UNIVERSITAT POLITÈCNICA DE CATALUNYA BARCELONATECH

cttc Centre Tecnològic de Telecomunicacions de Catalunya

PhD dissertation

A Flexible System-on-Chip FPGA Architecture for Prototyping Experimental GNSS Receivers

Author

Marc Majoral Ramoneda

PhD Advisors

Dr. Carles Fernández Prades and Dr. Javier Arribas Lázaro
Centre Tecnològic de Telecomunicacions de Catalunya
Parc Mediterrani de la Tecnologia
Av. Carl Friedrich Gauss 7, Building B4 08860 Castelldefels

PhD Tutor

Prof. Ana Isabel Pérez Neira
Universitat Politècnica de Catalunya (UPC)
Campus Nord UPC, Edifici D5
Jordi Girona, 1-3
08034 Barcelona

Universitat Politècnica de Catalunya

A la meva família i amics.

Abstract

Global Navigation Satellite System (GNSS) technology is evolving at a rapid pace, necessitating the use of advanced prototyping tools for researching new and innovative signals and systems. Prototyping plays a crucial role in the design and development process as it allows researchers to test and validate their ideas before large-scale deployment. It facilitates the exploration of concepts, enabling researchers to identify drawbacks and areas for improvement early in the development process. Moreover, it aids in demonstrating the feasibility and potential of a concept, making it easier to attract funding and support for further research.

Prototyping with readily available GNSS receivers presents challenges. Presently, GNSS receivers are mainly built-up in Application-Specific Integrated Circuits (ASICs), providing low power consumption with small size and low cost, but offering limited flexibility. While some commercial devices utilize Software-Defined Radio (SDR) techniques, they often incorporate proprietary code, restricting reconfigurability through a predefined Application Programming Interface (API).

Free and Open Source Software (FOSS) GNSS receivers have emerged as valuable resources in the realm of research and development, particularly for those in the field of satellite navigation. These software-based receivers are highly appreciated for their customization and flexibility, allowing researchers to tailor the software to suit specific experimental requirements or to innovate new signal processing algorithms. However, compared to their hardware-based counterparts, these receivers are generally less power-efficient and offer lower performance. This is because they usually operate on general-purpose processors, like those in regular computers, which are not designed for low-power usage.

This thesis focuses on designing and developing a low-cost System on Chip (SoC) Field Programmable Gate Array (FPGA) architecture for prototyping experimental GNSS receivers. This architecture addresses the limitations of commercial GNSS receivers by emphasizing customization, flexibility, and reprogrammability, and it achieves improved power efficiency over conventional designs reliant on general-purpose processors. The core idea is to combine the versatility and adaptability of SDR concepts with the massive parallelism and improved power consumption of reprogrammable hardware, offering the best of both worlds. This combination enables the development of compact, portable, multi-band, and multi-constellation GNSS receivers, facilitating the development of embedded devices suitable for field testing. Additionally, the core GNSS processing engine is based on a free and open-source software receiver, enabling detailed access to the signal processing path and thorough cross-examination of the underlying mechanisms.

This thesis also introduces a comprehensive design methodology for developing novel GNSS signal processing algorithms and concept demonstrators, utilizing the newly proposed SoC FPGA architecture. This methodology places significant emphasis on code reuse, which is relevant for reducing developmental costs and time. Reusing code increases the added value of the proposed architecture by enabling the progressive incorporation of new features into the architecture, further enhancing its adaptability and efficiency. This facilitates the adaptation of GNSS receivers across various research applications, thereby streamlining and economizing the development process.

The practical applications of this architecture have been demonstrated through three developments: a spaceborne GNSS receiver, a GNSS rebroadcaster, and a High Sensitivity (HS) GNSS receiver.

The spaceborne receiver is a low-power, multi-frequency, multi-constellation concept demonstrator designed for space applications, featuring a form factor suitable for CubeSats. It has been tested using live signals, assessing the quality of the navigation solutions, and demonstrating its capability to process GNSS signals in Low Earth Orbit (LEO) scenarios.

The GNSS rebroadcaster is a low-power, Small Form Factor (SFF) signal generator and regenerator capable of transmitting and retransmitting GNSS signals with minimal delay. It can modify the retransmitted navigation solutions in real-time, potentially enabling the replication of movement dynamics in multiple scenarios. The rebroadcaster has undergone testing to assess the quality and the latency of the regenerated signals. This design integrates advanced capabilities for GNSS signal generation and regeneration within the proposed SoC FPGA architecture.

The HS GNSS receiver is a portable device designed for processing weak signals. It is capable of acquiring and tracking severely attenuated GNSS signals with a Carrier-to-Noise Density Ratio (C/N_0) down to 20 dB-Hz in real time. The HS GNSS receiver is ideal for developing and testing experimental GNSS algorithms, particularly those related to positioning in weak signal conditions. It has been evaluated for its ability to acquire and track weak signals and obtain navigation solutions.

The novel approach presented in this thesis facilitates the development of flexible and portable GNSS receiver and signal generator prototypes with non-standard capabilities. These prototypes are designed for operation in battery-powered devices, making them suitable for both laboratory experiments and field testing.

Resum

La ràpida evolució en la tecnologia de navegació per satèl·lit (GNSS) requereix eines de prototipatge avançades per a l'exploració de nous senyals i el desenvolupament de sistemes innovadors. El prototipatge és essencial en el procés de disseny i desenvolupament, ja que permet als investigadors provar i perfeccionar les seves idees abans de dur a terme una implementació a gran escala. Els models preliminars faciliten l'exploració de conceptes i ajuden a identificar limitacions i oportunitats de millora des de les primeres fases de desenvolupament d'un producte. A més, contribueixen a demostrar la viabilitat i el potencial dels dissenys, la qual cosa facilita l'obtenció de finançament i suport per a futures investigacions.

El prototipatge utilitzant receptors GNSS comercials planteja diversos reptes. En l'actualitat, aquests receptors es basen majoritàriament en circuits integrats d'aplicació específica (ASICs), els quals es caracteritzen per un consum energètic reduït, dimensions compactes i un cost baix, però ofereixen una flexibilitat limitada. Tot i que alguns dispositius comercials incorporen tècniques de ràdio definida per programari (SDR), aquests freqüentment contenen codi propietari que en restringeix la reconfiguració mitjançant una interfície de programació d'aplicacions (API) establerta pel fabricant.

Els receptors GNSS basats en programari lliure i codi obert han esdevingut recursos molt valuosos en el camp de la recerca i desenvolupament, especialment en el camp de la navegació per satèl·lit. Aquests receptors són molt valorats per la seva adaptabilitat i flexibilitat, permetent als investigadors adaptar el programari a necessitats experimentals específiques o desenvolupar nous algoritmes de processament de senyal. Tanmateix, els receptors definits per programari solen ser menys eficients energèticament en comparació amb els receptors basats en maquinari, ja que operen en processadors de propòsit general, que no estan optimitzats per a un baix consum energètic.

Aquesta tesi se centra en el disseny i desenvolupament d'una arquitectura de baix cost per al prototipatge de receptors GNSS experimentals, basada en sistemes en un xip amb matrius de portes lògiques programables in situ (SoC FPGA). Aquesta arquitectura supera les limitacions dels receptors GNSS comercials en termes d'adaptabilitat, flexibilitat i capacitat de reprogramació, i ofereix una eficiència energètica millorada en comparació amb els receptors basats en programari que depenen de processadors de propòsit general. L'estratègia consisteix a combinar la versatilitat de la ràdio definida per programari amb el paral·lelisme intensiu i el consum energètic optimitzat del maquinari reprogramable, proporcionant el millor de tots dos mons. Aquesta fusió permet el desenvolupament de receptors GNSS compactes, portàtils i capaços de treballar simultàniament en diverses bandes de freqüència i amb diferents sistemes GNSS, facilitant així el prototipatge de dispositius encastats adequats per a proves de camp. A més, el nucli de processament GNSS es basa en una implementació de programari lliure i obert, que proporciona un accés detallat a la cadena de processament de senyal i permet una exploració i modificació sense restriccions dels algoritmes utilitzats.

Aquesta tesi també presenta una metodologia de disseny per al desenvolupament de nous prototips i nous algoritmes de processament de senyal GNSS basats en l'arquitectura SoC FPGA que es proposa. Aquesta metodologia posa especial èmfasi en la reutilització de codi, aspecte clau per a reduir els costos i temps de desenvolupament. La reutilització de codi augmenta el valor afegit de l'arquitectura, permetent la incorporació progressiva de noves funcionalitats i millorant-ne la adaptabilitat i eficiència. Això facilita l'adaptació dels receptors GNSS a una àmplia gamma d'aplicacions de recerca, optimitzant i economitzant el procés de desenvolupament.

Les aplicacions pràctiques d'aquesta arquitectura s'han demostrat a través de tres prototips: un receptor GNSS per a l'espai, un retransmissor de senyals GNSS, i un receptor GNSS d'alta sensibilitat.

El receptor per a l'espai és un receptor GNSS multifreqüència i multi-constel·lació, dissenyat per a aplicacions espacials, de baix consum d'energia i amb un tamany adaptable per a CubeSats. Ha estat provat amb senyals a temps real, evaluant-ne la qualitat de les solucions de navegació, i demostrant la seva capacitat per processar senyals GNSS en escenaris d'òrbita baixa terrestre (LEO).

El retransmissor de senyals GNSS és un generador i regenerador de senyals de baix consum i tamany reduït, capaç de transmetre i retransmetre senyals GNSS amb un retard mínim. Aquest regenerador pot modificar les solucions de navegació retransmeses en temps real, permetent la simulació de dinàmiques de moviment en multiples escenaris. El retransmissor ha estat provat per verificar-ne la qualitat dels senyals regenerats i per avaluar-ne la latència. Aquest disseny integra funcionalitats avançades per a la generació i la regeneració eficient de senyals GNSS, adaptades específicament a l'arquitectura SoC FPGA que es proposa en aquesta tesi.

El receptor GNSS d'alta sensibilitat és un dispositiu portàtil dissenyat per processar senyals febles. És capaç d'adquirir i fer seguiment en temps real de senyals GNSS extremadament atenuats amb una relació entre la potència de la portadora i la densitat espectral de soroll (C/N_0) tant baixa com 20 dB-Hz. Aquest receptor és ideal per al desenvolupament i prova d'algoritmes GNSS experimentals, especialment en condicions de senyal feble. Ha estat provat per demostrar la seva capacitat per adquirir i fer seguiment de senyals febles, i per evaluar-ne la qualitat de les solucions de navegació.

L'enfocament innovador presentat en aquesta tesi facilita al desenvolupament de prototips experimentals de receptors i generadors de senyals GNSS flexibles i portàtils, aptes tant per a experiments de laboratori com per a proves de camp.

Acknowledgements

This thesis was conducted within the Signal Theory and Communications Department (TSC) of the Universitat Politècnica de Catalunya (UPC), concurrently with full-time employment at Centre Tecnològic de Telecomunicacions de Catalunya (CTTC). The research presented in this thesis was supported by a diverse array of funding sources and collaborations, highlighting the multi-institutional efforts involved.

The European Commission provided support under grant agreement H2020-687367-AUDITOR. The European Space Agency supported this research through contract AO/2-1647/17/NL/CRS. Additionally, part of the work was carried out in the framework of the European Space Agency contract 4000133576/20/NL/CRS with SENER Aeroespacial S.A., with the CTTC serving as the subcontractor, highlighting the project's industry collaboration.

Additional financial support for this research was provided by grant RTI2018-099722-B-I00, funded by the Spanish Ministry of Science and Innovation, and grant PID2021-128373OB-I00, funded by the Spanish Ministry of Science and Innovation through MCIN/AEI/10.13039/501100011033, and by the European Regional Development Fund (ERDF) "A way of making Europe". Additional support, also from the Spanish Ministry of Science and Innovation, was received through project CPP2021-008648, titled "Receptor Abierto y Flexible de Investigación por Satélite".

I would like to thank my thesis advisors, Dr. Carles Fernández-Prades, Dr. Javier Arribas, and Prof. Ana Isabel Pérez-Neira for their invaluable guidance and support throughout the journey of my thesis. Their expertise and encouragement have been pivotal in shaping both the direction and substance of my research. Carles and Javi's rigorous approach to scientific inquiry, alongside their keen insights, has not only enriched my work but also fostered my growth as a researcher. Their dedication and commitment to excellence have inspired me at every step, making this endeavor both enlightening and rewarding. I am profoundly thankful for their mentorship in my academic development. Finally, I would like to express my gratitude to all the individuals at CTTC who encouraged and supported me during this research endeavor.

Contents

List of Figures

List of Tables

Acronyms

C/N_0	Carrier-to-Noise Density Ratio
5G	5th Generation
A-GNSS	Assisted Global Navigation Satellite System
ADC	Analog to Digital Converter
AGC	Automatic Gain Control
AI	Artificial Intelligence
AltBOC	Alternative Binary Offset Carrier
AMBA	Advanced Microcontroller Bus Architecture
AMD	Advanced Micro Devices
API	Application Programming Interface
ARNS	Aeronautical Radio Navigation Service
ASIC	Application-Specific Integrated Circuit
AXI	Advanced eXtensible Interface
AXI4	Advanced eXtensible Interface 4th generation
BDT	Beidou Time
BPSK	Binary Phase Shift Keying
BRAM	Block Random Access Memory
C/A	Coarse Acquisition
CAF	Cross-ambiguity function
CBOC	Composite Binary Offset Carrier
CEP	Circular Error Probability
CFAR	Constant False Alarm Rate
CFO	Carrier Frequency Offset
CI	Coherent Integration
CLB	Configurable Logic Block
CMOS	Complementary Metal-Oxide-Semiconductor
CMT	Clock Management Tile
COTS	Commercial Off-The-Shelf
CPU	Central Processing Unit
CRC	Cyclic Redundancy Check
CSAC	Chip Scale Atomic Clock
CSV	Comma-Separated Values
CTTC	Centre Tecnològic de Telecomunicacions de Catalunya
CW	Continuous Wave
DAC	Digital-to-Analog Converter
DDR	Double Data Rate

DDR3	Double Data Rate 3
DDR4	Double Data Rate 4
DFT	Discrete Fourier Transform
DLL	Delay-Locked Loop
DMA	Direct Memory Access
DOP	Dilution of Precision
DPDI	Differential Post-Detection Integration
DRMS	Distance Root Mean Square
DSP	Digital Signal Processor
DSSS	Direct Sequence Spread Spectrum
E	Early
ECEF	Earth-Centered, Earth-Fixed
ENU	East-North–Up
eSDK	Extensible Software Development Kit
FDE	Fault Detection and Exclusion
FF	Flip-Flop
FFT	Fast Fourier Transform
FIFO	First In, First Out
FLL	Frequency-Locked Loop
FOSS	Free and Open Source Software
FPGA	Field Programmable Gate Array
FSM	Finite State Machine
GEO	Geosynchronous Earth Orbit
GIC	Generic Interrupt Controller
GIS	Geographic Information Systems
GLONASS	GLObalnaya NAvigatsionnaya Sputnikovaya Sistema
GLONASST	Glonass Time
GNSS	Global Navigation Satellite System
GPDIT	Post-Detection Integration Truncated
GPDITSD	Post-Detection Integration Truncated with Squaring Detector
GPIO	General Purpose Input/Output
GPL	General Public License
GPS	Global Positioning System
GPST	GPS System Time
GPU	Graphics Processing Unit
GPX	GPS Exchange Format
GST	Galileo System Time
HD	Hard Disk
HDL	Hardware Description Language
HLS	High-Level Synthesis
HS	High Sensitivity
I	In-Phase
I/O	Input/Output
I&D	Integrate and Dump
IC	Integrated Circuit
IF	Intermediate Frequency

IFFT	Inverse Fast Fourier Transform
IMU	Inertial Measurement Unit
INS	Inertial Navigation System
IO	Input/Output
ION	Institute of Navigation
IoT	Internet of Things
IP	Intellectual Property
IRQ	Interrupt Request
JTAG	Joint Test Action Group
KML	Keyhole Markup Language
KPI	Key Performance Indicator
L	Late
LAN	Local Area Network
LEO	Low Earth Orbit
LHCP	Left-Handed Circular Polarization
LNA	Low-Noise Amplifier
LS	Least Squares
LUT	Lookup Table
LUTRAM	Lookup Table Random-Access Memory
LVDS	Low Voltage Differential Signaling
MEO	Medium Earth Orbit
ML	Machine Learning
MMU	Memory Management Unit
MPSoC	Multi Processor System-on-Chip
MRSE	Mean Radial Spherical Error
Msps	Mega samples per second
NCI	Non-Coherent Integration
NCO	Numerically Controlled Oscillator
NMEA	National Marine Electronics Association
NPDI	Non-Coherent Post Detection Integration
NTP	Network Time Protocol
NTRIP	Network Transport of RTCM via Internet Protocol
OCXO	Oven-Controlled Crystal Oscillator
OpenCL	Open Computing Language
OS	Operating System
OTG	On-The-Go
P	Prompt
PC	Personal Computer
PCB	Printed Circuit Board
PCPS	Parallel Code Phase Search
PDI	Post-Detection Integration
PFA	Probability of False Alarm
PL	Programmable Logic
PLL	Phase-Locked Loop
POD	Precise Orbit Determination
PPA	Power, Performance and Area

ppb	parts per billion
ppm	parts per million
PPS	Pulse Per Second
PRN	Pseudo Random Noise
PS	Processing System
PVT	Position, Velocity and Time
Q	In-Quadrature
QSPI	Quad Serial Peripheral Interface
RAIM	Receiver Autonomous Integrity Monitoring
RAM	Random-Access Memory
RF	Radio Frequency
RFFE	Radio Frequency Front-End
RHCP	Right-Handed Circular Polarization
RINEX	Receiver Independent Exchange Format
RMSE	Root Mean Square Error
RNSS	Radio Navigation Satellite Service
ROM	Read Only Memory
RTCM	Radio Technical Commission for Maritime Services
RTL	Register Transfer Level
SAS	Spherical Accuracy Standard
SD	Secure Digital
SDK	Software Development Kit
SDR	Software-Defined Radio
SEP	Spherical Error Probable
SFF	Small Form Factor
SFTP	Secure File Transfer Protocol
SMP	Symmetric Multi-Processing
SNR	Signal-to-Noise Ratio
SoC	System on Chip
SoM	System on Module
SPI	Serial Peripheral Interface
SRAM	Static Random Access Memory
SSH	Secure Shell
TC	Telecommand
TCP	Transmission Control Protocol
TCP/IP	Transmission Control Protocol/Internet Protocol
TCXO	Temperature-Compensated Crystal Oscillator
ToA	Time of Arrival
ToF	Time of Flight
ToT	Time of Transmission
TOW	Time of Week
TTFF	Time to First Fix
UART	Universal Asynchronous Receiver / Transmitter
UIO	User Input/Output
USB	Universal Serial Bus
UTC	Coordinated Universal Time

VE	Very Early
VHDL	Very High Speed Integrated Circuit Hardware Description Language
VL	Very Late
WGS	World Geodetic System
XML	Extensible Markup Language

Chapter 1

Introduction

Over recent years, Global Navigation Satellite System (GNSS) technology has significantly grown in importance, exerting a profound influence across a diverse array of global sectors. GNSS is a system of satellites and ground-based stations that provide signals used for precise positioning and navigation anywhere on Earth's surface. It consists of multiple satellites in orbit around the Earth, which transmit signals that are received and used by GNSS receivers and other GNSS devices to determine their precise location, velocity, and time information. Examples of GNSS include Europe's Galileo, the United State's Global Positioning System (GPS), Russia's GLONASS and China's BeiDou. It is widely used in applications that require precise positioning, navigation, and timing synchronization across various domains, including disaggregated advanced 5th Generation (5G) radio access networks. GNSS technology is essential in transportation, enhancing safety and efficiency in vehicle, aircraft, and maritime navigation. In agriculture, it enables precision farming, leading to more effective resource use and improved crop management. Additionally, GNSS is essential in emergency services for providing precise and timely responses during disasters. Its significance also extends to scientific research, aiding in environmental monitoring and space exploration, thus demonstrating its broad impact on various aspects of modern life and industrial operations [1,2].

GNSS systems are rapidly evolving due to a confluence of technological advancements, increasing demands for precision, and global geopolitical factors. The need for more precise and reliable positioning, navigation, and timing services grows as industries and technologies advance. This demand drives continuous improvements in satellite technology, including more sophisticated payloads and extended satellite lifespans, enhancing the accuracy and reliability of GNSS services. New generations of satellites, such as those from the United States'

GPS, Russia's GLObalnaya NAvigatsionnaya Sputnikovaya Sistema (GLONASS), Europe's Galileo, and China's BeiDou, offer improved capabilities including better signal quality, increased accuracy, and expanded coverage areas. Enhancements often feature new capabilities like multi-frequency signals for civilian use, providing greater accuracy and resistance to interference. Furthermore, the addition of more satellites enhances the redundancy and reliability of GNSS signals.

In addition, GNSS technology is increasingly being integrated with emerging technologies, including augmented reality [3], intelligent transportation and robotics [4], Inertial Navigation System (INS) [5], cellular networks [6], and Internet of Things (IoT) devices [7]. This integration not only expands the capabilities and applications of GNSS but also fosters innovation within the technology itself, leading to new and improved functionalities [1, 2].

1.1 Motivation

The swift progress in GNSS technologies necessitates prototyping tools for the exploration of new signals and systems. Prototyping is crucial in research, serving as a bridge between abstract theories and practical applications, and offering a hands-on approach to experimentation and analysis. A key advantage of prototyping is its ability to speed up the research and development process, enabling researchers to quickly transition from concept to application. This contrasts with traditional research methods that often involve extensive theoretical study before any practical testing. Prototyping allows for faster testing, refining, and iteration of ideas, leading to a deeper, intuitive understanding of a product's potential and functionality. Furthermore, prototyping plays a vital role in risk mitigation by allowing the early identification and resolution of potential issues, thus avoiding the pursuit of unfeasible or ineffective ideas. It also enhances communication, as a physical model provides a tangible representation of a concept, making it easier to understand and facilitating the exchange of ideas, feedback, and collaborative problem-solving.

Prototyping with commercial GNSS receivers in research environments often presents challenges. These difficulties stem from the inherent limitations of commercial receivers, which are typically designed for general mass-market use rather than specialized research applications. Commercial GNSS receivers usually come with fixed, proprietary firmware and software, which restricts the ability to modify or customize their functionality to meet specific research needs. They often operate as black boxes, with their internal processing methods not disclosed to the researcher. This lack of transparency can be particularly problematic in advanced GNSS research, where specialized algorithms and processing techniques are often required, and a deep understanding of the receiver's signal processing chain is crucial for tasks such as error analysis or the development of new positioning methods. Research activities involving GNSS technology, usually require non-standard features from the receivers and detailed knowledge of the signal processing tasks that are invovled in the computation of the GNSS products. If the commercial receiver supplies the required measurements, researchers can employ statistical characterization and fitting models in their analysis. Nevertheless, researchers encounter the obstacle of constructing their own receiver in the absence of available measurements. This can be problematic, as receiver prototyping typically falls outside their area of expertise or field of focus [8].

Software-Defined Radio (SDR) techniques offer a solution to the constrained adaptability of commercial receivers. With its programmable and adaptable nature, SDR technology can be customized precisely to fulfill the user's specific needs. Moreover, access to inspection and modification of the full SDR receiver's signal processing chain can be addressed using Free and Open Source Software (FOSS). FOSS allows users and programmers to edit, inspect, and modify the software's source code. The open philosophy of FOSS is well suited for education and research, since the source code of the software is available for examination and derivative software can be written without any copyright issues [8]. With FOSS, researchers can freely share and reuse their work, optimizing the development process of applications and increasing productivity in software teams. This collaborative approach is not only beneficial for advancing the technology but also provides an educational resource for students and new researchers, helping them to understand practical applications and gain hands-on experience. Code reuse allows developers to focus on the implementation of new features instead of reinventing the wheel for each project. Moreover, it encourages clean, modular, and efficient software, which decreases the likelihood of introducing bugs or inconsistencies. The flexibility, adaptability, and collaborative potential of FOSS GNSS software receivers, combined with their cost-effectiveness and educational value, make them particularly suited for research environments where deep technological understanding, innovation, and customization are key.

However, GNSS software-defined receivers have a notable drawback in terms of power consumption. They typically do not match the power efficiency of hardware-based receivers, mainly because they run on general-purpose processors found in computers. These processors are not optimized for low-power operation, leading to excessive power consumption, particularly when executing complex algorithms. The intensive computational demands of real-time GNSS signal processing result in significant power consumption, making these systems less suitable for scenarios that require prolonged battery operation. This aspect is relevant for field-based GNSS applications, where power efficiency is needed. Such applications involve the practical use of GNSS technology in outdoor or real-world environments, rather than in controlled settings like laboratories.

It is thus interesting to investigate SDR architectures that can improve the trade-off between efficiency and flexibility. The semiconductor industry historically relied on Central Processing Unit (CPU) scaling to increase performance and reduce power consumption. This practice was largely governed by Moore's Law and Dennard Scaling, which together provided a predictable roadmap for performance improvements in semiconductor devices. Moore's Law predicted that the number of transistors on a chip would double approximately every two years, leading to increased performance, reduced costs, and greater energy efficiency [9]. Dennard Scaling complemented Moore's Law by outlining how, as transistors were made smaller, they would consume less power, produce less heat, and operate faster, allowing for more transistors to be packed into the same chip size without increasing overall power consumption [10].

Through these principles, the semiconductor industry was able to consistently deliver significant performance improvements from generation to generation of integrated circuits. However, as transistor sizes approached the atomic scale, the industry began encountering physical and technical limitations that made continued scaling as per Moore's Law and Dennard Scaling more challenging [11]. Issues such as heat dissipation, quantum tunneling, and variability increased, diminishing the returns on traditional scaling efforts.

As a result, the industry has shifted towards new strategies for performance improvement. This shift includes adopting increasingly parallel architectures and creating hardware that better matches the applications, exploring heterogeneous computing and advanced chip architectures previously limited to extreme performance segments [12, 13]. In this sense, there is a growing focus on specialized processors designed for specific tasks, such as vector-based processing with Digital Signal Processors (DSPs) and Graphics Processing Units (GPUs), as well as fully parallel programmable hardware like Field Programmable Gate Arrays (FPGAs).

An FPGA is a type of digital Integrated Circuit (IC) that can be reprogrammed by the user after manufacturing to perform a wide variety of digital functions. FPGAs are made up of an array of programmable logic blocks, interconnects, and Input/Output (IO) blocks that can be configured to execute complex combinational functions, implement state machines, or even mimic the architectures of entire CPUs. FPGAs offer precise customization for distinct computational tasks, rendering them ideal for latency-critical real-time applications. They outperform processors in terms of computing power and energy consumption for tasks that are highly parallelizable and require low latency and high throughput. FPGAs achieve this due to the nature of their architecture and design. Their reconfigurable matrix of logic blocks enables the execution of a diverse array of digital functions, including SDR algorithms. Designed for parallel processing, FPGAs can concurrently manage multiple tasks, enhancing their overall efficiency. This parallelism reduces the overall processing time and decreases the power consumption compared to sequential processing by CPUs. On top of that, FPGAs do not carry the overhead associated with managing complex instruction sets and control logic, which is typical in CPUs. As a result, they can perform specific tasks efficiently with minimal unnecessary operations, leading to lower power consumption.

GNSS receivers exploit parallelism to efficiently process signals from multiple satellites at once, thereby enhancing the speed and precision of Position, Velocity and Time (PVT) solutions. By utilizing multiple channels that operate in parallel, these receivers can simultaneously process signals from various satellites. This approach to parallel processing is essential for achieving accurate positioning, as data from at least four satellites are needed for a precise three-dimensional location. The integration of FPGA technology further enhances GNSS signal processing efficiency. FPGAs, with their intrinsic parallel architecture and adaptability, are suitable for GNSS applications, offering an effective method for signal management. The adaptability of FPGAs is also a key feature. As GNSS technology advances or as processing needs change, the hardware can be adjusted to meet these new demands. This flexibility ensures the system remains relevant and efficient, offering both cost-effectiveness and energy efficiency.

Integrating FPGAs with embedded general-purpose processors enhances the flexibility and power efficiency of software-defined GNSS receivers. This configuration allows for the optimal allocation of signal processing tasks to the most appropriate processing units, ensuring the system's adaptability to evolving requirements. System on Chip (SoC) FPGAs and Radio Frequency (RF) SoC FPGAs are particularly suited for developing architectures that merge software-defined systems with specialized hardware. At the heart of an SoC FPGA is the integration of a general-purpose processor with a reconfigurable FPGA fabric. Typically, SoC FPGAs include high-speed interconnects, such as buses or dedicated interfaces, to ensure efficient communication between the FPGAs and the embedded processor. These interconnects facilitate rapid data transfer, enhancing the interaction between hardware and software components. The integration of both an embedded processor and an FPGA into a single chip reduces power consumption, as internal chip connections consume less power compared to the external links of separate processor and FPGA systems. Moreover, RF SoC FPGAs further integrate RF Analog to Digital Converters (ADCs) and Digital-to-Analog Converters (DACs), further improving flexibility and reducing board space and power consumption [14, 15].

Still, adopting FPGAs presents challenges. Their performance can lag behind general-purpose processors for tasks that are inherently sequential or involve complex branching and decision-making processes. Furthermore, FPGA development is generally more time-consuming than traditional software development, encompassing stages like simulation, synthesis, and place-and-route, and debugging. This process not only extends development timelines but also demands a high level of specialized knowledge and expertise in hardware design, making FPGA programming less accessible to those without a background in this area.

Despite these challenges, strategically partitioning software between the FPGA and CPU can leverage their combined strengths, enhancing both power efficiency and performance. This approach involves dividing computational tasks, with certain tasks designated for the CPU and others offloaded to the FPGA, to achieve an optimal power-performance tradeoff in SoC FPGAs. Meticulous design and analysis are essential for this process. Typically, tasks suited for FPGAs require high throughput, parallel processing, and real-time execution. In contrast, the CPU is better equipped for handling complex control logic, decision branching, and sequential processing. Such optimal hardware-software partitioning not only improves the power consumption-performance balance but also enables efficient GNSS signal processing. The software-defined portion, managed by the processor, provides fast adaptability, whereas the FPGA offers hardware-accelerated functions. This dual capability is essential for GNSS receiver functionalities that require both adaptability and precise processing.

1.2 Objectives

The main objective of this thesis is to design and implement a scalable SoC-FPGA architecture for prototyping experimental, portable multi-constellation and multi-frequency GNSS receivers using Commercial Off-The-Shelf (COTS) products. The proposed architecture overcomes the flexibility limitations of commercial receivers by enabling the implementation of GNSS processing engines on SoC-FPGA platforms. It capitalizes on the embedded processors' flexibility and the FPGAs' massive parallelism and energy efficiency, facilitating the creation of real-time, portable receivers. These receivers can be adapted to a wide range of SoC-FPGA devices with varying FPGA and embedded processor sizes, with a focus on optimizing for low power consumption, price-performance ratio, or catering to high-capacity and high-performance needs based on the application.

In the proposed architecture, the SoC-FPGA embedded processor runs GNSS-SDR, a popular and well-regarded FOSS GNSS SDR receiver that is readily available online [16, 17]. The adoption of open-source software enables thorough inspection and modification of the receiver's chain, fostering freedom, collaboration, and innovation, while also reducing implementation costs. Conventionally, GNSS-SDR operates as a host-based GNSS receiver: an Radio Frequency Front-End (RFFE) tuned to GNSS frequency bands receives satellite signals, performs RF-to-baseband down-conversion and sampling, and then streams the samples to a Personal Computer (PC) running GNSS-SDR. This thesis introduces a significant enhancement by adding the capability to cross-compile GNSS-SDR for its execution on SoC-FPGA embedded devices. With this enhancement, GNSS-SDR can offload the most computationally intensive tasks to the FPGA, thus realizing an SoC-FPGA-based embedded receiver. In this configuration, the FPGA processes the incoming sample stream from the RFFE, utilizing hardware accelerators for the most demanding algorithms, notably acquisition and tracking. These FPGA hardware accelerators are designed as reusable Intellectual Property (IP) cores, making them adaptable to a wide range of FPGAs.

This thesis also proposes a design methodology for implementing GNSS signal processing algorithms and concept demonstrators based on the SoC FPGA architecture. This methodology is based on a hardware/software design flow where FPGA and software development can largely proceed independently. In this way, software can be implemented using a complete FOSS toolchain, independently of the FPGA development tools. Researchers have the option to release FPGA IP cores under either FOSS or proprietary licenses. This flexibility allows for the monetization of research outcomes while simultaneously enhancing research impact and reputation. The FPGA IP cores developed within the framework of this thesis are available under proprietary terms, which restrict access to their source code. However, access to the internal aspects of the signal processing algorithms is possible by inspecting the software version of the FPGA hardware accelerators in GNSS-SDR, along with the documentation provided by the developers of the FPGA IP cores. This approach facilitates code reusability, which is essential for minimizing development time and costs.

