presented, and the results are analyzed and discussed.
- **Chapter 6, Conclusion** - Concluding remarks are made on the work carried out, followed by a summary of potential future work.

In addition to the chapters described, the following appendices complement the work conducted with some detailed technical elements.

- **Appendix A, Real-Time Kernel Tunning and Low Jitter Computing Techniques** - A real-time low-jitter computing platform is configured for benchmarking high-performance applications.
- **Appendix B, High-Performance DSP Software Module** - A software implementation of a high-performance single precision FIR filter of arbitrary order using AVX-512 SIMD operations, and other non-optimized DSP functions are presented.
- **Appendix C, Numerical Approximation and Optimized Implementation for Signal Power Calculation** - A numerical approach for calculating signal power is described, its associated error is studied, and a software implementation is detailed, optimized, tested, and validated.
- **Appendix D, System Implementation for On-Site Deployment** - A walkthrough of the design migration process to a different, field-ready board.

Chapter 2

Spectral Probes

2.1 Introduction

For a better understanding of the design and implementation of an SDR-based spectral sensing platform, this chapter presents an overview of some key concepts and techniques. First, the ideal architecture of an SDR receiver will be discussed and compared to the most common implementable architecture. Next, some of the spectrum sensing campaigns carried out over the last few decades will be presented, and the main techniques used will be mentioned, as well as the setups used, exploring the hardware architectures and the associated tradeoffs. Finally, a set of concepts and figures of merit is presented that will later serve to characterize the performance of the system implemented in this work.

2.2 Software Defined-Radio Receivers

An SDR is a radio communication system that employs reconfigurable software-based components for processing and conversion of digital signals. While the idealized SDR receiver, shown in Figure 2.1, serves as a conceptual foundation, practical implementations diverge due to a plethora of limitations such as limited ADC bandwidth, dynamic range, and resolution. There are several architectures for RF receivers, each with its own set of advantages and disadvantages. In this context, an advantage corresponds to a feature that resembles the ideal device, while a disadvantage represents a feature that detracts from it. This work specifically focuses on discussing the most common architecture in the context of SDR receivers, namely the Direct-Conversion Receiver (DCR).

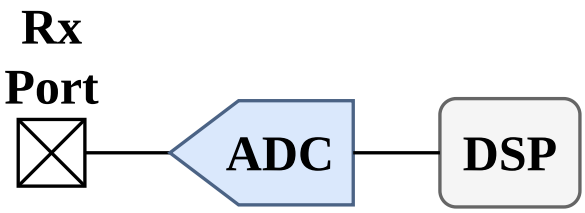

Figure 2.1: Ideal SDR receiver block diagram.

2.2.1 Direct-Conversion Receivers

The DCR, also referred to as Zero-Intermediate Frequency (IF) or Homodyne Receiver, is a radio receiver architecture that directly converts RF signals to baseband. Instead of using an intermediate frequency, as in traditional superheterodyne architecture, the DCR tunes a Local Oscillator (LO) to the same frequency as the desired RF signal, resulting in a zero intermediate frequency.

Modern Direct-Conversion Receivers DCR implement two separate In-Phase and Quadrature (IQ) channels, as can be seen in Figure 2.2. A dual channel scheme is essential for demodulating modern modulation schemes like Quadrature Amplitude Modulation (QAM), which encode data in both signal amplitude and phase.

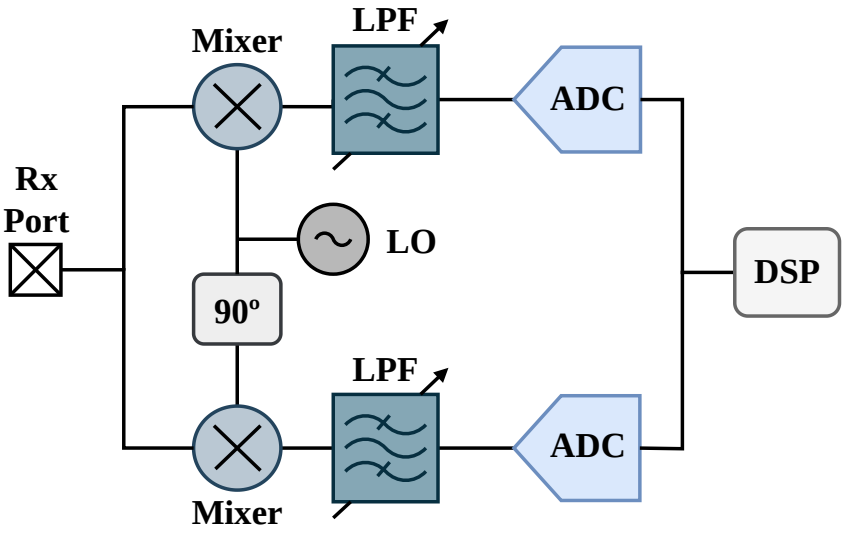

Figure 2.2: DCR receiver block diagram with two IQ channels and without the RF front-end.

The shift towards the DCR architecture has been largely driven by

the advent of SDR. SDRs seek flexibility, efficiency, and adaptability, and the DCR design aligns well with these objectives. By eliminating the need for an intermediate frequency, DCRs simplify the receiver design, reduce cost and power consumption, and are especially suited for on-chip integration, favoring miniaturization and multifunctionality in modern radio applications [8].

2.3 Spectrum Sensing

In the last two decades, several measurements of spectral occupancy in urban environments around the globe have been made [11, 12, 13, 14, 15, 16, 17]. The results suggest that the current spectrum utilization could be more efficient, especially in the licensed bands, due to the static approach used for spectrum allocation. Thus, to increase the efficiency of spectrum usage, the concepts of Cognitive Radio (CR) and Dynamic Spectrum Access (DSA) arose [18, 19].

In the literature, CR is envisioned as an intelligent wireless communication system that can sense, learn, and adapt to the surrounding spectrum environment. The mentioned technology uses advanced algorithms and machine learning techniques to monitor the RF spectrum, identifying unused portions (spectrum holes or white spaces) and dynamically adjusting its transmission parameters [18]. In turn, the main objective of DSA is to enable unlicensed Secondary User (SU), to access licensed frequency bands when they are not being actively used by PU [19]. Essential DSA-enabling techniques, such as spectrum sensing and geolocation databases are crucial in achieving this goal.

2.3.1 Spectrum Sensing Techniques

For a DSA-capable device to detect the aforementioned white spaces or if the licensed bands are not being used by a PU it needs spectrum sensing capabilities.

In this context several techniques arise namely: Energy Detection (ED), Matched Filter, Cyclostationary Feature Detection, and Eigenvalue-based Detection [10]. In the literature, studies demonstrate that when Signal-To-Noise Ratio (SNR) levels are at 40 dB or higher, all techniques yield 100% accuracy. However, when considering a simplified channel model with the addition of white Gaussian noise, performance decreases. For lower SNR values, it can be concluded that ED yields the highest probability of detection [9]. In addition, it is also the easiest and fastest performance technique [10]. Despite these attractive properties, the predominant factor for its use in spectrum occupancy campaigns [11, 12, 12, 13, 14, 15, 16, 17] is the fact that the receiver does not require any information about the transmitted signal [9, 10]. The ED method compares the received energy

in a particular frequency band to a usually empirically defined threshold to determine if the band is being used [23]. For reasons of simplicity of implementation and lack of information about the captured signals, this will be the preferred technique in this work.

2.3.2 Spectrum Sensing Hardware

Concerning spectrum occupancy, measurement campaigns presented in several studies [11, 12, 13, 14, 15, 16, 17] are conducted using two main types of equipment, namely the Spectrum Analyzer (SA) [11, 12, 13, 14], and Universal Software Radio Peripheral (USRP) [15, 16, 17].

In essence, a SA is a device designed to measure the amplitude of input signals as a function of frequency. Most SAs are superheterodyne receivers, functioning essentially as highly sensitive and selective radio receivers. The typical architecture of an SA includes a tunable RF front end, mixers that convert signals to an intermediate frequency for easier processing, followed by filters and digitizers [43]. This working principle is employed to achieve exceptional performance in instrumentation applications, but it comes with drawbacks. These include a very high cost, larger form factor compared to SDR, reduced flexibility in system integration, and susceptibility given their design tailored for laboratory settings.

USRP is a range of SDRs designed and sold by Ettus Research and its parent company, National Instruments. These devices are tailored to facilitate SDR implementations, offering notable flexibility compared to traditional spectrum analyzers. They consist of RF front ends and an interface to a host computer, allowing most of the signal processing to be executed in software. While USRPs provide increased adaptability over dedicated spectrum analyzers, their hardware abstraction mechanism paradoxically limits certain flexibilities. Additionally, when matched against a system built directly on an FPGA and an SDR transceiver with similar specifications, the USRP might present a higher cost due to the added layers of abstraction and integration.

Table 2.1 contains a summary of the advantages and disadvantages of the three typologies discussed.

2.4 RF Receiver Performance Characteristics

The technological advancement of wireless communication systems has led to their operation with ever-lower power signals in order to save energy or reduce interference. The power of these signals can in some cases be on the order of nW or pW, which is barely above noise level. The performance of the receiver is thus critical for a communication system to be able to operate with such low power. The aim of this section is to provide a basic introduction to the key concepts and figures of merit that are essential for characterizing

Criteria	FPGA + Transceiver	USRP	SA
Accuracy	Cost-dependent	Cost-dependent	High
Cost	Low to Moderate	Moderate to High	High
Flexibility	High	Moderate	Low
Form Factor	Low to Moderate	Low to Moderate	High
Development Complexity	High	Medium	Low

Table 2.1: Comparison of spectral probes based on FPGA with transceiver, USRP, and SA.

the performance of a spectral probe receiver. The key concepts and metrics that will be used to characterize an RF receiver in this work are:

- Noise Floor

  The noise floor is the signal resulting from the summation of all unwanted signals in a measurement system. The noise floor consists of noise from a number of sources, including thermal noise, atmospheric noise, and noise from the components used to build the measurement system [57]. Despite the existence of many different variations of noise sources, it is not necessary to delve into their physical properties to describe their ultimate impact on system performance. A simplified noise model consisting of a single theoretical noise generator can be used [58]. Whilst there are multiple sources of noise that can be combined into a unified source, thermal noise establishes a baseline for the lowest level that a receiver can measure. Thermal noise arises from vibrations of conduction electrons and holes due to their finite temperature. Some of the vibrations have spectral content within the frequency band of interest and contribute noise to the signals. The noise spectrum produced by thermal noise is uniform over RF and microwave frequencies [59]. The power delivered by a thermal source into an impedance-matched load in Watt is given by:

  $$P = kTB \qquad \text{(W)} \tag{2.1}$$

  where k is the Boltzmann's constant (1.38×10^{-23} J/K), T is the absolute temperature in Kelvin, and B is the bandwidth in Hertz. In the literature, a reference temperature is defined as T_{ref} = 290 K (16.85 ºC). The equation 2.1 shows the linear relationship between thermal noise power, temperature, and bandwidth. If the temperature is increased significantly, for example by 20 K, the noise power value rises from -93.975 dBm to -93.685 dBm, which in some applications makes the temperature variation negligible, as is the case in this work. To

represent very small and very large power values in a compact and easy-to-interpret format, the dBm unit is widely used in the telecommunications industry. The equation for the calculation of the noise floor in dBm is:

$$\text{Noise Floor} = 10\log_{10}(kTB) + 30 + NF \qquad \text{(dBm)} \tag{2.2}$$

where the term NF is the noise factor in dB and is the next concept detailed. Considering an ideal receptor, with NF = 0 dB at the reference temperature equation 2.2 simplifies to:

$$\text{Noise Floor} = -174 + 10\log_{10}(B) \qquad \text{(dBm)} \tag{2.3}$$

Equation 2.3 yields that at the reference temperature, considering a bandwidth of 1 Hz the noise floor is -174 dBm. The mentioned equation also reveals an important relationship: increasing the bandwidth by one decade results in a tenfold increase in the noise floor. This consideration is critical because, ideally, a spectral probe receiver would have as much bandwidth as desired, but this additional bandwidth comes at the expense of performance, resulting in a trade-off. For example, considering a bandwidth of 100 MHz, the maximum value achievable in this work, the noise floor rises to -94 dBm, assuming an ideal receiver. This value is of the same order of magnitude as those used in certain telecommunications technologies which, in order to operate at such low powers, naturally use reduced bandwidths.

- Noise Factor/Noise Figure

  The noise factor is a figure of merit used to indicate the degradation of the SNR introduced by devices in a signal chain. This figure of merit can be used to evaluate the performance of an amplifier or a complete radio receiver. Mathematically, it is defined as:

$$F = \frac{SNR_i}{SNR_o} \tag{2.4}$$

  where SNR_i and SNR_o are the input and output SNR, respectively. This formula assumes that the characterized component or system is at the reference temperature T_{ref}. The noise factor is a unitless ratio. The Noise Figure (NF) is the logarithm of the noise factor:

$$NF = 10\log_{10} F \tag{2.5}$$

  Ideally, a device such as an amplifier would not change the SNR, only amplify the input signal by its gain. In practice, the amplifier will

add its own noise to the signal, making the output SNR always lower than the input SNR. The NF therefore becomes a crucial aspect in the design of a signal chain for very low-power applications, where the signal level is already slightly above the noise floor. In this scenario, each component's contribution must be low enough to minimize the degradation of the SNR. To analyze the impact of each component in a multi-stage cascade system, a formula introduced by H. T. Friis is used. For a two-stage system, the formula is as follows:

$$\mathrm{F_{Receiver}} = F_{A1} + \frac{(F_{A2} - 1)}{G_1} \tag{2.6}$$

where F_{A1} is the noise factor of the first amplification stage with a gain of G_1 and F_{A2} is the noise factor of the second amplification stage with a gain of G_2 as shown in Figure 2.3.

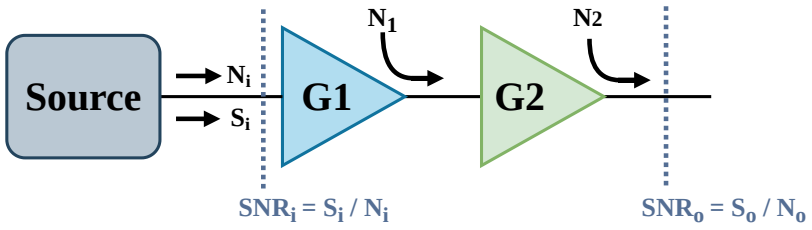

Figure 2.3: Cascaded amplifiers block diagram - adapted from [56].

The formula 2.6 reveals that the contribution of the first stage is dominant for the resulting noise factor, assuming that its gain G_1 is high, typically higher than 20 dB.

With this in mind, the first stage in an RF receive chain should be an amplifier with the lowest possible noise factor/figure and high gain. This component is typically referred to as Low-Noise Amplifier (LNA). The formula is rewritten as follows:

$$\mathrm{F_{Receiver}} = F_{LNA} + \frac{(F_{Rem} - 1)}{G_{LNA}} \tag{2.7}$$

where F_{Rem} is the NF of the remainder of the RF chain.

- Minimum Discernible Signal

  The Minimum Discernible Signal (MDS) is the lowest signal detectable by a receiver above the noise floor. However, to actually detect the signal it is required that the power level should be greater than the noise floor by an amount that varies with the application. In some

literature, MDS is also referred to as sensitivity [59], and it is defined as:

$$\text{MDS}_{\text{dBm}} = \text{Noise Floor}_{\text{dBm}} + \text{SNR}_{\text{min}} \tag{2.8}$$

where the Noise $\text{Floor}_{\text{dBm}}$ is calculated using equation 2.2 and SNR_{min} is the minimum acceptable signal-to-noise ratio, in dB, for the receiver to effectively detect and process the signal. For example, the minimum SNR recommended by Cisco for a Wireless Local-Area Network (WLAN) 802.11b/g access point is 20 dB, and 25 dB if a wireless voice telephony system is desired [22].

- Dynamic Range

  The dynamic range of a radio receiver is essentially the range of signal power levels over which it can operate. Ideally, a receiver would be characterized by constant linearity and exhibit an infinite dynamic range, enabling it to efficiently process input signals across a vast spectrum of power levels. However, reality dictates a different scenario. Systems, in their operational linear mode, are confined within a finite range. They are intrinsically constrained, with limitations manifesting at lower and upper thresholds (Figure 2.4), each characterized by distinct, inherent phenomena. At higher power levels, amplifiers start to reach so-called compression points, causing a flattening of the output signal or mixers form intermodulation distortion where multiple frequencies mix and create unwanted signals. At the other end, when the received signal power is comparable to or below the noise floor, distinguishing the intended signal from the noise floor becomes unfeasible.

As previously mentioned at the beginning of this section, the discussed metrics will be used to characterize the receiver later in Subsection 5.3.2.

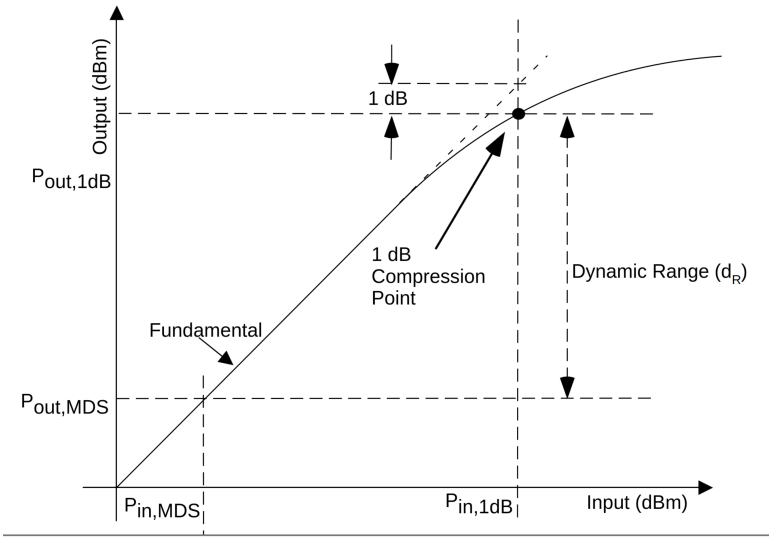

Figure 2.4: Output power as a function of input of a RF receiver - retrieved from [55].