Finally, this thesis validates the practicality of the proposed architecture and design methodology by implementing various real-time concept demonstrators. These include a spaceborne receiver for processing GNSS signals in LEO scenarios, a GNSS rebroadcaster for regenerating live signals, and a High Sensitivity (HS)-GNSS receiver designed for weak signal conditions. The prototypes are evaluated based on hardware resources, size, and power consumption, highlighting the scalability of the proposed architecture. They are tested with live signals to demonstrate their capability to fulfill the requirements.

The software-defined spaceborne GNSS Receiver offers a fully customizable platform capable of processing GPS L1 C/A, GPS L5, Galileo E1b/c, and Galileo E5a signals. It can produce navigation solutions and deliver GNSS products in standard formats in real time, even in high-dynamic scenarios such as Low Earth Orbit (LEO). Designed for space applications, this receiver features a form factor suitable for CubeSats.

The rebroadcaster integrates a closely coupled receiver and transmitter within a single device, capable of generating, receiving, and regenerating GPS L1 C/A and Galileo E1b/c signals with very low latency. It can regenerate the received satellite signals in real time while simultaneously rebroadcasting a PVT solution that differs from the PVT fixes obtained by the receiver. This device can be used to test the addition of new features in GNSS signals, and to simulate the channel dynamics in various scenarios. The GNSS rebroadcaster features a specialized component within the FPGA, known as the telemetry symbol link. This link directly forwards telemetry data, estimated by the receiver's tracking multicorrelator hardware accelerators, to the signal generators in the FPGA, bypassing any need for processor intervention. The telemetry symbol link can be enabled for Galileo E1b/c signals. When enabled, the telemetry data is rebroadcasted with a latency that can be set below 30 ms. This facilitates maintaining the consistency with other sensors, e.g., an Inertial Measurement Unit (IMU), where available.

The HS GNSS receiver is designed for processing weak signals. It implements two operating modes: high-sensitivity mode, and normal-sensitivity mode. When operating in high-sensitivity mode, the receiver is capable of acquiring and tracking Galileo E1b/c signals with a Carrier-to-Noise Density Ratio (C/N_0) as low as 20 dB-Hz (equivalent C/N_0 observed at the post-correlation level), enabling the derivation of navigation solutions. On the other side, when operating in normal-sensitivity mode, the receiver processes GPS L1 C/A, Galileo E1b/c, GPS L5 and Galileo E5a signals with an acquisition sensitivity of approximately 37 dB-Hz. To enhance the availability of satellite signals, the receiver is capable of processing Galileo E1b/c signals in high-sensitivity mode, while simultaneously processing GPS L1 C/A, GPS L5 and Galileo E5a signals in normal-sensitivity mode. This receiver is engineered to process severely attenuated GNSS signals and has the potential to test novel algorithms that address common indoor GNSS challenges, such as multipath interference [18]. However, it currently does not incorporate these algorithms.

To sum up, the objectives of the research carried out in this dissertation are the following:

- To develop a state-of-the-art architecture that is both flexible and scalable, leveraging SoC FPGA technology for the prototyping of multi-frequency and multi-system GNSS receivers. This architecture seeks to improve the power efficiency typically seen in software-defined receivers while maintaining their flexibility and offering an alternative to the limited adaptability found in commercial devices, streamlining the exploration and advancement of cutting-edge GNSS algorithms.
- To propose a design methodology that integrates advanced design tools, aimed at streamlining the development of experimental GNSS receivers by leveraging the proposed architecture. This methodology encompasses the phases of research, conceptualization, iteration, prototyping, and implementation, with a particular focus on incorporating unconventional features for research purposes.
- To demonstrate the practical applicability of the proposed architecture by developing three innovative and groundbreaking GNSS receiver prototypes incorporating non-standard features aimed at conducting research on current GNSS topics, including a spaceborne GNSS receiver, a GNSS rebroadcaster, and a HS-GNSS receiver for moderate indoor positioning.

The proposed research aims to develop a framework that streamlines the design and development of portable and compact experimental GNSS receivers. This framework will specifically facilitate the implementation and field testing of innovative algorithms for research purposes.

1.3 Outline of the dissertation

This section provides a summary of the content of each chapter included in this thesis.

Chapter 2 establishes the essential knowledge and context necessary for the research presented in this thesis. It offers a background and analysis of GNSS systems and receivers. This is further complemented with an introduction to the GNSS-SDR software receiver. The chapter then delves into the fundamental concepts of FPGA and SoC FPGA technology, exploring key components such as configurable logic blocks, interconnects, programmable routing resources, and design flows. This is followed by a discussion on design forces and Key Performance Indicators (KPIs) relevant to GNSS technology. These KPIs facilitate the discourse on the proposed architecture and design methodology, playing a crucial role in the evaluation of the proposed concept demonstrators in subsequent chapters. Finally, the chapter concludes with a review of existing GNSS receiver implementations, placing special emphasis on their underlying processing units. This highlights the evolution and diversity of architectures employed in GNSS receivers, showcasing advancements and variations.

Chapter 3 delivers an in-depth analysis of the proposed SoC-FPGA receiver architecture and its design methodology. The chapter begins with a description of the proposed architecture, explaining how the FPGA and the embedded processor work in tandem to achieve optimized performance and efficiency. Then, it focuses on the FPGA design, emphasizing the design and execution of the acquisition and tracking multicorrelator hardware accelerators. Subsequently, this chapter delves into the GNSS-SDR software architecture, with a focus on how the software is modified for execution in the SoC FPGA platform. Upon exploring the software aspect, this chapter introduces a design methodology for creating innovative GNSS demonstrators that utilize the proposed architecture, with a focus on code reuse. Separate development flows are adopted for software and hardware, enabling the use of FOSS development tools and a FOSS compiler toolchain for creating software signal processing blocks. This approach ensures their independence from FPGA design tools. The development of new algorithms is structured into three phases: initial software validation, FPGA implementation for computationally demanding algorithms, and the subsequent integration of software and FPGA components on a compact, portable, SoC-FPGA platform. Finally, this chapter discusses the proposed receiver architecture and design methodology, addressing several design forces — scalability, testability, portability, maintainability, reproducibility, and openness — and KPIs, all of which were introduced in Chapter 2. These aspects are crucial for understanding how the architecture aligns with the KPIs and meets overall design objectives.

Chapter 4 reports on the design, proof-of-concept implementation, and preliminary performance evaluation of a low-cost, software-defined, multi-band, multi-system spaceborne GNSS receiver. This chapter outlines the specific objectives pursued through the implementation of the spaceborne receiver and provides a detailed description of the system's design, highlighting its key components and the rationale behind its architecture. Following this, the chapter presents test results evaluating the receiver's performance against various KPIs. These include accuracy, availability, efficiency, flexibility, interoperability, reliability, and precision. Finally, the chapter concludes with a comprehensive summary and discussion of the findings. It offers insights into the implications for future GNSS receiver technology and outlines potential research directions.

Chapter 5 details the design, proof-of-concept implementation, and preliminary performance assessment of the GNSS rebroadcaster. This chapter describes the rebroadcaster, with an emphasis on its detailed design, particularly focusing on signal regeneration and the telemetry symbol link. It then presents the test results of the rebroadcaster, evaluating the latency and the quality of the rebroadcasted signals. The chapter concludes with a discussion of the findings.

Chapter 6 presents the architecture, prototype development, and initial performance analysis of the HS GNSS receiver. This chapter delves into the receiver's features and operating modes, offering a comprehensive examination of its implementation. Special emphasis is placed on innovative features tailored to process severely attenuated signals, including significant enhancements to the software and FPGA components of the architecture. The chapter proceeds by presenting test results that assess the receiver's ability to process weak signals, focusing on several KPIs, including availability, flexibility, interoperability, reliability, and the precision of navigation solutions derived. It culminates in a detailed discussion of these results, highlighting their implications for receiver performance.

Finally, **Chapter 7** presents the conclusions and directions for future work.

1.4 Research contributions

The research contributions of this Dissertation are pointed out in the summary available at the end of each chapter. The full list of publications is provided hereafter.

1.4.1 Journal papers

- [19] M. Majoral, J. Arribas, and C. Fernández-Prades, "Implementation of a High-Sensitivity Global Navigation Satellite System Receiver on a System-on-Chip Field-Programmable Gate Array Platform," *Sensors*, vol. 24, no. 5, 2024, Art. no. 1416. doi: 10.3390/s24051416
- [20] M. Majoral, C. Fernández-Prades, and J. Arribas, "A Flexible System-on-Chip Field-Programmable Gate Array Architecture for Prototyping Experimental Global Navigation Satellite System Receivers," *Sensors*, vol. 23, no. 23, 2023, Art. no. 9483. doi: 10.3390/s23239483

1.4.2 Peer-Reviewed International Conferences

- [21] M. Majoral, J. Arribas, and C. Fernández-Prades, "Implementation of a GNSS Re-broadcaster in an All-Programmable System-On-Chip Platform," in *2022 10th Workshop on Satellite Navigation Technology (NAVITEC)*, Noordwijk, Netherlands, April 2022, pp. 1–9. doi: 10.1109/NAVITEC53682.2022.9847537
- [22] C. Fernández-Prades, J. Arribas, M. Majoral, A. Ramos, J. Vilá-Valls, and P. Giordano, "A Software-Defined Spaceborne GNSS Receiver," in *2018 9th ESA Workshop on Satellite Navigation Technologies and European Workshop on GNSS Signals and Signal Processing (NAVITEC)*, Noordwijk, Netherlands, December 2018, pp. 1–9. doi: 10.1109/NAVITEC.2018.8642697

1.4.3 International Conferences (Non-Peer Reviewed)

- [23] M. Majoral, C. Fernández-Prades, and J. Arribas, "Implementation of GNSS Receiver Hardware Accelerators in All-programmable System-On-Chip Platforms," in

Proceedings of the 31st International Technical Meeting of the Satellite Division of The Institute of Navigation (ION GNSS+ 2018), Miami, FL, USA, September 2018, pp. 4215–4230. doi: 10.33012/2018.16082

1.4.4 Open-Source GNSS Receiver Contributions

- Developer of the GNSS-SDR software-defined receiver [24] between 2018 and 2024. Designed and implemented an FPGA offloading strategy for executing GNSS-SDR in embedded SoC FPGAs. Additionally, enhanced the acquisition, tracking, and telemetry decoding blocks to process weak signals in SoC FPGAs.

Chapter 2

Background and State of the Art

This chapter provides a comprehensive context for the research conducted in this thesis, delving into the fundamentals of GNSS technology, the intricacies of FPGA and SoC FPGA design, and reviewing existing implementations of software-defined GNSS receivers. Additionally, it introduces and details KPIs critical to GNSS systems. These KPIs will be instrumental in subsequent chapters, guiding our discussion on the proposed architecture and the assessment of the receiver prototypes developed in this study.

The chapter begins with an exploration of GNSS positioning principles, detailing the signal structure and frequency bands of GNSS satellites, with particular focus on the GPS and Galileo systems. This focus is due to their relevance to our concept demonstrators, specifically designed for processing signals like GPS L1 C/A, GPS L5, Galileo E1b/c, and Galileo E5a.

Subsequently, the architecture of a typical GNSS receiver is outlined, highlighting its core components. Following this, the software-defined GNSS-SDR receiver is introduced, which plays a key role in our proposed architecture. This introduction underscores the flexibility and programmability that GNSS-SDR offers, setting it apart from traditional GNSS receivers.

The discussion then transitions to FPGA and SoC FPGA technologies, elucidating their architecture, design methodologies, and the challenges associated with using FOSS FPGA design tools. It delves into FPGA IP cores, which are reusable units of logic, commonly integrated into FPGAs.

After that, this chapter provides a detailed examination of the critical elements that influence the architecture and performance of GNSS technologies. The discussion is twofold, initially focusing on the concept of *design forces*. These encompass a range of factors, including technological constraints and user requirements, which play a significant role in shaping the development and implementation of GNSS technologies. Secondly, the chapter explores KPIs pertinent to GNSS. KPIs serve as quantifiable measures that offer a way to assess the efficiency, accuracy, reliability, and overall performance of GNSS technologies.

In conclusion, this chapter includes a review of state-of-the-art software-defined GNSS receiver implementations, with an emphasis on the underlying processing units.

2.1 Basics of GNSS Positioning

GNSS positioning is based on trilateration, a geometric method that determines the position of an object by measuring its distance from multiple known locations. This process involves calculating the distances to satellites orbiting the Earth. The signals transmitted by GNSS satellites carry very precise and accurate time stamps to synchronize the receiver on Earth and determine the exact location of the user. This accurate timekeeping is achieved by the atomic clocks on board the GNSS satellites, ensuring synchronized timekeeping across the system. Each GNSS System adheres to its own time reference, which includes GPS System Time (GPST) for GPS, Galileo System Time (GST) for Galileo, Glonass Time (GLONASST) for Glonass, and Beidou Time (BDT) for Beidou. These time references are essential for maintaining the accuracy of the GNSS positioning.

GNSS receivers use Time of Arrival (ToA) ranging for position determination: they estimate the time when a satellite signal is received and compute the Time of Flight (ToF) (the propagation time from the satellite to the receiver) by subtracting the ToA from the Time of Transmission (ToT):

$$\mathrm{ToF} = \mathrm{ToT} - \mathrm{ToA}\ . \tag{2.1}$$

This time, multiplied by the speed of light, gives the distance to the satellite. Navigational messages embedded in the satellite signals provide data on the satellites' orbits, allowing the receiver to calculate the satellite coordinates.

The ToT and the ToA are determined using two non-synchronized clocks: the satellite clock and the receiver clock respectively. The receiver clock will generally have a bias error from system time. Because of this error, the range measurements are called pseudoranges. The pseudorange measurements contain errors including the time difference between the GNSS system time and the user clock, the difference between the system time and the satellite clock. Considering that T_s is the system time when the satellite transmitted the signal, T_u is the system time when the signal reached the receiver,δt is the offset of the satellite clock from system time, t_u is the offset of the receiver clock from system time, $T_s + \delta t$ is the satellite clock reading when the satellite transmitted the signal, $T_u + t_u$ is the user receiver clock reading the signal reached the receiver, and c is the speed of light, then the geometric range r is determined as

$$r = c(T_u - T_s) = c\Delta t\ , \tag{2.2}$$

and the pseudorange measurement ρ is given by

$$\rho = c[(T_u + t_u) - (T_s + \delta t)] = c(T_u - T_s) + c(t_u - \delta t) = r + c(t_u - \delta t) \ . \tag{2.3}$$

The GNSS ground monitoring network determines corrections for the satellite clock offset δt and the GNSS satellites broadcasts this clock correction data such that the receivers can apply this data. For this reason, the satellite clock offset can be considered very small compared to the user clock offset.

GNSS mainly operates using the Earth-Centered, Earth-Fixed (ECEF) coordinate system. ECEF is a coordinate system which represents positions as X, Y, and Z coordinates. In this system, the origin $(0, 0, 0)$ is at the center of the Earth, the X-axis extends from the Earth's center to the intersection of the equator and the prime meridian, the Y-axis is perpendicular to the X-axis and lies in the plane of the equator, and the Z-axis coincides with the mean Earth rotational axis, pointing towards the North Pole. The ECEF system is advantageous for terrestrial navigation because it remains fixed relative to the Earth's surface, making it directly applicable for most location-based applications.

Figure 2.1 illustrates the positions of the receiver and a GNSS satellite relative to Earth's center of mass. To determine the user position, the receiver needs to compute vector **u**. Vector **s** is the known satellite position, which has coordinates x_s, y_s, z_s in ECEF coordinates.

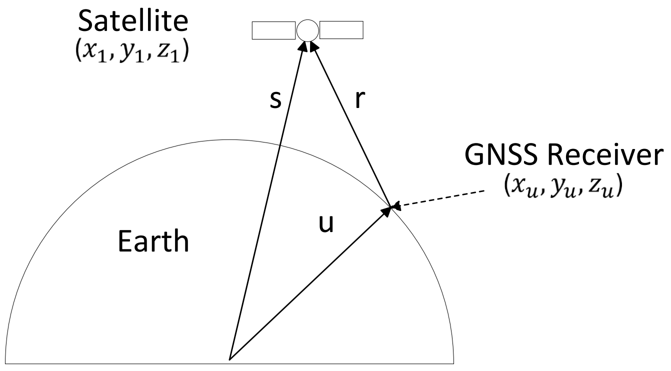

Figure 2.1 Vector representation of the receiver position [25].

Vector **r** is the satellite position with respect to the user. Considering (2.3), and not considering the satellite clock offset with respect to system time, the satellite to user vector is determined by

$$r = s - u = \rho - ct_u \ , \tag{2.4}$$

where t_u is the receiver clock offset [25, 26]. The user's position can be determined by performing pseudorange measurements with at least four visible satellites. This process results in a system of four equations with four unknowns—namely, the user's three-dimensional coordinates (x_u, y_u, z_u) and the receiver's clock offset (t_u), as detailed in:

$$
\begin{aligned}
\rho_1 &= \sqrt{(x_1 - x_u)^2 + (y_1 - y_u)^2 + (z_1 - z_u)^2} + ct_u \\
\rho_2 &= \sqrt{(x_2 - x_u)^2 + (y_2 - y_u)^2 + (z_2 - z_u)^2} + ct_u \\
\rho_3 &= \sqrt{(x_3 - x_u)^2 + (y_3 - y_u)^2 + (z_3 - z_u)^2} + ct_u \\
\rho_4 &= \sqrt{(x_4 - x_u)^2 + (y_4 - y_u)^2 + (z_4 - z_u)^2} + ct_u \ .
\end{aligned}
\tag{2.5}
$$

In these equations, ρ_1, ρ_2, ρ_3, ρ_4 represent the pseudoranges measured by the receiver with respect to satellites 1, 2, 3, and 4 respectively, while (x_1, y_1, z_1), (x_2, y_2, z_2), (x_3, y_3, z_3), and (x_4, y_4, z_4) are the coordinates of these satellites [25].

Each pseudorange corresponds to a sphere, with the satellite at its center and the pseudorange as the radius. The receiver's position is located at the intersection of the satellite-ranging spheres.

In practice, GNSS receivers compute navigation solutions using more than four satellites. These solutions are initially refined through iterative techniques based on linearization and further enhanced by applying Kalman filtering and least-squares estimation algorithms. [25, 26].

2.2 GNSS Satellite Signals

GNSS satellites transmit information using radio signals. The transmission of signals from both Galileo and GPS satellites employs Direct Sequence Spread Spectrum (DSSS) modulation [25]. DSSS is a method that extends a signal over a bandwidth wider than what is essential for transmitting information. This expansion is achieved by modulating the message symbols with a spreading sequence, also referred to as a Pseudo Random Noise (PRN) sequence. Each element of this sequence is known as a *chip*, which has a shorter duration than the original message symbols. Through this modulation process, the message symbols are scrambled and dispersed across the spectrum. Consequently, the bandwidth of the overall signal matches that of the spreading sequence, effectively broadening its spectral footprint and reducing its power spectral density. This spreading process enhances the system's resistance to interference and multipath fading, key advantages in satellite communication.

2.2.1 Frequency Bands

The satellites from both the Galileo and GPS systems broadcast their signals within the L frequency band, using Right-Handed Circular Polarization (RHCP), and utilizing the range of frequencies shown in Figure 2.2 [27–29]. GNSS signals are categorized under Radio Navigation Satellite Service (RNSS), because they fundamentally provide radio-based navigation and positioning services using satellites. Furthermore, some GNSS signals are included in Aeronautical Radio Navigation Service (ARNS), due to their critical role in aviation navigation and safety. Using multiple frequencies offers several advantages over their single-frequency counterparts. One of the key benefits is improved accuracy. By accessing more than one frequency, these receivers can more effectively correct errors caused by the ionosphere, a major source of GPS signal delays and inaccuracies. The ionosphere affects signals at different frequencies in different ways, so by comparing the delay across multiple frequencies, the receiver can accurately calculate and compensate for these errors [25]. Another advantage of multi-frequency receivers is better resistance to interference and jamming. Since receivers can access multiple frequencies, they are less likely to lose all satellite connections due to interference on a single frequency.

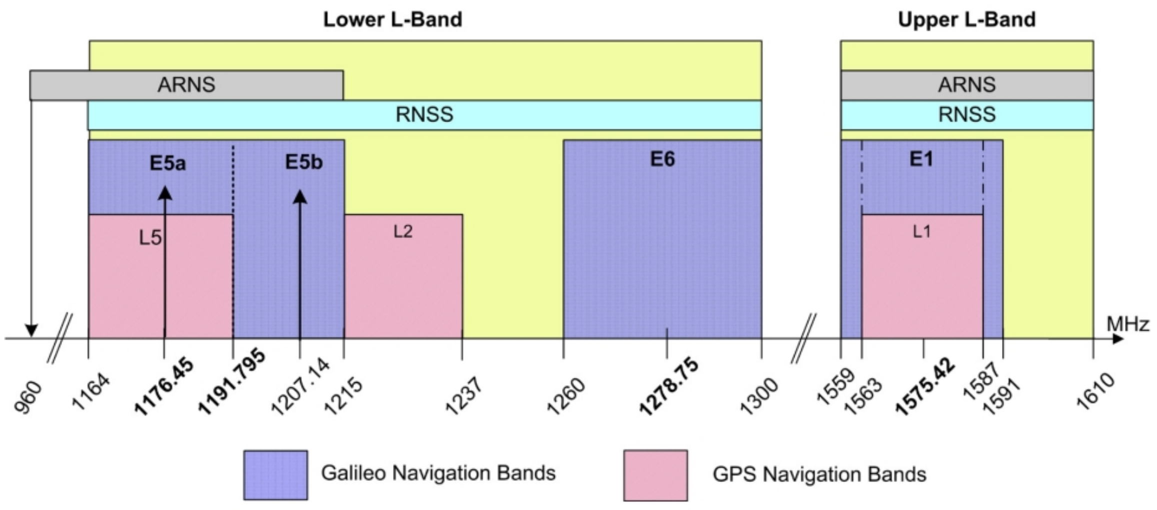

Figure 2.2 GPS and Galileo frequency bands (source: [29]) .

The carrier frequencies, bandwidths, and the minimum received signal power on the Earth's surface for the GPS L1 C/A, GPS L5, Galileo E1b/c, and Galileo E5a signals are detailed in Table 2.1 [30].

Table 2.1 GNSS signal characteristics.

Signal	GPS L1 C/A	Galileo E1b/c	GPS L5	Galileo E5a
Carrier Frequency (MHz)	1575.42	1575.42	1176.45	1176.45
Frequency Band	L1/E1	L1/E1	L5/E5a	L5/E5a
Bandwidth (MHz)	2.046	20.46	14.322	20.46
Minimum Received Power (dBW)	−158.5	−157.9	−157.25	−155.25

2.2.2 Spreading Codes

Each GNSS system and each specific signal within these systems uses unique spreading codes that are orthogonal to one another. This orthogonality ensures that the codes do not interfere with each other, allowing GNSS receivers to accurately distinguish and lock onto signals from different satellites. Some GNSS signals use long spreading codes generated by a tiered code construction, whereby a secondary code sequence is used to modify successive repetitions of a primary code. Tiered codes enhance GNSS performance through improved receiver sensitivity, superior cross-correlation properties compared to other GNSS signals, increased resistance to narrow-band jamming, and faster bit synchronization. [31].

In tiered codes, the secondary code modulates the primary PRN code by altering its polarity: each bit of the secondary code determines whether a corresponding period of the PRN code remains unchanged or is inverted. The tiered code generation process is illustrated in Figure 2.3, where the primary code sequence has a length of N chips, and the secondary code sequence has a length of N_s chips. This results in a tiered code with a total length of $N \cdot N_s$ chips.

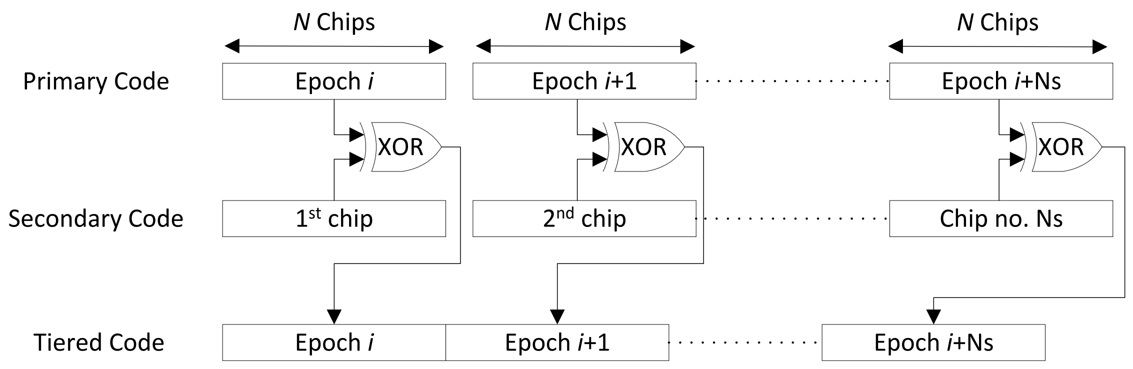

Figure 2.3 Tiered codes generation.

GNSS satellites employ unique PRN codes for the data and pilot components of the transmitted signals. These codes are orthogonal, ensuring they are unique to each satellite, and they serve to identify both the satellite itself and the separate data and pilot components of the signal.

2.2.3 Signal Structure

The mathematical model of GNSS signals encompasses carrier waves, code modulation, and navigation data. A pilot signal is transmitted within the Galileo signals, including Galileo E1b/c and E5a, as well as in modernized GPS signals, such as GPS L5. This enhances the signal acquisition and tracking capabilities [27–29]. In this section, we summarize the structure of satellite navigation signals, including GPS L1 C/A, Galileo E1b/c, GPS L5, and Galileo E5a, highlighting various aspects such as modulation type, sub-carrier usage, and frequencies where applicable. This summary also covers the composition of the pilot and data components, detailing the length of the primary PRN code, its chipping rate, and the lengths of secondary PRN codes for both data and pilot signals.

2.2.3.1 GPS L1 C/A

The legacy civil signal, GPS L1 C/A signals, use Binary Phase Shift Keying (BPSK) modulation. BPSK is a phase modulation technique where each symbol to be transmitted represents one of two possible phase shifts of a carrier wave, typically 0 degrees (for the binary "0") or 180 degrees (for the binary "1"). The GPS L1 C/A, transmitted by the k-th satellite, can be modeled as

$$s_k(t) = \sqrt{2P} c_k(t) d_k(t) \cos(2\pi f_c t) \ . \tag{2.6}$$

In this equation, $s_k(t)$ denotes the signal transmitted, P signifies the transmitted signal power, $c_k(t)$ and $d_k(t)$ represent the PRN code and the data symbols transmitted by the k-th satellite, respectively, and f_c is the carrier frequency for GPS L1 C/A.

Table 2.2 summarizes the most relevant characteristics of the GPS L1 C/A signals [27].

Table 2.2 GPS L1 C/A signal characteristics.

GPS L1 C/A	
Modulation	BPSK
PRN code chipping rate	1.023 Mchips/s
PRN code length	1023 chips
Data Rate	50 bps

2.2.3.2 GPS L5 and Galileo E5a

The GPS L5 and Galileo E5a signals, which comprise both data and pilot components, are transmitted both in-phase and in quadrature. The main properties of these signals are captured by the following equation:

$$s_k(t) = \sqrt{2P}[c_{i,k}(t) d_k(t) \cos(2\pi f_c t) + c_{q,k}(t) \sin(2\pi f_c t)] \ . \tag{2.7}$$

Here, $s_k(t)$ denotes the signal from the k-th satellite, with P indicating the transmitted signal power for both phase and quadrature components. $c_{i,k}(t)$ and $c_{q,k}(t)$ represent the PRN codes for the phase and quadrature components, respectively, transmitted by the k-th satellite. The data transmitted by the same satellite is $d_k(t)$, and f_c is the carrier frequency.

The data and pilot PRN codes are composed of combinations of primary and secondary codes, as detailed in

$$\begin{aligned} c_{i,k}(t) &= c_{i_p,k}(t) c_{i_s}(t) \\ c_{q,k}(t) &= c_{q_p,k}(t) c_{q_s}(t) \ . \end{aligned} \tag{2.8}$$

Specifically, $c_{i_p,k}(t)$ and $c_{q_p,k}(t)$ represent the primary PRN codes for the in-phase and quadrature components, respectively, of the k-th satellite's signal. Similarly, $c_{i_s}(t)$ and $c_{q_s}(t)$ represent the secondary PRN codes.

The GPS L5 signal uses BPSK modulation, whereas the Galileo E5a signal uses Alternative Binary Offset Carrier (AltBOC) modulation. An explanation of the AltBOC modulation can be found in [32].

Table 2.3 and Table 2.4 summarize the most relevant characteristics of the GPS L5 and Galileo E5a signals [28, 29].

Table 2.3 GPS L5 signal characteristics.

	GPS L5	
Signal component	Data	Pilot
Modulation	BPSK	BPSK
Primary PRN code chipping rate	10.23 Mchips/s	10.23 Mchips/s
Primary PRN code length	10230 chips	10230 chips
Secondary PRN code length	10 bits	20 bits
Data Rate	100 bps	-

Table 2.4 Galileo E5a signal characteristics.

	Galileo E5a	
Signal component	Data	Pilot
Modulation	AltBOC	AltBOC
Subcarrier frequency	15.345 MHz	15.345 MHz
Primary PRN code chipping rate	10.23 Mchips/s	10.23 Mchips/s
Primary PRN code length	10230 chips	10230 chips
Secondary PRN code length	20 bits	100 bits
Data Rate	50 bps	-

2.2.3.3 Galileo E1b/c

The Galileo E1b/c signals comprise both a data component and a pilot component, which are transmitted in-phase and anti-phase, respectively. The Galileo E1b/c signals use the Composite

Binary Offset Carrier (CBOC) modulation. An explanation of the CBOC modulation can be found in [33]. The composite signal of Galileo E1b/c can be written as

$$\begin{aligned} s_k(t) = \frac{1}{\sqrt{2}} = \big[& e_{E1-B_k}(t)(\alpha sc_{E1-B,a}(t) + \beta sc_{E1-B,b}(t)) \\ & - e_{E1-C_k}(t)(\alpha sc_{E1-C,a}(t) - \beta sc_{E1-C,b}(t)) \big] \ , \end{aligned} \tag{2.9}$$

where $s_k(t)$ denotes the signal transmitted by the k-th satellite. Here, $e_{E1-B_k}(t)$ represents the data component, modulated by subcarriers $sc_{E1-B,a}(t)$ and $sc_{E1-B,b}(t)$, and $e_{E1-C_k}(t)$ represents the pilot component, modulated by subcarriers $sc_{E1-C,a}(t)$ and $sc_{E1-C,b}(t)$.

The α and β parameters are set to $\alpha = \sqrt{\frac{10}{11}}$ and $\beta = \sqrt{\frac{1}{11}}$ respectively. The shape of the binary subcarriers $sc_x(t)$ may be formulated as

$$sc_x(t) = sgn(\sin(2\pi R_{s,x} t)) \ , \tag{2.10}$$

Where $sgn()$ is the sign function. The parameter $R_{s,x}$ shown in (2.10) is determined by Table 2.5.

Table 2.5 E1 Composite Binary Offset Carrier (CBOC) sub-carrier rates.

Component (Parameter Y)	**Sub-carrier Type**	**Sub-carrier Rate**	
		$\mathbf{R_{S,E1-Y,a}}$ **(MHz)**	$\mathbf{R_{S,E1-Y,b}}$ **(MHz)**
B	CBOC, in-phase	1.023	6.138
C	CBOC, anti-phase	1.023	6.138

Table 2.6 summarizes the most relevant characteristics of the Galileo E1b/c signals [29].

Table 2.6 Galileo E1 signal characteristics.

	Galileo E1b/c	
Signal component	Data	Pilot
Modulation	CBOC	CBOC
Subcarrier frequency	1.023 MHz and 6.138 MHz (2 subcarriers)	1.023 MHz and 6.138 MHz (2 subcarriers)
Primary PRN code chipping rate	1.023 Mchips/s	1.023 Mchips/s
Primary PRN code length	4092 chips	4092 chips
Secondary PRN code length	-	25 bits
Data Rate	250 bps	-

2.2.4 Doppler frequency

GNSS receivers experience Doppler shift due to the relative motion between the satellites, which are the source of the signals, and the receivers, which are observing these signals. The satellites in the GNSS systems are in a Medium Earth Orbit (MEO) and move at high speeds. This high velocity causes the frequency of the signals they emit to be altered by the time they reach the receiver. In addition, the movement of the GNSS receiver itself, whether in a vehicle, aircraft, or in a person's hand, further modifies the frequency of the incoming signal.