Chapter 3

System Architecture and Development Platform

3.1 Introduction

In this chapter, the SDR-based spectral probe is described. In Figure 3.1, the high-level block diagram of the spectral probe architecture is presented. This architecture allows exploring different trade-offs between centralized and distributed processing using both software and hardware-accelerated techniques.

First, the individual blocks that comprise the system are introduced and explained to clarify their role in the overall design. Then, a holistic view of the entire system is presented.

3.2 General Architecture

The overall architecture was designed to include multiple spatially distributed probes that can sense, process, and send RF signals to the central processing unit. Figure 3.1 outlines the proposed scenario.

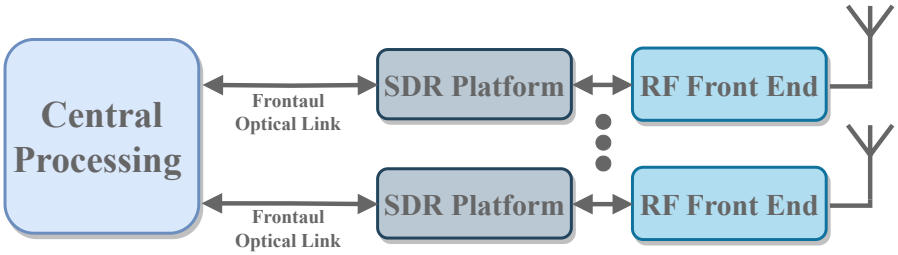

Figure 3.1: High-level block diagram of distributed scenario.

The following sections contain detailed explanations of the structure, function, and operation of the four main blocks that make up the system.

3.3 SDR Platform

In the following subsections, a detailed exploration of the RF transceiver and FPGA components is undertaken, highlighting their crucial roles in the proposed spectral probe architecture.

3.3.1 RF Transceiver

As previously discussed in subsection 2.2.1 the fundamental component in the architecture of a spectral probe is the RF receiver, crucial for the accurate capture of radio frequency signals.

The radio front end adopted in this work consists of a single AD9371 flexible broadband transceiver that had been acquired prior to the commencement of this project. The mentioned hardware operates from 300 MHz to 6000 MHz and covers most of the cellular bands, with up to 100 MHz and 250 MHz for the receiver and transmitter instantaneous bandwidths, respectively. The transceiver also provides a dynamically re-configurable receiver gain of up to 30 dB [24]. The receiver is based on the previously explained DCR architecture. The ADCs included in the receiver follow the $\Delta - \Sigma$ topology, featuring a maximum sampling frequency of 122.88 Msps, and have a resolution of 14 bits, which is a suitable value for a spectral probe designed to measure signals in the framework of modern communication technologies. This type of ADC is known to offer a good compromise between resolution and sampling rate.

The AD9371's maximum instantaneous receiver bandwidth is sufficient to capture entire Time Division Duplex (TDD) bands, or either the uplink or downlink bands in the case of FDD, assigned to modern cellular technologies. In the specific case of the TDD band n78 defined by the 5G New Radio (NR) standard, which ranges from 3400 MHz to 3800 MHz in Portugal, the AD9371's bandwidth cannot cover the entire band in a single capture. However, this part of the spectrum is divided into spectrum slices among different telecom operators, with each slice extending up to a maximum of 100 MHz contiguously [20][21]. Considering these factors, it can be argued that the AD9371 is a suitable choice for developing a spectral probe capable of determining the spectral occupation of modern wireless communication systems up to 6000 MHz.

The demands for real-time data processing in spectral probing are met by the low latency characteristics of the AD9371. The transceiver's design ensures that data is acquired and made available, through its high-speed serial interface for analysis with minimal delay which is critical given the system requirements.

3.3.2 FPGA

Receiving, processing, and sending the radio samples captured by the receiver requires a high-performance platform. The inherent flexibility and parallelism of FPGAs make them the best choice for this task. Recent advancements place FPGAs at the forefront of decentralizing computational tasks and enabling real-time processing closer to data sources. Such dynamics create a challenging design scenario for engineers to determine which computations are best suited for edge processing and which should be moved to a centralized processing unit. This dynamic has added a new layer of complexity and strategy to digital design, emphasizing the balance between localized and centralized computing power.

3.4 Fronthaul

As mentioned in subsection 3.3.1, the high throughput that can reach Gbit/s generated or consumed by the AD9371 transceiver must be efficiently transported to or from the BaseBand Unit (BBU) via a point-to-point optical link. In this context, the IEEE 1904.3 RoE standard emerged. This open and free standard specifies an encapsulation format and transport protocol for the transport of time-sensitive radio streams over Ethernet-based networks [26]. In the context of RoE, Evolved Common Public Radio Interface (eCPRI) provides a streamlined protocol that enables efficient transportation of radio signals over Ethernet networks. The eCPRI is a key protocol in the RoE framework. eCPRI was introduced as an evolution of the earlier Common Public Radio Interface (CPRI) protocol, with the primary goal of meeting the growing demands of 5G and future mobile networks. The transition from CPRI to eCPRI represents a paradigm shift from a more rigid, fixed-bandwidth interface to a flexible, scalable, and more efficient interface [46].

3.4.1 Radio over Ethernet

An eCPRI packet in an Ethernet frame primarily consists of an eCPRI header and its payload. Notably, the eCPRI payload contains the Protocol Control Identifier (PC ID), Sequence Identifier (SEQ ID), and the eCPRI Data as depicted in Figure 3.2.

The PC ID, acts as a unique identifier, distinguishing multiple protocol entities on an Ethernet network. It ensures that data is correctly directed to specific radio equipment. Essentially, it provides precise coordination among multiple layers of radio equipment and functions. The SEQ ID field serves as a sequence number for the messages, ensuring data integrity and continuity. By tracking the sequence of incoming packets, systems can quickly identify if any packets are missing or out of order, crucial for maintaining a consistent

data flow in radio communications. The most significant portion, the eCPRI payload, carries the actual data, which can range from IQ samples to control information.

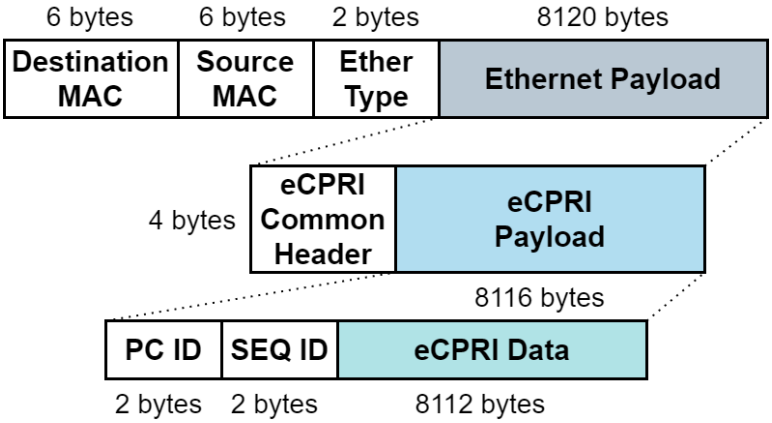

Figure 3.2: eCPRI packet structure.

3.5 Centralized Processing

In contrast to the hardware-accelerated nature of the FPGAs, the flexibility of the software allows rapid development of flexible applications. This component is particularly useful in scenarios where timing constraints are not as strict.

In the proposed architecture, the high throughput generated by the SDR platform is sent to a server, where the data is processed. The server unit is based on a Dell PowerEdge R650xs rack server equipped with an Intel Xeon Gold 6336Y Central Processing Unit (CPU) configured to run 16 physical cores at 3.1 GHz with 128 GB of Double Data Rate 4 (DDR4) Synchronous Dynamic Random-Access Memory (SDRAM).

The chosen CPU is an advanced processor tailored for high-performance tasks and server environments. One of its notable features is its support for Single Instruction Multiple Data (SIMD) (Single Instruction, Multiple Data) instructions, specifically the Advanced Vector Extensions (AVX). AVX-512 can significantly increase computing power by allowing the processor to perform parallel operations on large data sets, making it invaluable for applications that require intensive data processing, such as DSP tasks.

To handle high-throughput communication with the Remote Radio Head (RRU), the server features a Broadcom BCM57410 dual-port GbE 10 Small Form-factor Pluggable (SFP)+ network adapters with support for Jumbo frames (up to 9600 Bytes) [32].

It should be noted that all the hardware comprising the Centralized Processing unit was purchased before the start of this work.

3.6 Analog RF Front-End and Antenna

The versatility of the testbed is highlighted by the adaptability of the RF front end, which is capable of receiving and/or transmitting signals. The front end can be adapted to suit a wide range of scenarios by adjusting key characteristics, including antenna, filtering, and amplification chains.

As discussed in Section 2.4, LNAs are essential to boost the performance of software-defined radio SDR receiver systems. SDRs are typically designed to wideband and therefore the NF is not optimized for any one particular frequency. With an LNA and appropriate filtering, performance can be improved to a particular range of frequencies.

The antenna is an essential component in a spectral probe, responsible for capturing electromagnetic waves from the surroundings. For a spectral probe, it's imperative to select or design an antenna that has a wide frequency response and is sensitive across the intended spectrum of interest. The antenna's radiation pattern, gain, and efficiency can significantly affect the system's ability to detect and measure signals. In addition, depending on the probe application, the antenna may need to be directional, focusing on signals coming from a specific direction, or omnidirectional, capturing signals from all directions.

Figure 3.3 contains the block diagram illustrating the integration of the analog RF front-end and antenna into the RF platform.

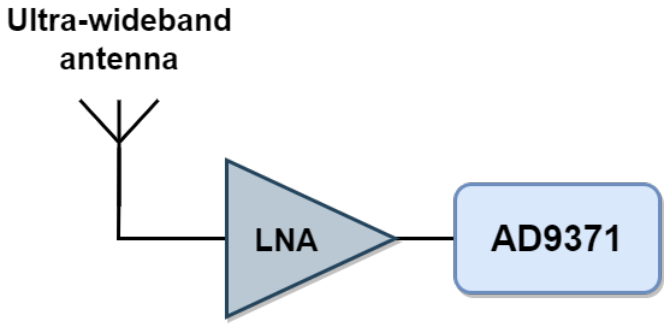

Figure 3.3: Block diagram of analog RF front-end and antenna.

3.7 Final Architecture

After detailing the various blocks that make up the system, Figure 3.4 illustrates the final architecture. The details of the implementation and integration of the subsystems will be presented in the next chapter.

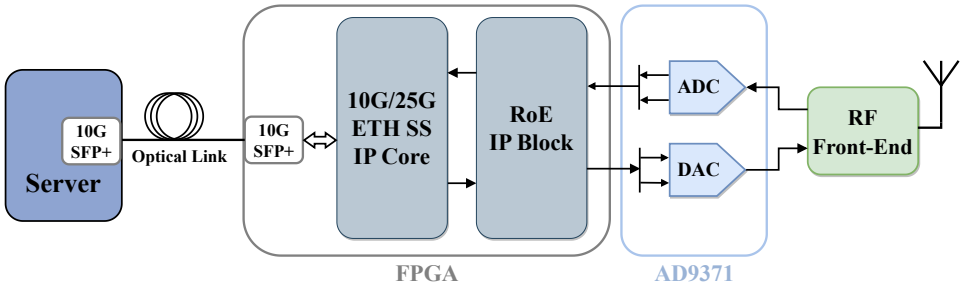

Figure 3.4: Final system architecture block diagram.

Chapter 4

Implementation

4.1 Introduction

This chapter follows the architecture described in the previous chapter and details the implementation and integration of the various subsystems that make up the entire spectral probe. It covers the FPGA-based SDR platform, including distinct design implementations. Key topics include embedded Linux, network raw sockets, and Centralized Processing. The chapter concludes with insights into the critical role of the analog RF-front end in achieving an optimal noise floor.

4.2 SDR Platform

The implementation of the FPGA-based SDR platform started from AMD's RoE and Analog Devices' AD9371 example designs. The implementations of both designs were initially done in parallel and independently.

The first AD9371 design was implemented in a bare metal system, that is, without an operating system. The sample design requires an external 30.72 MHz reference clock to derive its 122.88 MHz operating clock. In its default configuration, the AD9371 outputs 14-bit resolution IQ samples at 122.88 Msps, providing an instantaneous receiver bandwidth of 100 MHz. Each IQ component is encoded as a 16-bit value, with a pair of IQ samples totaling 32 bits sampled at 122.88 MSPS. This results in a data rate of approximately 3.93 Gbit/sec. It is important to note that this value depends on the profile of the AD9371, which is chosen according to the desired bandwidth of the receiver.

The implementation of the RoE Intellectual Property (IP) Core, a solution developed for the Zynq UltraScale+ MPSoC that utilizes both hardware and software to provide an efficient yet flexible platform, was also implemented in a bare-metal design, running at 156.25 MHz. Note that the implementation of this IP core implies the inclusion of the Ethernet

subsystem IP core, which operates at the same frequency.

The RoE IP Core was configured to transport packets with the maximum allowed eCPRI payload of 8112 bytes, which translates to 2028 IQ pairs (32-bit each pair). This payload size minimizes packet overhead and offloads the Centralized Unit (CU) processing, maximizing throughput. Furthermore, Ethernet encapsulation is done by the 10G/25G Ethernet Subsystem (ETH SS) contained in the RoE example design, carrying eCPRI packets over Ethernet frames.

The process of integrating the two designs resulted in two distinct implementations, called Loopback Design and Standard Design. In both designs, the blocks are connected via standard 64-bit AXI4 stream interfaces. The ADC input is connected to port RX1 of the AD9371, and the output of the Digital-to-Analog Converter (DAC) is connected to the TX1 port. Whereas the RoE IP Core is connected to an SFP+ multi-Gigabit transceiver.

4.2.1 Loopback Design

The Loopback Design was developed to facilitate easy testing and validation of the transmission and front-haul link. Due to the 2:1 relation between the DAC and the ADC in their transmit and receive datapaths, an upsampling block with a factor of 2 was implemented in the FPGA, followed by a Finite Impulse Response (FIR) low-pass interpolation filter with 200 taps and a 100 MHz cutoff frequency. This configuration enables the connection of the ADC output to the DAC, thus creating an internal loopback, through an asynchronous clock domain crossing First In First Out (FIFO), bridging the clock domains of the framer and deframer. As a result of this modification, the output of the upsampling block is 64-bit, which doubles the previously mentioned bit rate to approximately 7.86 Gbit/s. The Data Gen/Sink block is an additional block created to generate synthetic data to validate the correct data framing, followed by a fiber loopback and subsequent deframing and sink of the data to verify the matching values. Given the above bit rate and the total size of the eCPRI packet of 8134 bytes, this results in a packet being sent every 8.27 μs, or a packet rate of 120.857 kpackets/s.

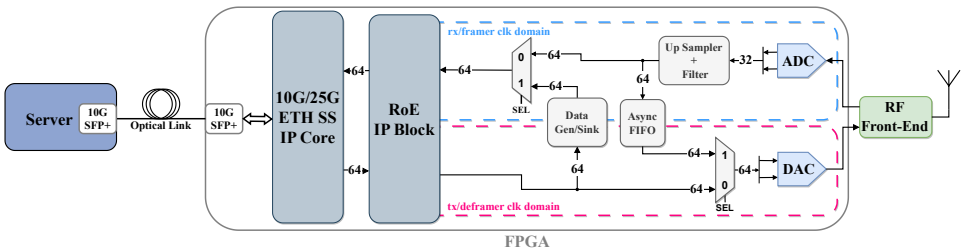

Figure 4.1: Block diagram of Loopback Design implementation.

4.2.2 Standard Design

The Standard Design was developed later, as it doesn't have the disadvantage of generating double the throughput, further restricting the timings for receiving packets on the server side. In this case, the period between packets is then 16.55 μs, equivalent to a packet rate of 60,428 kpackets/s. In this design, the blocks added to the previous design have been removed, with only the need to create a block to concatenate two consecutive 32-bit samples of the ADC to form a 64-bit data stream compatible with the input of the RoE framer. This way, data is only made available to the framer block every two samples of the ADC. This design thus becomes a direct implementation of the architectural block diagram shown earlier in Figure 3.4.

4.2.3 Embedded Linux

To facilitate development and testing, the system is running an embedded Linux kernel built with PetaLinux tools. This enables a file system with user-defined applications and easier integration with Analog Devices' AD9371 device drivers. The file system is also very practical, as it allows for the storage of files that have been preloaded or sent over the network. This feature enables full configuration profiles to be stored for the AD9371 so that the spectral probe receiver can be completely reconfigured at run time.

The use of embedded Linux also allows for an agile configuration at run time of the transceiver parameters, via a Command Line Interface (CLI). This configuration via CLI can be done manually or programmatically by sending commands via the serial device.

4.2.4 Network Sockets

At the time of writing, RoE's IP Core does not support the calculation of the User Datagram Protocol (UDP) checksum [45]. The absence of checksum calculation makes receiving packets with this transport protocol on the server side impractical, as the network card detects the malformation of the packet when calculating the checksum using accelerated hardware, and rejects the packet. To circumvent this problem, it would be necessary to implement a software sniffer capable of analyzing all packets arriving at the network card. The inherent inefficiency of this implementation is therefore made unfeasible by the system's time-critical requirements. Bearing these considerations in mind, the use of the transport layer is known in the Open System Interconnection (OSI) model as Layer 4 was eliminated. The network architecture is simplified to point-to-point connections implemented through the Data Link Layer (Layer 2 in the OSI model) [47].