For a static receiver on the Earth's surface, the Doppler frequency shift arises from the component of the satellite's velocity directed towards the user. The maximum Doppler shift occurs when the satellite is at the horizon. Specifically, when receiving signals from GPS satellites, the maximum velocity component directed towards a static receiver is approximately 929 m/s, as detailed in [26]. Consequently, the maximum Doppler shift experienced by a GNSS receiver, $f_{d_{L1},\max}$, can be computed as

$$f_{d_{L1},\max} = \frac{f_{c_{L1}} v_d}{c} = \frac{1575.42 \cdot 10^6 \cdot 929}{3 \cdot 10^8} \approx 4.9 \text{ kHz} . \tag{2.11}$$

In this equation, $f_{d_{L1},\max}$ represents the maximum Doppler shift, $f_{c_{L1}}$ is the L1 carrier frequency, v_d is the maximum velocity component directed towards the user, and c denotes the speed of light [26].

The maximum rate of change of the Doppler frequency at the L1 carrier frequency, known as the Doppler rate, can be estimated to be $\delta f_d|_{max} \approx 1$ Hz/s [26].

Similarly, the Doppler frequency shift experienced by the receiver at the L5 carrier frequency can be calculated as

$$f_{d_{L5},\max} = \frac{f_{c_{L5}} v_d}{c} = \frac{1176.45 \cdot 10^6 \cdot 929}{3 \cdot 10^8} \approx 3.7 \text{ kHz} , \tag{2.12}$$

in accordance with the procedure described above. Here, $f_{d_{L5},\max}$ represents the maximum Doppler shift, and $f_{c_{L5}}$ denotes the L5 carrier frequency.

2.2.5 Navigation Data

GNSS satellites are continuously broadcasting data to enable GNSS receivers to determine their location and time. This data is transmitted in the form of coded signals and includes the following key components:

- Ephemeris Data: This is precise orbital and clock correction data for each satellite. The Ephemeris data allows a GNSS receiver to calculate the precise position of the satellite at any given time. This data is usually valid for only a few hours.
- Almanac Data: This includes information about the status of satellites in the GNSS constellation, their current and predicted orbits for several days, and general system

health. The Almanac data is not precise enough for calculating the receiver position, but it helps the receiver to know which satellites to listen for and aids in quicker signal acquisition.

- Time Information: Each satellite transmits its own precise time, generated by an onboard atomic clock. This timing information is crucial for calculating distances based on the time it takes for the signal to travel from the satellite to the receiver.
- Satellite Health and Status Information: This includes information about the operational status of the satellite (whether it's functioning properly or not).
- Ionospheric Data: Some signals include information about the ionosphere, which can affect the speed of the GNSS signals as they pass through the Earth's atmosphere, and thus impact positioning accuracy.

GNSS receivers process this data to determine their exact position (latitude, longitude, and altitude), speed, and time. The system is designed so that this information is constantly updated and globally available.

2.3 Fundamentals of GNSS Receivers

GNSS receivers use signals from satellites to determine their geographic location. The basic architecture of a GNSS receiver typically includes an antenna, a RFFE, and several signal processing blocks, as shown in Figure 2.4 [34].

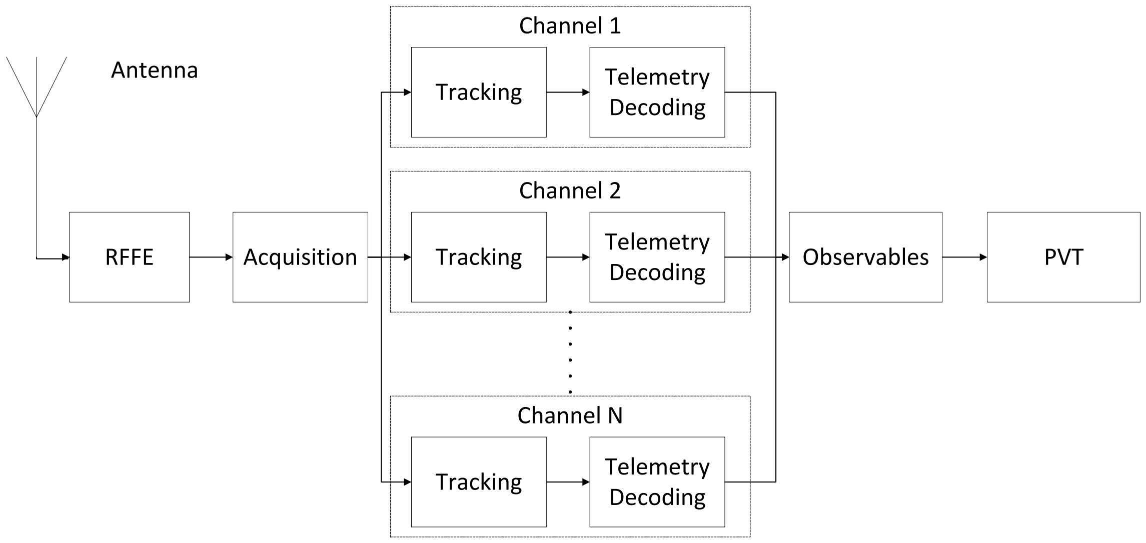

Figure 2.4 GNSS receiver block diagram.

The processing of GNSS signals starts with the antenna, designed to capture signals from the satellites. This antenna is tuned to the GNSS-specific frequencies, and its quality and type significantly influences the receiver's accuracy and sensitivity.

Following the antenna, the RFFE amplifies the received signals and converts them from their high-frequency state to a lower intermediate frequency or directly to baseband for simpler processing. This stage encompasses filtering, initial signal conditioning, and ADCs.

At the core of the GNSS receiver is the baseband signal processing chain. An acquisition block detects the signals received from visible satellites, and the receiver implements several channels, each capable of tracking and demodulating the telemetry messages from one of those signals. An observables block performs the basic GNSS measurements, including pseudoranges, carrier phase, and Doppler shift. Finally, the PVT Engine computes the receiver's position, velocity, and time. It employs algorithms to solve navigation equations using data such as satellite positions and the basic GNSS measurements.

2.3.1 GNSS Antennas

Antennas used in GNSS systems are generally omnidirectional, enabling them to capture a wide array of satellite signals effectively. In applications where the antenna remains stationary, the antenna design typically focuses on receiving signals primarily from the upper hemisphere. This approach helps to exclude signals arriving from below the horizon, which are often prone to interference or jamming, particularly those coming from lower elevation angles. In contrast, for dynamic environments, antennas are designed to maintain a uniform gain pattern extending below the horizon. This feature is essential to counteract the effects of vehicle movements, such as rolling and pitching [34].

As mentioned in subsection 2.2.1, GNSS signals are transmitted using RHCP. However, upon reflection off surfaces such as buildings or the ground, these signals can change their polarization from RHCP to Left-Handed Circular Polarization (LHCP). Such reflected signals can degrade the quality of signal reception. To mitigate this issue, GNSS antennas are often engineered to have lower gain for LHCP signals. This design minimizes the impact of multipath effects, where reflected signals interfere with direct GNSS signals, thereby ensuring more accurate and reliable signal reception [26, 34].

Active antennas incorporate a Low-Noise Amplifier (LNA) to compensate for potential signal loss when a long cable is used to connect the antenna to the receiver.

The thermal noise power at the receiver antenna is given by

$$N = kT_aB \,, \qquad \text{[W]} \quad (2.13)$$

where k is the Boltzmann's constant ($1.38 \cdot 10^{-23}$ J/°K), B is the bandwidth of the receiver in Hz, and T_a is the antenna noise temperature. The antenna noise temperature is a measure of the noise being produced by an antenna in a given environment. It refers to the temperature of a resistor that would produce the same thermal noise as the antenna. This temperature depends of the gain, radiation pattern and the noise that the antenna picks up from the surrounding environment.

For GNSS, the signals received at the antenna include a small amount of noise from the satellite and additional background radiation from other sources, resulting in an effective noise temperature (T_a) of typically about 130 K [35]

As shown in section 2.2.1, the minimum received power level of the GNSS signals is around −160 dBW. Considering a bandwidth of $B = 10$ MHz, and an antenna noise temperature of 130 K, the Signal-to-Noise Ratio (SNR) at the receiver antenna can be computed as

$$\begin{aligned} SNR &= 10log_{10}(\frac{P}{N}) = 10log_{10}(P) - 10log_{10}(kT_aB) \\ &= 10log_{10}(160) - 10log_{10}((1.38 \cdot 10^{-23}) \cdot 130 \cdot 10 \cdot 10^6) \\ &= -22.5 \text{ dB} \ . \end{aligned} \tag{2.14}$$

In this equation, P is the power of the received signal, and N is the noise power at the antenna.

Negative SNRs in GNSS are a common occurrence due to the nature of satellite communication and the calculation of SNR. The GNSS receiver acquisition and tracking algorithms are designed to process these weak signals, extracting the required signal from the noise. In GNSS, the received signal power is commonly expressed in terms of the C/N_0, which is the ratio of carrier power C to noise power density N_0, and denotes the strength of the power of carrier wave relative to the noise. The C/N_0 does not depend on the receiver bandwidth. The C/N_0 is related to the SNR as described by the following relationship:

$$C/N_0 = SNR \text{ [dB] } + 10log_{10}(B) \ . \qquad \text{[dB − Hz]} \tag{2.15}$$

Standard outdoor working conditions are typically characterized by nominal C/N_0 values of typically ≥ 44 dB-Hz [18].

2.3.2 Radio Frequency Front-End (RFFE)

The RFFE handles the initial processing of the signals received from GNSS satellites. RFFEs perform low noise amplification, filtering, downconversion, and Automatic Gain Control (AGC). The first stages of the front-end are a band pass filter and a LNA. The LNA amplifies the received signals while adding as little additional noise as possible, a crucial step in preserving signal integrity. After amplification, the signals are passed through filters. These filters are designed to remove unwanted noise and interference from frequencies not used by the GNSS signals, thus ensuring that only the relevant signal frequencies are processed further. In addition to the filtering process, the signals undergo a frequency downconversion: a mixer in the front-end performs frequency shifting to an Intermediate Frequency (IF) or directly to baseband, making the GNSS more manageable for the subsequent stages. Finally, an AGC automatically regulates the gain in the RFFE, ensuring a steady output level and minimizing quantization losses in the ADC.

A simplified block diagram of a GNSS receiver's analog front-end is depicted in Figure 2.5. In this diagram, each component is characterized by its gain and noise figure. The noise figure represents the degradation of the SNR caused by the components within the signal chain. It is defined as the ratio of the SNR at the component's input to the SNR at its output. For a more comprehensive depiction, refer to [35].

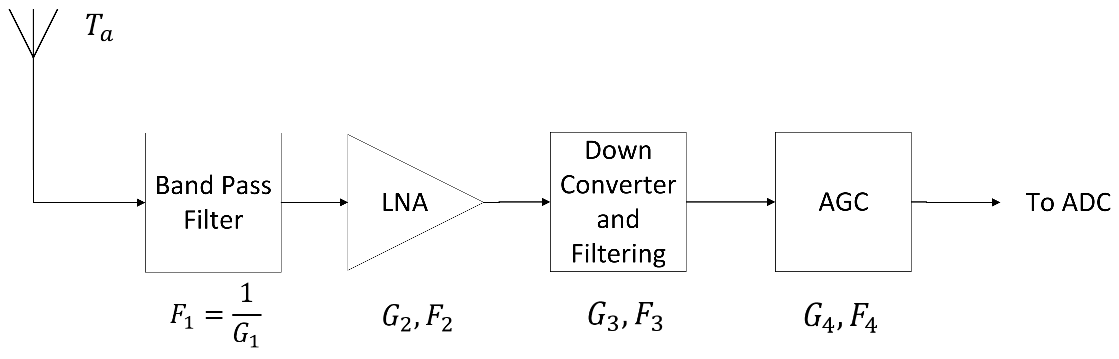

Figure 2.5 Example Block Diagram of a GNSS receiver's RFFE.

The entire RFFE can be characterized by its front-end noise figure and the effective noise temperature, T_{eff}, according to the Friis formulas for noise [35]. The effective noise temperature represents the temperature of a passive element that would generate the same amount of thermal noise power as all the components in the analog front-end combined. The front-end noise figure F is given by

$$F = F_1 + \frac{(F_2 - 1)}{G_1} + \frac{(F_3 - 1)}{G_1 G_2} + \frac{(F_4 - 1)}{G_1 G_2 G_3} \ . \tag{2.16}$$

The front-end effective noise temperature T_{eff} can be calculated as

$$T_{eff} = T_a + (F_1 - 1)T_0 + \frac{(F_2 - 1)T_0}{G_1} + \frac{(F_3 - 1)T_0}{G_1 G_2} + \frac{(F_4 - 1)T_0}{G_1 G_2 G_3} \ . \tag{2.17}$$

Moreover, the noise power density can be determined as

$$N_0 = kT_{eff} \ . \qquad \text{[W/Hz]} \tag{2.18}$$

Finally, the noise power at the output of the AGC in Figure 2.5 can be derived as

$$P = k \cdot T_{eff} \cdot B \ . \qquad \text{[W]} \tag{2.19}$$

Note that Figure 2.5 assumes a passive band-pass filter is placed after the antenna; consequently, $F_1 = \frac{1}{G_1}$.

2.3.3 Analog to Digital Converter (ADC)

The ADC converts the analog signals to a digital format. A way to express the quantization losses in the ADC is to measure the degradation of the SNR caused at the correlator outputs [36]. In GNSS receivers, where the power of the received signals is weaker than the surrounding thermal noise, the AGC primarily responds to the ambient noise. For this reason, given a specific number of quantization bits in the ADC, the quantization losses mostly depend on the ratio between the maximum quantization threshold L and the input noise standard deviation σ. This quantization loss is not influenced by the received GNSS signals, as the levels of these signals are considerably low and insignificant compared to the noise floor at the input of the ADC [36]. When the optimal ratio is used, the signal degradation is about 1.96 dB when using 1-bit quantization, 0.54 dB when using 2-bit quantization, and further decreases for higher bit quantization. Consequently, the dynamic range required for the ADC is not excessively high. This is why many commercial GPS receivers employ ADCs with just 1 or 2 bits [36]. The only exception to that occurs when GNSS receivers are built with antijamming capacity [26].

2.3.4 Signal Acquisition

The acquisition process detects the presence or absence of GNSS signals, each signal distinguished by unique PRN codes. When a positive detection occurs, the acquisition process yields a rough estimate of the PRN code phase and the Doppler shift, with sufficient accuracy to initiate the tracking loops. The code phase represents the relative timing offset between a local replica of the PRN code generated internally in the receiver and the PRN code in the signal received from a satellite. Achieving accurate timing synchronization, or aligning the code phase, is crucial for the receiver to correctly begin the tracking process.

The acquisition involves a two-dimensional search across various trial Doppler frequencies and code phases. The receiver adjusts the timing of the local replica of the PRN code to match the incoming signal, modifies the residual frequency shift of the local replica to compensate for the Doppler frequency of the received signal, and performs a cross-correlation between the received signal and the local replica. The duration over which the receiver sums or integrates the signal in a coherent manner when performing the cross-correlation is referred to as Coherent Integration (CI) time. GNSS satellites use orthogonal PRN codes, which exhibit high autocorrelation only at zero lag. Therefore, the receiver achieves maximum cross-correlation when the received signal and the local replica are perfectly aligned, and the Doppler frequency of the received signal is effectively wiped-off.

A way to perform this two-dimensional search is to perform a serial search, systematically testing different combinations of frequency offsets and code phases one after the other. However, this results in a long acquisition time as each potential combination must be checked sequentially, making the process time-consuming, especially in scenarios with weak signal conditions or high levels of interference. As an example of this, GPS L1 C/A signals have PRN codes with a duration of 1 ms, and a length of 1023 chips, therefore the receiver has to search through 1023 possible code phases. On top of that, as explained in subsection 2.2.4, a static GNSS receiver on the earth surface may experience a Doppler frequency of $\approx\ \pm 5$ kHz. A typical value of the search step of the Doppler frequency used in the local replica is $f_{step} = \frac{1}{2T_{CI}}$ [37], and a typical value for T_{CI} is the duration of the PRN code, 1 ms in this case, or a multiple of it, resulting in T_{CI} = 500 Hz. If, for instance, a sampling frequency of 4 Mega samples per second (Msps) is employed, and a code phase search with a granularity of one sample is conducted, this results in a very large number of combinations:

$$(1\text{ms}) \cdot (4\text{Msps}))(2\frac{5000\ \text{(max Doppler Hz)}}{500\ \text{(Doppler step Hz)}} + 1) = 40000\ \text{combinations}\ . \tag{2.20}$$

In practice, to improve the speed and efficiency of signal acquisition, techniques are used to parallelize the search across one of the parameters: the code phase or the Doppler frequency. The Parallel Code Phase Search (PCPS) algorithm is usually more efficient in scenarios with stable signal conditions [38].

The PCPS algorithm performs a circular correlation between the incoming signal and the local replica of the PRN code in the frequency domain. The computation of the circular cross-correlation is performed as follows: the discrete Fourier transforms of the finite length sequences $x(n)$ and $y(n)$, both of length N, may be formulated as

$$X(k) = \sum_{n=0}^{N-1} x(n) e^{-\frac{j2\pi kn}{N}} , \tag{2.21}$$

and

$$Y(k) = \sum_{n=0}^{N-1} y(n) e^{-\frac{j2\pi kn}{N}} , \tag{2.22}$$

respectively. Multiplying $X(k)$ and $Y(k)$ yields the circular cross-correlation of $X(k)$ and $Y(k)$ in the frequency domain. This relationship can be expressed as

$$\begin{aligned} Z(K) = X^*(K)Y(K) &= \sum_{m=0}^{N-1} x(m) e^{-\frac{j2\pi km}{N}} \sum_{n=0}^{N-1} y(m+n) e^{-\frac{2\pi k(m+n)}{N}} \\ &= \sum_{n=0}^{N-1} \sum_{m=0}^{N-1} x(-m) y(m-n) e^{-\frac{2\pi kn}{N}} . \end{aligned} \tag{2.23}$$

Performing the inverse Fourier transform on $Z(K)$ produces the circular cross-correlation sequence between two finite length sequences $x(n)$ and $y(n)$ both with length N and with periodic repetition:

$$z(n) = \frac{1}{N} \sum_{n=0}^{N-1} x(-m) y(m-n) = \sum_{m=0}^{N-1} x(m) y(m+n) . \tag{2.24}$$

Figure 2.6 shows a block diagram of the PCPS algorithm [38].

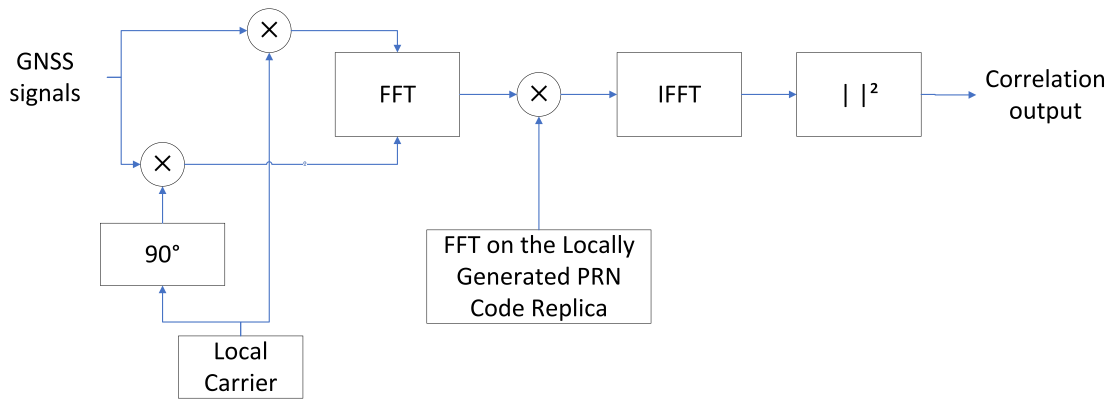

Figure 2.6 Parallel Code Phase Search (PCPS) algorithm [38].

The receiver correlates the received signal with the local replica across all trial Doppler frequencies to obtain the Cross-ambiguity function (CAF). The CAF is a two-dimensional function correlating the received signal with the local replica by varying both time delay and Doppler shift. A peak in the correlation output indicates the precise timing of the received code and its Doppler frequency. After acquisition, the receiver transitions to the tracking phase, where it continuously adjusts the timing and frequency of its local code replica to maintain alignment with the satellite signal, as both the satellite and receiver are in motion.

More in-depth analysis of the acquisition algorithm can be found in [25, 26, 34, 38].

2.3.5 Signal Tracking

The primary purpose of tracking is to closely follow the evolution of the signal synchronization parameters of a satellite signal and to demodulate the navigation data contained within it. These synchronization parameters are specific elements of the received signal that enable the receiver to maintain a stable lock on the signal, including for instance the code phase, Doppler shift and carrier phase.

The tracking process executes the tracking loops, which are feedback systems that continuously adjust the receiver's internal code, frequency, and phase to match those of the incoming satellite signals. Two tracking loops are needed to track the received signals: one loop is used to track the Doppler shift and is referred to as the carrier loop. The other one is used to track the PRN code and is referred to as the code loop. Typical GNSS receivers rely on scalar tracking techniques employing Delay-Locked Loop (DLL) and Phase-Locked Loop (PLL) architectures (*i.e.*, first- or second-order DLLs and second- or third-order PLLs), for code and carrier phase tracking, respectively.

A typical GNSS receiver implements coupled code and carrier tracking loops. Figure 2.7 depicts a block diagram of the tracking loops. The code tracking loop comprises the generation of the PRN code replica, the code discriminator, and the code filter, while the carrier tracking loop encompasses the Numerically Controlled Oscillator (NCO), the carrier discriminator, and the carrier filter.

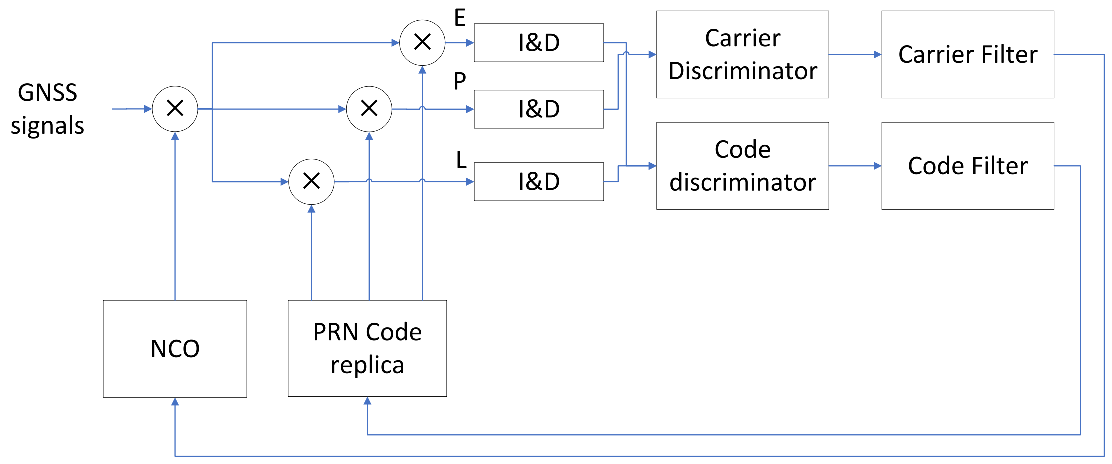

Figure 2.7 Tracking Loop.

The receiver performs Doppler wipeoff by adjusting the frequency and phase of the NCO to match the Doppler shift. The digitized complex input baseband signal, comprising the In-Phase (I) and In-Quadrature (Q) components, is multiplied by The output of the NCO.

The receiver generates three replica codes: Early (E), Prompt (P), and Late (L), with typically half-chip phase differences between them. The Doppler-corrected I and Q components are correlated with these three replica codes by undergoing multiplication with the E, P, and L codes, followed by coherent integration through the Integrate and Dump (I&D) block.

The correlation outputs are subsequently processed by both a code discriminator and a carrier discriminator. These discriminators compare the integrators' outputs to estimate code phase, and carrier phase (or frequency) errors, respectively. Various techniques are available for acquiring code phase differences and carrier phase differences, each with differing computational loads and accuracy levels. Code phase discriminators include noncoherent early minus late power, quasi-coherent dot product power and coherent dot product [25]. Carrier phase discriminators include PLLs and Costas Loops. PLLs are sensitive to data bit transitions, whereas Costas Loops are designed to be insensitive to such transitions. Both PLLs and Costas loop discriminators produce phase error estimates at their respective outputs [25].

A Frequency-Locked Loop (FLL) discriminator can also be employed for GNSS tracking. FLL discriminators generate a frequency error estimate. While the PLL and the Costas loop are known for their high accuracy, they are more susceptible to dynamic stress, which encompasses changes in carrier phase and Doppler shift resulting from motion and acceleration. In practice, tracking loops are typically initiated using short coherent integration times in conjunction with an FLL and a wideband carrier loop filter. Subsequently, these tracking loops smoothly transition into a Costas loop when data modulation is present on the carrier signal or a PLL when tracking dataless pilot signals. This transition process involves gradual adjustments: the coherent integration time is gradually aligned with the period of data transitions, and the carrier tracking loop bandwidth narrows down to the extent allowed by the anticipated dynamics [25].

Code and carrier filters are applied to the discriminators' outputs to reduce noise. Subsequently, the filtered code phase and carrier phase errors are used to drive the generation of the PRN code replica and the NCO, facilitating the refinement of the alignment between the PRN code replicas and Doppler compensation. Various types of filters are discussed in more detail in [25].

The number of PRN code replicas used in GNSS receivers is optimized for the best tracking performance of the received signals. For GPS L1 C/A signals, three replicas – E, P, and L – are typically employed. When tracking GNSS signals with pilot components, such as GPS L5 and Galileo E5a signals, four replicas are used: E, P, and L for the pilot component, plus an additional Prompt P for the data component and navigation data demodulation. For Galileo E1 signals, the use of the CBOC(6,1, 11) modulation introduces correlation ambiguities. To mitigate the risk of tracking a local maximum instead of the global maximum, discriminators incorporating two additional samples of the cost function, named Very Early (VE) and Very Late (VL) can be employed [39].

More in-depth analysis of the tracking loops can be found in [25, 26, 34, 38, 40].

2.3.6 Telemetry Decoder

The telemetry decoder processes the data produced by the tracking loops, extracting and decoding the data bits from the navigation messages broadcast by GNSS satellites, and obtaining the satellite ephemeris data, almanac data, time information, satellite status information and the broadcasted ionospheric data.

To synchronize with the navigation messages, the telemetry decoder initially locates the telemetry preambles. These preambles consist of known bit patterns transmitted within the navigation messages. They facilitate the identification of the beginning of new frames or subframes, and help the receiver identify and align with the specific navigation message of a satellite.

2.3.7 Observables

The Observables block gathers synchronization data from all the active processing channels and calculates the fundamental GNSS measurements based on this data. These key measurements include pseudorange, carrier phase (or its equivalent in phase-range form), and Doppler shift (alternatively expressed as the pseudorange rate).

2.3.8 Position, Velocity, and Time (PVT) computation

The PVT block computes the user's position, velocity, and time based on the GNSS basic measurements. It processes these measurements to calculate navigation solutions, providing this information in formats suitable for further processing or representation.

The primary function of the GNSS PVT system is to ascertain the geographic location of the receiver. This determination involves calculating the receiver's coordinates (latitude, longitude, and altitude) using signals from at least four GNSS satellites. The process, detailed in section 2.1, utilizes trilateration based on pseudoranges, which are the distances measured by the time it takes for signals to travel from the satellites to the receiver.

Furthermore, the PVT system derives the receiver's velocity (both speed and direction) by analyzing the Doppler shift of GNSS signals. This accurate velocity information is indispensable for applications like vehicle navigation and tracking.

GNSS systems provide not only precise location and velocity data but also essential timing information. This timing information is crucial for synchronizing operations across various sectors, including telecommunications and power distribution networks. The accuracy of this timing information stems from the synchronization of satellite signals with onboard atomic clocks. Thus, GNSS PVT systems are foundational, offering comprehensive solutions for location, movement, and time synchronization across a wide range of applications, from personal navigation to critical infrastructure management.

2.4 GNSS-SDR Software Receiver

GNSS-SDR is a widely recognized open-source GNSS software receiver, freely accessible online and released under the GNU General Public License (GPL) license [16,17]. It processes raw signal samples collected through an RFFE, computes GNSS basic measurements, such as pseudoranges, pseudorange rates, phase ranges, signal strengths, and obtains navigation solutions. GNSS-SDR integrates with GNU Radio, a widely used platform for signal processing and software-defined radio projects [41]. This integration gives GNSS-SDR access to an extensive library of signal processing blocks and a robust runtime environment. In the SoC FPGA receiver architecture proposed in this thesis, GNSS-SDR serves as the core baseband processing engine, enhanced by the capability to offload computationally intensive tasks to the FPGA, optimizing performance and efficiency.

The source code of GNSS-SDR is written in C++ and released under the GNU GPL. The software receiver can be built using popular and freely available compilers. As a result, the source code can be freely inspected and modified.

2.4.1 Receiver Configuration

GNSS-SDR is highly configurable, allowing users to tailor the software to meet a wide variety of needs and applications. This flexibility stems from its architecture, which is designed to be modular and arbitrarily extensible. Users can adjust parameters related to signal processing algorithms, tracking loops, and data output formats, among others. This configurability enables the adaptation of GNSS-SDR to different GNSS signals, receiver hardware, and processing environments, from low-cost, resource-constrained systems to high-performance computing platforms. Users can specify the satellite systems and signal types they wish to process through configuration files, allowing for a high degree of customization for specific projects or experiments.

The receiver's settings and operational parameters are determined through a configuration text file. This file allows users to define the behavior of the receiver without modifying the source code, making it a user-friendly way to adapt the software to different requirements and scenarios.

2.4.2 Receiver Architecture

The GNSS-SDR software receiver is designed around the principle of task parallelization, where execution threads are distributed across multiple computing nodes to execute concurrently, handling different threads or processes on the same or varied data sets. This design aims to spread out processing tasks across several threads, a strategy critical for avoiding processing or memory access bottlenecks that could impede the entire processing chain from achieving real-time operation.

The software's architectural approach draws inspiration from Kahn's formal representation of process networks and his work on defining a programming language that simplifies the structuring of dynamically evolving networks of processes [41, 42]. In this model, processes within a network communicate exclusively through channels, sending and receiving data elements or tokens. A process waits when attempting to fetch data from an empty channel, ensuring that the network's behavior remains deterministic. In this way, the sequence of tokens produced does not depend on the processes' execution order. With the correct scheduling policy, it's possible to create software-defined radio networks that are non-terminating and strictly bounded, meaning they can run indefinitely without deadlock situations and maintain a bounded number of data elements in communication channels, regardless of execution order.

This concept is encapsulated in the idea that software-defined radios can be conceptualized as a flow graph comprising nodes (representing signal processing blocks) and links (representing data flows). This flow graph is essentially an acyclic directional graph with no closed cycles, featuring source blocks to introduce samples, sink blocks to terminate or export samples, and various signal processing blocks in between. This structure underpins the GNSS-SDR's design, ensuring efficient and reliable processing of satellite navigation signals.

GNSS-SDR is composed of two main components: a control plane, responsible for overseeing the receiver's operations, interfacing with the operating system, external applications, and user-machine interactions; and a signal processing plane, dedicated to deriving information from raw signal samples [16].

The Control plane creates a flow graph in which a sample stream goes through a network of connected signal processing blocks up to the position fix. The flow graph is created according to the GNSS-SDR configuration file mentioned in Section 2.4.1. The configuration file allows users to define the flow graph (type of signal source, number of channels, algorithms to be used for each channel and each module, strategies for satellite selection, type of output format, etc.)

The signal processing plane is the collection of blocks that implement digital signal processing algorithms . Integrating with the GNU Radio framework [41], GNSS-SDR leverages a modular design where each component in a flow graph is designed to handle data and asynchronous messages through multiple input and output connections. This architecture enables the software to efficiently process digital signals by dynamically scheduling tasks across its components. Specifically, a sophisticated runtime scheduler ensures that data units are effectively passed from data sources to destinations. Here, individual processing blocks operate independently in separate threads, working diligently to process incoming data from their buffers as swiftly as possible, without being constrained by the rate of incoming data. Every block operates with its own distinct scheduler, alongside an asynchronous messaging system that facilitates communication with both upstream and downstream blocks. The underlying scheduler orchestrates the movement of data across the graph, from sources to sinks, optimizing the flow to maintain high throughput. This method allows GNU Radio-based software receivers to operate at peak efficiency, leveraging the full potential of the processing hardware, data flow, and available buffer capacity to handle the input signal at the highest possible processing speed [16].

Figure 2.8 shows a diagram of the modules that form the GNSS software receiver. GNSS-SDR implements the baseband signal processing chain, from sample capture up to the computation of the PVT solution and the generation of GNSS products in standard formats, enabling interoperability and integration with other systems. The flowgraph contains several processing blocks: signal source, signal conditioner, acquisition, tracking, navigation message decoder, observables, and PVT, enabling easy addition, modification, and replacement of GNSS receiver algorithms. A Control Thread reads the GNSS-SDR configuration file and operates in parallel with the flow graph, receiving notifications and triggering changes in the receiver's state. A channel Finite State Machine (FSM) controls the interaction of the various channel blocks. The receiver uses GNU radio's streaming data flow model and messaging system to pass data and asynchronously communicate events between blocks. The signal source, signal conditioner, acquisition, tracking, telemetry decoder, observables and PVT blocks are regular GNU radio signal processing blocks.

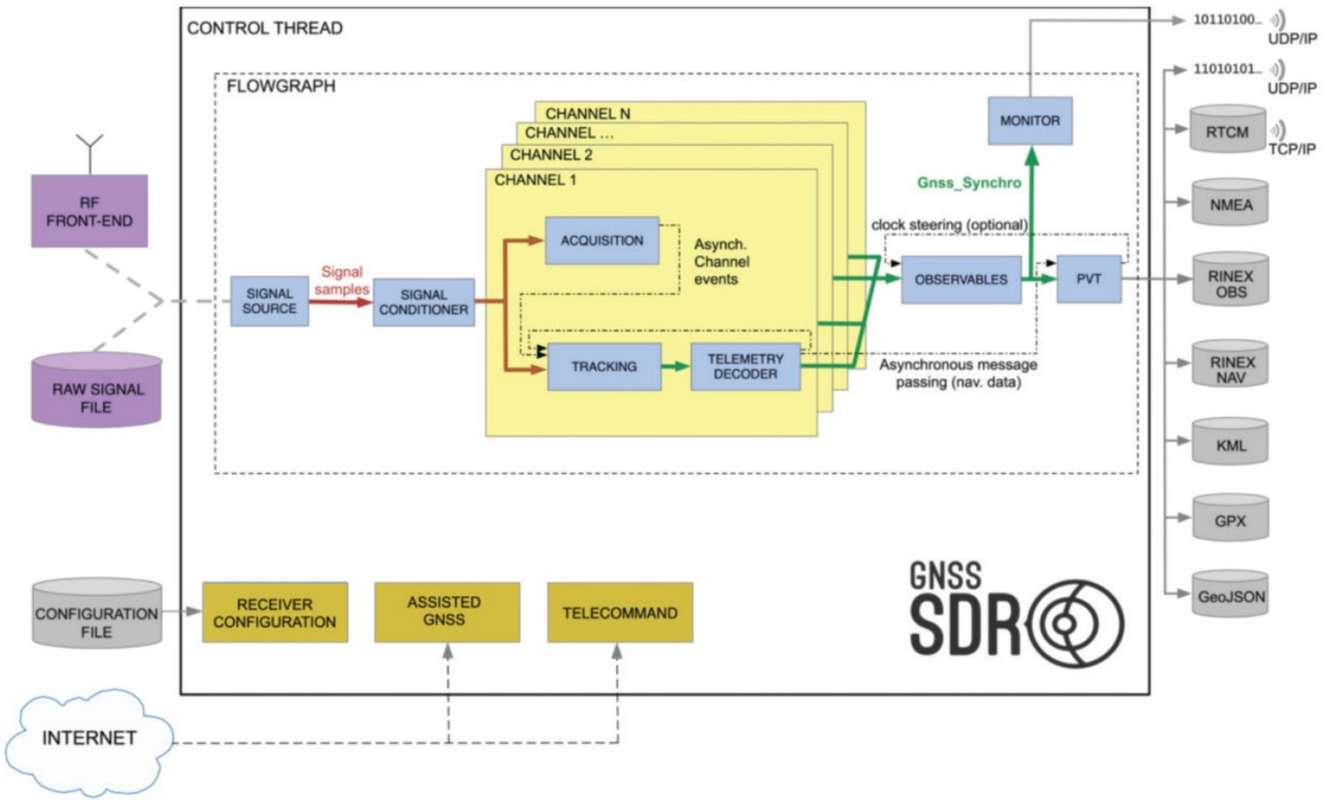

Figure 2.8 Software architecture of the GNSS-SDR software receiver (source: [17]).

2.4.3 Supported Output Formats

The proposed receiver produces GNSS signal products in standard open formats, including Geographic Information Systems (GIS)-oriented, standard, and application-specific messages, as well as observation and navigation data files. These formats include Receiver Independent Exchange Format (RINEX) Files, Radio Technical Commission for Maritime Services (RTCM) messages with configurable rates, GPS Exchange Format (GPX), Keyhole Markup Language (KML), GeoJSON, and National Marine Electronics Association (NMEA)-0183 messages for sensor integration. PVT solutions are generated at a configurable rate in World Geodetic System (WGS)-84 based on Least Squares or Kalman filtering. For each visible satellite, the receiver provides time-tagged measurements of pseudorange (in m), carrier phase (in cycles) or phase range (in m), Doppler shift (in Hz) or pseudorange rate (in m/s), received signal strength (in dB-Hz), Dilution of Precision (DOP), raw navigation data, tracking correlators' output, GPS Time, Galileo Time, and Coordinated Universal Time (UTC) time. This facilitates integration with other positioning technologies.

Table 2.7 lists some of the most relevant output formats.

Table 2.7 Support of output formats

Type of output format	Output format
GIS-oriented	• KML [43] • GeoJSON [44]
Standard and application-specific messages	• NMEA 0183 [45] • GPX [46] • RTCM-104 v3.2 [47]
Observation and navigation data files	• RINEX v3.02 [48]

2.4.4 Cross-compiling GNSS-SDR

GNSS-SDR can be cross-compiled across multiple architectures, including but not limited to Intel x86-64 [49] and ARM [50].

Embedded GNU/Linux systems suitable for operating GNSS-SDR can be created using the Yocto project. The Yocto project is an open-source building framework for the creation of customized embedded GNU/Linux systems, providing tools, processes, templates, and methods to rapidly create and deploy embedded platforms [51]. The Yocto Extensible Software Development Kit (eSDK) is a component of the Yocto Project, designed to streamline the addition and modification of packages in an embedded Operating System (OS), as well as facilitating the rebuilding of the final OS image. It provides a comprehensive and configurable Yocto Project environment, allowing for a more flexible and efficient development workflow.

The Yocto Project uses the OpenEmbedded build system [51]. The core of this system is the BitBake tool, along with a set of metadata known as OpenEmbedded-Core (OE-Core). The Yocto Project and OpenEmbedded share OE-Core, which is a collection of recipes, classes, and associated files used to build various software packages and images for embedded devices. This build system facilitates the organization of metadata into multiple layers. These layers serve to segregate various customizations applicable to different applications and system components. Each layer is a repository containing a related set of instructions that guide the build system's actions. This layered approach not only allows for the integration of hardware and software components but also supports collaboration and customization.

An OpenEmbedded layer named *meta-gnss-sdr* is available to facilitate the development and installation of GNSS-SDR across various embedded platforms [52]. Furthermore, a repository named *oe-gnss-sdr-manifest* [53] provides repo manifests for setting up the OpenEmbedded build system specifically for building systems based on the meta-gnss-sdr layer. Repo is a tool that enables the management of many repositories given a single manifest file [54].

Finally, a customized GNU/Linux distribution named *Geniux* (GNSS-SDR for Embedded GNU/Linux) supports the development and operation of GNSS-SDR on embedded devices [55]. Based on the Yocto Project, this distribution includes a carefully selected set of tools, libraries, and drivers optimized for a broad spectrum of SDR applications, thus facilitating their advancement to production readiness. The foundation of Geniux is the meta-gnss-sdr layer, which specifies the distribution's packages and provides the recipes for downloading and building these components.

In addition to this, configurations and instructions for creating Docker images are available online, specifically designed for FPGA and GNU/Linux design tools [56]. These Docker images facilitate the setup of the design tools required for designing embedded systems based on GNSS-SDR.

2.5 Introduction to FPGAs

FPGAs are a type of digital integrated circuit that can be reconfigured and reprogrammed after manufacturing, which gives them their unique *field-programmable* attribute. These programmable semiconductor devices are based on a matrix of Configurable Logic Blocks (CLBs), connected through programmable interconnects. This design allows them to perform a wide range of logical functions, ranging from simple logic gates to complex combinational functions and state machines. The interconnects facilitate various configurations of these blocks, making them similar to a digital breadboard in their flexibility and reusability. FPGAs also feature specialized configurable hardware accelerators for specific operations, such as DSP slices for fast complex addition and multiplication, Random-Access Memory (RAM) blocks for data storage, and Clock Management Tiles (CMTs) for clock generation, clock distribution, and clock manipulation.

FPGAs are highly valued for their reprogrammability, making them suitable for a variety of applications such as rapid prototyping, product development, and systems that require frequent updates. Although FPGAs are fully manufactured devices, their architecture remains design-independent, providing exceptional flexibility for implementing a wide range of designs. This versatility allows them to adapt to evolving technological needs and project specifications

The flexibility and rapid processing capabilities of FPGAs are their main advantages. FPGAs achieve parallelism by executing various tasks concurrently through the utilization of diverse configurable logic blocks, leading to improved system performance and reduced processing time.

Programming an FPGA typically involves using a Hardware Description Language (HDL) such as Very High Speed Integrated Circuit Hardware Description Language (VHDL) or Verilog. These languages enable designers to describe the desired logic and circuitry in a textual format. Designing with FPGAs can be challenging, requiring significant expertise in both hardware design and programming. Their effective use demands a deep understanding of digital circuit design and specialized programming languages.

FPGAs are extensively used in various areas, including the prototyping of integrated circuits, digital signal processing, telecommunications, and safety-critical domains such as the automotive and aerospace industries. FPGAs are more cost-effective than Application-Specific Integrated Circuits (ASICs) for low-volume production. Unlike ASICs, which are custom-built for specific tasks, FPGAs offer adaptability but are less efficient for high-volume production due to their general-purpose nature.

2.5.1 Basic Building Blocks of an FPGA

The main elements of an FPGAs are the CLBs, the programmable interconnect, the Input/Output (I/O) blocks, and specialized hardware units, including DSP blocks, RAM blocks and CMT blocks. Additional specialized hardware units include high-speed trasnceivers for high-speed serial communication, supporting a wide range of communication protocols.

A CLB is a fundamental component in FPGAs, serving as the primary unit for implementing a variety of logical functions. Each CLB consists of programmable logic gates, Lookup Tables (LUTs), and flip-flops. The LUTs in a CLB can be configured to perform a broad spectrum of combinatorial logic circuits, ranging from simple logic gates to complex multi-bit arithmetic functions. Flip-flops are used for storing states or the results of logic operations, facilitating the creation of sequential logic circuits. Additionally, CLBs are instrumental in implementing distributed RAM within FPGAs. This type of RAM is a distributed form of memory, composed of individual memory elements. Each element is essentially a flip-flop capable of storing a single bit of data. These elements are strategically dispersed throughout the FPGA, enabling tight integration with the programmable logic elements, which is crucial for efficient processing in various applications.

The I/O blocks are programmable units in FPGAs that manage the interface between the FPGA's internal logic and external circuits. These blocks enable the FPGA to interact with other devices and systems, receiving input signals from and sending output signals to them. The I/O blocks in an FPGA are organized into groups, with each group capable of independently supporting different I/O standards. Through software configuration, these blocks can be adapted to various electrical standards and physical I/O characteristics, such as adjusting drive current size and modifying resistance during signal transitions. Additionally, many FPGAs are equipped to support Double Data Rate (DDR) register technology, which allows for faster data transfer by processing data on both the rising and falling edges of the clock signal.

The DSP blocks facilitate the implementation of complex algorithms for filtering, modulation, demodulation, Fourier transforms, and other signal-processing functions. The RAM blocks facilitate fast data access, buffering, and storage. Both DSP and RAM blocks are typically organized into columns on the FPGA device. [15].

FPGAs are intricately designed as a two-dimensional grid of configurable elements, as shown in Figure 2.9. The specialized hardware elements, including DSP and RAM blocks, are not depicted in this figure.

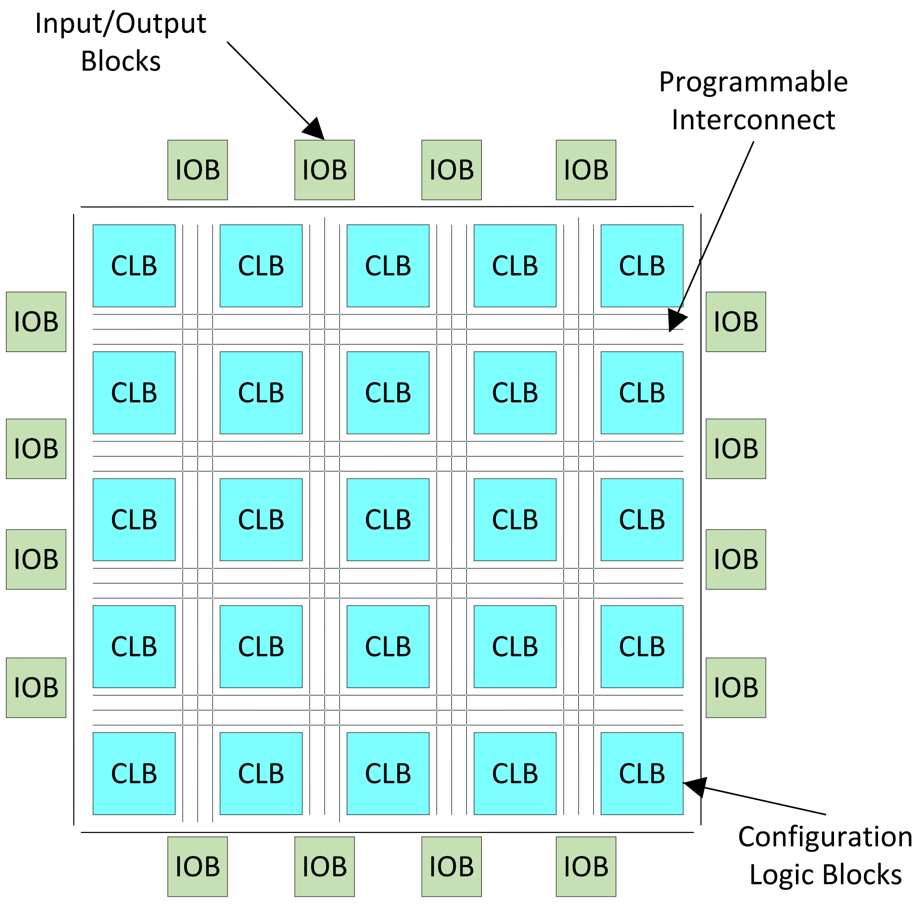

Figure 2.9 Simplified FPGA block diagram (DSP blocks, RAM blocks and other specialized hardware blocks not shown).

The programmable interconnect consists of electrically programmable interconnections, which include horizontal and vertical routing tracks, switch matrices, and multiplexers. These components collectively provide the routing paths for the FPGA's programmable logic blocks. The routing paths themselves are made up of wire segments of various lengths, which can be interconnected through programmable switches. The configurability of these paths is achieved by manipulating these switches, enabling them to open or close connections between different lines. The operation of these switches is controlled by the configuration data loaded into the

FPGA, allowing for the customization of the FPGA's internal circuitry according to specific needs [57].

2.5.2 FPGA Intellectual Property (IP) Cores

IP cores are pre-designed and pre-verified functional blocks or modules that can be integrated into FPGAs. These cores add specific functionalities or features to FPGA designs without the need to develop these functions from scratch. Provided by FPGA manufacturers, third-party vendors, or created in-house, IP cores streamline the development process by offering pre-built, tested, and optimized components. They are particularly useful for commonly used functions like communication interfaces, processors, or signal processing blocks, saving significant time and effort in the design process. Furthermore, the modularity of IP cores enhances the ease of maintenance and updates within a design. Many IP cores can be customized or configured to meet specific project requirements, offering scalability and adaptability. By utilizing these pre-existing cores for standard functionalities, developers can focus more on the unique aspects of their projects. This approach promotes innovation and distinctiveness in the final product, enabling developers to create advanced systems efficiently without reinventing the wheel.

Each FPGA vendor offers a distinct selection of IP cores, which can be categorized as hard, firm, and soft. Users have the capability to implement and package their own soft or firm IP cores for code reuse. Soft IP cores offer the greatest flexibility, while hard IP cores provide maximum efficiency and performance. The distinctions between these types are elaborated in the following subsections [57].

2.5.2.1 Soft IPs

Soft IP cores are provided as synthesizable Register Transfer Level (RTL) models. RTL is a level of abstraction in digital circuit design that describes the movement of data between registers and the logical operations performed on this data. This abstraction is typically implemented using HDLs, such as VHDL or Verilog. Synthesizable code, written in an HDL, refers to a specific type of code used in the design of digital circuits that can be implemented in FPGAs. It possesses unique characteristics that enable it to be directly converted into a physical hardware design. Synthesizable code is written in a way that unambiguously describes the circuit's behavior. This means the code is clear and deterministic in terms of how data flows and how operations are performed within the circuit. While synthesizable code describes the logic and structure of the circuit, it also can include timing constraints. These constraints guide the synthesis tool in optimizing the design for the desired performance characteristics.

Soft IPs offer several advantages in the design and development of FPGAs, including enhanced flexibility for customization, reduced development time and costs, ease of integration, and the ability to be easily modified and optimized for specific applications. Since they are not tied to a specific process technology, soft IP cores can be reused across multiple projects and different types of semiconductor fabrication processes. This reusability reduces the need for redesigning the IP for each new project.

Soft IP cores can be packaged using the standard IP-XACT format. IP-XACT is an Extensible Markup Language (XML) schema that provides a uniform method to define and describe these

IPs, facilitating their integration into larger integrated circuit designs [58]. This format is essential for ensuring that IPs from different sources can be effectively combined, enabling code reuse and streamlining the design process in electronic circuit development.

The SoC FPGA receiver architecture proposed in this thesis employs soft FPGA IP cores to implement algorithms with the highest computational demands, notably those associated with acquisition and tracking multicorrelators. These FPGA IP cores are packaged using the IP-XACT format, facilitating the reuse of the hardware accelerators across multiple SoC FPGA devices of the same manufacturer. This is described in more detail in Chapter 3.

2.5.2.2 Firm IPs

Firm IP cores are gate-level netlists that offer the flexibility to place modules within the FPGA based on usage, with minimal user-configurable options. A gate-level netlist is a representation of a digital electronic circuit at a very low level of abstraction. It describes the circuit in terms of individual logic gates and the connections (nets) between them. Each logic gate in the netlist corresponds to a specific digital function, such as AND, OR, NOT, or others. Firm cores offer a balance between flexibility and performance. They provide some level of customization but may not offer the same level of flexibility as soft cores.

2.5.2.3 Hard IPs

Hard IP cores are physically instantiated in the silicon. This means they are actual, tangible layouts embedded into the chip. They are highly optimized for Power, Performance and Area (PPA). Because they are physically part of the chip, they can be fine-tuned to achieve maximum efficiency in terms of speed, energy consumption, and space usage on the silicon. For this reason, using hard IP cores result in a more efficient design than using soft IP cores. Using dedicated hard IP blocks results in lower power consumption compared to implementing the same operations in general-purpose logic using CLB blocks, especially for power-critical applications.

However, the core functionality of hard IP cores is fixed, thereby limiting their flexibility and degree of configurability compared to soft IP cores. Because they are physically part of the chip, hard IP cores are typically tied to a specific fabrication process, and they are not as portable as soft IP cores.

Common examples of hard IP cores include the specialized hardware blocks mentioned in Section 2.5.1, including, DSP units, complex I/O interfaces, and specific memory types like Static Random Access Memory (SRAM). Hard IP cores are typically used in applications where high performance is crucial and where the volume of production is sufficient to offset the higher costs of design and manufacturing.

Memory elements like Block Random Access Memorys (BRAMs) and DSP units are methodically placed in rows or columns within the FPGA. This strategic placement ensures their even distribution and accessibility from CLBs, enhancing data storage, retrieval, and processing efficiency.

2.5.3 FPGA Design Flows

The FPGA design flow is a process that transforms a conceptual design into a hardware implementation in the FPGA [59, 60]. This process begins with the conceptualization of the system's functional requirements, followed by the design entry phase where the design is described using a HDL such as VHDL or Verilog. High-Level Synthesis (HLS) tools may also be employed to describe the design using higher abstraction level languages like C or C++.

Following design entry, the synthesis phase takes place. During synthesis, the HDL code is translated into a gate-level representation, known as a netlist, which is specific to the FPGA architecture being targeted. This netlist is then optimized to meet specified constraints including area, power, and performance objectives.

To ensure the design behaves as intended, simulation is conducted at various stages. Pre-synthesis simulation helps identify and correct logical errors in the HDL code, while post-synthesis simulation verifies that the synthesis process has not introduced any errors and that the design meets its intended functionality with real FPGA constraints applied.

The implementation phase is next, where the synthesized netlist is further processed to fit onto the physical FPGA device. This involves mapping the netlist to the specific resources available on the FPGA, and then placing and routing these mapped elements within the FPGA's architecture. Placement assigns each logic element a specific location on the FPGA, and routing establishes the interconnections between these elements. This step is crucial for meeting performance constraints such as timing. In addition, post-placement and routing simulation may be carried out to verify the implemented design. This stage of simulation is typically very time-consuming.

Timing analysis is conducted to ensure the design meets all timing requirements. This analysis is critical for verifying the performance of the design before it is programmed onto the FPGA.

Once the design is verified to meet the necessary constraints and performance requirements, a configuration file or bitstream is generated. This bitstream contains the data needed to program the FPGA with the specific design. The FPGA is then programmed using this bitstream through a hardware programmer, configuring its logic blocks and interconnects to implement the desired functionality.

The final stage involves verification and testing of the programmed FPGA in the target system to ensure it functions correctly in real-world conditions. Functional testing and in-system verification are conducted to monitor performance and identify any issues that need to be addressed.

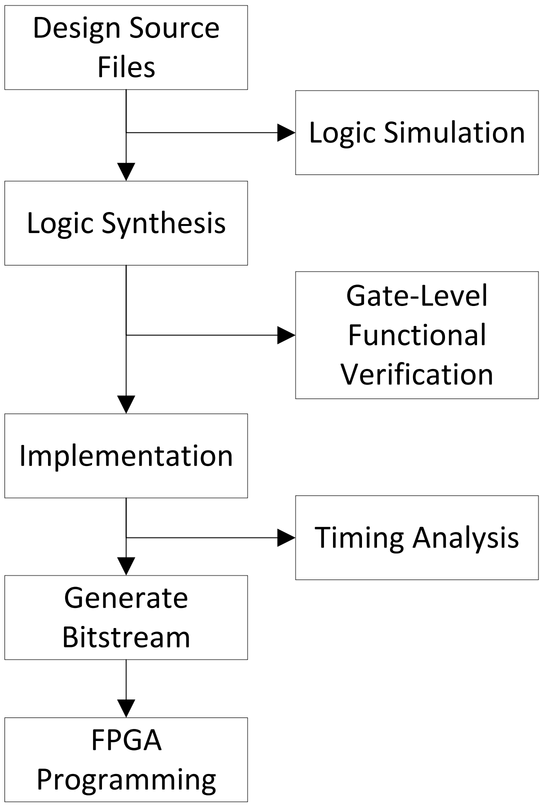

Figure 2.10 Simplified FPGA design flow.

Throughout the FPGA design flow, designers may iterate back to earlier stages to refine the design based on testing outcomes or to meet additional constraints. This iterative process, supported by effective use of simulation and timing analysis tools, is key to reducing development time and ensuring the quality of the final FPGA implementation.

2.5.4 FPGA Design Tools

FPGA development often relies on proprietary tools rather than pure software development tools. This preference is due to the complexity of FPGA architectures, limited availability of open documentation, and the necessity for IP protection. Major FPGA manufacturers such as Advanced Micro Devices (AMD), Intel, Microsemi, and Lattice Semiconductor each provide specialized software tools tailored to their hardware. These tools encompass synthesis, place and route, simulation, bitstream generation, and debugging.

Porting an FPGA design to an FPGA device of another manufacturer is a complex task due to several factors. These include not only the above mentioned unique vendor-specific toolchains, but also differences in FPGA architectures, variations in IP core compatibility, variations in clocking and timing constraints, resource utilization disparities, distinct I/O interfaces, and the presence of vendor-specific features. Each vendor has its strengths and weaknesses, therefore a careful selection of an FPGA vendor is essential.

2.6 SoC FPGAs

A SoC FPGA is a semiconductor device that contains a Programmable Logic (PL) and a Processing System (PS) within a single encapsulation. The PL consists of an FPGA, enabling flexible hardware implementations. The PS comprises an embedded processor, typically including memory controllers and various peripheral interfaces for communication with external components. Figure 2.11 illustrates this architecture. This design merges the programming ease of a processor with the flexibility and performance advantages of programmable logic fabric. Consequently, they provide a comprehensive processing system and programmable logic on a single chip, supporting a broad range of applications.

The programming logic is a standard FPGA, providing CLBs, memory blocks, DSPs blocks, and I/Os blocks. It is used for custom hardware acceleration, interfacing, and real-time processing.

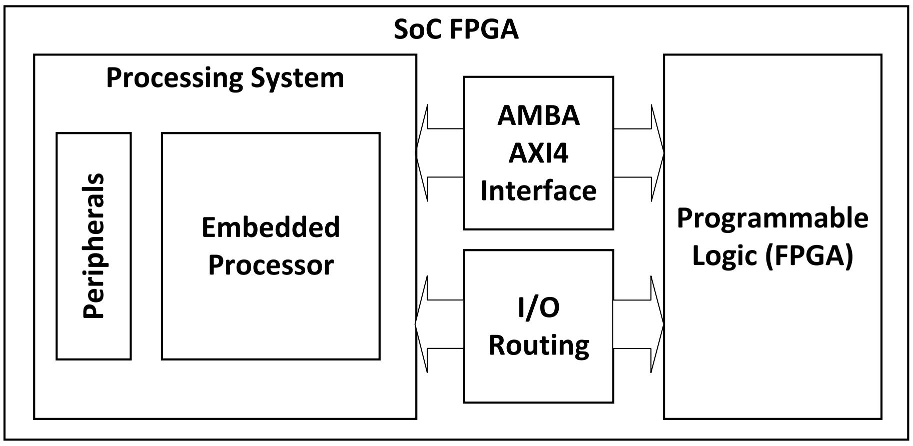

Figure 2.11 Simplified SoC FPGA block diagram.

Some SoC FPGAs feature additional processing units, such as GPUs, and specialized hardware units like Artificial Intelligence (AI) engines and video codecs. A specific category of SoC FPGAs, known as RF SoC FPGAs, also includes configurable RF ADCs and DACs.

Comparatively, SoC FPGAs offer several advantages over traditional FPGAs and microcontrollers. They combine the high-speed, adaptable hardware capabilities of an FPGA with the software-driven functionality of a microcontroller, thereby offering enhanced versatility. The PS and PL in an SoC FPGA are tightly coupled through various signals and interfaces, including on-chip communication bus protocols like the Advanced Microcontroller Bus Architecture (AMBA) Advanced eXtensible Interface (AXI) [61], I/O routing connections for directly connecting peripherals in the processing system to the FPGA, and interrupt lines. This arrangement enables efficient communication and seamless integration of functionality. Such integration facilitates faster data exchange between the processor and programmable logic, significantly reducing latency. Moreover, integrating both components on a single chip often leads to more efficient power usage.

Combining SDR techniques with FPGA technology allows the creation of adaptable signal processing systems. SDR techniques facilitate the processing of radio signals through software algorithms, and FPGAs deliver the essential hardware acceleration to execute these algorithms in real time. The flexibility and reprogrammability of FPGAs enables rapid reconfiguration of the SDR algorithms. Tasks or processes that are computationally intensive or time-critical can be offloaded to the FPGA fabric, where they can be executed faster than in a general-purpose CPU.

A detailed description of SoC-FPGAs, providing more details on the elements shown in Figure 2.11, can be found in [14, 15, 62, 63].

2.6.1 SoC FPGA Design Flows

The prototyping of SoC-FPGA-based systems encompasses both hardware and software design components. The hardware design is intricately mapped onto the physical resources available on the SoC device. Concurrently, the software is executed on one or more of the processors within the system.

The focus of hardware design is on creating custom hardware circuits. These are specifically mapped to the FPGA's reconfigurable array of logic blocks and routing resources. This approach allows designers to develop hardware accelerators, interfaces, and other logic elements that are precisely tailored to meet the needs of various applications. It has an impact on power management: FPGAs, tend to be more power-efficient than CPUs for algorithms that are highly parallelizable and require low latency and high throughput.

Software design involves the development of application software running on the system's processor cores. This software is responsible for the overall functionality of the system, including the management of the FPGA operations and handling intricate signal processing tasks. The software design also involves the development of drivers and firmware. This software facilitates communication, control, and data exchange between the CPU and the FPGA's custom hardware components.

Several design methods are available for the development of hardware and software, with the main approaches to designing SoC-FPGA-based systems being the separate hardware/software design flow and the software-oriented hardware/software co-design [15]:

- Hardware/software design flow: The system is divided into hardware and software sections that are designed independently, using dedicated tools for each. The designer searches for an optimal partitioning and assignment of tasks between the software running in the embedded processor and the hardware implemented in the FPGA, with the objectives of minimizing power consumption. Figure 2.12 illustrates a simplified workflow for this approach, where hardware and software development can largely proceed in parallel. Following their individual conclusion, Software and hardware development are followed by integration testing, where the FPGA and the software components are combined and tested to confirm that they interact according to their requirements [15]. This method allows each component to be developed independently, without waiting for the other's completion.
- Software-oriented, hardware/software co-design flow: the functionality of the whole system is described at a high level of abstraction using software code or block-based

design techniques, using advanced tools. The tools can then quickly partition hardware and software elements in the SoC-FPGA in different ways according to the designer's commands, and all the communication interfaces between the FPGA and the software are automatically managed by the tools [15]. The software functionality is then strategically distributed between the hardware and software components of the System on Chip (SoC), considering the available resource capabilities and guided by the designer's decisions. This approach can be markedly quicker, as the tools are capable of rapidly generating various configurations based on the designer's optimization preferences. Additionally, the integration of software and hardware elements is managed by the tools, simplifying the overall design process. Presently, there's considerable focus in research on design space exploration, the systematic process of hardware-software partitioning to optimize heterogeneous systems like SoC FPGAs for specific performance goals, using Machine Learning (ML) heuristics to decide which tasks should run on the software and which on the hardware portion of the chip.

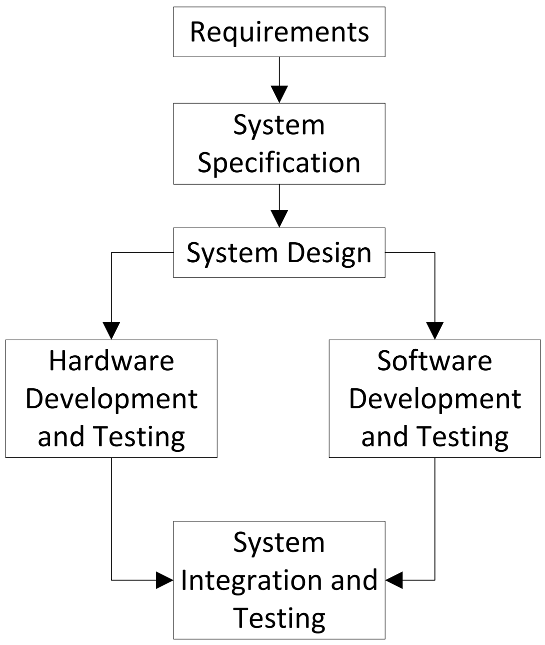
Requirements
System Specification
System Design
Hardware Development and Testing
Software Development and Testing
System Integration and Testing

Figure 2.12 A simple hardware/software design methodology [15].