The need to access and manipulate the fields in eCPRI packages requires a level of control that traditional socket programming cannot adequately provide, leading to the adoption of raw sockets. Raw sockets allow direct access to the communication protocols, bypassing the traditional protocol stack [48].

To implement raw sockets, the choice was made to use the Berkeley interface, a decision based on its recognized flexibility and efficiency in network communication. Berkeley sockets originated from the Berkeley Software Distribution UNIX operating system and have become an industry standard Application Programming Interface (API) to create and use sockets [49].

Initially, the raw socket implementation commenced with Python, capitalizing on its inherent capacity to swiftly prototype scripts. Python's rich library ecosystem and intuitive syntax facilitated the developmental phases, ensuring that ideas were promptly translated into code. However, for the sake of performance optimization and reduction in jitter, a transition to the C programming language was essential. C, with its low-level access and low function overhead, ensured that raw sockets were executed with optimal performance. The computational efficiency of C significantly minimized latency, delivering a level of performance that not only met but exceeded the stringent requirements for real-time processing and management of eCPRI packets in complex network environments.

This trend of software development based on an initial prototyping and validation stage in Python and subsequent implementation in C to achieve high performance was prevalent throughout the work.

4.3 Centralized Processing Implementation

As mentioned in the 3.5 Section, the flexibility of software development makes it an attractive approach for implementing and testing different algorithms, namely Digital Signal Processing (DSP) techniques. While one can argue that the high sample throughput of RF makes it attractive to implement processing on FPGA, it's also true that modern CPU can efficiently implement algorithms with high performance and reduced variability using a few specialized techniques. All programs developed in this thesis are run on a fine-tuned low-jitter computing platform described in detail in Section A. The Python language was used for initial testing and prototyping of the algorithms because of its flexibility, rich libraries, and ease of development. Once validated, the algorithms were reimplemented in C for maximum performance and determinism.

It should be mentioned that, although a multi-threaded multi-core producer -consumer application was developed using classical POSIX pthread mutual exclusion and synchronization mechanisms implemented in C with mutexes and condition variables, this approach introduced higher variabil-

ity in execution times. Therefore, a single-threaded, single-core approach was preferred. Developing such applications composed of multiple threads running on different CPU cores with such strict time requirements on the order of μs units, with low execution time variability would be a challenge for a master's thesis on its own.

To support the 8134-byte eCPRI packets, the network card's maximum transfer unit setting has been configured accordingly.

The first step was to reconstruct the IQ samples from the payload of the packets since they follow the big-endian network byte order. This required reordering the bytes of the 16-bit encoded IQ component of each sample.

4.3.1 Real-Time Digital Signal Processing

Although there are several libraries in Python that use C implementations of various DSP algorithms, their interpreted and dynamically typed nature prevents them from being used in applications with time-critical requirements. The C implementation of these algorithms used in these Python libraries, such as the well-known *SciPy* library, can be consulted and the documentation itself mentions that these implementations can be optimized [33].

With this in mind, and in the absence of a native C DSP library with an FIR filter implementation featuring SIMD AVX512 instructions to achieve maximum performance, a custom software module, described in detail in Appendix B, was developed with the appropriate algorithms. It should be noted that a library platform/architecture agnostic that contains kernels of SIMD code for different mathematical operations, including convolution called *Vok* (used in the GNU Radio software toolkit), already existed when the FIR filter was implemented [34]. However, at the time of writing, this library only supports up to AVX-256 instructions, limiting the maximum performance achievable to half. Given that the filtering operation is one of the most computationally demanding functions, it made sense to invest resources in its implementation in order to obtain the best possible performance. In addition, at the time of this writing, the Intel Integrated Performance Primitives library was discovered, which provides a comprehensive set of application domain-specific, highly optimized functions, including those for signal processing with digital filters implemented. However, it's worth noting that the source code is not available, only the binaries, which limits the scope of the analysis. Nevertheless, the application of advanced techniques using SIMD instructions on high-end CPUs is an intriguing educational exercise. As future work, it would be interesting to compare the performance of the developed method with the function provided by Intel.

Another fundamental DSP tool is the Discrete Time Fourier Transform (DTFT), which already has an implementation optimized for several CPU architectures, the *FFTW* (Fastest Fourier Transform in the West). This C

subroutine library allows the computation of DTFT in one or more dimensions, of arbitrary input size, and of both real and complex data, as well as the discrete cosine/sine transform. This software won the H. Wilkinson Prize for Numerical Software in 1999 and is still used as the mainframe for these computations, namely in MATLAB and Python Fast Fourier Transform (FFT) implementations [44].

Digital Filtering

Since most of the bands allocated for wireless communications have different bandwidths, sometimes less than the default 100 MHz receiver bandwidth, it is necessary to filter out the adjacent bands. Another possible scenario would be to simply analyze the occupancy of different telecom operators which would also require filtering out different slices of the spectrum.

Figure 4.2 illustrates both scenarios described above, where the width of a given band is 60 MHz, less than the maximum value of the receiver, and each telecom operator has slices of the spectrum with different widths.

Figure 4.2: 2100 MHz 4G LTE n1 FDD downlink band allocation in Portugal - retrieved from [4].

To isolate the desired frequency band, it's imperative to either center the local oscillator frequency accordingly or implement a software-based frequency shift. After this adjustment, a Low-Pass Filter (LPF) should be applied, characterized by a bandwidth that matches the specific frequency band of interest.

Although the FFT is used to implement digital filtering in some scenarios by converting the convolution operation in the time domain into a multiplication in the frequency domain, this approach only becomes advantageous for signals with higher numbers of samples than those used in this dissertation, 2028 samples. For this number of samples, the computational load of FFT proved to be too high to meet the tight time requirements.

In the process of designing the digital LPF, the first decision was the choice between FIR and IIR filters. Each option has distinct advantages. While IIR filters offer increased rejection with lower-order filters, they can become unstable, especially when the cutoff frequencies are well below the sampling frequency. Furthermore, in practice, using more than a dozen recursion coefficients can also push the filter toward instability [35]. On the other hand, FIR filters offer simplicity of implementation, unconditional

stability, and consistent frequency response, especially when implemented in single precision. While phase response wasn't a significant factor for the application, the rejection factor was. A notable advantage of single-precision implementation is the ability to perform twice as many parallel operations using SIMD compared to double-precision.

The figures 4.3 and 4.4 highlight the effect of the floating point precision on the frequency response of the IIR and FIR filters. Both filters were designed for a 122.88 MHz sampling frequency and a 10 MHz cutoff frequency. The response of the FIR filter remained practically unchanged when implemented in single-precision, while the IIR filter became unstable.

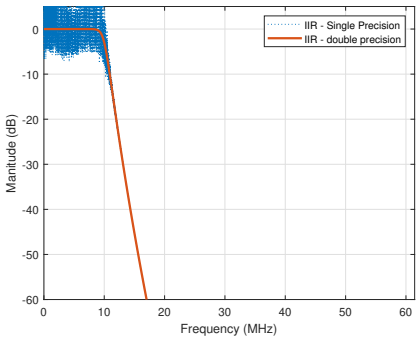

Figure 4.3: Frequency response of a 12th order IIR filter implemented with single and double precision.

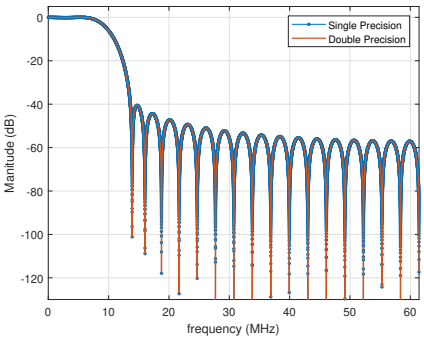

Figure 4.4: Frequency response of a 29th order FIR filter implemented with single and double precision.

Taking all this into account and mainly weighing up the factor of unconditional stability, greater invariability depending on the implementation in single or double precision, and filter design with a narrower cut-off frequency, the FIR filters were considered more suitable.

The next step in designing the FIR filter was to choose the windowing method. The choice of windowing is critical to minimizing ripple in the ripple in the passband and maximizing attenuation in the rejection band.

The Kaiser window is a one-parameter family of window functions that maximizes the ratio of the main lobe energy to the side lobe energy. Another advantage of Kaizer windows is that by keeping the window length constant, the shape factor, known as the β parameter, can be used to design the passband and stopband ripples [36]. This is an important advantage over other windows where there is a three-way trade-off between window length, ripple size, and passband bandwidth [37]. Larger β values provide lower side-lobe levels but at the price of a wider main lobe. Taking all this into account, the β value chosen was 3.5.

As a final note, it should be noted that despite all the techniques to optimize the implementation of the FIR filter, this is still a task with a high

computational burden. This creates a compromise between the order of the filter and therefore its rejection performance and the computational load.

4.3.2 Signal Power Calculation

After reconstructing the IQ samples and filtering the bands of interest, one use case already discussed is to determine the power in that band to allow spectrum occupancy analysis. This simple technique, known as ED, is discussed in the Subsection 2.3.1.

In order to achieve maximum performance, a careful signal power approximation method is introduced, implemented, tested, optimized, and benchmarked in the appendix C.

4.3.3 Remote Access

As mentioned in the 4.2.3 Subsection, the SDR platform runs embedded Linux to add certain convenient features to the system. These features allow remote and flexible access from the server to the SDR platform via the Accelerated Processing Unit (APU), which serves as a communication middleman. This architecture makes it possible to develop programs that run on the server and automatically reconfigure the probe's parameters. A simple and powerful example of this mechanism is for the server to be able to change the frequency of the receiver oscillator, capture and process the samples, and thus scan the spectrum.

To establish this connection, the server first needs to establish an SSH connection to APU, which has been implemented programmatically using the *Paramiko* library in Python and *libssh* in C. Once it has access to the APU, the server sends commands for the APU to execute, but the commands sent are set to write directly to the serial device corresponding to the USB UART connection that exists between the APU and the SDR platform. This emulates a direct connection from the server to the APU. It should be noted that the server and the APU have to be on the same network. Figure 4.5 illustrates the proposed strategy.

The main advantage of this strategy for ensuring remote access programmatically is flexibility, since the server has access to the Linux CLI the configuration of parameters becomes entirely the responsibility of the AD9371 drivers. The main disadvantage is the latency of the whole process, from sending the Secure Shell (SSH) command to interpreting the command by the AD9371 drivers to reconfiguring the probe's parameters. This whole process is in units of milliseconds, which depending on the application can be several orders of magnitude above the requirements.

Another disadvantage that could be pointed out is the need for the APU, but this is already integrated into the posts for other purposes.

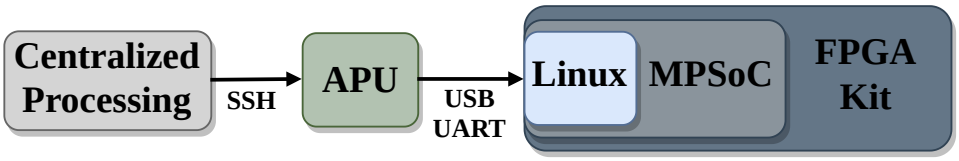

Figure 4.5: Block diagram of the setup used to enable remote access.

4.4 Analog RF-front end

Although the analog RF chain is not the primary focus of this paper, its importance in achieving the lowest possible noise floor cannot be overstated. The RF chain is fundamental to the overall performance of the system and directly affects the sensitivity and accuracy of signal detection, as mentioned in 2.4.

For proof-of-concept, the LNA considered is the QPL9057 from Qorvo, in which the bandwidth is compatible with the AD9371. In the 600 MHz to 4000 MHz frequency range, the LNA presents more than 20 dB of gain and a noise figure of less than 0.6 dB [27]. Additionally, it exhibits a high linearity, and a gain variation of less than 2.0 dB across this range, ensuring consistent performance over this frequency range. Therefore the analog RF front-end jointly with the AD9371 receiver provides a suitable solution for sensing cellular bands.

To quantify receiver performance, the equations 2.3 and 2.7 were adopted, considering a laboratory temperature of 21 ^{o}C and a bandwidth of 100 MHz. This approach yielded values shown in Table 4.1 for the expected cascaded NF and noise floor based on the AD9371 and LNA gains and NF values retrieved from the datasheets [24] [27].

Freq. (GHz)	AD Gain (dB)	AD NF (dB)	LNA Gain (dB)	LNA NF (dB)	NF (dB)	Noise Floor (dBm)
0.8	21.0	0.6	30.0	12.0	1.0	-92.9
2.6	23.5	0.5	30.0	13.5	0.9	-92.8
3.5	22.8	0.6	30.0	14.0	1.0	-92.9

Table 4.1: RF chain cascaded NF, and noise floor calculations for different frequencies.

Considering the noise floor values from Table 4.1, it is also possible to calculate the dynamic range of the receiver. This calculation takes into account a maximum input power of -10 dBm at the AD9371 RX port, resulting in a dynamic range of approximately -60 dB.

The RF signal was captured using a previously developed Ultra-Wideband (UWB) planar elliptical antenna (Figure 4.7), with a frequency range within the AD9371 supported range, but it has been specifically matched for the

2.6 GHz and 3.5 GHz frequencies, which correspond to the 3rd Generation Partnership Project (3GPP) LTE B7 and the 3GPP 5G NR n78 bands, respectively.

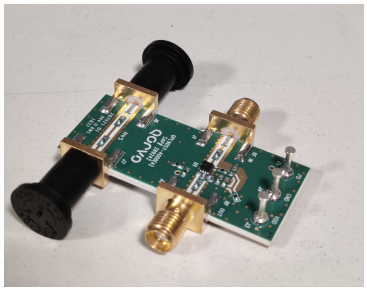

Figure 4.6: QPL9057 evaluation board.

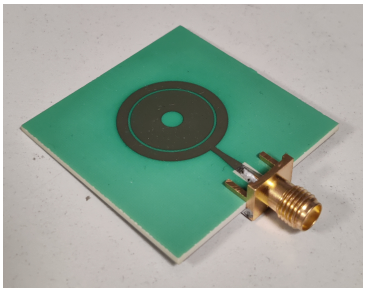

Figure 4.7: UWB antenna.

The gains at different frequencies of the pre-designed antenna are shown in Table 4.2.

Frequency (GHz)	**Antenna Gain (dBi)**
2.0	1.81
2.5	2.98
3.0	2.76
4.0	3.67

Table 4.2: UWB antenna gain at different frequencies.

4.5 Physical Setup

With the implementation of all the subsystems described, the physical lab setup is now introduced.

As far as FPGA is concerned, the choice is limited to boards supported by RoE IP Cores which is developed targeting the Zynq UltraScale+ MPSoC. Within this restricted set is the ZCU102 Evaluation Kit, which has extensive documentation and support for the AD9371, making this kit the ideal choice. The ZCU102 Evaluation Kit is a platform designed by Xilinx to showcase the Zynq UltraScale+ MPSoC, a state-of-the-art technology that merges a user-programmable FPGA with a quad-core Arm Cortex-A53, and a dual-core Cortex-R5F real-time processors. The kit is equipped with a wide range of high-speed connectivity, memory interfaces, and peripherals, making the ZCU102 a very flexible kit for prototyping and testing designs. Of all the features identified in Figure 4.8, of particular relevance to this work are the Ethernet, USB UART, and Joint Test Action Group (JTAG) connectivity,

the SFP+ connectors for the optical link, and the FMC connector for the interface with the AD9371.

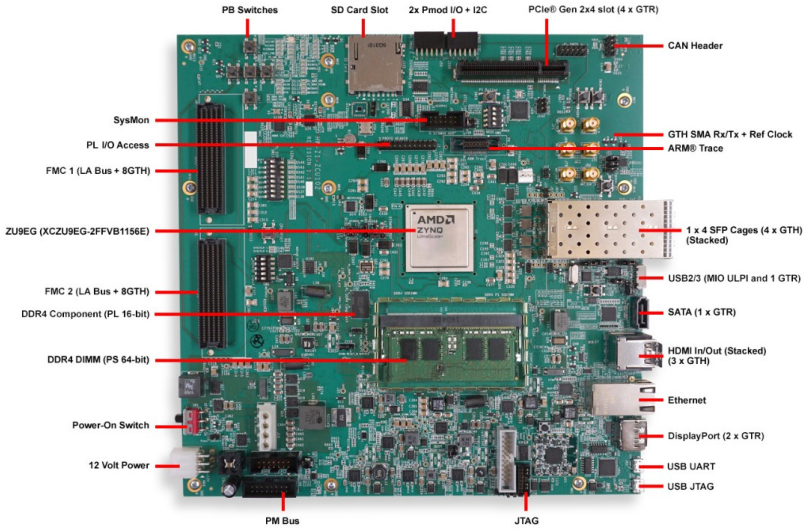

Figure 4.8: Zynq UltraScale+ MPSoC ZCU102 Evaluation Kit [50].

The centerpiece of this spectral probe, the AD9371 transceiver, is embedded in the ADRV9371-WPCBZ radio card, which already has an internal power supply and provides the interface with its four reception channels, two of which are auxiliary and two transmission channels via SMA connectors. As previously mentioned, the interface with the FPGA is entirely provided by the FMC connector.

To generate its internal operating clock of 122.88 MHz, the AD9371 requires a reference clock signal of 30.72 MHz. To this end, a programmable clock generator module based on the SI570 integrated circuit was used to generate the reference clock (Figure 4.10). The clock signal was delivered to the AD9371 using an SMA cable.