The advantage of the hardware/software co-design approach is its ability to allow designers to test hardware and software components together early in the design cycle. This early integration facilitates the identification and swift resolution of issues or bottlenecks, contrasting with traditional hardware design where changes might be more time-consuming and costly.

Conversely, traditional separate design approaches provide the benefit of utilizing distinct, specialized tools for hardware and software. This offers flexibility, particularly advantageous for those who prefer employing a complete FOSS toolchain for software development, independent of hardware design tools.

The design methodology proposed in this thesis embraces a separate hardware/software design flow. This choice is motivated by the flexibility it offers, enabling the use of a FOSS toolchain for software development that is independent of the FPGA design tools. This approach not only offers flexibility but also potentially simplifies the development process by allowing for the independent evolution and optimization of software components.

More detailed explanations of the hardware/software design flow and the hardware/software co-design methodologies are provided in Chapter 3.

2.7 Design Forces and Key Performance Indicators (KPIs)

2.7.1 Design Forces

The GNSS receiver architecture and design methodology, alongside the concept demonstrators presented in this thesis, have been designed considering a range of design forces. These forces encompass factors, requirements, constraints, and considerations aimed at enhancing the performance of the proposed GNSS receiver prototypes. The design forces considered, as outlined in [64], include scalability, testability, portability, maintainability, reproducibility, openness, accuracy, precision, availability, efficiency, flexibility, interoperability, and reliability. These design forces are explained below:

- Scalability refers to the proposed architecture's ability to incorporate new features and process new satellite signals.
- Testability refers to the extent to which the proposed architecture supports testing.
- Portability refers to the ability of the proposed architecture to operate across various hardware systems, ensuring usability regardless of the underlying platform.
- Maintainability is defined by the ease with which the implemented receivers can be maintained, encompassing both error resolution and the implementation of new features.
- Reproducibility concerns the ability to achieve reproducible builds.
- Openness measures the degree to which the proposed architecture allows access for viewing, modifying, and using it, promoting transparency and collaboration.
- Accuracy refers to how close the measured or calculated position of a receiver is to the true or actual position.
- precision is related to the consistency of repeated measurements, indicating the spread around the average value.
- Availability measures the proportion of time the receiver is operational and capable of providing navigation solutions.
- Efficiency encompasses the receiver's ability to process multiple signals in parallel while optimizing power consumption.
- Flexibility denotes the receiver's adaptability to various scenarios and user requirements.

- Interoperability describes the capability of a GNSS receiver to utilize signals from multiple satellite constellations and to exchange raw GNSS data effectively.
- Reliability assesses the dependability of the navigation solutions provided by the receiver.

2.7.2 Key Performance Indicators (KPIs)

In the context of GNSS, KPIs are essential metrics for evaluating the performance and effectiveness of GNSS services and devices. They provide a quantifiable and measurable gauge of a product's progress towards achieving its objectives. These KPIs are designed to assess receiver performance, taking into account a range of design forces—both technical and operational challenges—that the proposed GNSS receivers must effectively address [64].

The design forces listed in Section 2.7.1 shape the context and objectives of the design process, while KPIs offer ways to measure success in meeting those objectives. Below is an examination of the KPIs considered for each design force. These KPIs are sourced from [64].

Table 2.8 outlines the KPIs considered for evaluating the architectural and design methodology in Chapter 3, alongside the design forces. Although these KPIs are not measured, they form the foundation of our discussion.

Table 2.8 KPIs for architectural and design methodology evaluation [64]

Design force	KPIs
Scalability	Quasi-linear acceleration with the number of processors available in the computing platform.
	Arbitrarily scalable architecture: unlimited addition of new GNSS signals and algorithms.
	Arbitrarily scalable configuration system.
Testability	Availability of a testing framework.
	Availability of an application-level logging system.
	Availability of a flexible configuration mechanism.
Portability	Supported processor and SoC-FPGA architectures.
	Supported operating systems.
Maintainability	Change Request Response Time
	Time to Fix Defects
Reproducibility	Availability of a software development process that ensures that a given source code will always produce the same binary or output (e.g., executable file, package, or any other form of software product)
Openness	Use of free and open source licenses and/or availability of a technical description of the algorithms implemented in the receiver

Table 2.9 illustrates the KPIs considered for the the concept demonstrator's evaluation, along with the design forces.

Table 2.9 KPIs for concept demonstrator evaluation [64]

Design force	**KPIs**
Availability	Acquisition sensitivity for each targeted GNSS signal, in C/N_0: minimum signal strength required for the receiver to perform acquisition and achieve a position fix.
	Tracking sensitivity for each targeted GNSS signal, in C/N_0: minimum signal strength required for the receiver to perform tracking and achieve a position fix.
	Time to First Fix (TTFF), in seconds: time required for a receiver to determine its PVT after being turned on.
Efficiency	Number of parallel channels that the software receiver can sustain in real time, given the targeted signals (GPS L1 C/A, Galileo E1b/c, GPS L5, Galileo E5a.
	Power consumption (in Watts)
Flexibility	Possibility to process signals either in real time or in post-processing time
	Possibility to easily define and interchange implementations and parameters for each processing block
	Availability of operation modes, as combinations of single and multiple frequency bands, and single or multiple constellations
Interoperability	Number of GNSS signals, defined as combinations of frequency band and channel or code, from which GNSS observables can be generated
	Sampling frequency
	Sample bit length and interpretation
Reliability	Availability of Receiver Autonomous Integrity Monitoring (RAIM) mechanisms
Precision	Stand-alone receiver's positioning precision.
Accuracy	Stand-alone dynamic position accuracy

The design forces and KPIs mentioned above are relevant for the continuous improvement and innovation within the field of satellite navigation. Further elaboration on the TTFF, as well as the precision and accuracy of the navigation solutions, can be found in the subsections below.

2.7.3 Time to First Fix (TTFF)

The TTFF indicates how quickly the device can start providing accurate location data after being powered up or after losing its previous lock on satellite signals. It can be specified in the following scenarios [64]:

- Cold start: This is when the GNSS receiver has no prior knowledge of its time, location, or the satellites' positions. It might occur when the device is turned on for the first time or has been inactive for a long time.
- Warm start: The receiver has access to the current time and the almanac, which provides the schedule and rough orbital information of the satellites. Additionally, it has a coarse

estimation of its position, accurate to within 100 km. However, it lacks the current ephemeris data, which is essential for precise satellite positioning.

- Hot start: The receiver has almost all the current information it needs, including the current time and almanac, a coarse estimation of its position, accurate to within 100 km, and the current ephemeris data.

2.7.4 Precision of the Navigation Solutions

The quality of the navigation solutions produced by the GNSS concept demonstrators developed in this thesis is assessed using standard positioning precision measurements and corresponding static confidence regions. Precision, which indicates how closely a solution aligns with the mean of all obtained solutions, reflects repeatability or the spread of the measurement. Key confidence measurements include Distance Root Mean Square (DRMS) and Circular Error Probability (CEP) for 2D positioning, as well as Spherical Accuracy Standard (SAS), Mean Radial Spherical Error (MRSE), and Spherical Error Probable (SEP) for 3D positioning. Formulas for 2D and 3D position confidence regions are provided in Tables 2.10 and 2.11, respectively [65, 66].

Table 2.10 Most common 2D precision measures.

Measure	**Formula**	**Confidence region probability**
2DRMS	$2\sqrt{\sigma_E^2 + \sigma_N^2}$	95 %
DRMS	$\sqrt{\sigma_E^2 + \sigma_N^2}$	65 %
CEP	$0.62\sigma_N + 0.56\sigma_E$ (accurate if $\frac{\sigma_N}{\sigma_E} > 0.3$)	50 %

Table 2.11 Most common 3D precision measures.

Measure	**Formula**	**Confidence region probability**
99 % SAS	$1.122(\sigma_E + \sigma_N + \sigma_U)$	99 %
90 % SAS	$0.833(\sigma_E + \sigma_N + \sigma_U)$	90 %
MRSE	$\sqrt{\sigma_E^2 + \sigma_N^2 + \sigma_U^2}$	61 %
SEP	$0.51(\sigma_E + \sigma_N + \sigma_U)$	50 %

These measurements involve converting the receiver's latitude, longitude, and height coordinates into a local East-North–Up (ENU) coordinate system, with the WGS-84 reference ellipsoid as the reference [67]. Standard deviations for the East (E), North (N), and Up (U) coordinates are computed. The standard deviation for East coordinates, denoted as $E[n]$, using the mean value $\overline{E}$ of the East coordinates and the number of position fixes, L, can be computed as

$$\sigma_E^{(precision)} = \sqrt{\frac{1}{L-1}\sum_{l=1}^{L}(E[n]-\overline{E})^2} \, . \qquad [\mathrm{m}] \tag{2.25}$$

The mean value of the East coordinates is calculated as

$$\overline{E} = \frac{1}{L}\sum_{l=1}^{L} E_l \, . \tag{2.26}$$

The standard deviation calculations for North (N) and Up (U) coordinates followed the same procedure as (2.25) and (2.26) for East coordinates.

2.7.5 Accuracy of the Navigation Solutions

Accuracy refers to how close the measured or calculated position of a GNSS receiver is to the true or actual position. Its measurement requires a reference position in the case of static positioning and a controlled mobile platform in the case of dynamic positioning [64]. The most common accuracy measurements are shown in tables 2.12 and 2.13. They are computed in a similar way as the 2D and 3D precision measurements shown in section 2.7.4, however the standard deviations are computed with respect to a reference location, as shown in

$$\sigma_E^{(accuracy)} = \sqrt{\frac{1}{L-1}\sum_{l=1}^{L}(E[n]-E_{ref})^2} \, , \qquad [\mathrm{m}] \tag{2.27}$$

for the East coordinates $E(n)$, where E_{ref} is the reference point, and L is the number of position fixes [65, 66].

Table 2.12 Most common 2D accuracy measures.

Measure	Formula	Confidence region probability
2DRMS	$2\sqrt{\sigma_E^2 + \sigma_N^2}$	95 %
DRMS	$\sqrt{\sigma_E^2 + \sigma_N^2}$	65 %
CEP	$0.62\sigma_N + 0.56\sigma_E$ (accurate if $\frac{\sigma_N}{\sigma_E} > 0.3$)	50 %

Table 2.13 Most common 3D accuracy measures.

Measure	Formula	Confidence region probability
99 % SAS	$1.122(\sigma_E + \sigma_N + \sigma_U)$	99 %
90 % SAS	$0.833(\sigma_E + \sigma_N + \sigma_U)$	90 %
MRSE	$\sqrt{\sigma_E^2 + \sigma_N^2 + \sigma_U^2}$	61 %
SEP	$0.51(\sigma_E + \sigma_N + \sigma_U)$	50 %

2.8 Review of Software-Defined GNSS Receivers

Advancements in software and hardware technology have significantly accelerated the development of software-defined GNSS receivers, as evidenced by an increasing volume of textbooks and publications on the topic. Textbook [68] focuses on GPS receiver theory and practice, while [26, 38] discuss the benefits of the SDR approach, providing Matlab implementations for a complete GPS receiver. More recently, ref. [69] introduces the Beidou and GPS dual-system software receiver algorithms, and ref. [70] details the construction and operation of multi-GNSS and multi-frequency software receivers using advanced techniques. Tutorials and detailed discussions on software GNSS receiver architectures are available in several publications [71–73]. Ref. [74] reviews the history of GNSS SDR development over the last decade. It highlights key public SDR implementations and contributions to the field, discusses the standardization of intermediate-frequency sample data and metadata, and suggests updates to the SDR Standard by the Institute of Navigation (ION).

Numerous publications have documented the design, implementation, and preliminary performance evaluation of software-defined GNSS receivers. These systems vary in both their intended use and the underlying technology. Many are implemented entirely in software, without any programmable logic, and operate on PCs or other computing devices [16, 75–80]. Alternatively, some software receivers leverage the parallel computing capabilities of GPUs to enhance performance, as demonstrated by [81, 82]. Additionally, software libraries facilitating the development of software-defined GNSS receivers have been introduced by [83]. While these systems are highly flexible and scalable, purely software-based implementations are usually not energy efficient. GNSS receivers implementing computationally expensive algorithms may not be suited for battery-powered embedded devices.

Some research focuses on developing FPGA-based platforms for rapid prototyping of GNSS receiver algorithms, as seen in [84–88], or on platforms that combine FPGAs with DSPs [89]. While these platforms offer significant flexibility, they often require large, power-intensive devices that lack portability. Conversely, other studies introduce FPGA-based solutions tailored for specific uses, such as GNSS applications in space [90, 91]), safety-of-life GNSS receivers [92], multi-antenna GNSS receivers [93], and ASIC designs [94]. These implementations are highly optimized for their respective applications, offering less versatility as general-purpose platforms for experimenting with a wide range of GNSS receiver algorithms.

Some publications address the trade-off between efficiency and flexibility by leveraging multi-core parallelism, aiming to reduce power consumption. Ref. [95] proposes a multi-core GNSS baseband processing architecture that features numerous processor cores operating at lower clock frequencies. Similarly, [96] introduces a multi-core architecture with FPGA support, where the processing cores are implemented within FPGA logic. This approach of using processor cores embedded in FPGA logic, known as soft processor cores, offers the benefit of full customizability and inclusion of only essential features. However, compared to hard processor cores fabricated directly into silicon—like those in SoC FPGAs—soft processor cores tend to be slower and less power-efficient. Additionally, an Open Computing Language (OpenCL)-based implementation suggested by [97] seeks to find a middle ground between flexibility and efficiency, by distributing various signal processing tasks across different hardware resources (PC, GPU, FPGA).

More recently, a variety of publications have introduced SoC-FPGA-based designs tailored for specific GNSS applications. These designs leverage the extensive parallelism and energy efficiency of FPGAs alongside the versatility of embedded processors. For instance, [98] demonstrates a SoC-FPGA-based receiver's capability to acquire and track GPS L1 C/A satellites using recorded signals. Ref. [99] advances this approach with a dual-frequency receiver designed to support the GPS L2 and L5 bands, enhancing tracking capabilities following the acquisition of L1 C/A signals. Furthermore, [100] explores a receiver designed for reflectometry applications, showcasing the adaptability of SoC-FPGA-based systems to specialized GNSS tasks. In these implementations, the FPGA component primarily handles the most computationally intensive operations, allowing the SoC processor to focus on calculating basic GNSS measurements and navigation solutions.

The materials referenced span a variety of platforms, including CPUs, GPUs, FPGAs, DSPs, and SoC-FPGAs, utilized either individually or in combination to achieve optimal performance. CPU-based systems often run on PCs or embedded processors, showcasing the versatility of CPU applications. There are also devices that integrate CPUs with FPGAs, functioning on both PCs and embedded processors, some of which utilize soft processors implemented within FPGA logic for enhanced customization. Table 2.14 organizes these references by processing units and publication dates, compiled from a basic search targeting the most relevant publications. Initially, software-based implementations predominantly relied on CPUs. However, there seems to be a tendency towards heterogeneous platforms, like SoC-FPGAs, which combine various processors and co-processors within a single system. This trend underscores a strategic move to optimize system performance by delegating specific tasks to the most appropriate processing units, thereby enhancing efficiency and adaptability.

Table 2.14 Table comparing related work based on publication date and underlying processing units .

Year	CPU	CPU, GPU	FPGA, DSP	FPGA	CPU, FPGA	CPU, FPGA, GPU	SoC FPGA
1997	[75]						
2003	[76]						
2004	[83]				[84]		
2006			[89]				
2008		[81]					
2009	[77]						
2010	[78]	[82]	[90]				
2011	[16]				[91, 94, 95]		
2012			[92, 93]				
2013					[96]		
2014					[85, 86]		
2015	[79]						
2018	[80]						
2019					[87]		[98]
2020						[97]	
2021							[99]
2022				[88]			[100]

The research articulated in this dissertation is distinguished by a unique combination of features not previously brought together in this field:

- A flexible SoC-FPGA-based GNSS receiver architecture aiming to enhance the balance between flexibility and energy efficiency.
- A generic design methodology for the development of portable, experimental GNSS receivers, intended for research purposes that demand non-standard features. The proposed methodology facilitates code reuse and allows for the development of GNSS receivers across a range of research applications.
- The integration of a SoC FPGA architecture with GNSS-SDR, a popular FOSS-based software-defined receiver. This approach facilitates code reuse and simplifies the implementation of modifications within the signal processing path. In this way, researchers can save time by avoiding repetitive problem-solving and focus immediately on developing the specific features they require. The software running in the embedded processor is portable to various processor architectures, including the Intel x86-64 [49] and the ARM architectures [50]. Therefore, novel algorithms can be first tested in software, followed by an implementation in an ARM-based SoC FPGA.
- Flexibility in the licensing types for FPGA hardware accelerators: In the proposed designs, FPGA IP cores are not restricted to FOSS licenses, providing opportunities to

monetize research while enhancing research impact and reputation. While the FPGA IP cores developed in this thesis are under proprietary licenses—limiting public access to their HDL source code—the underlying signal processing algorithms remain accessible. They can be examined through their software implementation in GNSS-SDR and detailed in the documentation of the cores.

2.9 Summary

This chapter established the foundational context for this thesis, providing the essential knowledge crucial to the work presented. It introduced GNSS systems and the fundamentals of GNSS receivers, including an overview of the widely recognized and publicly available FOSS GNSS-SDR software receiver. This software is a critical component of the proposed SoC FPGA receiver architecture.

Additionally, the chapter delved into FPGA and SoC FPGA technology, highlighting their main features and design flows, thus setting the stage for their application within this research. FPGAs enhance the power efficiency of software-defined GNSS receivers by enabling parallel processing of receiver channels within the FPGA. Additionally, they enable the development of a flexible architecture, allowing for updates and modifications without the need for hardware redesign. However, FPGA design presents challenges due to the complexity of design tools and the necessity of using HDL languages, which demand a thorough understanding of parallel computing, timing, and hardware behavior. These factors contribute to longer development times.

After exploring GNSS and FPGA technologies, this chapter introduced the design forces and KPIs that are pivotal in subsequent chapters for discussing the proposed receiver architecture and methodology, as well as for evaluating the concept demonstrators.

Finally, the state of the art of existing software-defined GNSS receiver implementations is reviewed, showing a trend towards the use of heterogeneous platforms that combine different types of processing units to optimize performance and assign signal processing tasks to the most suitable units.

The literature review presented in this chapter has been published in:

- [20] M. Majoral, C. Fernández-Prades, and J. Arribas, "A Flexible System-on-Chip Field-Programmable Gate Array Architecture for Prototyping Experimental Global Navigation Satellite System Receivers," *Sensors*, vol. 23, no. 23, 2023, Art. no. 9483. doi: 10.3390/s23239483

Chapter 3

System Design and Methodology

This chapter elaborates on the proposed GNSS receiver architecture and design methodology, beginning with a design overview that highlights the architecture's components, including the RFFE, SoC FPGA, and communication interfaces. The discussion narrows down to the FPGA architecture, detailing how it processes digitized signals from the RFFE through sample conditioning, buffering, and hardware accelerators for the acquisition and tracking multicorrelator algorithms. Additionally, the software architecture is examined, particularly how the GNSS-SDR software receiver delegates resource-intensive tasks to the FPGA.

Next, the design methodology is outlined. The prototyping of SoC-FPGAs-based systems incorporates both software and hardware design aspects. This process entails programming the GNSS-SDR signal processing blocks (software design) and developing FPGA hardware accelerators (hardware design). A separate software and hardware design flow is adopted to enable the concurrent development of the software receiver and the FPGA hardware accelerators. Additionally, this approach facilitates the development of the software receiver using a FOSS toolchain, which operates independently of the FPGA design tools.

The proposed design methodology incorporates several key components introduced in Section 2.4.4: the Yocto Project framework and the OpenEmbedded build system [51], along with various tools including the Geniux GNU/Linux customization [55], the meta-gnss-sdr Yocto layer [52], the oe-gnss-sdr-manifest [53], and the Docker images [56]. While the design and development of the SoC FPGA architecture and methodology are central to this thesis, the development of the aforementioned tools — specifically, the Geniux customization, the meta-gnss-sdr layer, the repo manifest, and the Docker images — falls outside its scope.

Finally, the proposed architecture is discussed in terms of several design forces, including scalability, testability, portability, maintainability, reproducibility, and openness.

3.1 Proposed GNSS receiver architecture

The proposed architecture is based on SoC FPGAs, strategically distributing the signal processing tasks between the FPGA and the SoC's integrated processor. This distribution is carefully planned, considering the processing units' capabilities, the specific demands of computing tasks, and the algorithms' frequent update requirements. By doing so, the architecture effectively leverages the distinct advantages of both FPGAs and embedded processors, creating a synergistic effect that enhances overall performance. Additionally, it streamlines the implementation process, ensuring a more efficient development timeline.

To optimize performance and minimize power usage, it's essential to allocate tasks judiciously according to their inherent characteristics. Tasks that leverage parallel processing, requiring high throughput and minimal latency, are ideally assigned to the FPGA. In contrast, tasks requiring less throughput, and characterized by complex, sequential processing and decision-making branches, which challenge parallelization, are more aptly handled by the embedded processor. Simultaneously, from the perspective of minimizing development time, tasks that undergo frequent changes and require experimentation are more suitably implemented on the embedded processor. Conversely, specialized tasks that do not need frequent updates are ideally placed on the FPGA.

In this two-dimensional scenario, tasks that do not clearly fall into the established categories should be assigned to either the FPGA or the embedded processor, after carefully weighing their respective advantages and disadvantages. This decision is taken at design time. The GNSS-SDR software receiver can be utilized to profile these tasks and determine the optimal hardware-software partitioning.

In GNSS receivers, the tasks that demand the most computational effort are those involved in processing the received signals at the baseband sampling rate. Primarily, these tasks occur during the initial stages of the software-defined signal processing chain, namely, the acquisition and tracking of multi-correlator processes. These critical tasks are focused on the relentless correlation of the incoming signals with the local copies of the PRN codes, essential for the accurate detection and tracking of signals. Unlike other stages, these initial tasks do not involve complex decision-making processes, highlighting their specialized, computationally intensive nature without the need for intricate branching decisions.

The remaining parts of the signal processing chain (the control of the tracking loops, the telemetry decoding, the computation of the observables, and the PVT) typically process the received signals at the symbol rate or a lower rate. These tasks are also the most complex ones in terms of decision branching and on top of that they require frequent updates, when testing new algorithms for research purposes.

Considering these facts, the FPGA is tasked with sample conditioning and buffering, as well as with the acquisition and tracking multicorrelator processes, capitalizing on its ability for massive parallel processing and energy efficiency. Conversely, the embedded processor manages the control of tracking loops, telemetry decoding, computation of the observables, and PVT calculations, taking advantage of its quicker development cycles and simpler architecture, which are crucial for facilitating research and development of new GNSS receiver algorithms.

3.1.1 SoC FPGA Architecture

As outlined in Chapter 2, a SoC FPGA combines programmable logic with a processing system, which includes various peripheral interfaces. This is depicted in Figure 3.1. The peripheral interfaces include Universal Asynchronous Receiver / Transmitter (UART) and Ethernet ports for connecting external devices. This setup enables the implementation of an embedded receiver that can disseminate GNSS data to remote devices using universally recognized formats like RINEX, RTCM (with adjustable rates), and NMEA-0183 messages, ensuring compatibility and ease of integration with other systems. The support for these output formats is detailed in Table 2.7 in Chapter 2. Both Ethernet and UART interfaces offer versatile communication options, with Ethernet providing the added advantages of enabling receiver remote control and the transmission of signal snapshots to an off-site computer for further analysis. Signal snapshots are discrete segments of the received GNSS signals captured at specific points in time.

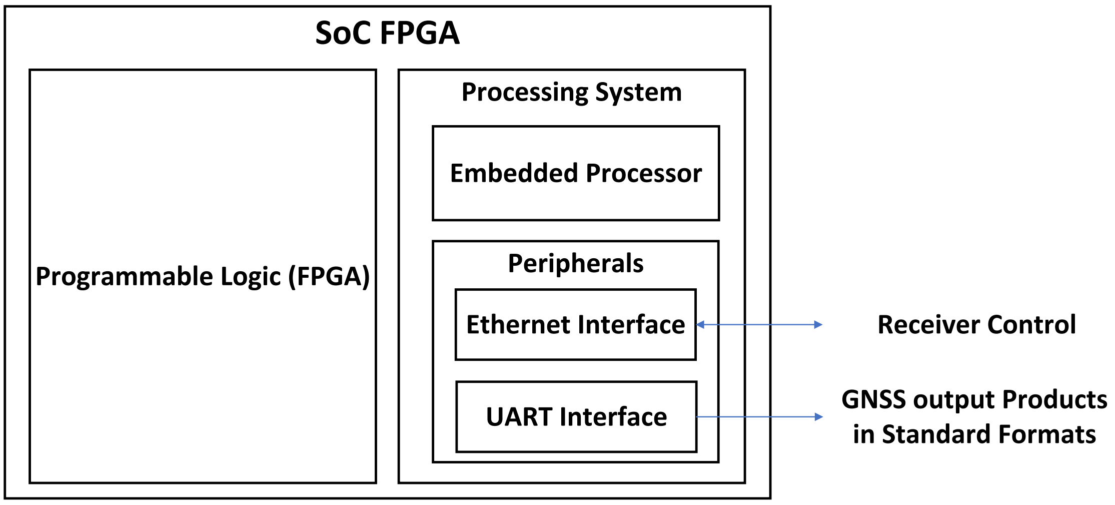

Figure 3.1 SoC-FPGA Receiver block diagram.

Figure 3.2 shows a block diagram of the GNSS receiver architecture. The diagram highlights the main components of the GNSS receiver: the RFFE and the SoC-FPGA. The illustration also depicts the FPGA logic and the Embedded Processor, alongside the primary functional blocks of the GNSS receiver within both components.

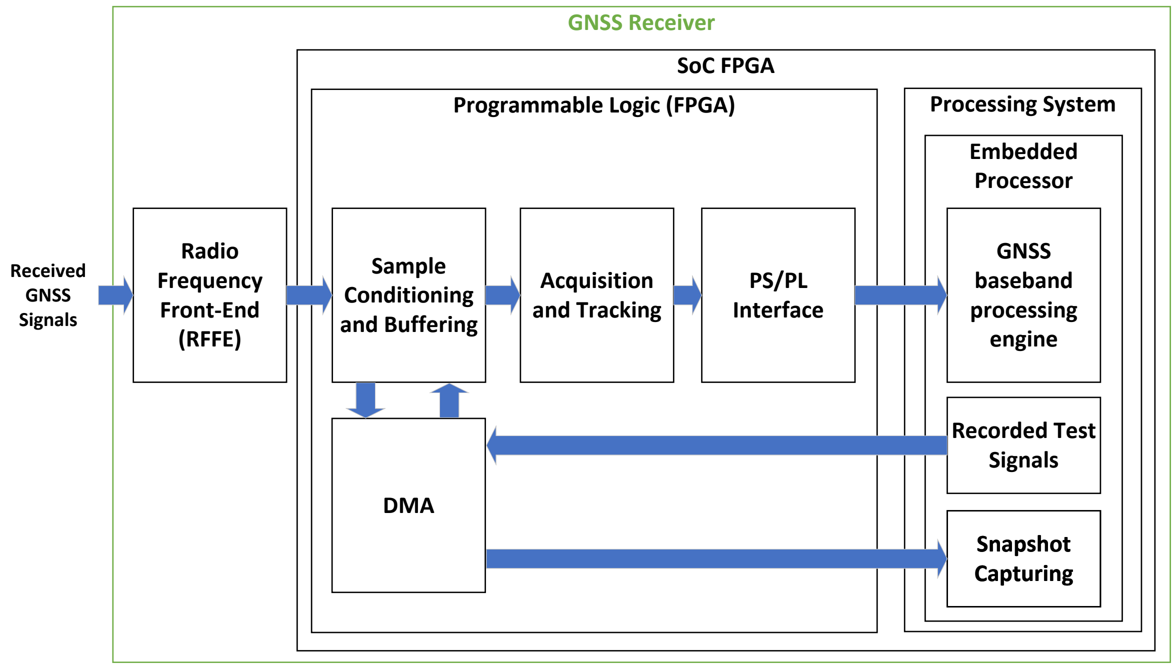

Figure 3.2 Receiver architecture.

The RFFE in Figure 3.2 is tuned to the desired GNSS frequency bands, performs direct RF to baseband conversion, and digitizes the received signals. Utilizing multi-frequency RFFEs enables the implementation of GNSS receivers that support multiple frequencies.

The FPGA Logic comprises four functional blocks, as illustrated in Figure 3.2: sample conditioning and buffering, acquisition and tracking, Direct Memory Access (DMA), and PS/PL interface. These blocks are briefly explained below:

1. Sample Conditioning and Buffering: The sample conditioning and buffering block receives the samples coming from the RFFE and implements sample buffering and clock conversion between the front-end interface and the FPGA hardware accelerators. The sample conditioning and buffering block also implements a bit selector, used to dynamically requantize the GNSS signals to map the dynamic range of the incoming samples to the dynamic range of the acquisition and tracking hardware multicorrelators. The bit selector dynamically selects the most significant bits of the digitized signal based on the received signal power, as explained in more detail in Section 3.1.3.

2. Acquisition and Tracking: The FPGA incorporates hardware accelerators for the algorithms with the highest computational cost: the acquisition and tracking multicorrelators [101]. The most computationally expensive algorithms are the signal processing stages that process the digitized signals at the sampling rate. However, it is also possible to offload any other processor-intensive algorithms to the FPGA. The acquisition and tracking hardware multicorrelators are explained in more detail in Sections 3.1.4 and 3.1.5 respectively.

3. DMA: the FPGA implements a bi-directional direct memory access. The DMA can be used to run the receiver in post-processing mode using recorded GNSS files, to record

the received GNSS signals into files, to capture snapshots, or to send the received GNSS signals to an external device in real-time.

4. PS/PL Interface: Interface between the FPGA and the processing system. This interface is implemented using the AMBA Advanced eXtensible Interface 4th generation (AXI4) memory-mapped bus. The GNSS baseband engine controls the execution of the acquisition and tracking multicorrelators using a set of memory-mapped registers, and a set of interrupts going from the FPGA to the processing system.

The processor shown in Figure 3.2 runs GNSS-SDR on a customized GNU/Linux OS. The processor cores are configured for Symmetric Multi-Processing (SMP). GNSS-SDR implements the baseband GNSS processing engine. GNSS-SDR has an option to offload the most computationally demanding tasks to the FPGA. This option can be used when cross-compiled for execution in embedded processors, enabling the execution of the GNSS-SDR in real-time using portable devices.

When working in post-processing mode, the receiver processes recorded GNSS signals. The recorded GNSS signals are usually stored in a non-volatile device such as Secure Digital (SD) card. The embedded processor transfers the recorded signals from the SD card to the system's DDR memory, and programs the DMA in the FPGA to simultaneously transfer the signals from memory to the hardware accelerators in the FPGA. Transferring the recorded signals directly from the SD card to the hardware accelerators using only the FPGA would increase efficiency. However, this approach would lead to a more complex implementation, necessitating the development of a dedicated controller within the FPGA. The OS running in the embedded processor has proper drivers to enable easy access to external system components such as the SD card, facilitating the use of the embedded processor for transferring the samples from the SD card to the system's memory. For this reason, the current implementation uses both the DMA and the embedded processor to read the recorded signals. The involvement of the PS in both reading samples from the recorded signals and in the GNSS signal processing algorithms hinders real-time processing speeds of the recorded signals. However, that is not a problem, as real-time operation is usually not required when processing recorded signals, and the processing speed adapts to the system capabilities.

The process of capturing snapshots also involves the use of DMA and the embedded processor. The utilization of the embedded processor leverages the OS Transmission Control Protocol/Internet Protocol Transmission Control Protocol/Internet Protocol (TCP/IP) software stack, facilitating the utilization of the Ethernet interface for sending the captured snapshots from the system's memory to a remote computer (Snapshot Capturing in Figure 3.2). The sample transfer fully occupies the processor, rendering simultaneous real-time operation with GNSS-SDR unfeasible during the transfer.

The proposed architecture has been implemented and tested using the AMD's Zynq 7000 All Programmable SoC [102] and the Zynq Ultrascale+ Multi Processor System-on-Chip (MPSoC) [63] families, demonstrating the flexibility and the scalability of the design. Various types of GNSS receivers can be implemented on a wide range of SoC-FPGAs, starting from a Zynq 7000 SoCs featuring a single-core Cortex A9 ARM processor, 23 k logic cells and 66 DSP slices [103], and up to a Zynq Ultrascale+ MPSoC with quad-core ARM Cortex-A53 processors, 1143 k logic cells and 2520 DSP slices [104]. The scalability of the Zynq 7000 and the Zynq Ultrascale+ devices enables the implementation of a wide range of GNSS receivers, starting from low-power and small form-factor single-frequency and single-constellation devices and up to multi-frequency, multi-constellation, high-performance receivers implementing highly complex algorithms.

The subsections below provide a more detailed description of the FPGA architecture, the acquisition and tracking multicorrelator hardware accelerators, and the software design.

3.1.2 FPGA Architecture

Figure 3.3 shows a schematic representation of the FPGA design, providing a comprehensive view of the functional blocks outlined in Figure 3.2: the sample conditioning and buffering, the acquisition and tracking, the PS/PL interface, and the DMA. The block diagram in Figure 3.3 showcases the implementation of a multi-frequency multi-constellation GNSS receiver capable of processing GPS L1 C/A, Galileo E1b/c, GPS L5 and Galileo E5a signals.

In Figure 3.3, a dual-channel RFFE is tuned to the L1/E1 and L5/E5a frequency bands. The RFFE performs RF to baseband conversion, implements AGC, digitizes the received signal using ADCs, and forwards the digital samples to the FPGA. The FPGA performs signal conditioning and buffering of the received samples. A separate sample buffer is used for each frequency band (labeled as L1/E1 Buffer, and L5/E5a Buffer). The sample buffers are followed by several multicorrelator hardware accelerators. Each tracking multicorrelator is preceeded by a small sample buffer (labeled as Channel Buffer). In this example, the GNSS receiver implements 24 multicorrelator hardware accelerators for each frequency band, and therefore the receiver can potentially track up to 48 GNSS signals simultaneously.

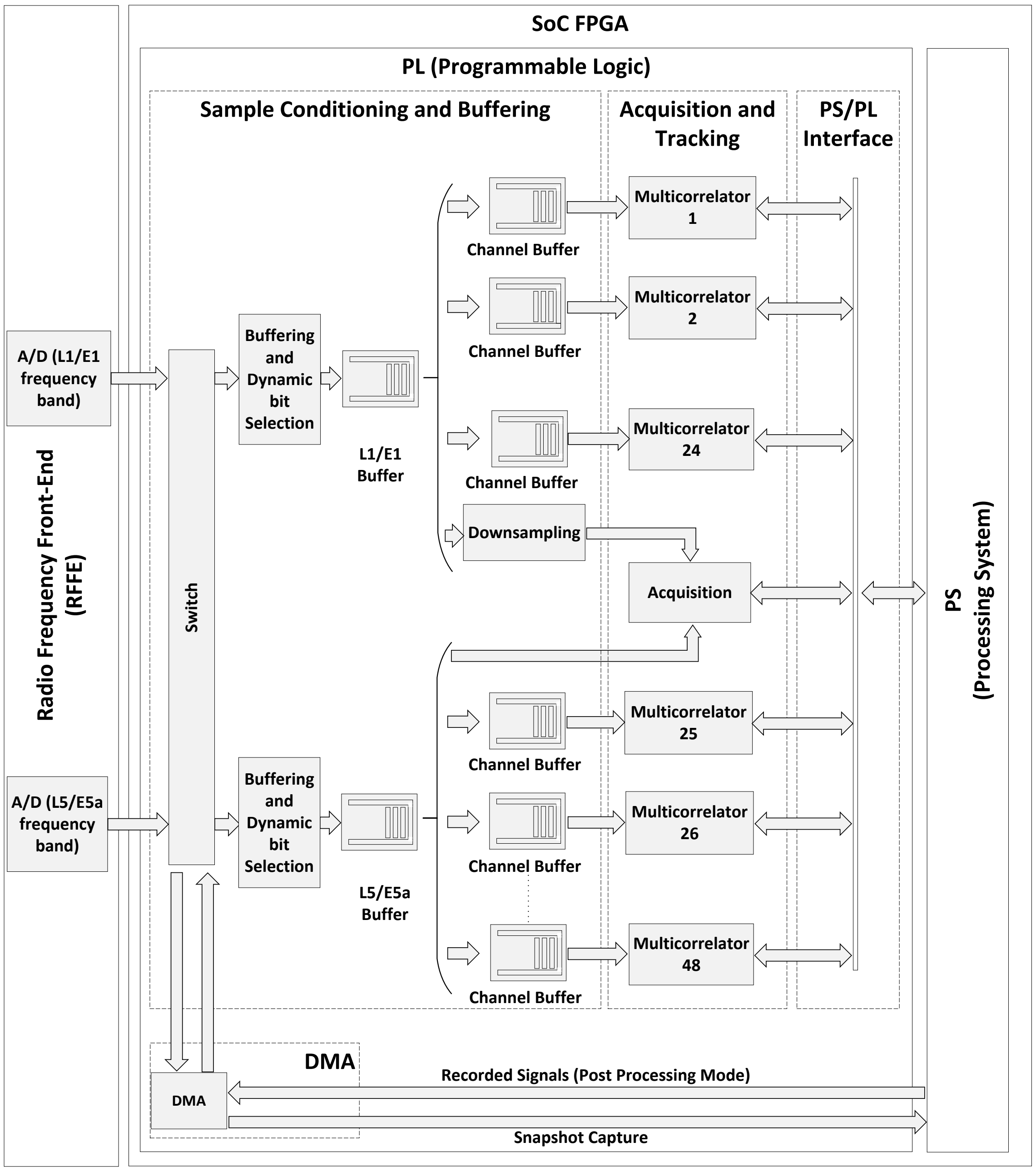

Figure 3.3 Detailed FPGA design.

The L1/E1 and the L5/E5a buffers also deliver the received samples to the acquisition hardware accelerator. The input of the acquisition hardware accelerator in the L1/E1 band is preceded by a downsampling filter. By placing the downsampling filter in front of the acquisition, the receiver employs a reduced sampling frequency for the acquisition of the GNSS signals in the E1/L1 frequency band, and subsequently, a higher sampling frequency for tracking the same signals. This arrangement aligns with the fact that the signal detection is not significantly benefited from operating within a bandwidth exceeding that necessary to capture the main lobe of the received signals, which is approximately 2 MHz for GPS L1 C/A signals and 4 MHz for Galileo E1b/c signals, due to the increased noise in the captured signal [105]. However, using a large bandwidth for tracking GNSS signals facilitates a more precise determination of the carrier and code phases of the received signals. Using a reduced sampling frequency for the acquisition of GNSS signals in the E1/L1 frequency band also speeds up the acquisition process.

The FPGA also implements the bi-directional DMA engine. The DMA engine can be used to transfer recorded GNSS signals from the processing system memory to the L1/E1 and the L5/E5a buffers and in this way run the receiver in post-processing mode. The DMA can also be used to capture snapshots from the RFFE and transfer them to the processing system's memory.

The PS/PL interface in Figure 3.3 implements an efficient mechanism to enable, disable, configure, and exchange information between the processing system and the hardware accelerators in the FPGA. Each hardware accelerator presents a series of accessible read/write registers via an AXI4 bus connection with the processing system. In order to reduce the computational load, individual interrupt signals are directed from each hardware accelerator to the processing system. The interrupts trigger acquisition and tracking channels callbacks in the processing system to read the results and to reload the acquisition and tracking parameters for processing the next batch of samples. The processing system controls the tracking loops, cycling through the following steps:

1. The software running in the processing system configures the multicorrelator hardware accelerators with updated parameters: coherent integration time, Doppler frequency correction, etc.

2. The multicorrelator hardware accelerator captures a new batch of samples and processes the samples on the fly.

3. The multicorrelator hardware accelerator finishes processing the received samples.

4. The multicorrelator hardware accelerator interrupts the processing system and waits for the multicorrelation results to be read. The software running in the processing system reads those results and operates the tracking loop.

The sample buffers are essential for temporarily storing incoming samples while the multicorrelator hardware accelerators are awaiting the processing system's response. This necessity arises due to the embedded OS's response time to hardware interrupts. The processing system runs GNU/Linux. In this way, GNSS-SDR can be easily cross-compiled and used in the embedded platform. However, GNU/Linux is not a real-time operating system. For this reason, the processing system may not immediately react to the interrupts coming from the tracking multicorrelators. When the processing system does not promptly respond to the interrupts, the multicorrelators that are in a waiting state block the flow of samples, exerting back pressure on the channel buffers. During normal operation, when processing GNSS signals in real-time, the channel buffers are never completely filled in. However, should a temporary CPU overload occur and result in channel buffer overflow, the L1/E1 and the L5/E5a buffers will store the incoming samples, reducing the probability of sample loss. The L1/E1 buffer and the L5/E5a buffer are identical. Having a separate buffer for each frequency band reduces the probability of sample loss when a temporary channel buffer overflow occurs, as the channel buffer temporarily obstructs the flow of samples in one frequency band only.

The FPGA hardware accelerators are implemented as soft IP cores enabling portability across many AMD Zynq 7000 All Programmable SoC and Zynq Ultrascale+ MPSoC variants [63, 102].

The subsections below explain the dynamic bit selection process, and the acquisition and tracking multicorrelator hardware accelerators in more detail.

3.1.3 Dynamic Bit Selection

The dynamic bit selector selects the most significant bits in the digitized signal based on the incoming signal's power. Its necessity stems from the mismatch between the number of bits per sample provided by most commercial ADCs and the quantization depth required for processing GNSS signals in the acquisition and tracking multicorrelator hardware accelerators.

Most commercial ADCs not specifically designed for GNSS receivers typically utilize a large number of bits per sample. For instance, the AD9361 transceiver—used in the concept demonstrators presented in Chapters 4, 5, and 6—uses 12 bits per sample [106]. However, as detailed in Section 2.3.3, GNSS receivers without antijamming capabilities do not significantly benefit from a high bit depth. In line with this, the acquisition and tracking multicorrelator hardware accelerators process GNSS signals using a selectable small bit quantization depth (2 or 4 bits per sample), as determined by the user.

Even with the RFFE implementing AGC, a fixed selection of bits from the samples delivered by the ADC may not lead to optimal bit usage, due to the discrepancy between the ADC's output bit depth and the bit depth utilized by the acquisition and tracking hardware accelerators.

The dynamic bit selection process in the FPGA dynamically maps the input of the acquisition and tracking hardware accelerators to the most significant bits of the samples coming from the ADC, minimizing quantization loss.

3.1.4 Acquisition Hardware Accelerator

The purpose of the acquisition hardware accelerator is to detect GNSS signals coming from the visible satellites and to estimate the code phase and the carrier frequency of received signals. The acquisition implements the PCPS algorithm, which is detailed in Section 2.3.4. The PCPS algorithm performs a step-by-step search for the Doppler frequency, while concurrently parallelizing the code phase search, resulting in reduced signal acquisition time and enhanced GNSS receiver performance. Table 3.1 serves as a guide for the notation and definitions utilized in describing the FPGA implementation of the PCPS acquisition algorithm.

Table 3.1 Notation table for the acquisition algorithm description.

Variable	Definition
f_{min}	Minimum tested Doppler frequency
f_{max}	Maximum tested Doppler frequency
$[f_{min}, f_{max}]$	Doppler frequency span
$f_{d_{test}}$	Tested Doppler frequency
f_{step}	Doppler search step
$x_{IN}[n]$	Received GNSS signal input sample stream
T_s	Sampling period
$D[K]$	FFT of the PRN code
$\|R_{xd}(f,\tau)\|$	Cross Ambiguity Function (CAF)
f_d	Estimated Doppler frequency
τ	Estimated code phase

The implementation of the PCPS algorithm in the FPGA is described in Algorithm 1 and shown in Figure 3.4. The first step of the acquisition algorithm is to buffer the received samples. Then, for each tested Doppler frequency, the acquisition performs the Doppler wipe-off using a local carrier generated with an NCO and computes the magnitude of the circular cross-correlation between the received signal and a local replica of the satellite's PRN code. The circular cross-correlation is computed in the frequency domain. At the end of this process, the acquisition obtains the CAF, a two-dimensional function of the code delay and Doppler frequency. The presence of a signal results in a significant peak detected in the CAF. The acquisition uses a first peak vs. second peak statistic to assert the presence or absence of a satellite signal, searching for the second peak value in the same frequency bin of the highest peak, as proposed in [38]. Comparison against a predefined Probability of False Alarm (PFA) is not implemented.

The acquisition hardware accelerator runs the complete acquisition algorithm (Algorithm 1) with no processor intervention. The FPGA uses a power-of-two Fast Fourier Transform (FFT), with a configurable FFT size. The acquisition hardware accelerator performs zero padding when the combination of sampling frequency and length of the PRN codes results in the need to compute a FFT that is not a power of two. The result is equivalent to using an arbitrary length Discrete Fourier Transform (DFT), but using a more efficient calculation. The CAF computation is carried out in parallel to the search for the peak value. The acquisition hardware accelerator detects GPS L1 C/A, Galileo E1b+c, GPS L5, and Galileo E5a signals, and it can acquire either the data or the pilot components of the GNSS signals. The current implementation of the acquisition can be configured to use two or four bits per sample.

Algorithm 1 Acquisition hardware accelerator

```
 1: Buffer the received samples $x_{IN}[n]$
 2: for $f_{d_{test}} = f_{min}$ to $f_{d_{test}} = f_{max}$ in $f_{step}$
 3:     Perform Doppler wipe-off: $x[n] = x_{IN}[n] \cdot e^{-j2\pi f_{d_{test}} n T_s}$, for $n = 0, \dots, N-1$
 4:     Compute $X[K] = FFT_N(x[n])$
 5:     Compute $Y[K] = X[K] \cdot D[K]$, for $k = 0, \dots, N-1$
 6:     Compute $R_{xd}(f_{d_{test}}, \tau) = \frac{1}{N^2} IFFT_N(Y[k])$
 7: end for
 8: Search the peak value and its indices in the search grid: $\{S_{max}, f_i, \tau_j\} = max_{f,\tau}|R_{xd}(f,\tau)|^2$
 9: Search the second peak value $S_{max_2}$ in the same frequency bin of the highest peak
    $|R_{xd}(f_i,\tau)|^2$, for $\tau = 0, \dots, N-1$
10: Compute test statistic $\Gamma = \frac{S_{max}}{S_{max_2}}$
11: Compare with threshold value: If $\Gamma > \gamma$
12:     Declare positive acquisition and provide $f_d = f_i$ and $\tau = \tau_j$
13: else
14:     Declare negative acquisition
15: end if
```

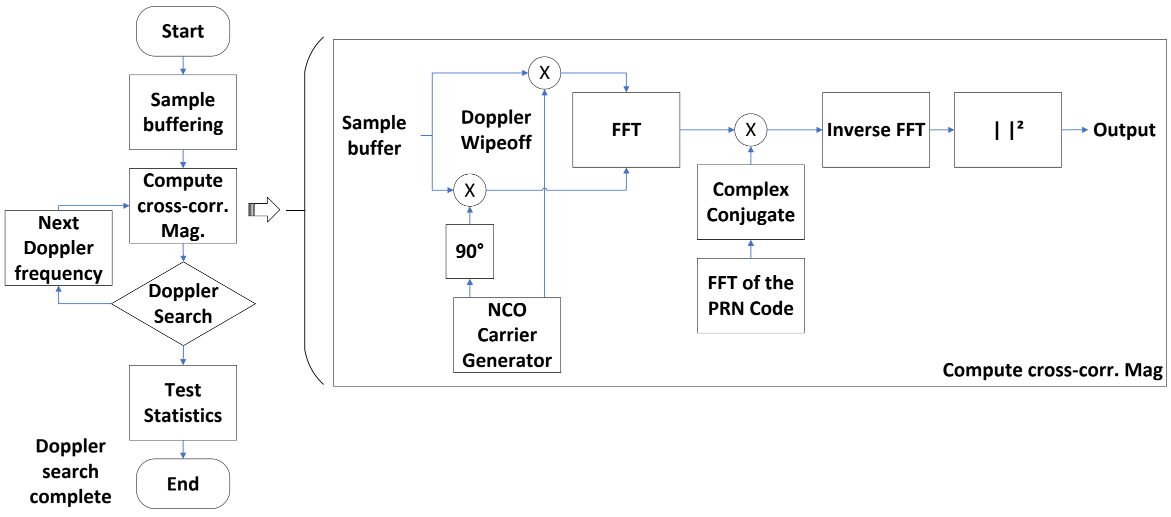

Figure 3.4 FPGA acquisition hardware accelerator.

3.1.5 Tracking Multicorrelator Hardware Accelerators

The tracking multicorrelator hardware accelerator computes the correlation of the incoming signal with various local replicas of the satellite's PRN code, $R_{xd}(\tau)$. The FPGA also wipes off the PRN secondary code. The tracking multicorrelator hardware accelerator implements the NCO, the generation of the PRN code replica, and the I&D blocks shown in Figure 2.7 in Chapter 2.

The software implements the tracking loop, which uses the correlation results obtained in the FPGA to synchronize the local replicas of the PRN code to the incoming signal and to follow the evolution of the signal synchronization parameters as accurately as possible. Table 3.2 serves as a guide for the notation and definitions utilized in describing the FPGA implementation of the tracking multicorrelator algorithm.

Table 3.2 Notation table for the tracking multicorrelator algorithm description.

Variable	Definition
N	Number of samples indicating coherent integration time.
f_d	Estimated Doppler frequency
$x_{IN}[n]$	Received GNSS signal input sample stream
T_s	Sampling period
$c[n]$	PRN code
$s[n]$	Secondary code
$R_{xd}(\tau)$	Correlation of the incoming signal with the PRN code
C_{VE}	Very Early correlator output
C_E	Early correlator output
C_P	Prompt correlator output
C_L	Late correlator output
C_{VL}	Very Late correlator output

The implementation of the tracking multicorrelator algorithm is shown in Figure 3.5 and described in Algorithm 2. The FPGA performs the Doppler wipe-off using a local carrier generated by an NCO, and multiplies the incoming signal with various code replicas having configurable spacing between them. As explained in Section 2.3.5, these code replicas are named VE, E, P, L, and VL [39]. The tracking multicorrelator algorithm implements complex correlators, separately integrating and dumping the I and Q components of the multiplication of the incoming signal with the code replicas.

The tracking loops use the multicorrelation results to estimate the derivative $\frac{dR_{xd(\tau)}}{d\tau}$ zero-crossing, which is used as a timing error detector [25].

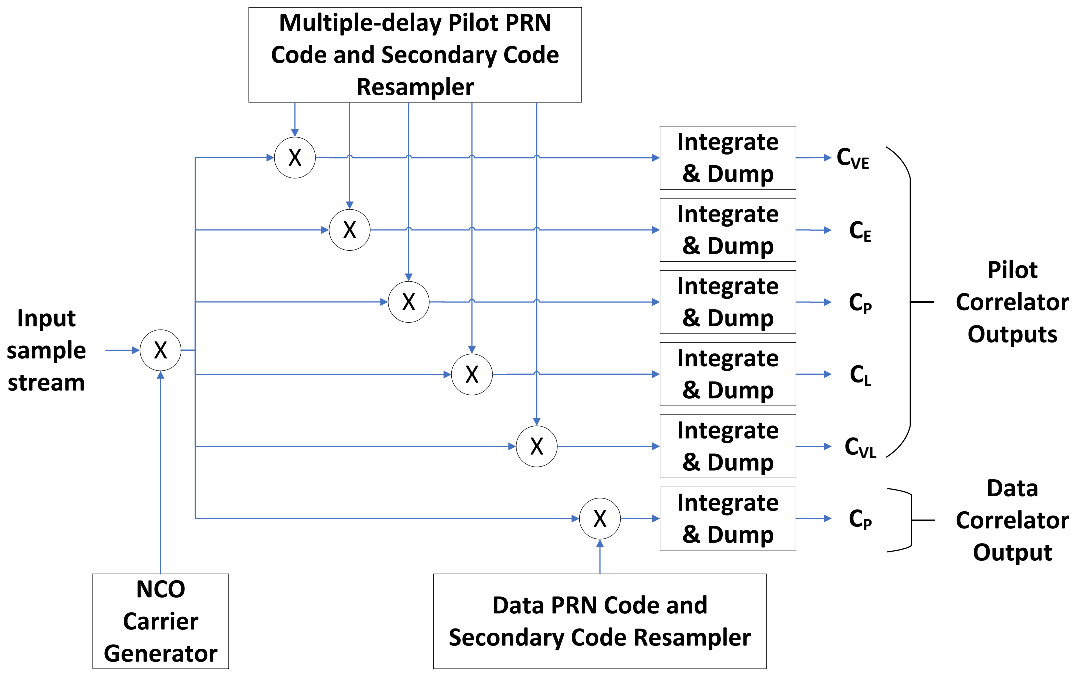

Figure 3.5 FPGA tracking multicorrelator hardware accelerator.

Algorithm 2 Tracking multicorrelator hardware accelerator

```
1:  Initialize the Pilot correlator outputs: $C_{VE_{Pilot}} = 0$, $C_{E_{Pilot}} = 0$, $C_{P_{Pilot}} = 0$, $C_{L_{Pilot}} = 0$, and $C_{VL_{Pilot}} = 0$
2:  Initialize the Data correlator output: $P_{Data} = 0$
3:  for $n = 0$ to $n = N - 1$
4:      Perform Doppler wipe-off: $x[n] = x_{IN}[n] \cdot e^{-j2\pi f_d n T_s}$
5:      For each pilot correlator, determine the following data point of the pilot's PRN sequence: $c[n]^{VE}_{Pilot}, c[n]^{E}_{Pilot}, c[n]^{P}_{Pilot}, c[n]^{L}_{Pilot}, c[n]^{VL}_{Pilot}$
6:      For each pilot correlator, determine the following data point of the pilot's secondary code sequence: $s[n]^{VE}_{Pilot}, s[n]^{E}_{Pilot}, s[n]^{P}_{Pilot}, s[n]^{L}_{Pilot}, s[n]^{VL}_{Pilot}$
7:      Determine the following data point of the data's PRN sequence: $c[n]^{P}_{Data}$
8:      Determine the following data point of the data's secondary code sequence: $s[n]^{P}_{Data}$
9:      Accumulate the pilot's correlator outputs:
10:         $C_{VE_{Pilot}} = C_{VE_{Pilot}} + x[n]c[n]^{VE}_{Pilot}s[n]^{VE}_{Pilot}$
11:         $C_{E_{Pilot}} = C_{E_{Pilot}} + x[n]c[n]^{E}_{Pilot}s[n]^{E}_{Pilot}$
12:         $C_{P_{Pilot}} = C_{P_{Pilot}} + x[n]c[n]^{P}_{Pilot}s[n]^{P}_{Pilot}$
13:         $C_{L_{Pilot}} = C_{L_{Pilot}} + x[n]c[n]^{L}_{Pilot}s[n]^{L}_{Pilot}$
14:         $C_{VL_{Pilot}} = C_{VL_{Pilot}} + x[n]c[n]^{L}_{Pilot}s[n]^{L}_{Pilot}$
15:     Accumulate the data correlator output:
16:         $C_{P_{Data}} = C_{P_{Data}} + x[n]c[n]^{P}_{Data}s[n]^{P}_{Data}$
17: end for
```

The computations of the multicorrelator algorithm are concurrently executed in the FPGA. The coherent integration time can be extended up to the duration of one data symbol without any processor intervention. When using a coherent integration time larger than one data symbol duration, the correlation results have to be further accumulated in software.

The tracking multicorrelators can be configured to use a variable number of code replicas to track the pilot and data components of the GNSS signals. The configuration shown in Table 3.3 is used. The current implementation of the tracking multicorrelator can be configured to use two or four bits per input sample.

Table 3.3 Tracking multicorrelator configuration.

Signal	**Configuration**
GPS L1 C/A	E, P, L
Galileo E1b/c	VE, E, P, L, VL (Pilot Component) P (Data Component)
GPS L5	E, P, L (Pilot Component) P (Data Component)
Galileo E5a	E, P, L (Pilot Component) P (Data Component)

3.1.6 Software Architecture

The baseband processing engine is based on the GNSS-SDR software-defined receiver released under the GNU GPL [16, 17]. As explained in section 2.4, GNSS-SDR implements the baseband signal processing chain, from sample capture up to the computation of the PVT solution and the generation of GNSS products in standard formats, enabling interoperability and integration with other systems. It consists of a modular design with several processing blocks: signal source, signal conditioner, acquisition, tracking, navigation message decoder, observables, and PVT, enabling easy addition, modification, and replacement of GNSS receiver algorithms.

The use of embedded devices requires the implementation of hardware accelerators in the FPGA for the computationally expensive tasks: the acquisition and the tracking processes [101]. The FPGA receives and processes the signals coming from the RFFE, replacing the GNSS-SDR's signal sources, and the flow graph structure is changed as shown in Figure 3.6. The software signal source blocks are removed, the acquisition block becomes a function that is called when GNSS-SDR proceeds to detect the presence or absence of a satellite signal, and the tracking blocks act as software signal sources, providing the FPGA multicorrelator's outputs to the telemetry decoder blocks. The acquisition and the tracking multicorrelator algorithms are performed in the FPGA with no processor intervention, but the PLL and DLL are managed in the software tracking blocks. GNSS-SDR configures and controls the execution of the acquisition and tracking FPGA hardware accelerators by writing and reading data to/from their memory-mapped registers. The acquisition and tracking accelerators issue an interruption to the processing system when they are ready with new data.

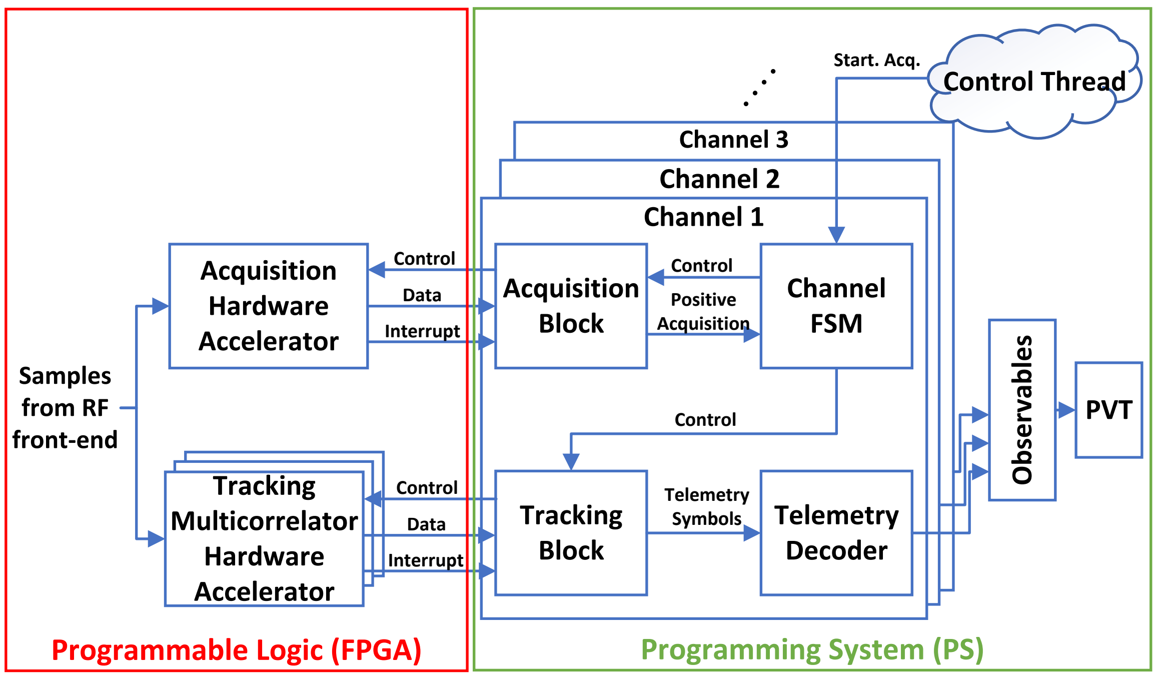

Figure 3.6 Interface between the FPGA IPs and the GNSS-SDR software receiver.

3.2 Proposed Design Methodology

To accelerate development and facilitate quicker iterations in the concept testing phase, a two-stage design process is proposed, as detailed in Table 3.4.

The first stage involves implementing, testing, and validating the GNSS algorithms in software, incorporating the novel algorithms in GNSS-SDR. The receiver can be tested by executing GNSS-SDR on a general-purpose processor, such as a laptop computer. The second stage includes cross-compiling GNSS-SDR for the SoC-FPGA platform.

If the embedded processor has sufficient computing power to execute the newly implemented algorithms in real-time, then it is not necessary to implement hardware accelerators for these algorithms. If a previous implementation of a SoC-FPGA GNSS receiver is already available, researchers can use the newly cross-compiled GNSS-SDR software receiver with the existing platform, and proceed to test and validate the algorithms using the SoC-FPGA platform.

However, if the algorithms being tested require high throughput and low latency, FPGA hardware accelerators may be implemented. These accelerators are integrated with an option in GNSS-SDR to offload the processing tasks to the FPGA. This integration phase includes comprehensive system testing of both GNSS-SDR and the FPGA hardware accelerators. The final step is field validation of the algorithms using the portable GNSS platform.

Many of these design steps can be partially automated using FOSS tools. The proposed method enables faster validation and refinement of algorithms in software before they are further optimized or integrated into a complex system like a SoC FPGA.

Table 3.4 SoC FPGA-based GNSS receiver design flow.

Design Stage	Design tasks
Stage 1: Software Design	• Implement and validate novel GNSS algorithms in software, using GNSS-SDR
Stage 2: SoC-FPGA Design	• Implement FPGA hardware accelerators. • Cross-compile the GNU/Linux OS for the embedded processor on the SoC-FPGA platform. • Cross-compile a Software Development Kit (SDK) for the embedded processor. • Modify the GNSS-SDR source code to use the hardware accelerators when the option to use an FPGA is enabled. • Cross-compile GNSS-SDR for the SoC-FPGA platform using the SDK • Integrate the embedded OS, the software and the FPGA components and test the complete system in a SoC-FPGA platform.

A design methodology based on separate software and hardware design flows is adopted. This approach facilitates the development of GNSS-SDR software signal processing blocks using a FOSS compiler toolchain that is independent of the FPGA development tools. As explained in Section 2.5, FPGA development often requires a deep understanding of hardware design and is generally done using hardware description languages like VHDL or Verilog. This requirement for specialized knowledge can steepen the learning curve and extend the development time frame for the FPGA-based components. Conversely, programming for CPUs typically involves high-level languages, which are easier to learn and can lead to faster development times for complex algorithms.

Code reuse is a fundamental aspect of the proposed design methodology. Implementing the first prototype can be a lengthy process involving FPGA development and the configuration and cross-compilation of an embedded GNU/Linux system for the targeted hardware. However, once the first prototype is implemented, adding new features or porting the existing architecture to other SoC-FPGA variants becomes a much quicker process, facilitating research on novel and experimental algorithms.

The proof-of-concept prototypes presented in this thesis were developed using AMD's SoC FPGAs, formerly Xilinx. Consequently, the development process utilized AMD's suite of development tools. This choice was based on our prior experience with these tools and motivated by the comprehensive support provided by these tools for both FPGA development and embedded system development, including support for creating and building embedded GNU/Linux systems.

The following subsections explain the proposed design methodology and the design tools used in more detail.

3.2.1 Stage 1: Software Design Process

Researchers may implement experimental receiver algorithms in the form of GNSS-SDR software blocks [16, 17], creating new software blocks or modifying the existing ones. The new software blocks can be easily integrated into GNSS-SDR. By default, GNSS-SDR implements a host-based receiver architecture as shown in Figure 3.7, where a general-purpose processor, usually a PC, implements the baseband signal processing chain. An external RF tuner featuring a LNA, an AGC, and an ADC can be used to receive live GNSS signals. GNSS-SDR can process GNSS signals in real-time, provided that the processor has sufficient computing power. Thus, researchers can test new signals and algorithms using a PC and an analog front-end. If the complexity of the implementation prevents real-time processing, researchers may use GNSS-SDR in post-processing mode using recorded signals.

GNSS-SDR can be compiled using a FOSS toolchain, including the two most popular open-source compilers, GCC and LLVM/CLang [8]. A GNSS-SDR website is available on the internet [17], containing tutorials, instructions, and examples of how to download, compile, use, and contribute to GNSS-SDR. Researchers are encouraged to contribute to GNSS-SDR and push the newly implemented changes to the GNSS-SDR repository. In this way, novel algorithms are available to the research community, allowing for review, code inspection, and further modification.