4.5.1 Radio Frequency Instruments

The SMW200A VSG was chosen because of its compatibility with the AD9371 in terms of frequency range and bandwidth, as well as the ability to generate test signals with different characteristics, namely configurable baseband content and finely adjustable output power in a dynamic range between -100 and -10 dBm. The FSW was chosen because of its flexibility to behave as either a Vector Signal Analyzer (VSA) or SA, compatibility with the frequency range and bandwidth, high flexibility in configuring measure-

Figure 4.9: ADRV9371-WPCBZ radio card [51].

Figure 4.10: Si570 Evaluation Board [52].

ments, input power range, and overall precision and accuracy. To perform average power measurements and system calibrations Keysight's N1913A power meter was used.

4.5.2 Fronthaul

For the fronthaul optical link, a single mode-fiber cable (Figure 4.11) with a lucent connector is connected using a bidirectional SFP+ transceiver, ensuring this way a 10 Gbit connection. On the server side is another bidirectional transceiver that uses wavelength-division multiplexing to share the

same optical fiber. In addition, to test the loopback design, the SFP+ loopback module shown in Figure 4.12 was used.

Figure 4.11: Bidirectional SFP+.

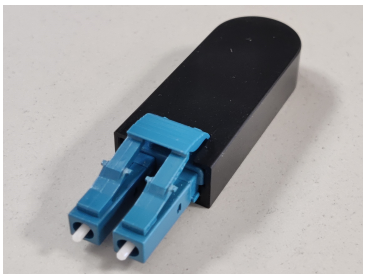

Figure 4.12: Loopback SFP+.

4.5.3 Final Setup

The laboratory setup is presented in Figure 4.13. To connect the LNA to the AD9371's Rx1 port, an SMA Male to SMA Male connector was used and the connection between the LNA and the antenna was made using a semi-rigid SMA cable to ensure that the position of the antenna was fixed in a practical way.

The present setup functioned as a development and prototyping iteration framework, leading to the implementation of the system prepared for installation in the lamppost wall boxes of the ATCLL. To this end, a similar system was assembled, this time utilizing the Trenz Electronic TEBF0808 carrier board. This choice was driven by considerations of cost and form factor. The TEBF0808 offered a more economical and compact solution while maintaining the required functionality. The process of migrating the design to this new carrier board, along with the associated challenges and solutions, is detailed in Appendix D.

Following the description of the system's implementation and final setup, the next chapter tests the various subsystems mentioned above individually and the system as a whole, analyzing the results to see if the performance meets the expectations.

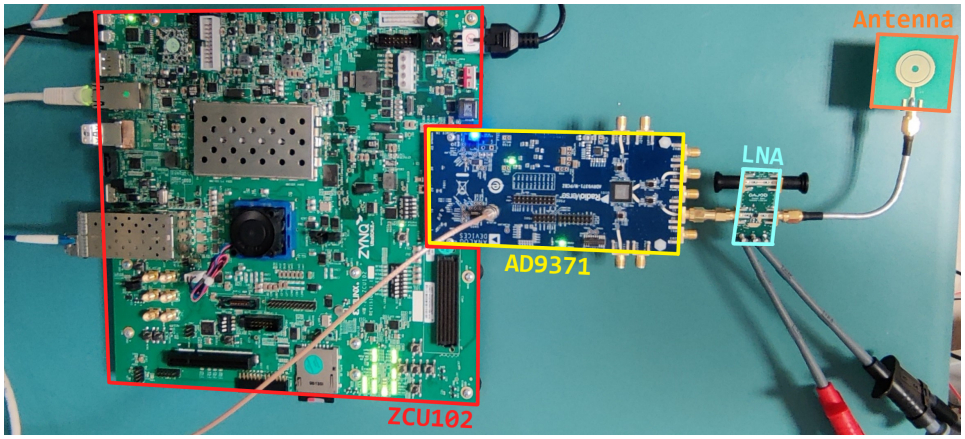

Figure 4.13: Laboratorial setup of the spectral probe with the LNA and antenna.

Chapter 5

Tests and Results

5.1 Introduction

With the implementation of the system detailed in the previous chapter, this chapter focuses on testing and validating the system in a phased approach, isolating the various subsystems before testing the system as a whole with a multiband spectral sensing campaign in an indoor environment.

5.2 Loopback Tests

The first loopback test was conducted to ensure that the RF signal was received correctly. This test consisted of ejecting the RF signal from the R&S SMW200A VSG into the AD9371's RX1 port and retransmitting the signal into the AD9371's TX1 port to the R&S FSW VSA/SA. Figure 4.13 shows the laboratory setup, where the SFP loopback module is only of relevance for the next test. In this test, the loopback is internal to the FPGA fabric. The IQ samples at the output of the ADC only pass through the upconversion and filter block and are directly connected to the input of the DAC, as shown in the diagram in Figure 4.1. The purpose of this test is to validate the integration of the AD9371 example design in FPGA and to baseline only the performance of the receiver and transmitter and the upconversion block and filter introduced. To conduct this test, various signals with different QAM constellation sizes, different bandwidths, and different carrier frequencies were used. The results using a 60 MHz signal modulated in 16-QAM constellation at 2.5 GHz are shown in Figure 5.1 where Error Vector Magnitude (EVM), estimated by VSA, is used as the performance metric. An EVM value close to 1% serves to validate the receiver's correct operation.

The second loopback test was carried out to guarantee the integration of the RoE IP, validating that the IQ signals validated previously are correctly framed and deframed according to the packet structure described in Figure

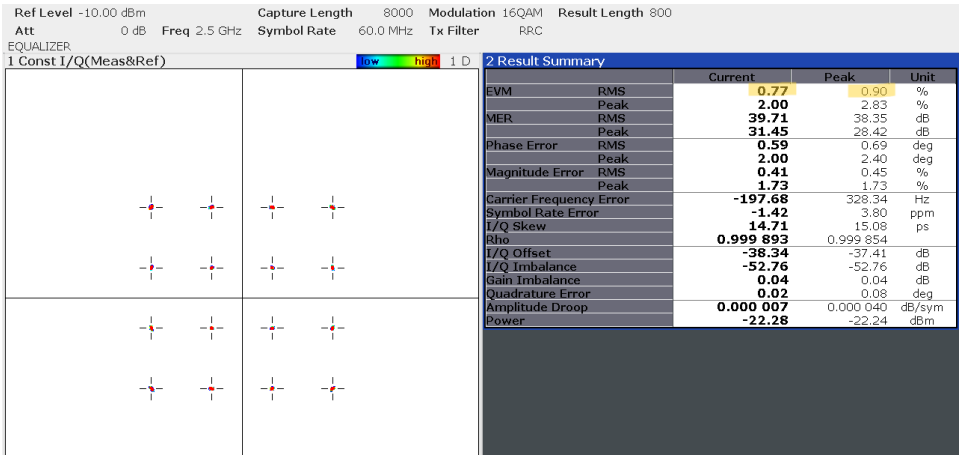

Figure 5.1: Internal loopback results.

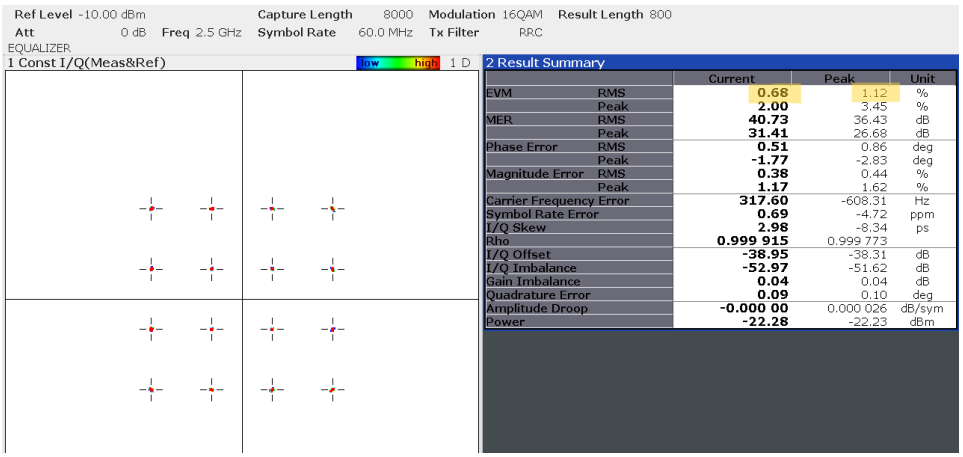

Figure 5.2: RoE loopback results.

3.2. The procedure for this test is the same as the first, the only difference is where the loopback is taking effect, which is no longer internal, but rather using the SFP+ fiber-optic loopback module shown in Figure 4.12. In this test, the acquired IQ samples are packed into eCPRI Ethernet packets and placed in the SFP+ transceiver, which converts the electrical signal into an optical signal. The loopback module sends the optical signal back to a transceiver, which converts the signal back to the electrical domain. This signal containing the packets is transported to the deframer block, which unpacks the IQ samples, which are then placed at the input of the DAC. The generated RF signal is then placed on the TX1 port and analyzed in the VSA. The results using a signal equal to the one mentioned in the previous test are shown in 5.2. The EVM value close to the previous result of 1% shows that the packing and unpacking of the samples and interfacing with

the optical transceiver have been successfully integrated into the previous design. The laboratory setup for both tests can be seen in Figure 5.3.

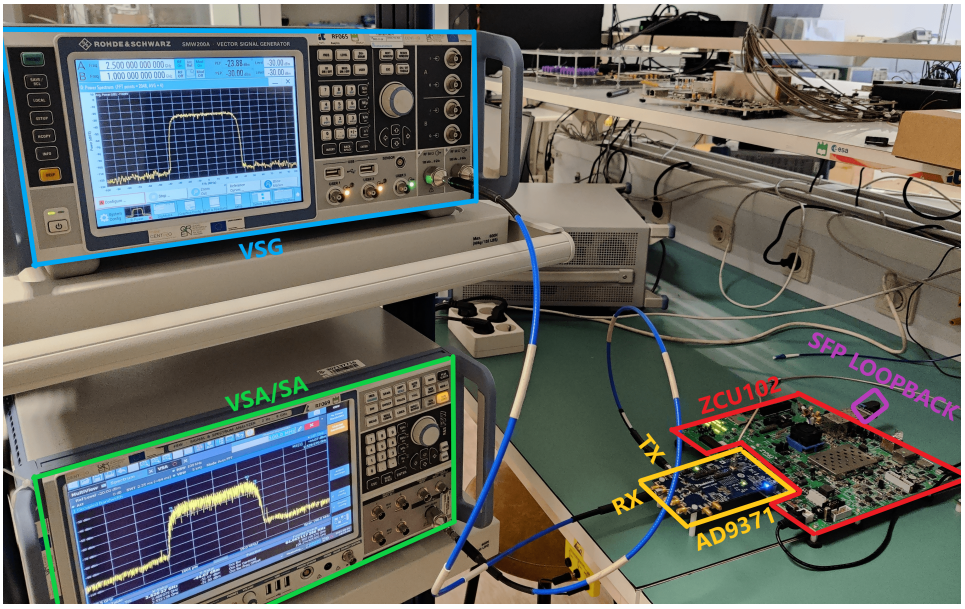

Figure 5.3: Laboratory setup for the loopback tests.

5.3 Centralized Processing Tests

After testing and validating the correct capture and packaging of the samples, the next step was to test the reception, decoding, and processing of the packets on the server. To ensure that the eCPRI packets are sent by the SDR platform and accepted by the network card, the TShark software was used. TShark is a terminal-oriented version of the well-known network protocol analyzer, Wireshark, designed for capturing and displaying packets when an interactive user interface isn't necessary or available. Once the packets were accepted by the network card, the next test was to programmatically capture and decode the packets using the network sockets described in Subsection 4.2.4.

5.3.1 Spectrum Analysis

The payload of the eCPRI packets containing the IQ samples was processed in order to retrieve the IQ samples. To validate this process, the frequency spectrum was computed using the FFT and then plotted using a Python script. The FFT was computed using 1, 10, 100, and 1000 packets (Figures 5.4, 5.5, 5.6, and 5.7).

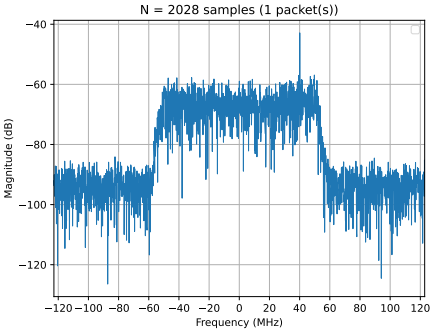

Figure 5.4: FFT of the received signal (N = 2.028 ksample) on the server.

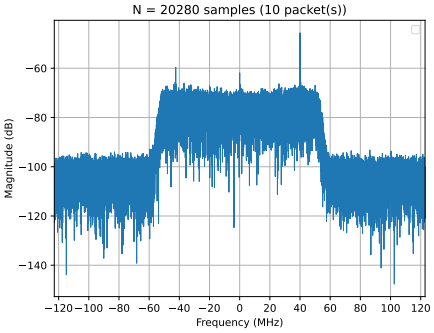

Figure 5.5: FFT of received signal (N = 20.28 sample) on the server.

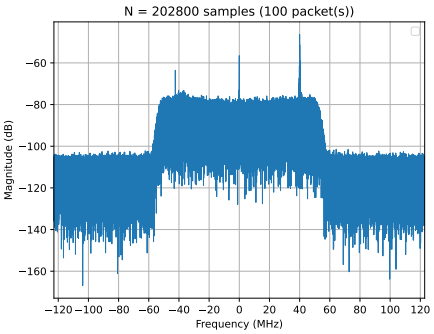

Figure 5.6: FFT of the received signal (N = 202.8 ksample) on the server.

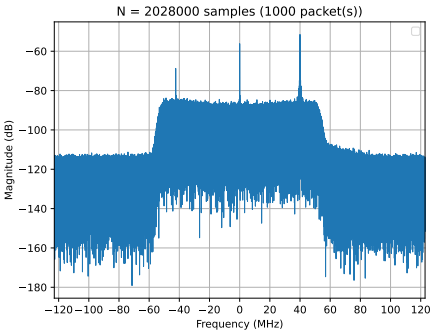

Figure 5.7: FFT of the received signal (N = 2028.0 ksample) on the server.

The spectrum shown in these Figures is that of a Pseudorandom Binary Sequence (PRBS) 100 MHz bandwidth signal 16-QAM modulated and a signal power of -80 dBm with a 3.5 GHz carrier frequency. Analyzing the spectrum with the naked eye, it looks the same as the spectrum compared with the VSA, which is configured to have as similar a number of points and frequency resolution as the instrument allows. Analysis of the figures also shows that as the number of FFT samples increases, the noise floor decreases. This was to be expected as it is a fundamental concept in DSP. By increasing the number of points in the FFT, the frequency resolution also increases, increasing the number of bins in the FFT. As a result, the power density of the noise in each bin decreases. At first glance, this technique appears to improve the distinction between noise and true signal peaks in the frequency representation. This is often called noise floor reduction and can help identify weak signals that would otherwise be hidden in the noise

when an FFT with fewer points is used. However, this technique makes two assumptions: the noise is uniformly distributed, which means it has a constant power density at all frequencies, and the signal must be stationary during the observation window, that is, the joint probability distribution is time-invariant, or at least the mean and variance are. Both of these assumptions must be taken with care, especially the second one, where the size of the capture window is critical to capture signals using modulations of modern mobile communications, namely Orthogonal Frequency-Division Multiplexing (OFDM) used in 4G LTE and 5G NR. In general, and in an attempt to maintain simplicity and performance, throughout the tests, the captures will be made using just one or ten packets, made up of 8112 samples sampled at 122.88 Msps, which is equivalent to a capture window of 66 μs or 660 μs, respectively.

5.3.2 Signal Power Measurements

After validating the spectrum of the received signal, the next step was to test and validate the method for calculating the power of the received signal.

The procedure used was as follows: using a PRBS 100 MHz bandwidth signal with a 16-QAM modulation and a 2.4 GHz carrier, the signal power generated by the VSG was changed. It is important to note that the 16-QAM modulation scheme used imposes an Peak to Average Power Ratio (PAPR) of 5.86 dB, i.e. the maximum power entering the receiver does not correspond to the average value configured in the equipment, but almost quadruples it. To avoid damaging the receiver, the maximum power was limited to a value that did not exceed 0 dBm. The power values shown in this Section correspond to the average power value of the signal, so the maximum value is four times higher but far from the maximum limit mentioned. It should also be noted that the approximation detailed in Appendix C was used for all the power calculations in this Subsection.

Table 5.1 shows the measured power values for signal strengths between -10 dBm and -50 dBm without using the AD9371's digital reconfigurable gain and the additional gain of the LNA. The results show that the receiver begins to saturate at values above -10 dBm. Although the compression value is still quite low, higher power values were not used for safety reasons. At the lower end of the power range, it can be seen that the system struggles to measure signal power below -43 dBm due to the increasing influence of noise. As the power continues to decrease, the reading does not drop below -47 dBm, which represents the noise floor. At these levels, the system is completely unable to distinguish the signal from the noise. Throughout this work, 1 dBm of absolute error was used as the criterion for defining the upper and lower limits of power. Once this criterion has been established, it can be said that the dynamic range of the system is 36 dBm. This inadequate

value is due to the receiver's significant noise floor, caused by the high noise figure mentioned above.

Signal Power (dBm)	**Measured Power (dBm)**	**Absolute Error (dBm)**
-10.00	−9.82	0.18
-20.00	−20.04	0.04
-30.00	−29.95	0.05
-40.00	−39.78	0.12
-43.00	−42.40	0.60
-46.00	−45.00	1.00
-49.00	−47.37	1.63
-50.00	−47.32	2.68

Table 5.1: Measurements of received signal power and absolute error for different signal powers without digital and RF chain gains.