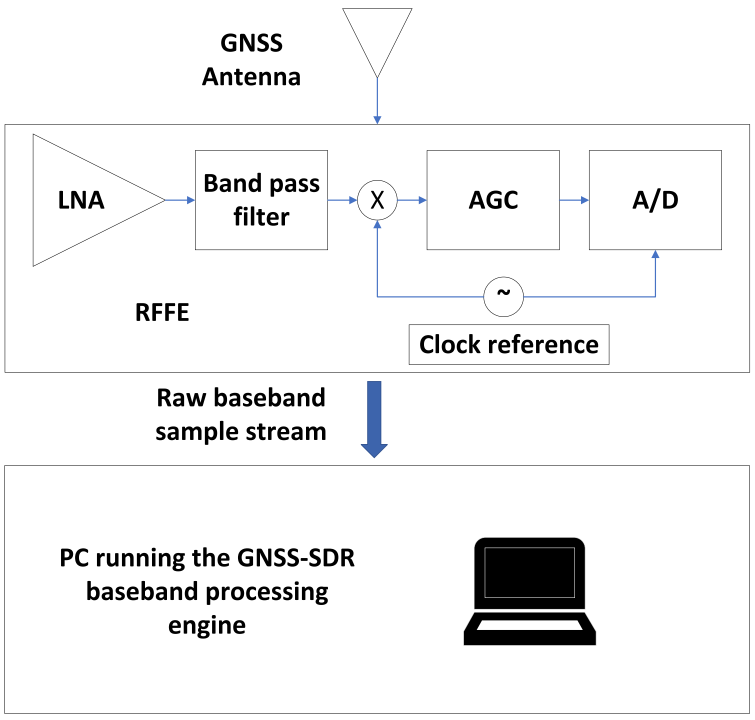
GNSS Antenna
LNA
Band pass filter
X
AGC
A/D
~
RFFE
Clock reference
Raw baseband sample stream
PC running the GNSS-SDR baseband processing engine

Figure 3.7 Host-based GNSS-SDR receiver.

3.2.2 Stage 2: SoC-FPGA Design Process

After validating experimental GNSS receiver algorithms in software, researchers in need of a battery-powered, compact, and portable device for field validation can utilize an SoC-FPGA-based receiver to meet these requirements.

A design based on an SoC-FPGA implements the receiver depicted in Figure 3.8. Often, the RFFE and the SoC-FPGA device can be combined on the same board, yielding a design that is both compact and efficient, thereby minimizing the system's physical footprint. The complete software receiver, encompassing all signal processing stages, operates within the SoC-FPGA. A PC is employed for remote control of the receiver via an Ethernet TCP/IP interface, serial port interface, or a similar communication interface. The SoC-FPGA computes GNSS products in standard formats and transmits them to the remote PC.

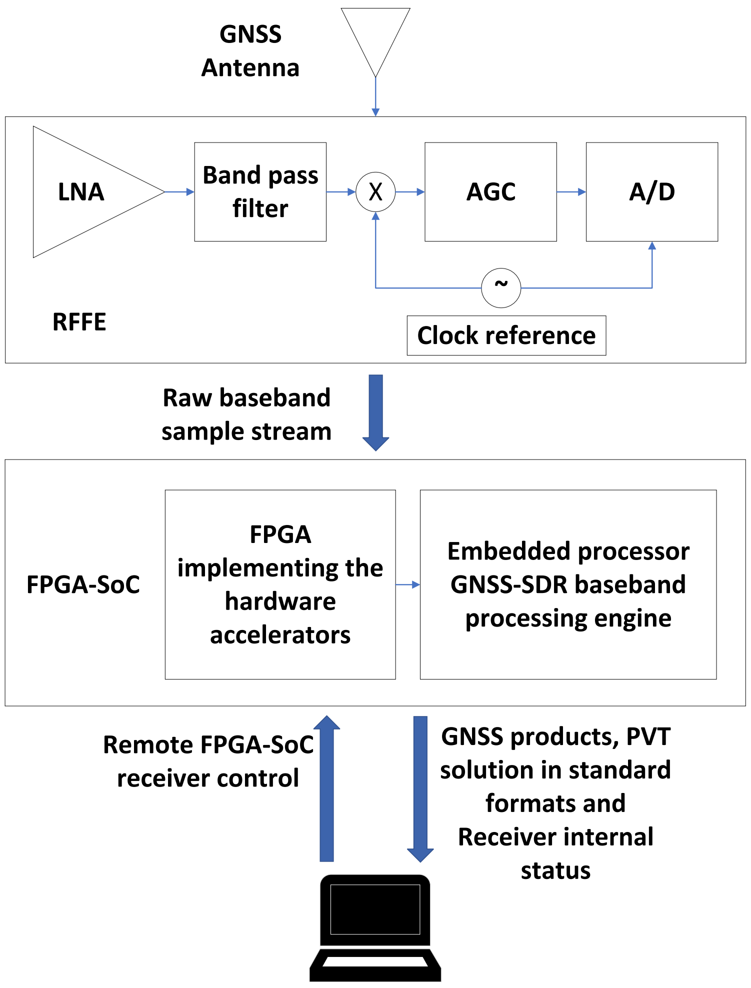
GNSS Antenna
LNA
Band pass filter
X
AGC
A/D
~
RFFE
Clock reference
Raw baseband sample stream
FPGA-SoC
FPGA implementing the hardware accelerators
Embedded processor GNSS-SDR baseband processing engine
Remote FPGA-SoC receiver control
GNSS products, PVT solution in standard formats and Receiver internal status

Figure 3.8 SoC-FPGA GNSS receiver.

As explained in Section 2.4.4, a customized GNU/Linux distribution named *Geniux* (GNSS-SDR for Embedded GNU/Linux) supports the development and operation of GNSS-SDR on embedded devices [55]. This distribution is based on the Yocto Project. It includes a carefully seleted set of tools, libraries, and drivers optimized for a broad spectrum of SDR applications, thus facilitating their advancement to production readiness.

The generation of Geniux images and Software Development Kits (SDKs) can be automated for specific versions, timestamps, and machines within a virtualized environment through the Yocto Geniux repository [55]. However, in its default setup, the Yocto-Geniux repository does not include any FPGA firmware images. Therefore, using this repository to create Geniux images in the standard configuration does not immediately enable the use of hardware accelerators in the FPGA.

For this reason, we use the PetaLinux tools [107] to configure and build a Geniux embedded system and to build a SDK for the target SoC-FPGA platform. PetaLinux is a platform for embedded GNU/Linux development that streamlines the process of customizing and building GNU/Linux systems. Using the PetaLinux tools, a Geniux OS can be compiled that incorporates the FPGA image needed to operate the FPGA hardware accelerators.

The PetaLinux tools, based on the Yocto Project [51] and incorporating the Yocto eSDK, offer a suite of high-level commands that are integrated with the Yocto Linux distribution. These tools ease the customization, building, and deployment of embedded GNU/Linux systems for a variety of AMD SoC-FPGA variants, including the Zynq-7000 and Zynq Ultrascale+ families [63, 102]. They support the incorporation of FPGA images with dedicated FPGA code, streamlining the process of embedding custom logic directly into the FPGA fabric.

An embedded system that incorporates the FPGA image with hardware accelerators can be created utilizing the Petalinux tools and the oe-gnss-sdr-manifest repository [53] introduced in Section 2.4.4. This repository is used to configure the OpenEmbedded build system, tailoring it to the meta-gnss-sdr layer. This approach effectively implements a Geniux customization, integrating the FPGA image with the hardware accelerators seamlessly.

The researcher can effectively implement a Geniux-based platform incorporating hardware acceleration by setting up a PetaLinux project that contains the essential GNU/Linux kernel and filesystem. This involves importing the FPGA firmware image and deploying the required layers, libraries, and dependencies provided by the oe-gnss-sdr-manifest repository mentioned above.

To facilitate the development process, a GitHub repository is also available for generating a Docker image specifically tailored for AMD Xilinx PetaLinux and Vivado/Vitis FPGA design Tools, using Ubuntu 18.04 as the base image [56]. It is named *docker-petalinux2*, and supports the creation of a Docker environment capable of handling PetaLinux and, optionally, Vivado and Vitis software. This tool requires Docker [108] to be installed on the user's machine and an AMD user ID for downloading the necessary installers. The repository provides detailed instructions for downloading and preparing the PetaLinux and Vivado/Vitis installers, and for building the Docker image, which can significantly simplify the development process with PetaLinux on various systems. The AMD Vivado and Vitis are design tools for FPGA development [109, 110]. They can be used to develop code for the SoC FPGA devices used in this thesis, including the AMD Zynq-7000 and Zynq UltraScale+ families [63, 102].

The steps involved in the SoC FPGA design process are detailed in the subsections below.

3.2.2.1 Implementation of the FPGA hardware accelerators

When developing novel GNSS receiver algorithms, researchers need to assess the feasibility and advantages of integrating new FPGA-based hardware accelerators. This evaluation should consider factors such as performance enhancement and cost-effectiveness. The performance enhancement usually depends on two factors: the algorithm's computational load and whether the algorithm directly processes the samples coming from the analog front-end. Algorithms on the left side of the flow graph shown in Figure 2.8 in Chapter 2, directly processing the samples coming from the analog front-end, or requiring a high computational load, cannot run in the embedded processor when the baseband processing engine is running in real-time. As opposed to this, signal processing blocks on the right side of the flow graph shown in Figure 2.8, between the tracking multicorrelators and the computation of the PVT solution, can usually be implemented in software.

The design of the FPGA hardware accelerators requires the use of specialized development tools. The proof-of-concept SoC-FPGA GNSS receivers were implemented in VHDL using the AMD Zynq-7000 and the Zynq Ultrascale+ families of SoC-FPGAs. Hence, the FPGA design process requires the use of the AMD Vivado FPGA Design Suite [109]. FPGAs can also be programmed using a higher level of abstraction, using software code. A process named HLS, briefly described in Section 2.5.3, converts software specifications into RTL models. Users wishing to implement FPGA hardware accelerators using HLS may use the AMD Vitis Unified Software Platform [111]. The FPGA design tools are not distributed using FOSS licenses. However, the academic and research community, which includes educators, researchers, and students, may be eligible for the resources provided by existing university programs [110].

The FPGA design process is devoted to the implementation of the architecture shown in Figure 3.3. The design of the FPGA architecture takes place in three steps:

- The initial step involves the implementation of FPGA acquisition and tracking hardware accelerators. These accelerators are implemented as separate FPGA projects and are encapsulated in the form of FPGA IP cores. In cases where existing acquisition and tracking hardware accelerators are available from a previous project, they can be reused, eliminating the need for new implementations. The developer must also design and integrate hardware accelerators for any newly developed algorithms that necessitate hardware acceleration.

- The second step is the creation of auxiliary FPGA projects implementing the sample conditioning, buffer monitors, and other cores that may be used for monitoring the internal status of the FPGA. These auxiliary IP cores are also implemented as separate FPGA cores. Again, the cores that are already available from other projects can be reused and they do not need to be implemented again.

- The third step entails creating an FPGA project that instantiates the acquisition, tracking, and auxiliary IP cores, alongside the PS/PL interface blocks and the sample buffers depicted in Figure 3.3. This project should also incorporate any additional FPGA IP cores required for hardware acceleration or essential for interfacing with the RFFE. The designer is tasked with importing the necessary IP cores into the FPGA project, establishing connections and configuring each core to align with the project's specifications, ensuring system-wide interoperability

Some manufacturers provide FPGA example designs implementing RFFE interfaces and DMA blocks. For instance, Analog Devices, provides FPGA example designs, libraries, and projects under FOSS licenses for various FPGA architectures [112]. These designs can be reused, reducing time and effort.

3.2.2.2 Cross-Compiling GNU/Linux for the SoC-FPGA Platform

The initial step in creating an embedded GNU/Linux OS is to install the PetaLinux tools on a PC and then create a PetaLinux project for the targeted SoC-FPGA platform [107].

Next, the FPGA design is integrated into the PetaLinux project using PetaLinux commands. This step automatically configures the embedded OS image to align with the FPGA's hardware design, including physical addresses of hardware accelerators and interrupt lines.

The subsequent step entails integrating additional layers—drivers, libraries, and applications—into the PetaLinux project, with a focus on those not included by default. To streamline this process, the oe-gnss-sdr-manifest repository [53] should be imported. Utilizing the Repo tool [54], the developer then synchronizes essential repositories for the integration of GNSS-SDR and its dependencies at their correct revisions, including the meta-gnss-sdr OpenEmbedded layer [52]. This process effectively creates a Geniux GNU/Linux customization with FPGA support.

The final step involves building the complete system and generating boot images for the target SoC-FPGA platform.

3.2.2.3 Cross-compiling an SDK for software development

Cross-compiling an SDK is essential for software development, as it enables the subsequent cross-compilation of GNSS-SDR on a PC for the target platform. Direct compilation of GNSS-SDR on the target platform would eliminate the need for an SDK, but this approach is impractical due to the extended duration it would take—often hours. This inefficiency stems from the SoC FPGAs' low-power embedded processors, which lack the computing power of typical PC processors.

Once the PetaLinux project is configured as explained in subsection 3.2.2.2, the SDK can be created using the PetaLinux command tools [107].

3.2.2.4 Creating GNSS-SDR processing blocks for FPGA offloading

When FPGA hardware accelerators are implemented, GNSS-SDR needs to establish communication with these accelerators to delegate signal processing tasks to the FPGA. As explained in Section 3.1.2, the hardware accelerators provide a set of memory-mapped registers accessible from the embedded processor and have the capability to generate interrupt requests for the embedded processor.

The memory-mapped registers are implemented using the AMBA AXI4 interface, as most SoC FPGAs support this interface for the communication between the FPGA and the embedded processors.

GNSS-SDR cannot directly access the memory-mapped registers in the FPGA hardware accelerators because the GNSS-SDR software receiver operates atop the Geniux OS , which enforces memory protection. Memory protection controls access rights to a computer's memory, ensuring the software in user space manages only virtual memory addresses. A specialized hardware unit within the embedded processor, the Memory Management Unit (MMU), examines all memory access requests. It translates these requests—from virtual memory addresses to physical addresses in the main memory. Consequently, a mechanism is required to map the actual memory addresses of these registers (the physical addresses) to a range of virtual addresses accessible by the software.

Linux offers a standard User Input/Output (UIO) framework for the development of user-space device drivers. This framework defines a small component in the operating system's kernel space that performs the following tasks [113]:

- Mapping the physical addresses of memory-mapped registers in the hardware accelerators to virtual memory addresses accessible by user-space applications.
- Registering FPGA interrupts and delivering interrupt notifications to user space.

To utilize the hardware accelerators, it is necessary to integrate GNSS-SDR blocks that employ the UIO framework. This approach enables direct access to the memory-mapped registers of the hardware accelerators and the management of interrupt requests. Consequently, researchers can adapt GNSS-SDR to delegate signal processing tasks to the FPGA, enhancing efficiency.

The latest GNSS-SDR release includes blocks to offload acquisition and tracking multicorrelator algorithms to the FPGA and to manage auxiliary FPGA IPs, such as the dynamic bit selector [24].

3.2.2.5 Cross-compiling GNSS-SDR for the SoC-FPGA platform using the SDK

After cross-compiling the SDK, as detailed in Section 3.2.2.3, and integrating GNSS-SDR blocks for FPGA offloading, as explained in Section 3.2.2.4, GNSS-SDR is ready for cross-compilation for the SoC-FPGA platform using the SDK on the PC.

3.2.2.6 Integrating the embedded OS, the software, and the FPGA components

The final step in the SoC-FPGA design process is the integration and testing of the complete system, which includes the FPGA hardware accelerators and the GNSS-SDR software receiver.

After building GNSS-SDR and the Geniux system, the next step involves copying the embedded OS, FPGA firmware image, and the cross-compiled GNSS-SDR software receiver onto a non-volatile storage medium, such as an SD card. This SD card is then to be inserted into the SoC-FPGA platform, which is capable of booting from SD cards. Upon booting the device, users can remotely access the system to execute GNSS-SDR.

The final assessment phase involves evaluating the performance of the hardware accelerators and testing the SoC-FPGA platform using live signals to ensure that the project requirements have been met.

3.3 Analyzing Design Force and KPIs in the Proposed Architecture

In this section, the proposed architecture is discussed in terms of several design forces, including scalability, testability, portability, maintainability, reproducibility, and openness, alongside KPIs related to these aspects.

3.3.1 Scalability

Scalability refers to the proposed architecture's ability to efficiently manage growing workloads and its capacity for expansion to accommodate future growth. Various KPIs gauge scalability, such as the system's ability to achieve nearly linear performance improvements in line with increases in the number of computing resources. Other indicators include the system's capability to seamlessly integrate new GNSS signals and algorithms, as well as the flexibility of its configuration system to scale without limits. These KPIs are analyzed below.

GNSS receivers can simultaneously and independently process multiple satellite signals or satellite measurements [114]. This capability allows for the replication of channel processing architectural units within the FPGA, resulting in a linear increase in the number of available channels based on the FPGA's resources and the maximum number of threads in the embedded processor. However, the observables and navigation solutions are computed using data obtained from all channels. Therefore, these algorithms do not scale linearly with the hardware resources.

In addition, the combination of a scalable configuration system and the flexibility of the FPGA allows for the unlimited addition of new GNSS signals and algorithms to the architecture. Each GNSS receiver implemented using this architecture may process a subset of these signals and algorithms based on the resources available within the FPGA and the energy budget.

Finally, the proposed receiver architecture capitalizes on the GNSS-SDR configuration method. This method involves using a configuration file to set operational parameters and settings for the GNSS-SDR system. The configuration file specifies all signal processing parameters for the receiver, including the choice of signal sources (e.g., recorded GNSS signals or the RFFE) and the GNSS signals to be processed. This scalable configuration system facilitates the integration of new FPGA hardware accelerators, enhancing support for additional GNSS signals. With the introduction of new hardware accelerators, the configuration file can be updated to include corresponding acquisition and tracking signal processing modules, thereby accommodating these advancements.

3.3.2 Testability

Testability refers to the ease with which the proposed receiver prototypes can be tested to ensure they function correctly and meet their design specifications. This important quality attribute affects the simplicity of designing and conducting tests, diagnosing problems, and validating system requirements, thereby impacting the effort, time, and resources required for testing activities. Testability is measured through several KPIs, such as the presence of a testing framework, application-level logging, and a flexible configuration mechanism.

In this context, the proposed receiver architecture utilizes GNSS-SDR's testing capabilities, which are built on the Google C++ testing framework, *GoogleTest* [16, 115]. This specialized library is employed for unit testing purposes. Tests are divided into two categories: unit tests and system tests. Unit tests are used for checking of certain functions and areas of the source code. System tests are performed on the complete, integrated receiver signal processing chain, to evaluate the system's compliance with its specified requirements.

Using this framework, developers can add new tests to verify the FPGA hardware accelerators, evaluate any aspect of the SoC FPGA receiver architecture, or test any newly implemented features. The main requirement for the FPGA hardware accelerators is that all required variables and status information must be accessible from the embedded processor. The test execution can be automated, in a way that any subset of tests can be performed, enabling the testing of multiple conditions within the same run. To implement tests, developers write test cases that execute specific checks, known as assertions. These assertions verify whether a condition is true and result in either success or failure. Test cases are sets of actions performed on the receiver to determine if it is functioning correctly. If a test case has a failed assertion, then it fails. Otherwise, it succeeds [17].

The proposed receiver architecture also leverages GNSS-SDR's logging mechanism. Logging is handled by the Google logging library known as *google-glog*, which implements application-level logging [16, 116].

In addition, GNSS-SDR's software receiver offers various flags enabling users to monitor and record internal receiver variables at different points in the signal path. Users can activate these flags through GNSS-SDR's flexible configuration mechanism, making them useful for overseeing the performance of signal processing blocks, especially those utilizing FPGA hardware accelerators. To demonstrate this capability, the spaceborne GNSS receiver presented in Chapter 4 was configured to capture the status of the GPS L1 C/A tracking loops in a static scenario. The results are depicted in Figure 3.9 and include the following information:

- The demodulated bits of the navigation message.
- A polar scatter plot of the I and Q signals. This plot is useful to check that the Doppler wipe–off is done correctly.
- The results of the E, P and L correlators. We can see that the P correlator has the highest values, which means that the receiver is correctly synchronized.
- Various plots showing the behavior of the PLL and the DLL, the estimated Doppler frequency, the carrier to noise density ratio, and the estimated code frequency of the received signal.

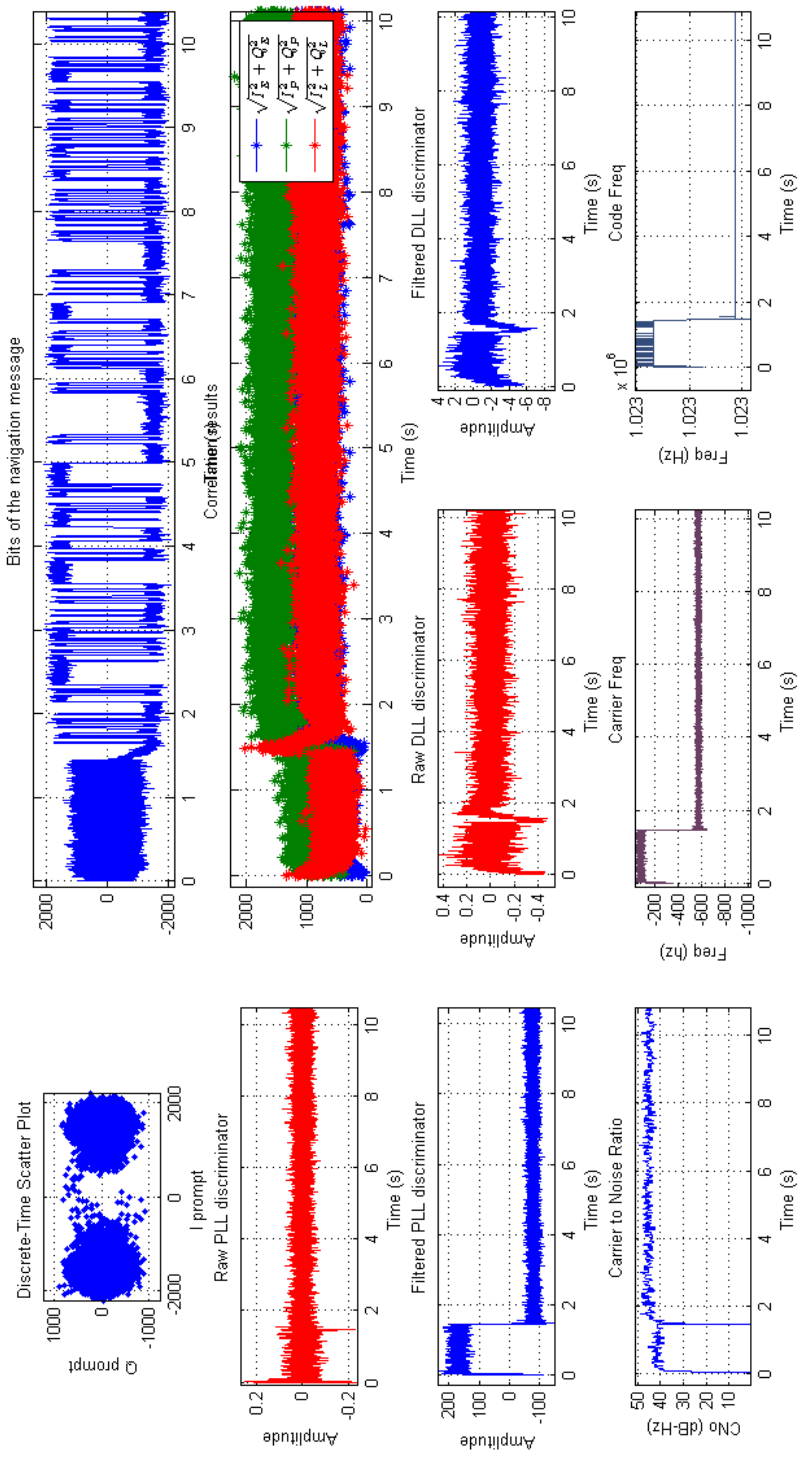

Figure 3.9 Status of the GPS L1 C/A tracking loops

3.3.3 Portability

Portability refers to the ease with which a software application or system can be transferred from one environment or platform to another. KPIs measuring the portability of the proposed architecture include the range of supported SoC-FPGA architectures and supported OSs.

Even though GNSS-SDR supports a wide range of computing platforms [17], the proposed architecture's adaptability across different platforms is primarily constrained by the FPGA's inherent portability limitations. As explained in section 2.5.4, FPGA design tools are often proprietary, with major FPGA vendors offering their own suites of development software. These proprietary tools are designed to support the specific features and architectures of their respective FPGA models, providing integrated environments for design, synthesis, simulation, placement, routing, and bitstream generation.

The design methodology proposed in this thesis can be applied to a large number of models of SoC FPGAs from the same manufacturer. This compatibility enables the implementation of a wide range of concept prototypes, accommodating devices that prioritize low power consumption as well as those designed for high performance. However, transferring an FPGA design to an FPGA device from a different manufacturer presents a challenging task due to various factors. These challenges include the vendor-specific toolchains mentioned above and the reliance of the existing code on FPGA IP cores exclusive to the original manufacturer. Therefore, the proposed architecture is not easily portable to SoC FPGAs of another manufacturer.

The proposed architecture is designed specifically for the embedded GNU/Linux OS. However, the inherent advantages of the GNU/Linux OS—its free availability and extensive customizability across a broad spectrum of computing platforms—provide a solid foundation for future adaptations.

3.3.4 Maintainability

Maintainability refers to the ease with which a software application can be modified to correct faults, improve performance, or adapt the product to a changed environment. It is a key aspect of software quality, encompassing the effort required to make updates, enhancements, or adjustments post-deployment. High maintainability is essential for extending the lifespan of software, ensuring it can evolve in response to new requirements, technologies, and user needs without excessive costs or complexity. KPIs measuring maintainability include change request response times, and time to fix defects.

To expedite the design process, the proposed methodology places a strong emphasis on the reuse of code. The main limitation of the proposed methodology remains the intricate design and the time needed for development. Creating the initial prototype can be a time-consuming task that may require specialized knowledge of hardware design, HDL, synthesis and place-and-route tools, as well as setting up an embedded GNU/Linux system for the targeted hardware. Nevertheless, once the first prototype is available, the addition of new features and the porting of the existing architecture to other SoC-FPGA variants of the same manufacturer becomes a more practical process, thanks to code reusability, facilitating research on novel algorithms.

The time required to fix errors can vary significantly, influenced by the problem's nature and the ease of its localization. However, diagnosing and resolving issues within an FPGA typically demands more time than addressing software-related problems. This increased complexity is attributable to several factors:

- As explained in Section 2.5, FPGA designs are typically more complex and specialized than general software, requiring deep knowledge of hardware description languages (HDLs) like VHDL or Verilog, as well as an understanding of the underlying hardware architecture. This can make diagnosing and fixing issues more time-consuming.
- Once the FPGA source code is modified to fix the bug, the modified design usually undergoes simulation to ensure that it behaves as expected. This simulation process can be time-consuming, lasting for tens of minutes or potentially more, especially for large and complex FPGA hardware accelerators.
- After the fixed code is validated in simulation, and before the design can be tested in the SoC FPGA, the design must be re-synthesized and implemented as explained in section 2.5.3, a process that can take from minutes to hours depending on the complexity of the design and the performance of the synthesis tools.

As opposed to this, software is generally more flexible than hardware. Bugs can often be fixed by changing code and reloading the software into the SoC FPGA nonvolatile memory. This process can be done very quickly, once the source of the problem is found and the code is fixed.

While the GNSS-SDR repository incorporates a continuous integration system featuring an automated build process and automated testing, this is not the case for the FPGA. A possible approach to improve maintainability is to implement a similar continuous integration system for the FPGA code. This system would reduce the time spent on manual testing of receiver prototypes, leading to increased productivity and shorter response times for implementation and change requests.

3.3.5 Reproducibility

Reproducibility in science refers to the ability to obtain consistent results using the same data and methods as were used in the original study. It's a cornerstone of the scientific method, ensuring that findings are reliable and can be trusted by other researchers and the wider community. A key KPI measuring reproducibility in software-based systems is the presence of a software development process that guarantees identical binaries or outputs, such as executable files, packages, or other software products, from the same source code across different builds or environments.

Reproducible builds in the context of FPGAs mean that the same source code or HDL files will produce the exact same binary configuration file (e.g., a FPGA firmware image) every time they are compiled or synthesized, regardless of the environment or the tools' version.

Achieving reproducible builds with FPGAs is challenging due to the inherent pseudo-randomness of modern FPGA design tools, making them less predictable compared to software development. Some steps in the FPGA build process, especially placement and routing, can involve heuristic algorithms that do not guarantee the same result on every run, even with the same input.

FPGA design tools implement a combination of different verification and validation techniques to ensure that the compiled FPGA file is compliant with the original design entry, including timing analysis, which ensures that the design meets all timing requirements, including setup and hold times, which are critical for the proper functioning of the FPGA in its target environment. Although the results are guaranteed to meet the tolerance ranges defined by the designer, the nature of FPGA design and development processes poses an extra challenge to reproducibility for FPGA firmwares defined by their VHDL model [117].

3.3.6 Openness

Openness refers to the degree to which a software project or platform is accessible, transparent, and inclusive to users, developers, and other stakeholders. A relevant KPI to measure openness is the use of open licenses and/or the availability of a technical description of the algorithms implemented in the receiver.

In this sense, as explained in Section 2.4, the proposed architecture utilizes the popular open-source GNSS-SDR software-defined receiver as the core GNSS engine, which is freely accessible online and released under the GNU GPL license [16, 17].

The FPGA hardware accelerators, however, are not necessarily required to be released under FOSS licenses. Researchers have the option to release FPGA IP HDL code under open-source licenses or proprietary licenses, which can serve as a means to monetize research while also enhancing research impact and reputation. For this reason, inspection of any internal aspect of the FPGA hardware accelerators relies on the availability of the software version of the algorithms within GNSS-SDR, alongside the documentation offered by the FPGA IP core developers. As explained in section 3.2.1, the software version of the algorithms is usually implemented in GNSS-SDR in the first stage of the design process.

3.4 Summary

This chapter introduced the proposed SoC FPGA receiver architecture and the design methodology for prototyping experimental GNSS receivers. both the FPGA and the software architecture were presented, with a specific emphasis on the FPGA architecture, describing how the FPGA processes the samples received from the analog front-end, implements sample conditioning and buffering, and executes acquisition and tracking hardware accelerators for processing the received signals. Then, the implementation of the acquisition and tracking hardware accelerators was described in detail, providing details on the acquisition and tracking algorithms implemented in the FPGA. Subsequently, the modifications made to the GNSS-SDR software receiver were described. These modifications enable the offloading of the most computationally demanding tasks to the FPGA. To enable communication between the embedded processor and the FPGA, the FPGA hardware accelerators expose a set of memory-mapped registers accessible from the embedded processor, which are used by the embedded processor to exchange data with or send commands to the FPGA. In addition, the FPGA implements interrupt lines from the hardware accelerators to the processing system. The hardware accelerators issue interrupt requests to the embedded processor to synchronize both the FPGA and the embedded processor operations.

The design methodology was also elucidated in this chapter. A separate hardware/software design flow was adopted, where both FPGA design and software design can be carried out concurrently, enabling the use of a complete FPGA toolchain for software development.

In the proposed design methodology, the implementation of novel algorithms can be performed in two stages. In the first stage, these algorithms are implemented in the GNSS-SDR software receiver, enabling rapid implementation, testing and validation of the experimental algorithms in a laboratory environment. In the second stage, GNSS-SDR is cross-compiled for the embedded platform, and FPGA hardware accelerators are implemented if the computational load required for the algorithms prevents their execution in the embedded processor. The implementation in the embedded platform enables deployment for portable and mobile test campaigns where power efficiency and size are relevant considerations.

In the concluding segment of this chapter, the proposed architecture is discussed in terms of several design forces, including scalability, testability, portability, maintainability, reproducibility, and openness, alongside KPIs related to these aspects.

The architecture is designed to be scalable, accommodating the integration of new signals and algorithms, albeit constrained by the availability of FPGA resources and computing power in the embedded processor. Notable features include a comprehensive testing framework, robust application-level logging, and a flexible configuration mechanism.

Despite its versatility, porting the architecture to other manufacturers' FPGA devices presents significant challenges due to factors such as proprietary FPGA tools, vendor-specific features inherent to FPGA platforms, and compatibility issues with IP cores.

Furthermore, while code reuse shortens development time, implementing new features or fixing defects in the FPGA takes longer than in software due to the nature of the FPGA design and implementation process. One potential solution to improve maintainability is the implementation of a continuous integration system for the FPGA code, incorporating build automation and automated testing. However, reproducible builds remain a challenge in FPGA development, primarily due to the use of non-deterministic processes in some steps and toolchain variability.