In order to lower the noise floor, the AD9371's maximum reconfigurable gain of 30 dB was employed, and the results are shown in Table 5.2. As we can see, the introduction of reconfigurable gain reduces the noise floor by around 24 dBm, while keeping the upper limit constant. The upper limit remained constant due to the receiver's internal compensation mechanism [25]. Thus, the dynamic range was increased to 60 dB.

Signal Power (dBm)	**Measured Power (dBm)**	**Absolute Error (dBm)**
-10.00	−10.00	0.00
-20.00	−20.04	0.04
-30.00	−29.98	0.02
-40.00	−39.91	0.09
-50.00	−49.95	0.05
-60.00	−59.97	0.03
-70.00	−69.14	0.86
-73.00	−71.50	1.50
-76.00	−73.64	2.36

Table 5.2: Measurements of received signal power and absolute error for different signal powers with 30 dB digital reconfigurable gain only.

Despite the reduction in the noise floor, the value of -70 dBm was not satisfactory, considering the power levels at which modern communication systems operate, as described in Section 2.4. Therefore, a gain was introduced through the external LNA mentioned in Section 4.4. At the carrier frequency of the signal used in this test, 2.4 GHz, the RF chain has a gain of 21.2 dB, including losses in the cable and connector.

Signal Power (dBm)	**Measured Power (dBm)**	**Absolute Error (dBm)**
-35.00	−34.97	0.03
-40.00	−39.98	0.02
-50.00	−50.02	0.02
-60.00	−60.04	0.04
-70.00	−69.99	0.01
-80.00	−79.94	0.06
-90.00	−90.44	0.44
-93.00	−91.72	1.28
-96.00	−93.57	2.43

Table 5.3: Measurements of received signal power and absolute error for different signal powers with 30 dB digital reconfigurable gain and 21.2 dB gain from the RF chain.

The results in Table 5.3 are as expected, with the noise floor dropping from -70 dBm to a satisfactory -90 dBm. Note that the thermal noise at a laboratory temperature of 294.15 K (21º C) and a bandwidth of 100 MHz is -93.9 dBm. Therefore, the value obtained comes dangerously close to the minimum noise floor achievable, considering only the existence of thermal noise. In practice, there are other noise sources that increase the noise floor. It could also be argued that with the introduction of the LNA the dynamic range was reduced to 55 dB. However, one could safely increase the power and test for -30 dBm average power, corresponding to a maximum power of -24.14. In this case, assuming a gain of 21.2 dB at 2.4 GHz, the AD9371's input power would not exceed 0 dBm. Also, assuming that the system would continue to operate in its linear range, the dynamic range would remain at 60 dB.

Finally, the gain of the RF chain and the noise floor of the receiver were measured for the various frequencies corresponding to the 4G LTE bands, the 2.4 GHz ISM band, and the n78 5GNR band. The results are shown in Table 5.4 and show that the noise floor is not only sufficiently low, nearly -90 dBm, but also invariant, making the system suitable for spectrum sensing of modern communication technologies.

The measured noise floor values are approximately 2 dBm higher than the theoretical calculations presented in Table 4.1. This discrepancy is not unexpected, since the measurements take into account not only the thermal noise analyzed but also all other sources of noise present in practical scenarios. Among these, an additional source of noise could be attributed to the relatively long power cables of the LNA, which are similar to the conditions existing in the ATCLL lamp posts and wall boxes.

Frequency (GHz)	RF gain (dB)	Noise floor (dBm)
0.8	21.4	-90
1.8	20.5	-89
2.1	21.0	-90
2.4	21.2	-90
2.6	22.0	-91
3.5	21.7	-91

Table 5.4: Measured RF chain gain and noise floor values at different frequencies.

5.3.3 Signal Filtering Results

To evaluate the performance of the filter implemented as described in Appendix B, the same signal from the previous test was used, with a power level of -80.30 dBm, was modulated with 16QAM scheme on a 2.6 GHz carrier with a bandwidth of 100 MHz. To ensure a consistent approach to testing the signal was recorded and stored in a file.

First, a filter with a cutoff frequency of 50 MHz was applied, expecting to halve the signal power, which corresponds to a reduction factor of -3 dBm, expecting a power level of -83.30 dBm. The spectrum of both the original and the filtered signal is shown in Figure 5.8. In this Figure, it is visible that although the transition band of the filter is not completely abrupt, the resulting power level is very close to the expected -83.52 dBm. This result highlights the effectiveness of the filter in terms of signal power.

For comparison with the previous value, the power of the filtered signal was calculated using a much higher-order filter with 100 taps. This filter has a much steeper transition band, as can be seen in Figure 5.9. However, the power level of the filtered signal shows only a slight variation, measuring -83.43 dBm. This result shows that a significant increase in the order of the filter, and thus its execution time, does not necessarily translate into a significant improvement in performance, especially in terms of signal power.

This procedure was replicated to evaluate the filter's performance for narrower bands, this time considering a cutoff frequency of 20 MHz. Once again, it was observed that although the spectrum has a visually more abrupt transition (as shown in Figures 5.10 and 5.11) when a higher order filter is used, the effect on signal power is not significant. The difference in power was only about 0.20 dBm, indicating that increasing the filter order does not significantly affect signal power in the context of narrower bandwidths.

The limitations of the filter, including its non-abrupt rejection and non-zero ripple in the passband, add an additional error term to the power calculation for the band of interest. It is critical to note that these filter non-idealities vary significantly with the cutoff frequency used and that the smaller the cutoff frequency relative to the Nyquist frequency, the greater

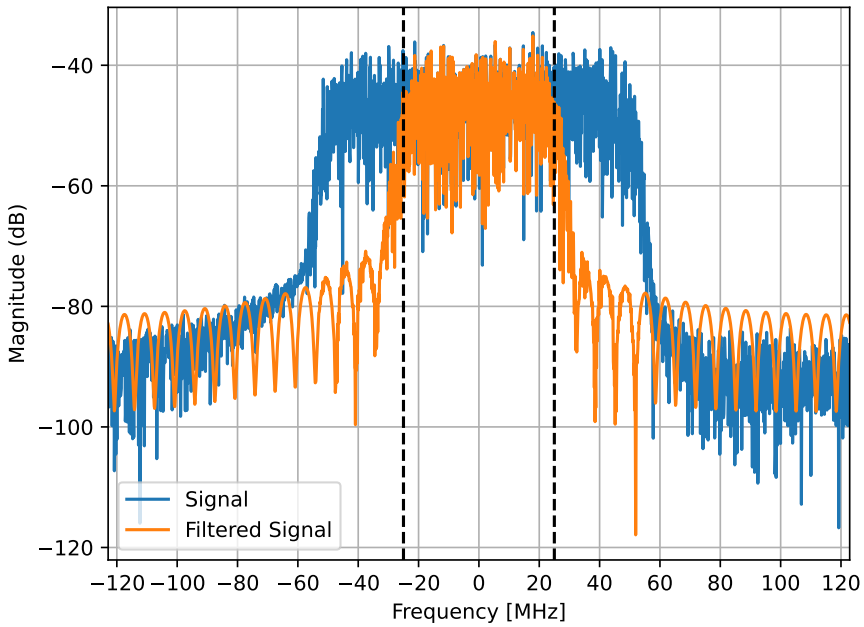

Figure 5.8: Spectrum of a 100 MHz signal before and after filtering using an FIR filter with 37 taps and 50 MHz cutoff frequency.

the deviation. The filter coefficients in this section were designed using a sampling frequency of 245.76 MHz, as per the Loopback Design described in subsection 4.2.1. In scenarios where the goal is to filter a narrower band of the spectrum, it would be appropriate to use the Standard Design, which operates at a sampling frequency of 122.88 MHz, and design filters for that frequency accordingly.

In summary, this filtering process represents an extra error to consider in the signal power calculation. In addition to the +/- 0.5 dB bandpass ripple inaccuracy of the AD9371 receiver when set to a 100 MHz bandwidth, the approximation error of the power calculation method described in the Appendix C, and the non-total flatness of the LNA gain over the maximum bandwidth of 100 MHz. All of these sources of error contribute to the definition of a conservative uncertainty estimate of +/- 2.0 dBm in the power calculations.

5.3.4 Execution Time Results

Having validated the implementation of the algorithms, the next step is to benchmark them. It's important to remember that in a scenario where Standard Design is employed, the server must be able to handle a through-

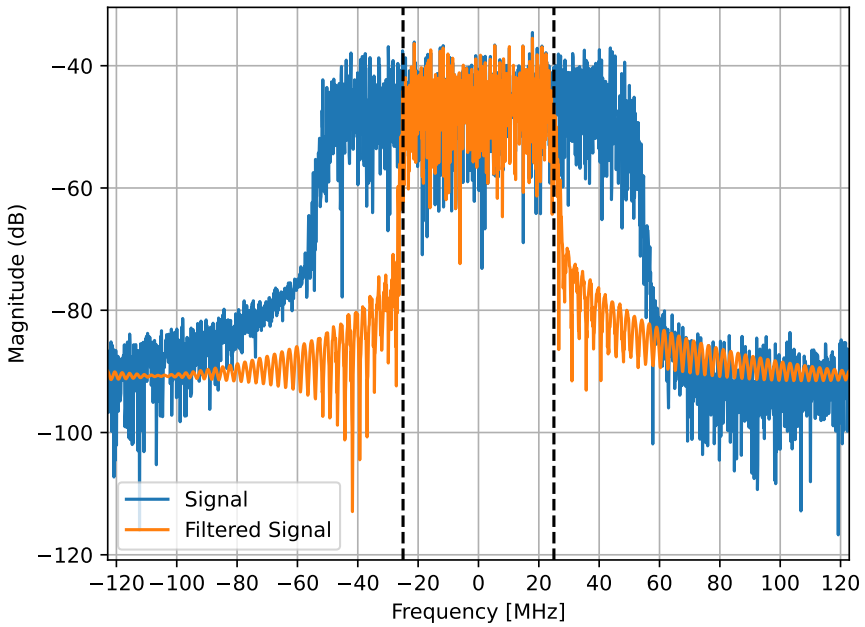

Figure 5.9: Spectrum of a 100 MHz signal before and after filtering using an FIR filter with 100 taps and 50 MHz cutoff frequency.

put of 3.93 Gbit/s, which implies that a new packet must be received and processed within 16.5 μs.

First, the performance of simply receiving network packets was analyzed using the network sockets described in subsection 4.2.4.

Trial	**Mean** (μs)	**STD** (μs)	**Max** (μs)
1	2.176	0.298	3.243
2	2.154	0.301	3.376
3	2.130	0.288	5.996
4	2.130	0.316	5.745
5	2.143	0.289	3.543

Table 5.5: Execution times for the *recvfrom* C function. Each of the 5 trials received 1000 packets.

As can be seen from the results in Table 5.5, the maximum value varies greatly from the average value, although the standard deviation is less than 15% of the average value. It should be noted that the network card stalls for more than 200 μs every 1200 packets and therefore the number of packets received in each experiment was lower than this value so that the maximum

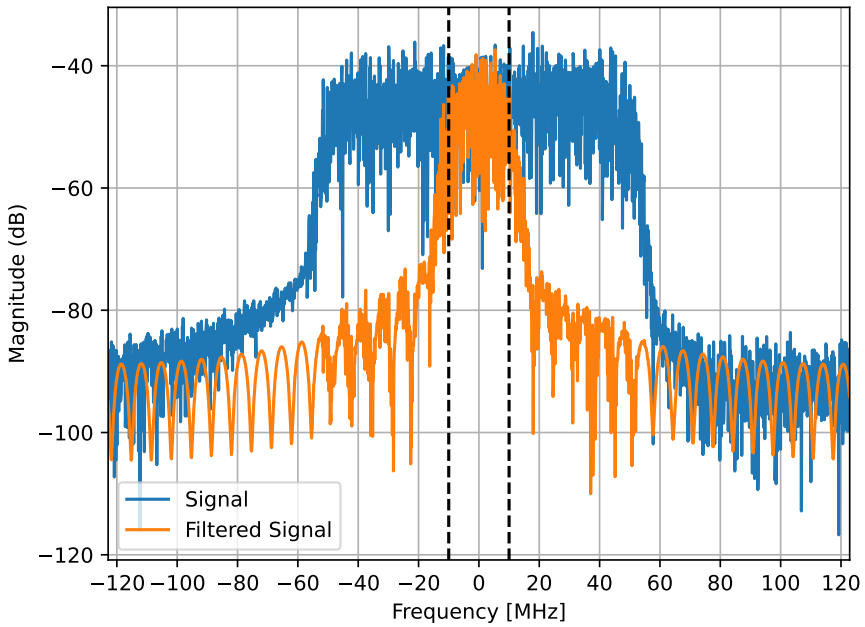

Figure 5.10: Spectrum of a 100 MHz signal before and after filtering using an FIR filter with 37 taps and 20 MHz cutoff frequency.

value was not completely dominated by this event. In order to mitigate these problems, it would be interesting to experiment with an ultra-low latency sub μs range network card, such as the NVIDIA Mellanox MCX512A-ACAT with a latency of 0.750 μs [61].

Trial	**Mean** (μs)	**STD** (μs)	**Max** (μs)
1	2.662	0.030	2.700
2	2.670	0.017	2.728
3	2.662	0.072	2.713
4	2.660	0.072	2.725
5	2.662	0.030	2.712

Table 5.6: Execution times for the byte shifting operation. Each of the 5 trials processed 10 million packets.

Table 5.6 shows the results of the execution times for the byte-shifting operation. The byte shifting of the IQ samples could be optimized using AVX-512 Vector Bit Manipulation Instructions. Alternatively, it would be much more practical to pre-invert the bytes directly in the FPGA fabric. The samples would still have to be separated and stored in an array, but

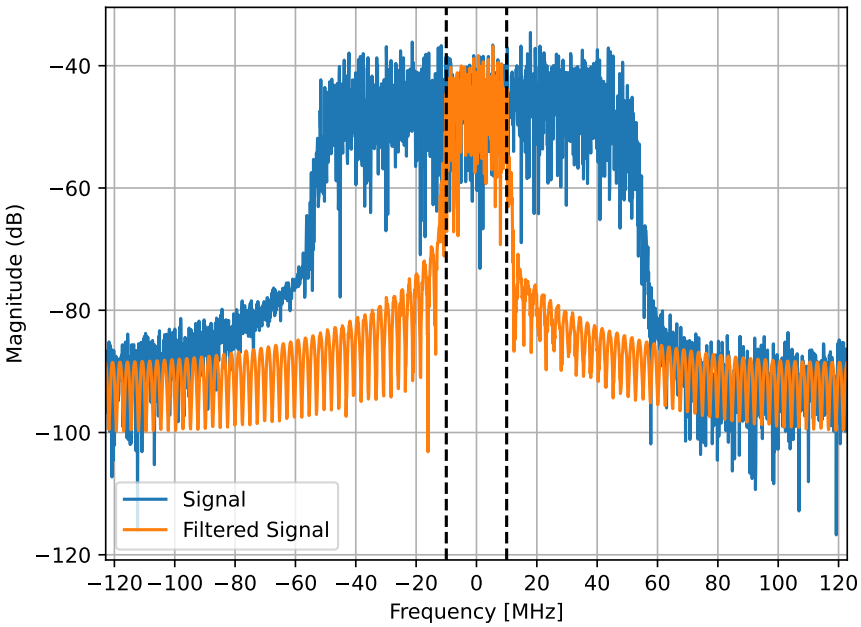

Figure 5.11: Spectrum of a 100 MHz signal before and after filtering using an FIR filter with 100 taps and 20 MHz cutoff frequency.

part of the processing would be reduced, and the use of SIMD instructions could speed up the process even more.

Benchmarking was also performed for the FFT, which achieved an average execution time of 21.849 microseconds processing IQ 2028 samples. For this reason, the calculation of FFT was not included in the processing pipeline in real-time scenarios. To meet the processing requirement within the 16.55 μs threshold, a reduction in the number of samples per FFT would be necessary. Reducing the number of samples to 1024 (the nearest smaller power of 2) significantly reduces the average execution time to 9.537 μs, which is within the acceptable range. This adjustment illustrates the trade-off between frequency resolution and processing speed which is critical to optimizing system performance.

The results in Table 5.7 once again show the indeterminism characterized by the significant increase in maximum time and standard deviation caused by the reception of packets. The most substantial share of the processing pipeline is digital filtering, which is benchmarked and described in more detail in Appendix B.

By eliminating the problem of the network card jamming every thousand or so packets with dedicated hardware for critical applications, the server

Trial	Mean (μs)	STD (μs)	Max (μs)
1	12.912	0.329	13.718
2	13.110	0.306	14.248
3	12.958	0.344	14.917
4	12.890	0.106	12.939
5	13.324	0.267	13.876

Table 5.7: Execution times for the whole software pipeline: receiving, byte shifting, filtering (with 37 filter taps), and signal power calculation. Each of the 5 trials processed 1000 packets

would be able to receive and calculate the signal strength in less than 5.5 μs, freeing time for some more demanding algorithms such as digital filtering. By implementing the remaining optimizations outlined above, it would be possible to reduce this time even further. Once this problem had been overcome, it could be claimed that the system was truly real-time.

5.4 Spectrum Sensing Campaign

Finally, to demonstrate the capabilities of the developed system, a multi-band indoor sensing campaign was conducted. To this end, the spectral occupancy of the 3GPP-defined LTE downlink bands n20 (800 MHz), n3 (1800 MHz), n1 (2100 MHz), n7 (2600 MHz), WLAN (2400-2483.5 MHz) contained in the unlicensed ISM, and the 3GPP-defined 5G NR n78 band (3500-3800 MHz) are probed for a user-defined period (20 hours in this case). To sense the n78 band, three segments of 100 MHz are considered, corresponding to four different operators [4].

Algorithm 1 contains the pseudo-code for the developed C program to conduct the spectrum occupancy campaign at the Telecommunications Institute. To estimate the power in each band, 10 RoE packets were used, which translates to 20280 samples. Note that in addition to using the pre-calibrated frequency variant gains for the RF chain, in this experiment the antenna gain was also subtracted when estimating the received signal power.

The spectral occupancy of 4G, 2.4 GHz ISM; and 5G bands are presented in Figs. 5.12 and 5.13 respectively. In both figures, a trend across all bands is noted. The highest occupancy occurs during the working period, while the minimum occupancy occurs during the dawn. This is expected since the campaign was carried out in a research institute.

Bear in mind that for these applications, execution times are not critical, and capturing packets at high rates on the server side easily creates data sets that exceed Gigabyte sizes. Large data sets are very inefficient to process, analyze, and generate meaningful graphs.

Concerning Figure. 5.12, the spectrum occupancy data are decimated

Algorithm 1: Algorithm used to perform the spectral occupancy campaign.

```
while elapsed time < desired capture duration do
    for frequency in frequencies_set do
        Send command to change frequency;
        Receive a given number of RoE packets;
        Decode IQ samples;
        Compute FIR filter response;
        Fetch frequency-dependent RF gains/attenuations;
        Compute signal power in dBm;
    end
    Save power values to timestamped file;
    Wait for a predefined idle time;
end
```

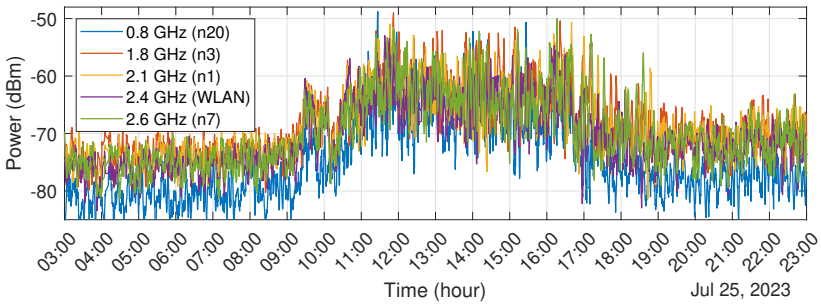

Figure 5.12: Spectral occupancy of 4G, and 2.4 GHz ISM bands.

and filtered using a second-order Savitzky-Golay filter with a span of 15 points. The data from Figure. 5.13 are not decimated, but the same filter type is used, with a span of 5 points. The mentioned filter type is adopted as it maintains amplitude accuracy without distorting the tendency of the signals.

The tests conducted in this chapter not only validate the performance of the individual subsystems but also confirm the overall effectiveness of the integrated spectral probe system in a controlled, multi-band indoor sensing campaign. These comprehensive tests, ranging from loopback verifications to centralized processing benchmarks, have solidified the system's capabilities in real-world scenarios. The following chapter will outline these results, draw conclusions, and highlight future lines of work.

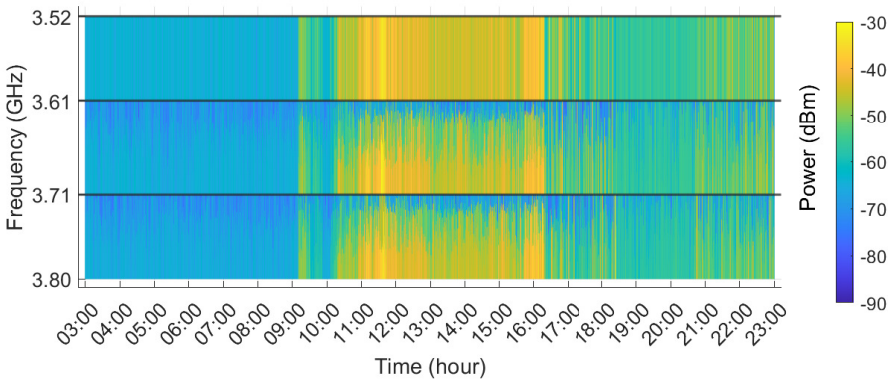

Figure 5.13: Spectral occupancy of 5G bands.

Chapter 6

Conclusion

6.1 Final Remarks

In conclusion, this master's thesis successfully developed a real-time flexible spectral probe, addressing the critical analysis of the increasingly contested electromagnetic spectrum. Through a comprehensive set of tests and validations, from loopback checks to centralized processing benchmarks, the system proved effective in an indoors multi-band spectral sensing measurement on the 4G LTE bands, 2.4 GHz ISM band, and the n78 5G band.

The implementation of this complex system required the study and familiarization of a wide range of subjects, including radio receivers and spectrum sensing techniques, RF instrumentation and measurements, state-of-the-art SDR technologies, real-time digital processing techniques, socket and network programming and configuring a real-time system Linux based-server.

Through this work, a solid foundation has been laid for future expansions and innovations in spectral detection. Implementing multiple probes in the Aveiro Tech City Living Lab represents an opportunity for conducting spatially distributed outdoor spectral occupancy campaigns, providing valuable insights into spectrum utilization in urban settings.

In summary, this work not only achieved its proposed objectives but also established a promising path for future investigations and developments in spectral detection. Through a combination of technological innovation and applied research, this project significantly contributes to the field of wireless communications and spectrum detection technologies.

6.2 Future Work

Apart from the most direct course of action, which is to expand the functionality and capabilities of the developed spectral probe, there are several emerging lines of work:

- Finish the installation of multiple probes on the Aveiro Tech City Living Lab (https://aveiro-living-lab.it.pt/) in the city of Aveiro, in order to allow us to carry out spatial distributed outdoor spectral occupancy campaigns.
- Unlock the platform's transmission capabilities, opening up new possibilities in which one or more systems behave as transmitters and others as receivers, in line-of- sight or non-line-of-sight scenarios. A testbed with such infrastructure would allow the research, development, experimentation, and validation of innovative radio architectures and techniques, with an emphasis on next-generation systems for smart city applications.
- Benchmark the packet reception using an ultra-low latency sub μs network card.
- Measure the noise floor of the system considering smaller bandwidths, for example, 10 and 20 MHz and perform sensing in more specific lower bandwidth scenarios.
- Optimize the remaining C functions using AVX-512 SIMD instructions to maximize performance.
- Incorporate the OpenGL high-performance C graphics library to enable the development of applications in C with a graphics component.
- Experiment and compare the performance of the FIR filtering function implemented using Intel's Integrated Performance Primitives library with the developed method and explore other DSP functions.

Appendix A

Real-Time Kernel Tunning and Low Jitter Computing Techniques

The technological development of modern CPUs, and operating systems has introduced a number of features aimed at maximizing throughput at the expense of latency and determinism [28][41]. These hardware and operating system features add several orders of magnitude to the variability in execution times of the same program. This variability is certainly unacceptable in real-time systems, especially with time constraints on the order of microseconds, as is the case in this work.

This appendix takes an in-depth look at configuring a low-jitter, real-time computing platform for benchmarking high-performance applications. The system is based on an Intel x86-64 Xeon® Gold 6336Y CPU running Ubuntu 22.04 LTS operating system with the 5.15 real-time kernel, and without a desktop environment. This configuration process requires full access to the system, both BIOS and root, and is specific to Intel x86-64 CPUs released after 2008, whose microarchitecture is codenamed Nehalem.

The first step in configuring the system was at the hardware/firmware level, disabling the following options from the Basic Input/Output System (BIOS):

- Disable all dynamic frequency scaling mechanisms (Intel's SpeedStep and Turbo Boost).
- Disable all power management/saving systems.
- Disable Intel's C-States.
- Disable Hyperthreading.
- Disable Virtualization.

- Disable System Management Mode.

In addition to disabling the above options, the CPU was configured to run at a fixed frequency of 3.1 GHz instead of the default of 2.4 GHz. This change reduced the number of available physical cores from 48 to 16 due to total power dissipation limitations. This reduction in the number of cores available doesn't affect the system's performance, as the idealized applications are mainly single-core single-threaded or multi-core multi-threaded but with a reduced number of cores (2 to 3). In this context, the gain from increasing the base frequency by 29% is a substantial advantage.

After the hardware configurations have been set up, the next step is to reduce variances at the kernel and operating system levels. The purpose of this process is to avoid any interruption of the intended application by a kernel or operating system task. In order to run a kernel-priority application, the Ubuntu 22.04 LTS operating system was installed with the latest pre-compiled 5.15 real-time kernel, with the optimized version for Intel processors. This specialized version of the real-time kernel is coupled with the Intel Time Coordinated Computing (TCC) and IEEE Time Sensitive Networking (TSN) technologies [38].

While the kernel used is a real-time kernel, it is important to note that an extensive tuning process is still required to ensure that a user-defined program does not suffer from preemption. Although there are guidelines for this tuning process [40] [41], it is essential to evaluate each system individually based on its specifications. Typically this process requires experimenting with different configurations until the desired behavior is achieved [39].

Kernel tuning started by modifying the GRand Unified Bootloader (GRUB), which is responsible for booting and loading the system kernel and is the default bootloader Linux systems. GRUB is essential for managing and selecting operating system boot options and parameters, allowing for initial kernel parameter adjustments that persist after rebooting. The changes made are detailed below:

- RCU Tuning:

  rcu_nocb_poll. This option enables polling for RCU callback offloading. Normally, the kernel might use a wake-up mechanism, but with this option, it uses CPUs polling, which can reduce latencies in real-time workloads.

  rcu_nocbs=4-6. Offloads RCU callback processing from the specified CPUs (in this case, CPUs 4, 5, and 6). It can be useful in real-time scenarios to offload this work from time-sensitive CPUs.

- Tickless Kernel:

nohz=on. Enables adaptive tickless mode in the kernel. When a CPU is idle, it stops the scheduler tick, saving power and possibly improving performance for some specific workloads.

nohz_full=4-6. Isolates CPUs 4, 5, and 6 from kernel timer ticks. The kernel stops sending timer ticks to the specified CPUs to minimize servicing interrupts and context switching on the specified CPUs.

- CPU Isolation: **isolcpus=managed_irq, domain,4-6**. This is used for CPU isolation. It specifies that CPUs 4, 5, and 6 should be isolated for user-defined applications. The **managed_irq** means that managed interrupts will not be delivered to the isolated CPUs.

- Thread and Interrupt Affinity:

  kthread_cpus=0,1. Restricts kernel threads to run only on the specified CPUs, in this case, CPUs 0 and 1. This is used to ensure that kernel threads don't run on isolated CPUs.

  irqaffinity=0,1: Specifies the default CPU affinity mask for the interrupt request (IRQ) handlers. Here, it's set to CPUs 0 and 1, meaning IRQs will by default be handled by these CPUs unless specifically reconfigured.

- Idle State: **idle=poll**. This specifies the idle loop behavior of the kernel. By setting it to poll, the CPUs will continuously poll while idle instead of entering a low-power state. This can improve wake-up times for latency-sensitive applications but at the cost of increased power consumption.

In addition to the changes made to the GRUB, a startup script was also developed that executes the following commands on startup:

sudo sysctl kernel.sched_rt_runtime_us=-1. This modifies the kernel's scheduler to allow real-time tasks unlimited access to the CPU, eliminating the default CPU time limit for these tasks.

sudo sysctl kernel.timer_migration=0. Disables timer migration between CPU cores, anchoring each timer interrupt to its initiating core to reduce latency.

for ((i=0; i<$num_of_cores; i++)); do echo performance > /sys/devices/system/cpu/cpu$i/cpufreq/scaling_governor; done. This sets the scaling governor to performance mode for each core.

Although the Linux real-time kernel enables applications to run with customizable kernel priorities and prevent them from being preempted by other kernel tasks, there is still the inherent preemption mechanism dictated by

the kernel's timer tick. The interrupt handler's frequency is set by the **CONFIG_HZ** constant, with a default value of 250 Hz [42]. This has the direct consequence of an unpreventable interruption every four milliseconds with a measured duration in the order of microsecond units. However, given the strict time requirements of the system, these interruptions are unacceptable. The workaround is to disable this mechanism by manually setting the kernel parameter **CONFIG_PREEMPT_NONE=y** and recompiling the Linux kernel. Although deactivating the real-time kernel may seem counterproductive, it enables the exploitation of the various real-time kernel mechanisms to increase the determinism of the system. This configuration is recommended for certain server applications [42].

After the hardware and kernel/operating system were configured meticulously, the next step was to develop a testing application. This application serves as an instrument for assessing the variability and performance. It allows for the assessment of the aforementioned hardware and kernel configurations, providing insights into their effectiveness and identifying areas that require further optimization for achieving the required system performance and reliability. In the development of the program designed for system determinism analysis and benchmarking, multiple techniques were employed to ensure optimal performance and accuracy. The program was written in C, a compiled language renowned for its efficiency and performance in critical applications. Memory locking was employed to maintain the program's memory in system memory, eliminating the latency and nondeterminism introduced by paging. The scheduling policy and kernel priority were defined to guarantee repeatability. Additionally, CPU affinity was set to bind the program to one of the pre-isolated cores, ensuring dedicated processing power and reducing potential interference from other system processes. The two programs below contain C functions that implement the memory lock, scheduling, and CPU affinity mechanisms, respectively.

```
1  [caption=C function to perform memory lock and set real-
       time scheduling policy and priority.]
2  void set_real_time()
3  {
4    // lock the program into RAM
5    if (mlockall(MCL_CURRENT | MCL_FUTURE) != 0)
6    {
7      perror("mlockall");
8      exit(EXIT_FAILURE);
9    }
10
11   // set scheduling policy and priority
12   struct sched_param param;
13   param.sched_priority = 99; // kernel priority
14
15   if (sched_setscheduler(0, SCHED_FIFO, &param) != 0)
```

```
16   {
17     perror("sched_setscheduler");
18     exit(EXIT_FAILURE);
19   }
20 }
```

The C programming language allows also for the development of multi-threaded applications in which each thread can be bound to a specific isolated CPU core. This approach guarantees that all threads are executed on a dedicated processor, which optimizes performance and increases deterministic execution times.

Listing A.1: C function to set CPU affinity to specific core.

```
1  void set_cpu_affinity()
2  {
3    static const uint16_t CPU_CORE = 5;
4
5    cpu_set_t set;
6    CPU_ZERO(&set);
7    CPU_SET(CPU_CORE, &set);
8
9    if(sched_setaffinity(0, sizeof(cpu_set_t), &set) < 0)
10   {
11     perror("sche_set_affinity");
12     exit(EXIT_FAILURE);
13   }
14 }
```

In order to benchmark and evaluate the temporal behavior of the system, it is essential to be able to count time with a sufficiently high resolution. In order to find out the resolution of the method, which is dependent on the architecture, the function *clock_getres()* was executed, which returned a value of 1 ns. This high resolution made the method look promising, so the following function was developed to count time in ns:

Listing A.2: C function using the *clock_gettime()* function from the C Time library.

```
1  static inline uint64_t nanos(void)
2  {
3    struct timespec ts;
4    clock_gettime(CLOCK_REALTIME, &ts);
5    uint64_t ns = (uint64_t)ts.tv_sec*1000000000ULL + (
       uint64_t)ts.tv_nsec;
6    return ns;
7  }
```

Note that the program was compiled using the gcc version 11.4.0 compiler with the -O2 optimization level because the -O3 option leads to too many

optimizations, which not only do not increase the program's performance, but increase its variability [60].

Listing A.3: Block of C code used for system analysis.

```
1     set_cpu_affinity();
2     set_real_time();
3
4     #define N 100000000 // 100M
5     static uint64_t time_arr[N];
6     static uint64_t start, end, time;
7     for(uint64_t i = 0; i < N; i++)
8     {
9       uint64_t start = nanos();
10
11      /* code to be measured:
12       volatile to prevent compiler optimizations */
13      volatile uint64_t tmp = nanos();
14
15      uint64_t end = nanos();
16
17      time = end - start;
18      time_arr[i] = time;
19    }
```

Trial	Time ns		
	Mean	STD	Max
1	96.0	10.1	60528
2	96.0	9.5	62236
3	96.2	10.0	62614
4	96.3	9.4	61438
5	96.0	9.5	60848

Table A.1: Execution performance time using the *clock_gettime()* function from the C Time library. Each of the five trials made 100 million function calls.

The results presented in Table A.1 revealed great variability, especially given the maximum value, in the method. It should also be mentioned that in all the tests the number of cycles exceeded 3100 cycles (1000 ns) more than 200 times.

This nondeterminism was excessive and so a lower-level alternative was sought, with as little overhead as possible. The alternative found is based on the Linux kernel's implementation of time measurement methods for modern x86 architectures by accessing the Time Stamp Counter (TSC) counter [31].

The TSC is a 64-bit register present on all modern x86 processors. The TSC register is monotolly incremented at every clock cycle [29]. As the

CPU was configured to run at a constant frequency of 3100 MHz, each clock cycle is approximately 0.3226 ns. Direct access to this register allows the development of a high-resolution, low-overhead timer, which is particularly important for performance benchmarking and real-time tasks.

Listing A.4: C function with inline Assembly code to read the TSC.

```
1  static inline uint64_t rdtscp()
2  {
3      uint64_t tsc;
4
5      __asm__ __volatile__ (  // begin an inline assembly
           block
6      "rdtscp;"           // read the tsc, store into rdx and
           rda registers
7      "shl\$32,%%rdx;"    // shift left 32 bits of TSC value
           into rdx
8      "or%%rax,%%rdx"     // combine the upper and lower TSC
           32 bits
9      : "=d"(tsc)         // output the TSC value to the tsc
           variable
10     :                   // no inputs
11     : "%rax", "%rcx"    // inform the compiler rdx and rda
           were modified
12     );
13
14     return tsc;  // Return the TSC value
15 }
```

Trial	Cycles			Time (ns)		
	Mean	STD	Max	Mean	STD	Max
1	38.0	0.0	49	12.3	0.0	15.8
2	38.0	0.0	78	12.3	0.0	25.2
3	38.0	0.0	55	12.3	0.0	17.7
4	38.0	0.1	75	12.3	0.0	24.2
5	38.0	0.1	85	12.3	0.0	27.4

Table A.2: Execution performance in cycles and time using the custom function based on the *rdtscp()* instruction. Each of the five trials made 100 million function calls.

The results from Table A.2 mainly show a very deterministic behavior and minimal overhead, since there are no library calls, both characteristics desired in this method. The near-zero standard deviation in cycles means that this method can achieve a resolution of 1 clock cycle (0.32 ns), providing extremely high precision for time measurement.

In conclusion, the computing platform was successfully configured, obtaining a system with the desired deterministic behavior.

Appendix B

High-Performance DSP Software Module

The main focus of this appendix is the implementation and optimization using SIMD instructions of FIR filters of arbitrary order assuming symmetric and antisymmetric coefficients.

To complement the aforementioned *fftw* library, some auxiliary functions were also implemented in C. These FFT support functions were not optimized since computing the FFT for N = 2028 samples is too time-consuming considering the system's time requirements, making optimization a non-priority. This simple software module implements the following functions in C:

- fftshift - rearranges a Fourier transform by shifting the zero frequency component to the center of the array.
- frequency shift - implements a frequency shift of a given value by multiplying the signal by a complex exponential.
- FIR filter - initialization, filtering, and deallocation.

The implementation of the FIR filter using SIMD AVX-512 instructions in C, as detailed in the above code, represents an optimized block processing approach suitable for handling data sets with a standard length of 2028 samples. This method takes advantage of the AVX-512's Vectorized Fused-Multiply Add instruction and 64-byte aligned memory arrays, optimizing memory by maximizing spatial locality. While implementing a filter exploiting symmetric coefficients could theoretically reduce by half the number of multiplications, in this SIMD context, such gains are outweighed by the inefficiencies of accessing non-contiguous memory addresses. This implementation highlights the balance between exploiting advanced hardware capabilities of modern high-end CPUs and managing the practical challenges of memory utilization for high-performance digital signal processing.

Note that the program was compiled using the gcc version 11.4.0 compiler with the -O2 optimization level and the mavx512f flag to generate code with the AVX512 extensions.

Listing B.1: Optimized C function with SIMD AVX-512 instructions to compute the response of an FIR filter of arbitrary order.

```
1  void filter_fir_avx_1d(const fir_filter_t * filt, float *
       y0, float * y1, const float * x0, const float * x1,
     uint32_t num_samples)
2  {
3    uint32_t taps = filt->taps;
4    const uint32_t vec_size = 512 / 32; // process 16x32-
       bit values once
5
6    uint32_t aligned_num_samples = (num_samples - taps) &
       ~(vec_size - 1);
7
8    for (uint16_t i = 0; i < aligned_num_samples; i +=
       vec_size)
9    {
10     __m512 y_real = _mm512_setzero_ps();
11     __m512 y_imag = _mm512_setzero_ps();
12     for (uint16_t j = 0; j < taps; j++)
13     {
14       /* broadcast b[j] to all 16x32 bit elems in 512-bit
            reg*/
15       __m512 coef = _mm512_set1_ps(filt->b[j]);
16
17       /* load 16x32-bit elems into 512-bit registers */
18       __m512 x_real = _mm512_load_ps(&x0[i + j]);
19       __m512 x_imag = _mm512_load_ps(&x1[i + j]);
20
21       /* fused multiply and add instruction for 16x32-bit
            floats */
22       /* y[n] = y[n] + (x[i-j] * b[j]) */
23       y_real = _mm512_fmadd_ps(x_real, coef, y_real);
24       y_imag = _mm512_fmadd_ps(x_imag, coef, y_imag);
25     }
26
27     _mm512_store_ps(&y0[i], y_real);
28     _mm512_store_ps(&y1[i], y_imag);
29   }
30
31   /* handle non-aligned samples */
32   for (uint32_t i = aligned_num_samples; i < num_samples
       - taps; i++)
33   {
34     float y_real = 0.0f;
35     float y_imag = 0.0f;
```

```
36
37      for (uint16_t j = 0; j < taps; j++)
38      {
39        y_real += x0[i + j] * filt->b[j]; // real part
40        y_imag += x1[i + j] * filt->b[j]; // imaginary part
41      }
42
43      y0[i] = y_real;
44      y1[i] = y_imag;
45    }
46  }
```

Filter Taps	**Mean** (μs)	**STD** (μs)	**Max** (μs)
5	1.867	0.018	1.977
10	2.98	0.015	3.134
20	5.285	0.025	5.559
30	6.759	0.013	7.012
40	8.358	0.025	8.591
50	10.583	0.030	10.88
60	11.614	0.024	11.835
70	13.668	0.028	13.890
80	16.035	0.031	16.249
90	16.724	0.028	16.988
100	18.924	0.029	19.073

Table B.1: Execution performance time of the custom FIR filter function for different filter tap counts. Each of the five trials made 1 million function calls.

The benchmark results shown in Figure B.1 show a low standard deviation and a maximum time close to the average execution time, indicating deterministic behavior, as desired. To analyze the dependency of the execution time on the number of filter taps, the graph in Figure B.1 has been plotted.

Analysis of Figure B.1 shows a clear linear relationship between the number of filter taps and the execution time for both optimized and non-optimized. This linearity is a consequence of the filter's computational complexity, which scales proportionally as the number of taps increases. In addition, the slopes of the lines in the graph provide valuable insight into the performance gains achieved by the optimized implementation. By comparing the slopes, it is possible to quantify the efficiency gain of about 13.35. In theory, the performance gain from using the AVX-512, which supports 16 simultaneous operations per register, would be expected to approach this number. In practice, however, this is diminished by the latency associated with these instructions and the fact that the number of samples is not a

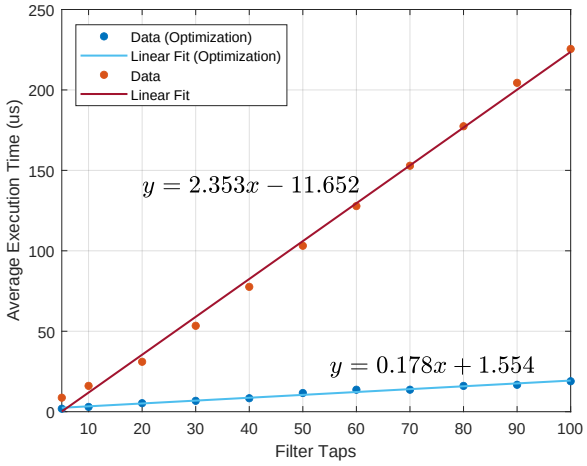

Figure B.1: Graph of the average execution time of the optimized and non-optimized FIR filter implementations as a function of the number of filter taps featuring linear regressions.

multiple of 16. Consequently, the remaining samples that do not fit into the 16-wide vectorized register must be processed using non-vectorized instructions, slightly reducing the overall performance gain.

By inverting the linear regression equation, the number of filter coeffi-

cients can be calculated based on the desired execution time. For example, with a target execution time of 8 μs, this method yields 37 taps. This inverse calculation is particularly useful for designing filters with performance constraints in mind, as it allows a direct relationship between execution time requirements and filter complexity.

Appendix C

Numerical Approximation and Optimized Implementation for Signal Power Calculation

This appendix explores a numerical approach to estimating the average power of a digital signal and its associated error for signals with different power levels and number of samples. Two implementations were developed to evaluate the performance gain, one trivial and one optimized using AVX-512 SIMD instructions.

The equation for calculating the average power of a discrete signal with N samples is given by [54]:

$$P = \frac{1}{N} \sum_{n=0}^{N-1} |x[n]|^2 \tag{C.1}$$

In the case of an IQ signal, each sample $x[n]$ of the signal x is represented as $x[n] = I[n] + Q[n]$.

Substituting in C.1 yields:

$$P = \frac{1}{N} \sum_{n=0}^{N-1} |I[n] + Q[n]|^2 \tag{C.2}$$

The IQ digital samples are encoded by the ADC in values between -8193 and 8192. To convert the values to an electrical quantity in V, the manufacturer specifies the conversion factor depending on the configuration of the transceiver [25]. For the default setting, the ADC range limits correspond to +/- 280 mV. Taking this into account, the following formula is used to map the ADC range values to voltage:

$$V_{\text{out}} = (val - \text{ADC\_MIN})\frac{(\text{V\_MAX} - \text{V\_MIN})}{\text{ADC\_MAX} - \text{ADC\_MIN}} + \text{V\_MIN} \quad \text{(C.3)}$$

As V_MAX = – V_MIN = V_AMP, substituting the values in C.3 leads to:

$$\begin{aligned} V_{\text{out}} &= \text{val} \cdot \frac{0.560}{16385} + \left(\frac{0.560}{16385}(-8193) - 0.280\right) = \text{val} \cdot M + K, \\ M &= \frac{0.560}{16385}, K = \left(\frac{0.560}{16385}(-8193) - 0.280\right) \end{aligned} \quad \text{(C.4)}$$

Equation C.4 demonstrates that converting each component of an IQ sample requires performing one floating-point multiplication and floating-point one addition.

Extrapolating the reasoning, for a signal of N IQ samples it is necessary to perform $2N$ floating-point multiplications and $2N$ floating-point additions.

After converting each sample to a voltage value, $v[n]$, the next step is to calculate the electrical power of the sample in Watt, denoted as $P[n]$, using the following formula:

$$P[n] = \frac{v[n]^2}{Z_0} \qquad \text{(W)} \quad \text{(C.5)}$$

Considering a normalized impedance $Z_0 = 1$ the equation (C.5) simplifies to:

$$P[n] = v[n]^2 \qquad \text{(W)} \quad \text{(C.6)}$$

The power in Watt of an IQ sample is then:

$$P[n] = |I_v[n] + Q_v[n]|^2 \qquad \text{(W)} \quad \text{(C.7)}$$

Where I_v and Q_v denote the I and Q components converted to voltage values.

Substituting $I[n]$ for $I_v[n]$ and $Q[n]$ for $Q_v[n]$ in C.1 gives the final equation for calculating the power in Watt of an IQ signal of N samples:

$$P = \frac{1}{N}\sum_{n=0}^{N-1} |I_v[n] + Q_v[n]|^2 \qquad \text{(W)} \quad \text{(C.8)}$$

Equation C.8 shows that in order to compute the power of an IQ signal with N samples, it is necessary to perform $2N$ floating-point multiplications plus $2N$ floating-point additions. Counting the $2N$ floating-point multiplications and $2N$ floating-point additions from the conversion process to

voltage values gives a total of $4N$ floating-point multiplications and $4N$ floating-point additions.

In order to reduce the computational load, it would be logical to perform as many operations as possible with the ADC's encoded values and then convert them to electrical power.

Substituting each component in equation C.2 with equation C.1 yields:

$$\begin{aligned}
P &= \frac{1}{N}\sum_{n=0}^{N-1} |(I[n]\cdot M + K) + (Q[n]\cdot M + K)|^2 \\
&= \frac{1}{N}\sum_{n=0}^{N-1} (I[n]^2\cdot M^2 + 2M\cdot K\cdot I[n] + K^2 + Q[n]^2\cdot M^2 + 2M\cdot K\cdot Q[n] + K^2) \\
&= \frac{1}{N}\sum_{n=0}^{N-1} \left[M^2(I[n]^2 + Q[n]^2)\right] + \sum_{n=0}^{N-1}[2MK(I[n] + Q[n])] + \sum_{n=0}^{N-1} 2K^2 \\
P &= \frac{M^2}{N}\sum_{n=0}^{N-1}(I[n]^2 + Q[n]^2) + \frac{2MK}{N}\sum_{n=0}^{N-1}(I[n] + Q[n]) + 2K^2 \quad \text{(W)} \qquad \text{(C.9)}
\end{aligned}$$

The equation C.9 enables computations to be performed without the need to convert voltage values and breaks down the power calculation into separate terms. Separating the calculation into terms gives rise to an approximation by considering only the term with the sum of the squared samples:

$$P \approx \frac{M^2}{N}\sum_{n=0}^{N-1}(I[n]^2 + Q[n]^2) \quad \text{(W)} \qquad \text{(C.10)}$$

The main advantage of this approximation is that it only requires $2N$ integer multiplications and $2N$ integer sums.

The approximation error is expected to be smaller the higher the signal power, with a non-linear decrease. To evaluate the approximation error of the C.10 equation depending on the signal power level, a simulation script was developed in Matlab using the Communications Toolbox. The script generates a 16-QAM signal with N samples with power levels ranging from -60 to -10 dBm and with a specified bandwidth, then simulates the ADC sampling, approximates the signal power, and ultimately calculates the error in the approximation. The size of the signals was chosen to match the number of samples carried in one packet and ten eCPRI packets.

The graph in Figure C.1 is as expected, as the power of the received signal increases, the approximation error decreases with a non-linear behavior.

An initial assessment might suggest that an error at a power level of -50 dBm of almost 0.1 dBm appears significant, especially given the inherently low power of the signals under observation. However, this perception changes when the gain provided by the receiver is taken into account. By factoring in a receiver gain of at least 50 dB, one can effectively map this

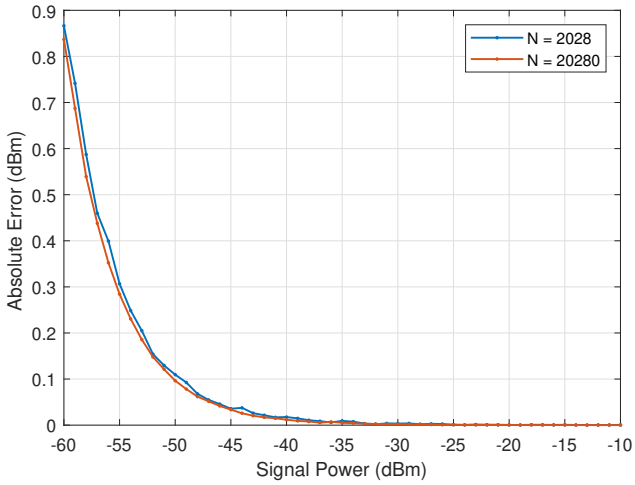

Figure C.1: Absolute error of the signal power approximation as a function of the signal approximation power for signals with 2028 and 20280 samples.

signal power to a -100 dBm power value. In this context, an error approximation of only 0.1 dBm is not only acceptable but significantly less than other inherent uncertainties in the system.

Listing C.1: Optimized C function with SIMD AVX-512 instructions to approximate the power of an IQ signal.

```
1  static inline double compute_signal_power_avx_float(const
       float * i_vec, const float * q_vec, const uint32_t
     num_samples)
2  {
3      const uint32_t vec_size = 512/32; // 16 floats
         processed at once
4      __m512 total_sum_squares_vec = _mm512_setzero_ps();
5
6      for (uint32_t i = 0; i < num_samples; i += vec_size)
7      {
8          /* load unaligned 16x32-bit floats */
9          __m512 i_data = _mm512_load_ps(&i_vec[i]);
10         __m512 q_data = _mm512_load_ps(&q_vec[i]);
11
12         /* square the i and q data */
13         __m512 i_square = _mm512_mul_ps(i_data, i_data);
14         __m512 q_square = _mm512_mul_ps(q_data, q_data);
15
16         /* sum the squares */
17         __m512 sum_squares = _mm512_add_ps(i_square,
             q_square);
18
```

```
19          /* accumulate the sum of squares */
20          total_sum_squares_vec = _mm512_add_ps(
                total_sum_squares_vec, sum_squares);
21      }
22
23      float sum_squares_array[vec_size];
24      _mm512_store_ps(sum_squares_array,
            total_sum_squares_vec);
25
26      /* add all the resulting 16x32-bit floats into a
            single float */
27      float sum_squares = 0.0f;
28      for (int i = 0; i < vec_size; ++i)
29      {
30          sum_squares += sum_squares_array[i];
31      }
32
33      double power_w = M_SQ * (double)sum_squares / (double
            )num_samples;
34      return 10 * log10(power_w) + 30.0;  // convert Watt
            to dBm
35  }
```

Trial	W/o Approximation		Approximation		Approx. + Opti.	
	Mean (μs)	STD (μs)	Mean (μs)	STD (μs)	Mean (μs)	STD (μs)
1	2.640	0.075	1.708	0.212	0.209	0.018
2	2.640	0.073	1.707	0.209	0.212	0.017
3	2.640	0.068	1.707	0.211	0.209	0.017
4	2.639	0.062	1.707	0.209	0.211	0.017
5	2.640	0.068	1.707	0.210	0.210	0.018

Table C.1: Execution performance time for different signal power calculation implementations considering a signal with 2028 IQ samples. Each of the five trials made 10 million function calls.

The execution time results presented in Table C.1, show the performance gains of the approximation implementation (1.5x) and the optimized approximation implementation (12.6x) over the baseline implementation, reaching an average execution time of around 200 ns. Note that all implementations were compiled using the same -O2 optimization level of gcc and executed on the platform described in Appendix A.

The performance gain obtained over an order of magnitude with an error of less than 0.1 dBm for signals with powers greater than -100 dBm makes this approximation an attractive method. By minimizing the computational load with this calculation, it is possible to invest more processing time in other more demanding tasks, such as filtering.

Appendix D

System Implementation for On-Site Deployment

The transition to on-site deployment within the ATCLL project required an arduous process of migration and adaptation, particularly with respect to the system's hardware architecture. This involved shifting the design to the Trenz Electronic TEBF0808 carrier board and the Trenz Electronic TE0808 UltraSoM+ TE0808 MPSoC module. This phase was particularly daunting due to the lack of documentation and, in some cases, inaccuracies in the information available.

The migration to the TEBF0808 board for field use as part of the ATCLL project initially presented several challenges, with limited and sometimes inaccurate documentation dictating a slow migration process. The task was further complicated by the need to manually remap pins to match the new development board, a time-consuming process. In addition, configuring the second-stage boot loader to meet the new board specifications was critical to ensure a seamless boot process. A significant amount of development effort was devoted to firmware customization, specifically configuring the on-board reference clock generator, which is essential for generating the 122.88 MHz clock for the AD9371. This was a modification from the lab setup, which used an external module for clock generation. In addition, the embedded Linux project underwent considerable customization to adapt to the new hardware environment. Critical to the deployment was the establishment of remote setup, reset, and reprogramming capabilities. System resets were performed via GPIO pins through the APU using the NCT5104D GPIO Linux driver, and both FPGA and processing system reprogramming was performed remotely via USB UART/JTAG using the Vivado hardware server running on the APU. This capability was critical for maintaining and updating the system in the field.

Finally, Figure D.1 shows one of the kits ready for installation. In total, four of these kits have been assembled and three are already deployed,

marking a significant milestone in the practical implementation phase of the project.

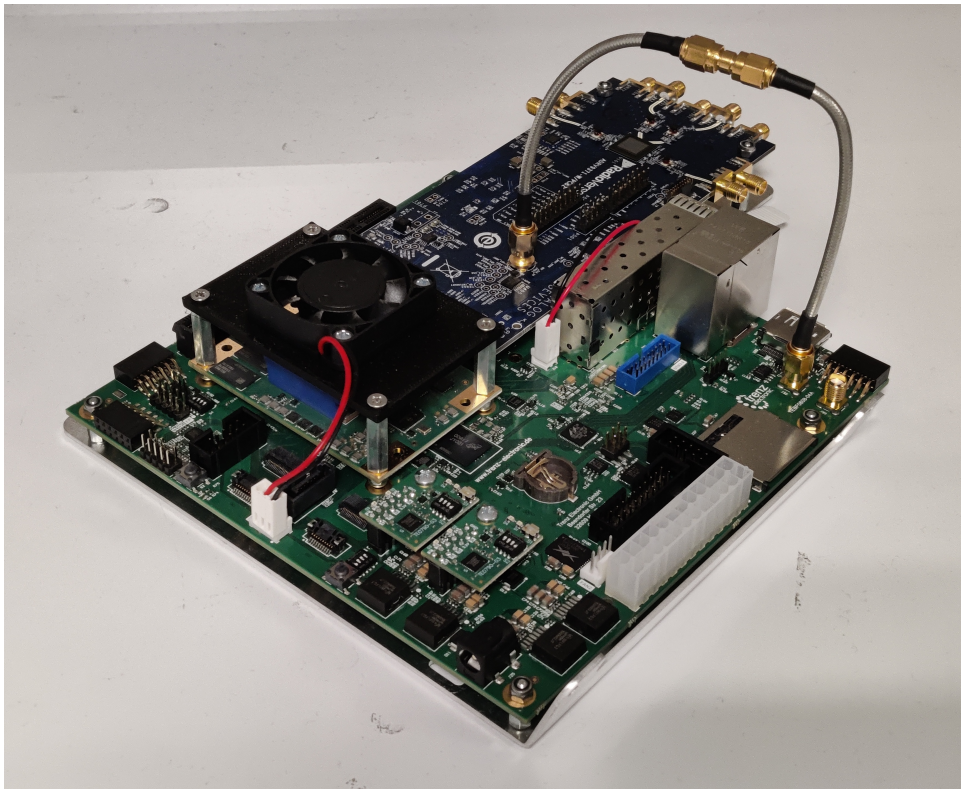

Figure D.1: Field-ready system setup.

Bibliography

[1] P. Rito et al., "Aveiro Tech City Living Lab: A Communication, Sensing, and Computing Platform for City Environments", in *IEEE Internet of Things Journal,* vol. 10, no. 15, pp. 13489-13510, 1 Aug.1, 2023, doi: 10.1109/JIOT.2023.3262627.

[2] Analog Devices, " ADALM-PLUTO Software-Defined Radio Active Learning Module", https://www.analog.com/en/design-center/evaluation-hardware-and-software/evaluation-boards-kits/adalm-pluto.html (Accessed July 21, 2023)

[3] innovationatwork.ieee.org, "Demand for Wi-Fi 6 is growing", https://innovationatwork.ieee.org/demand-for-wi-fi-6-is-growing/ (Accessed July 20, 2023).

[4] Spectrum Tracker, "Europe West", https://www.spectrum-tracker.com/Portugal (Accessed Oct. 27, 2023)

[5] U. Wetzker, I. Splitt, M. Zimmerling, C. A. Boano and K. Römer, "Troubleshooting Wireless Coexistence Problems in the Industrial Internet of Things", in *IEEE Intl Conference on Computational Science and Engineering (CSE, IEEE Intl Conference on Embedded and Ubiquitous Computing (EUC), and 15th Intl Symposium on Distributed Computing and Applications for Business Engineering (DCABES),* 2016, pp. 98-98, doi: 10.1109/CSE-EUC-DCABES.2016.167.

[6] G. Halfacree, "Semtech Announces 2.4GHz LoRa Product Plans for Single-Frequency Global Deployments" https://www.hackster.io/news/semtech-announces-2-4ghz-lora-product-plans-for-single-frequency-global-deployments-4f708d0ec116 (Accessed Oct. 26, 2023)

[7] C. Luo, J. Ji, Q. Wang, X. Chen and P. Li, 'Channel State Information Prediction for 5G Wireless Communications: A Deep Learning Approach," in *IEEE Transactions on Network Science and Engineering*, vol. 7, no. 1, pp. 227-236, 1 Jan.-March 2020, doi: 10.1109/TNSE.2018.2848960.

[8] M. Ratni, D. Krupezevic, Zhaocheng Wang, and J. -U. Jurgensen, "Broadband digital direct down conversion receiver suitable for software defined radio", in *IEEE Int. Symposium on Personal, Indoor and Mobile Radio Communications,* 2002, pp. 100-104 vol.1, doi: 10.1109/PIMRC.2002.1046669.

[9] S. Ziafat, W. Ejaz, and H. Jamal, "Spectrum sensing techniques for cognitive radio networks: Performance analysis" in *IEEE MTT-S Int. Microwave Workshop Series on Intelligent Radio for Future Personal Terminals,* 2011, pp. 1-4, doi: 10.1109/IMWS2.2011.6027191.

[10] K. Wasayangkool, et al., "A Performance Comparison of four Moderns Spectrum Sensing techniques under Noise Uncertainty and different User Accessing Time", in *2020 8th International Electrical Engineering Congress (iEECON),* 2020, pp. 1-4, doi: 10.1109/iEECON48109.2020.229568.

[11] M. A. McHenry, et al., "Chicago spectrum occupancy measurements & analysis and a long-term studies proposal", in *Proc. of IEEE TAPAS,* 2006, doi:10.1145/1234388.1234389

[12] M. H. Islam et al., "Spectrum survey in Singapore: Occupancy measurements and analyses," in *Proc. of IEEE CrownCom,* 2008, pp. 1-7, doi: 10.1109/CROWNCOM.2008.4562457.

[13] M. Wellens, J. Wu and P. Mahonen, "Evaluation of spectrum occupancy in indoor and outdoor scenario in the context of cognitive radio," in *Proc. of IEEE CROWNCOM,* 2007, pp. 420-427, doi: 10.1109/CROWNCOM.2007.4549835.

[14] N. Faruk, et al., "Large scale spectrum survey in rural and urban environments within the 50 MHz–6 GHz bands," *Measurement,* 2016, vol. 91, pp. 228–238, ISSN 0263-2241, doi:10.1016/j.measurement.2016.05.046.

[15] S. Subramaniam, H. Reyes and N. Kaabouch, "Spectrum occupancy measurement: An autocorrelation based scanning technique using USRP," in *Proc. of IEEE WAMICON,* 2015, pp. 1-5, doi: 10.1109/WAMICON.2015.7120376.

[16] F. Z. el Bahi, H. Ghennioui and M. Zouak, "Indoor spectrum occupancy in Morocco for cognitive radio applications: Measurements and analysis," in *Proc. of IEEE WINCOM,* 2019, Fez, Morocco, 2019, pp. 1-6, doi: 10.1109/WINCOM47513.2019.8942455.

[17] A. Martian, C. Vladeanu and I. Marghescu, "Novel software defined radio testbed for spectrum occupancy measurements," in *Proc. of IEEE ISETC,* 2020, pp. 1-4, doi: 10.1109/ISETC50328.2020.9301075.

[18] J. Mitola and G. Q. Maguire, "Cognitive radio: Making software radios more personal," *IEEE Personal Commun. Mag.,* vol. 6, no. 4, pp. 13-18, Aug. 1999, doi: 10.1109/98.788210.13–18,

[19] Federal Communications Commission, "Notice of proposed rule making and order: Facilitating opportunities for flexible, efficient, and reliable spectrum use employing cognitive radio technologies," ET Docket No. 03-108, Feb. 2005.

[20] Anacom, "Frequências", https://www.anacom.pt/render.jsp?categoryId=382989 (Accessed June 10, 2023).

[21] Spectrum Tracker, "Portugal", https://www.spectrum-tracker.com/Portugal (Accessed June 10, 2023).

[22] Cisco, "Signal-to-Noise Ratio (SNR) and Wireless Signal Strength", https://documentation.meraki.com/MR/Wi-Fi_Basics_and_Best_Practices/Signal-to-Noise_Ratio_(SNR)_and_Wireless_Signal_Strength (Accessed Oct. 3, 2023).

[23] I. Kakalou et al., "A survey on spectrum sensing algorithms for cognitive radio networks," in *Proc. of IEEE MOCAST*, 2018.

[24] Analog Devices,"AD9371 Integrated, Dual RF Transceiver with Observation Path" https://www.analog.com/media/en/technical-documentation/data-sheets/AD9371.pdf (Accessed July 20, 2023).

[25] Analog Devices,"AD9371/AD9375 System Development User Guide UG-992" https://www.analog.com/media/en/technical-documentation/user-guides/ug-992.pdf (Accessed July. 20, 2023).

[26] J. Korhonen, "IEEE P1904.3 Radio over Ethernet short introduction", https://www.ieee802.org/1/files/public/docs2016/new-roe-jik-ieee1904dot3intro-0316-v00.pdf (Accessed September 21, 2023).

[27] Qorvo,"QPL9057 Ultra-Low Noise Flat Gain Amplifier" https://store.qorvo.com/products/detail/qpl9057evb1-qorvo/631973/ (Accessed July 20, 2023).

[28] G. Paoloni, "How to Benchmark Code Execution on Intel IA-32 and IA-64 Instruction Set Architectures" https://www.intel.com/content/dam/www/public/us/en/documents/white-papers/ia-32-ia-64-benchmark-code-execution-paper.pdf (Accessed: September 11, 2023).

[29] Intel, "Intel® 64 and IA-32 Architectures Software Developer Manuals", https://www.intel.com/content/www/us/en/developer/articles/technical/intel-sdm.html (Accessed: 11 September, 2023).

[30] Intel, “Intel® Intrinsics Guide”, https://www.intel.com/content/www/us/en/docs/intrinsics-guide/index.html (Accessed: Oct. 4, 2023).

[31] L. Torvalds, “Linux kernel”, https://github.com/torvalds/linux/ blob/-master/arch/x86/kernel/tsc.c (Accessed: Oct. 5, 2023).

[32] Broadcom, “BCM57410 Ethernet NICs”, https://docs.broadcom.com/doc/12381557 (Accessed July 20, 2023).

[33] Scipy, “firfilter.c”, https://github.com/scipy/scipy/blob/main/scipy/signal/_firfilter.c (Accessed June 27, 2023).

[34] GNU Radio, “Volk - The Vector Optimized Library of Kernels”, https://github.com/gnuradio/volk/blob/main/kernels/volk/volk_32f_x2_dot_prod_32f.h (Accessed June 27, 2023).

[35] Steven W. Smith, ‘The Scientist and Engineer’s Guide to Digital Signal Processing”, Second Edition, California Technical Publishing, 1999, ISBN 0-9660176-7-6, ISBN 0-9660176-4-1, ISBN 0-9660176-6-8, chapter 19.

[36] J. Smith “Kaiser Window Beta Parameter” https://ccrma.stanford.edu/%7Ej/sasp/Kaiser_Window_Beta _Parameter.html (Accessed July 20, 2023).

[37] MathWorks, “Kaiser Window”, https://www.mathworks.com/help/signal/ug/kaiserwindow.html (Accessed: July 20, 2023).

[38] Canonical, “Real-time Ubuntu is now generally available”, https://ubuntu.com/blog/real-time-ubuntu-is-now-generally-available (Accessed June 20, 2023).

[39] E. Barbieri, “Technical deep-dive into a real-time kernel”, https://ubuntu.com/blog/real-time-kernel-technical (Accessed June 22, 2023).

[40] E. Barbieri, “Tuning a real-time kernel”, https://ubuntu.com/blog/real-time-kernel-tuning (Accessed June 22, 2023).

[41] E. Khartchenko, “Optimizing Computer Applications for Latency: Part 1: Configuring the Hardware”, https://www.intel.com/content/www/us/en/developer/articles/technical/optimizing-computer-applications-for-latency-part-1-configuring-the-hardware.html (Accessed June 22, 2023).

[42] E. Barbieri, “Low latency Linux for industrial embedded systems – Part II”, https://ubuntu.com/blog/industrial-embedded-systems-ii (Accessed Oct. 22, 2023).

[43] Rohde & Schwarz, "Measuring with Modern Spectrum Analyzers Educational Note", https://scdn.rohde-schwarz.com/ur/pws/dl_downloads/dl_application/application_notes/1ma201_1/1MA201_9e_spectrum_analyzers_meas.pdf (Accessed Oct. 25, 2023).

[44] Matteo Frigo and Steven G. Johnson, "FFTW Documentation", https://www.fftw.org/#documentation (Accessed May 7, 2023).

[45] AMD, "Radio over Ethernet Framer v3.0 Product Brief (PB056)", https://docs.xilinx.com/v/u/en-US/pb056-radio-over-ethernet (Accessed March 10, 2023).

[46] CPRI corporation, "Common Public Radio Interface: eCPRI Interface Specification", http://www.cpri.info/downloads/eCPRI_v_2.0_2019_05_10c.pdf (Accessed Oct. 20, 2023)

[47] Oracle, "Open Systems Interconnection (OSI) Reference Model", https://docs.oracle.com/cd/E19504-01/802-5886/intro-45828/index.html (Accessed May 7, 2023).

[48] IBM, "Raw Sockets", https://www.ibm.com/docs/en/zvm/7.2?topic=types-raw-sockets (Accessed May 7, 2023).

[49] Microchip, "Berkeley Sockets", https://microchipdeveloper.com/tcpip:berkeley-sockets (Accessed Oct. 2, 2023).

[50] AMD, "Zynq UltraScale+ MPSoC ZCU102 Evaluation Kit", https://www.xilinx.com/products/boards-and-kits/ek-u1-zcu102-g.html (Accessed Oct. 7, 2023).

[51] Analog Devices, "ADRV9371 Evaluation Board", https://www.analog.com/en/products/ad9371.html (Accessed Oct. 7, 2023).

[52] Digikey, "SI570-PROG-EVB", https://www.digikey.in/en/products/detail/skyworks-solutions-inc/SI570-PROG-EVB/4755476 (Accessed Oct.. 7, 2023).

[53] Intel, "Intel® Xeon® Gold 6336Y Processor 36M Cache, 2.40 GHz" https://www.intel.com/content/www/us/en/products/sku/215280/intel-xeon-gold-6336y-processor-36m-cache-2-40-ghz/specifications.html (Accessed 11 September, 2023).

[54] M. Kuhn, "Digital Signal Processing", https://www.cl.cam.ac.uk/teaching/1011/DSP/slides-2up.pdf (Accessed 11 Oct. 2023).

[55] L. Sangyoub, "Design and analysis of ultra-wide bandwidth impulse radio receiver", Ph.D. dissertation, University of Southern California, Los Angeles, 2002. [Online]. Available: https://www.researchgate.net/publication/35474287_Design_and_analysis_of_ultra-wide_bandwidth_impulse_radio_receiver

[56] Wikipedia, "Friis formulas for noise", https://en.wikipedia.org/wiki/Friis_formulas_for_noise (Accessed 12 Oct. 2023).

[57] everything RF, "What is Noise Floor?", https://www.everythingrf.com/community/what-is-noise-floor (Accessed 12 Oct. 2023).

[58] Anton Patyuchenko, "RF Signal Chain Discourse: Properties and Performance Metrics", https://www.analog.com/en/analog-dialogue/articles/rf-signal-chain-discourse.html (Accessed 12 Oct. 2023).

[59] Keysight, "Fundamentals of RF and Microwave Noise Figure Measurements", https://www.keysight.com/us/en/assets/7018-06808/application-notes/5952-8255.pdf (Accessed 12 Oct. 2023).

[60] Gnu Project, "Options That Control Optimization", https://gcc.gnu.org/onlinedocs/gcc/Optimize-Options.html (Accessed 15 Oct. 2023).

[61] Mellanox, "ConnectX®-5 EN Card Product Brief", https://resource.fs.com/mall/file/datasheet/connectx-5-en-ethernet-adapter card-datasheet.pdf (Accessed Oct. 26, 2023)