Lastly, regarding openness, although FPGA hardware accelerators may not be covered by FOSS licenses, accessibility to the signal processing details can be facilitated through the implementation of equivalent algorithms in GNSS-SDR and the documentation of FPGA IP cores. This approach promotes a thorough understanding of signal processing and offers insights into optimizing algorithms to meet specific requirements.

Future work may include exploring the use of the fully preemptible real-time Linux kernel (using the PREEMPT_RT patch) to provide enhanced system reliability. Experimental results demonstrate improved real-time support and significantly reduced kernel latencies, as shown in [118].

The results presented in this chapter were partially published in:

- [20] M. Majoral, C. Fernández-Prades, and J. Arribas, “A Flexible System-on-Chip Field-Programmable Gate Array Architecture for Prototyping Experimental Global Navigation Satellite System Receivers,” *Sensors*, vol. 23, no. 23, 2023, Art. no. 9483. doi: 10.3390/s23239483
- [23] M. Majoral, C. Fernández-Prades, and J. Arribas, “Implementation of GNSS Receiver Hardware Accelerators in All-programmable System-On-Chip Platforms,” in *Proceedings of the 31st International Technical Meeting of the Satellite Division of The Institute of Navigation (ION GNSS+ 2018)*, Miami, FL, USA, September 2018, pp. 4215–4230. doi: 10.33012/2018.16082

The work presented in this section was supported by the following funding sources and collaborators:

- Grant CPP2021-008648, titled “Receptor Abierto y Flexible de Investigación por Satélite”, funded by the Ministry of Science and Innovation.
- Grant PID2021-128373OB-I00, funded by the Spanish Ministry of Science and Innovation through MCIN/AEI/10.13039/501100011033, and by the European Regional Development Fund (ERDF) “A way of making Europe”.
- Grant agreement H2020-687367-AUDITOR, funded by the European Commission.

Chapter 4

Spaceborne GNSS Receiver

This chapter outlines the design, proof-of-concept implementation, and preliminary evaluation of a low-cost, software-defined spaceborne GNSS receiver developed with COTS products, following the design principles and methodology described in Chapter 3. This study aims to evaluate the feasibility of employing an SoC FPGA platform for space GNSS receivers.

This receiver is specifically designed to process GNSS signals and deliver navigation solutions under LEO conditions. Utilizing the GNSS receiver architecture proposed in this thesis, it integrates the versatility of the GNSS-SDR open-source software-defined receiver with the computational efficiency of FPGAs. Adopting an open-source software receiver not only fosters collaborative development of innovative algorithms but also enables a thorough examination of baseband processing techniques. Moreover, it creates a transparent and easily reproducible framework for scientific experimentation.

The development of the spaceborne receiver prototype was carried out within the framework of this thesis. This work involved implementing the FPGA architecture, porting the GNSS-SDR receiver with high-dynamic tracking loops to the SoC FPGA, and integrating an FPGA signal source block into GNSS-SDR. Additionally, it included conducting unit and system tests on the prototype in LEO scenarios. However, the implementation of high-dynamic tracking loops in GNSS-SDR fell outside the scope of this thesis.

The proposed design targets two distinct boards: Analog Devices' ADRV9361-Z7035 and AMD's ZCU102. The ADRV9361-Z7035 was selected for its high level of integration, featuring a full RFFE and an integrated SoC-FPGA within a single platform, with an Small Form Factor (SFF), measuring 10 cm $\times$ 6.2 cm. In contrast, the ZCU102 offers significant flexibility through its advanced platform, which includes a more powerful SoC-

FPGA combination, suitable for more demanding computational tasks. The ZCU102 does not integrate any RFFE. For this reason, Analog Devices' AD-FMCOMMS5-EBZ [119] was utilized in conjunction with the ZCU102. Compared to the ADRV9361-Z7035, both the ZCU102 and the AD-FMCOMMS5-EBZ are significantly larger; the ZCU102 measures $243.84 \times 237.49 \times 2.64$ mm, while the AD-FMCOMMS5-EBZ is approximately 140×90 mm.

Both implementations, one based on the ADRV9361-Z7035 System on Module (SoM) and the other on the ZCU102 development board, were tested, with this chapter detailing the outcomes. Measurements were conducted separately on each platform, utilizing the ADRV9361-Z7035 for certain tests and the ZCU102 development board for others.

This chapter is organized as follows: Section 4.1 provides an introduction. Section 4.2 presents the objectives of the spaceborne GNSS receiver design. This is followed by a detailed discussion on the implementation in Section 4.3, covering the RFFE, the FPGA architecture, as well as the implementation of the PLL and the DLL for tracking high dynamic GNSS scenarios. Section 4.4 covers the test results, and Section 4.5 concludes with a summary of the results obtained. Additionally, Appendices 4.A and 4.B present the receiver configurations used for static and LEO scenarios, respectively.

4.1 Introduction

Space-qualified GNSS receivers must deliver the exceptional measurement quality necessary for Precise Orbit Determination (POD). The AGGA-family is a notable example of such receivers [120–122]. However, these receivers are associated with high costs and a lack of reconfigurability.

To meet the requirements of low-cost space missions, such as CubeSats and MicroSats, it is essential to not only reduce the cost of space GNSS receivers but also enhance their configurability and flexibility. Such improvements would facilitate easy adaptation and updates for new missions, applications, platforms, and future GNSS signals [123]. In this context, software-defined GNSS receivers powered by consumer-grade, high-performance SoC technologies, present a promising alternative to traditional ASIC-based solutions.

Hybrid systems combining CPUs and FPGAs offer enhanced performance with an acceptable increase in size, power consumption, and cost, which is particularly essential for CubeSat missions. Additionally, these systems enable the possibility of in-flight reconfiguration [124]. FPGA technology has been widely utilized in space applications, and the practicality of using System on Chip SoC FPGA chips in space has been successfully demonstrated [125].

Modern FPGAs are highly suitable for the space environment, enabling reliable data processing and significant computational capabilities [126]. Recently, there has been noticeable growth in the availability of FPGAs and memory components specifically designed for space applications. Noteworthy examples include AMD's Virtex-5QV [127] and Kintex UltraScale XQRKU060 FPGAs [128], along with Microchip's Radiation-Tolerant FPGAs [129]. These products mark significant advancements in space-grade programmable logic devices, offering enhanced capabilities for space missions. Furthermore, European initiatives to boost competitiveness in the space services sector are propelling the research and development of advanced solutions, including radiation-hardened (rad-hard) FPGAs and SoCs FPGAs [124, 130].

Several FPGA-based space GNSS receivers for POD have been proposed in the literature, and some of them are accessible in the commercial market. Examples of this are referenced in [91, 131–133]. In addition, Ref. [134] proposes a SoC-FPGA based GNSS receiver.

As explained in Section 2.2.4, a static GNSS receiver experiences Doppler frequencies up to ±4.9 Hz and a Doppler rate up to 1 Hz/s [26]. The situation changes for a LEO space-borne GNSS receiver. LEO space-borne GNSS receivers move at high speeds, thereby increasing the experienced Doppler frequency and Doppler rates in the received signals. According to Newton's law of gravitation, the velocity of a LEO space-borne GNSS receiver increases as the orbital altitude decreases. Ref. [135] simulates several LEO orbit altitudes between 400 km and 2000 km. According to [135], a LEO space-borne GNSS receiver at a relatively low orbital altitude of 400 km experiences a maximum Doppler frequency shift of ±45.5 kHz and a maximum Doppler rate of ±80 Hz/s.

4.2 Objectives

The primary objective of implementing the spaceborne receiver was to validate the SDR approach in high-dynamic environments. Additionally, it aimed to demonstrate the feasibility of a low-power SFF SoC FPGA receiver, capable of processing GNSS signals in LEO scenarios. This design is based on the architecture and methodology proposed in Chapter 3.

To achieve accurate GNSS positioning, the system in focus features a dual-band GNSS receiver, operating at central frequencies of 1176.45 MHz and 1575.42 MHz, engineered to process GPS L1 C/A, Galileo E1b/c, GPS L5, and Galileo E5a signals. This dual-frequency capability is crucial for achieving accurate GNSS positioning, as it effectively counters the ionosphere's impact. The ionosphere, an atmospheric layer ionized by solar and cosmic radiation extending from approximately 60 km to 1000 km above Earth [136], significantly influences radio signal propagation. The mitigation of this ionospheric influence on GNSS positioning through dual-frequency signal processing, which exploits the frequency-dependent nature of ionospheric delays, is indispensable for precision applications. This strategy enables a substantial reduction in ionospheric delay effects, thereby enhancing the accuracy of navigation and positioning. LEO includes spacecraft orbital altitudes between 400 km and 1600 km [135]. POD for LEO satellites, utilizing GNSS, can be affected by ionospheric conditions [137]. For this reason, the proposed receiver applies, if applicable, double-frequency-based ionospheric corrections using the satellite-broadcasted ionospheric data when operating in double-frequency mode.

The receiver is designed to offer maximum flexibility and to enable comprehensive performance investigations under various operating conditions. It supports multiple operation modes, selectable across single and dual frequencies, including GPS-only, Galileo-only, or multi-constellation configurations that can encompass any combination of GNSS signals. Furthermore, the receiver accommodates up to 12 channels per signal, facilitating support for potentially up to 48 channels in a dual-frequency E1/L1 and L5/E5a configuration for both GPS and Galileo. The goal is to enable the receiver to simultaneously process signals from multiple satellites, thereby substantially improving the positioning data's robustness and reliability. Adding to its versatility, the receiver provides the option to track either the Galileo or GPS pilot or data signal components, with the choice being user-configurable.

Additionally, the receiver implements a narrow-correlator tracking architecture in order to minimise the noise and multipath impact. The severity of multipath effects decreases with altitude due to the geometry of signal reception. LEO satellites are less affected by multipath errors compared to ground-based receivers, but the effect is not entirely negligible [138].

The navigation solutions can be computed without any external aiding. Two types of PVT solution can be used: Least Squares (LS)-based and Kalman-filter-based.

To facilitate testing, the receiver is designed to operate in both real-time and post-processing modes. In real-time mode, GNSS signals are acquired directly from an antenna or a signal generator. In contrast, in post-processing mode, the receiver processes GNSS signals previously recorded and stored in files. Moreover, while in real-time mode, the receiver offers the capability to save the baseband I and Q samples, captured post-ADC, to external storage. This feature aids in debugging and facilitates further post-processing analysis. The receiver can also store the demodulated telemetry data in files for subsequent examination.

To facilitate interoperability with other devices and systems, the proposed concept demonstrator produces GNSS signal products using the standard output formats detailed in Section 2.4.3. The supported output formats include RINEX, RTCM, GPX, KML, GeoJSON, and NMEA, allowing for flexible integration with various applications. The receiver generates PVT solutions via Least Squares or Kalman filtering in WGS-84 at customizable rates. For every satellite in view, it provides comprehensive measurements such as pseudorange, carrier phase or phase range, Doppler shift or pseudorange rate, and signal strength, along with DOP, navigation data, correlators' output, and position and velocity in WGS-84. Additionally, it offers timing data including GPS Time, Galileo Time, and UTC time.

The receiver can be remotely controlled using Telecommands (TCs). The design utilizing the ADRV9361-Z7035, demonstrates the feasibility of developing an SoC-FPGA based GNSS receiver with a form factor suitable for CubeSats. These are small satellites consisting of units each measuring 10 cm on a side. CubeSats, primarily utilized for space research, are built in multiples of these cubic units. A typical CubeSat unit is limited to a weight of no more than 1.33 kg. Moreover, CubeSats often make use of commercially available, standard components for both their electronic systems and structural elements.

For the presented proof-of-concept, the targeted sampling rates were 12.5 MSps for L1/E1 and 12.5 MSps for L5/E5a signals, which provides sufficient capability to track GNSS signals in both the L1/E1 and L5/E5a bands [139].

4.3 System Design

This section details the spaceborne receiver design. Both the ADRV9361-Z7035 and the ZCU102 board-based implementations are presented.

4.3.1 Implementation on the ADRV9361-Z7035 Board

The receiver demonstrator based on the ADRV9361-Z7035 SoM development board is illustrated in Figure 4.1. This board integrates the AD9361 RF Transceiver [106] and the

AMD Zynq-7000 XC7Z035-L2FBG676I SoC FPGA [103], featuring a form factor of 100 mm x 62 mm. The AD9361 is a dual-channel 70 MHz to 6 GHz front-end IC with integrated 12-bit ADCs and DACs. The XC7Z035 is internally divided into the PL and the PS. The PL contains a Kintex-7 FPGA equipped with 275 k logic cells and 900 DSP slices. The PS houses a dual-core ARM Cortex-A9 processor running at 800 MHz, along with several peripheral interfaces, including Local Area Network (LAN) interface via a non-standard connector, which is used to remotely control the receiver. The system implements the architecture proposed in Section 3.1.1.

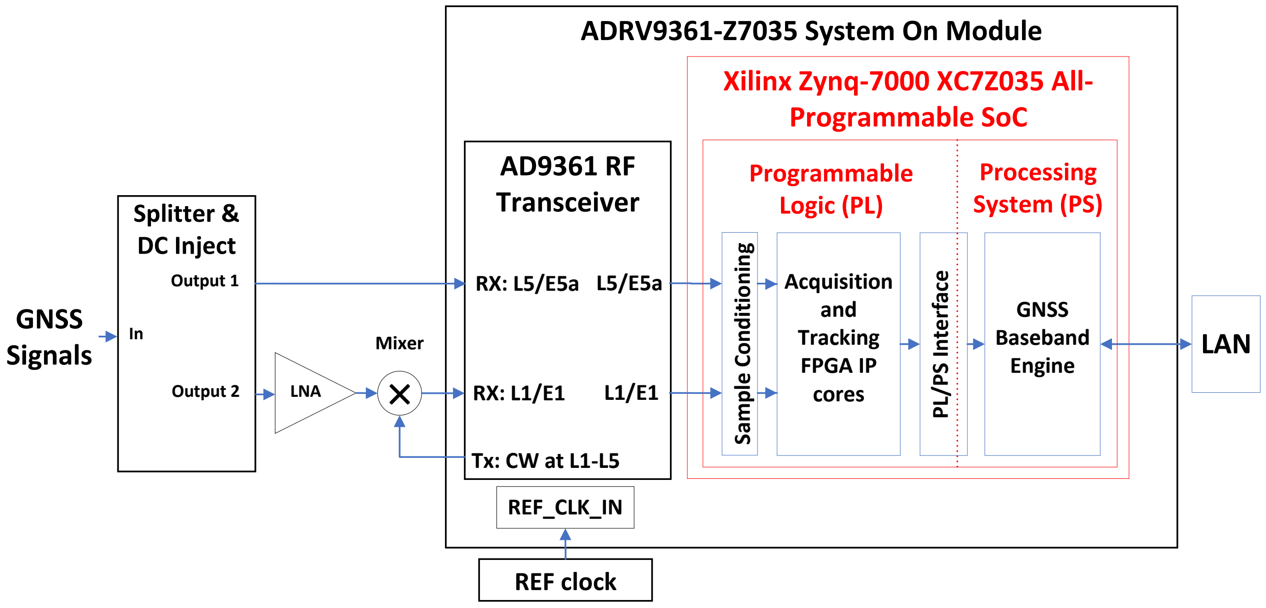

Figure 4.1 Spaceborne GNSS receiver block diagram using the ADRV9361-Z7035 development board

A splitter divides the GNSS signals received from the antenna into two outputs: output 1 and output 2, as illustrated in Figure 4.1. Both outputs are connected to the AD9361 RF transceiver. Together with an external mixer, this configuration allows for tuning to the L1/E1 frequency band in output 1, while output 2 is tuned to the E5a/L5 frequency band. This arrangement is elaborated upon in Section 4.3.1.1.

Subsequently, the baseband signals are forwarded to the FPGA, which implements sample conditioning and buffering, acquisition, tracking multicorrelator FPGA IP cores, and a PS/PL interface, as detailed in Section 3.1.1. Finally, the GNSS-SDR software-defined receiver processes the GNSS baseband signals, obtaining the PVT.

The system features 1 GB of Double Data Rate 3 (DDR3) RAM, utilized for the memory requirements of the embedded OS and to operate the GNSS-SDR baseband processing engine. Additionally, the ADRV9361-Z7035 SoM includes an SD card reader and 256 Mb of Quad Serial Peripheral Interface (QSPI) flash memory. This setup enables non-volatile storage for boot code, the OS, FPGA firmware, application code, and data. The SD card reader primarily serves prototyping needs, while the QSPI memory offers a more permanent storage solution.

The ADRV9361-Z7035 forms a direct connection between the AD9361 RF Agile Transceiver and the AMD Zynq-7000 Z7035 SoC FPGA using specialized high-speed data ports and clocks, along with an Serial Peripheral Interface (SPI) control interface and various other control and framing signals. The digital interface of the AD9361 includes dual parallel data ports, accompanied by several clocks, synchronization, and control signals for sample transfer between the AD9361 and the Zynq SoC. These signals can be configured either as single-ended Complementary Metal-Oxide-Semiconductor (CMOS) signals, or Low Voltage Differential Signaling (LVDS) for systems demanding rapid, low-noise data transmission. In its LVDS mode, the interface functions in a DDR configuration, enabling 12-bit samples from the AD9361 to be transmitted over two 6-bit lanes using differential pairs. The highest transfer rate achievable across this interface is constrained by the AD9361's peak data rate, which is 122.88 MSps.

To enhance the development, debugging, and testing processes of the GNSS receiver, the prototype utilizes the Analog Devices ADRV1CRR-BOB breakout carrier board [140]. This board is equipped with standard connectors for essential functions such as power supply, Ethernet, and Universal Serial Bus (USB) connections, as well as Joint Test Action Group (JTAG) programming interfaces. It also includes various features to aid in debugging. Furthermore, the carrier board offers numerous access points for supplying external I/O bank voltages and for measuring the power consumption of the SoM, thereby facilitating comprehensive monitoring and testing of the system.

The proposed design utilizes the SoM's LAN interface, accessed through an RJ45 connector on the ADRV1CRR-BOB board, for remote control of the receiver and delivering GNSS products. The LAN interface ensures connectivity to the embedded GNSS receiver. Additionally, the SoM includes a USB On-The-Go (OTG) interface, accessible via a USB connector on the ADRV1CRR-BOB board, offering an alternative connection method. Moreover, custom serial interfaces can be developed using the board's General Purpose Input/Output (GPIO) expansion bus. All connections for the PS/PL and the power supply are efficiently consolidated into JX Micro Headers.

The clock reference for the front-end PLLs and the sample clock for the Analog-to-Digital Converters ADCs in the ADRV9361-Z7035 can be sourced from either an internal local oscillator or an external clock. The standard onboard clock has a stability of 25 parts per million (ppm), which is insufficient for GNSS signal processing. To address this issue, the original oscillator was substituted with a high-precision, temperature-compensated crystal oscillator from TXC Crystal, specifically the model TXC 7Q series designed for GPS applications [141]. This oscillator operates at 40 MHz and offers improved frequency stability of 0.5 ppm. There are also other alternatives available for enhancing clock stability.

The GNSS receiver prototype, including all its components, was assembled into a plastic enclosure for convenient handling and transport. This enclosure, shown in Figure 4.2 was specifically designed for this purpose and produced using a 3D printer, ensuring a custom fit for all the included components.

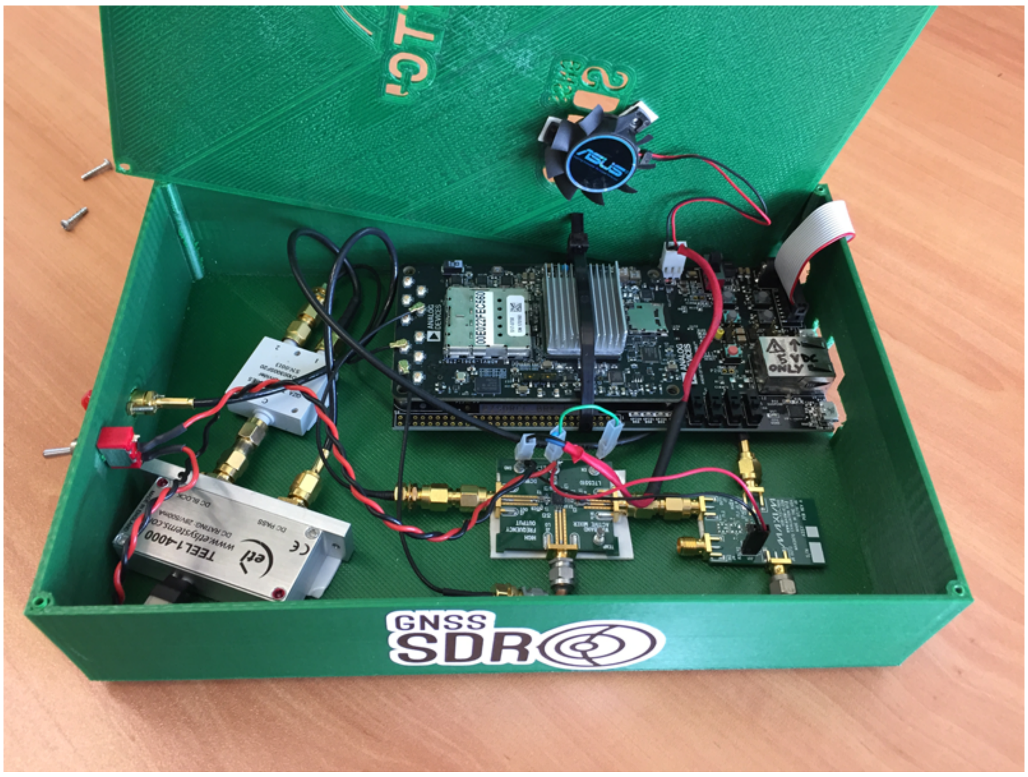

Figure 4.2 Laboratory prototype.

4.3.1.1 Radio Frequency Front-End

The RFFE is implemented using an Analog Devices AD9361 RF Agile Transceiver [106]. The AD9361 is a highly integrated dual-channel RF transceiver covering a wide frequency range from 70 MHz to 6 GHz, and supports tunable channel bandwidths from 200 kHz to 56 MHz. It features two independent direct conversion receiver channels, each with state-of-the-art noise figure and linearity. Each channel incorporates its own AGC, direct current offset correction, quadrature correction, and digital filtering, significantly reducing the need for these functions in the digital baseband. Additionally, two high dynamic range ADCs per channel digitize the received I and Q signals, which are then processed through configurable decimation filters and 128-tap finite impulse response filters to produce a 12-bit output signal at the desired sample rate.

The AD9361 RF Agile Transceiver [106] does not support tuning each channel to a different frequency band. This limitation comes from the fact that the AD9361 was originally designed for use in 3G and 4G base station applications. However, dual-band reception is needed to process GNSS signals both in the L1/E1 and in the L5/E5a frequency bands.

To enable dual-band reception, one of the AD9361's transmission channels is set to generate a Continuous Wave (CW) signal tuned at the difference between L1/E1 and L5/E5a center frequencies (that is, 398.97 MHz), and this signal is used to shift the L1/E1 band down to the L5/E5a center frequency using an external mixer. Hence, both AD9361 transmission channels can work at the same center frequency (that is, f_{L5} = 1176.45 MHz), while the whole system is still acting as a dual-band RF front-end. This is shown in Figure 4.3. The sampling frequency, configurable and consistent across both frequency bands, can be adjusted as needed.

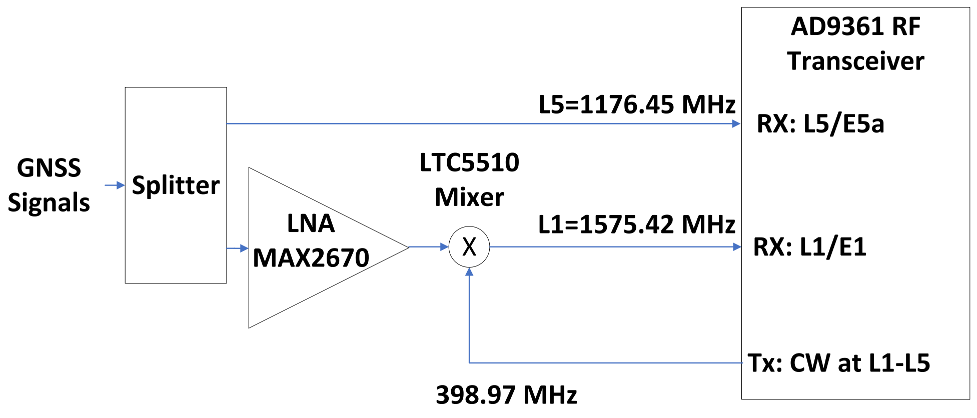

Figure 4.3 Making the AD9361 a dual frequency receiver.

4.3.1.2 FPGA Architecture

The FPGA implements the design as proposed in Section 3.1.1, following the outlined specifications and methodologies. It incorporates signal conditioning and buffering, an acquisition hardware accelerator, and 48 tracking multicorrelator hardware accelerators. These accelerators are distributed among 12 for GPS L1 C/A, 12 for Galileo E1b/c, 12 for GPS L5, and 12 for Galileo E5a, facilitating simultaneous tracking of up to 48 signals, in alignment with the specified design objectives.

In order to reduce the computational load and synchronize operations between the FPGA and the embedded processor, a dedicated interrupt request signal is routed from each FPGA hardware accelerator to the PS. The PS incorporates a PL390 Generic Interrupt Controller (GIC), as detailed in [102], capable of managing up to 16 interrupt signals from the PL. Given the need for more than 16 satellite channels, the capacity of the GIC is expanded by integrating two instances of AMD's Interrupt Controller IP core [142] within the FPGA, each configured with 32 interrupt request lines. This augmentation adds 64 interrupt lines in total, ensuring adequate support for all necessary Acquisition and Tracking blocks.

Figure 4.4 illustrates this setup, showing two AXI Interrupt Controllers that extend the PS's interrupt capacity by connecting to the PL's interrupt lines. The tracking multicorrelator hardware accelerator Interrupt Request (IRQ) lines are linked to the AXI Interrupt Controllers. Meanwhile, certain FPGA blocks, such as the acquisition block also depicted in Figure 4.4, are directly connected to the PL390 GIC in the PS. For the software running on the embedded processor, the connection method to the hardware accelerators—whether through the FPGA's AXI interrupt controllers or directly to the PL390 GIC in the PS—is transparent. Interrupt requests trigger callbacks within the software-implemented acquisition and tracking channels. These channels are responsible for reading the results and updating the parameters for the upcoming batch of samples, ensuring a seamless operation.

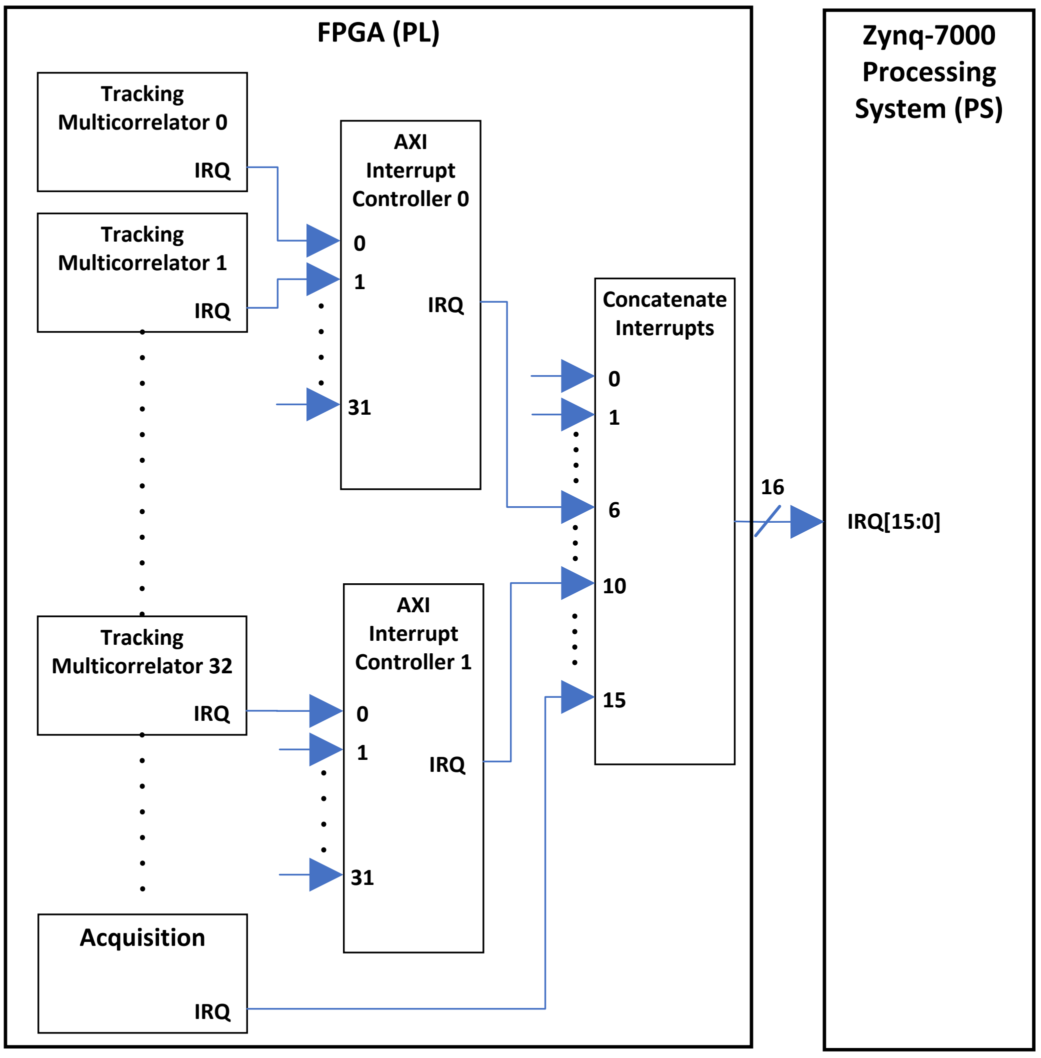
FPGA (PL)
Tracking Multicorrelator 0
IRQ
Tracking Multicorrelator 1
IRQ
Tracking Multicorrelator 32
IRQ
Acquisition
IRQ
AXI Interrupt Controller 0
0
1
31
IRQ
AXI Interrupt Controller 1
0
1
31
IRQ
Concatenate Interrupts
0
1
6
10
15
16
Zynq-7000 Processing System (PS)
IRQ[15:0]

Figure 4.4 Expansion of interrupt capacity through Multiple FPGA interrupt controllers

As explained in Sections 3.1.4 and 3.1.5, the acquisition and tracking multicorrelator hardware accelerators can be configured to use either 2 or 4 bits per sample.

Additionally, as discussed in Section 3.1.2, the tracking multicorrelators are preceded by sample buffers. Using a smaller bit depth in the tracking multicorrelators enables the storage of a larger number of samples in the FPGA sample buffers. For this reason, in the spaceborne GNSS receiver, the tracking multicorrelators are configured to use 2 bits per sample. This enables the storage of a larger number of samples in the FPGA buffers, whose size is limited by the FPGA memory resources. Specifically, as Section 2.3.3 describes, when the optimal quantization threshold values are used, the theoretical impact on signal degradation is limited to about 0.54 dB in the SNR at the correlator outputs. The acquisition does not require an extensive buffering at its input since it does not exert back pressure on the FPGA buffers. Therefore, quantization at the acquisition's input stage has less impact on the FPGA internal memory usage. For this reason, the acquisition is configured to use 4 bits per sample.

4.3.2 Implementation on the ZCU102 Board

A block diagram of the proposed ZCU102 board-based implementation is shown in Figure 4.5. The ZCU102 features the UltraScale+ XCZU9EG All-Programmable MPSoC, equipped with a quad-core ARM Cortex-A53 processor running at 1200 MHz, and an FPGA with a significant number of resources, comprising 600*k* logic cells and 2520 DSP slices [104]. A connection is provided for an antenna or a GNSS signal generator. A splitter optionally injects DC power to the active antenna and splits the signal in two. An Analog Devices AD-FMCOMMS5-EBZ RFFE receives the signals coming from the splitter. The AD-FMCOMMS5-EBZ , features two Analog Devices AD9361 RF Agile Transceivers [106]. One RF transceiver is tuned to the L1/E1 frequency band, and the other transceiver is tuned to the L5/E5a frequency band.

The SoC FPGA handles sample conditioning, acquisition, and tracking multicorrelators, in addition to providing an interface to the PS. The embedded processor executes GNSS-SDR, thereby effectively implementing the base-band signal processing system. This approach follows the guidelines of the proposed architecture as presented in Chapter 3, ensuring the delivery of GNSS products in standard formats via the LAN interface.

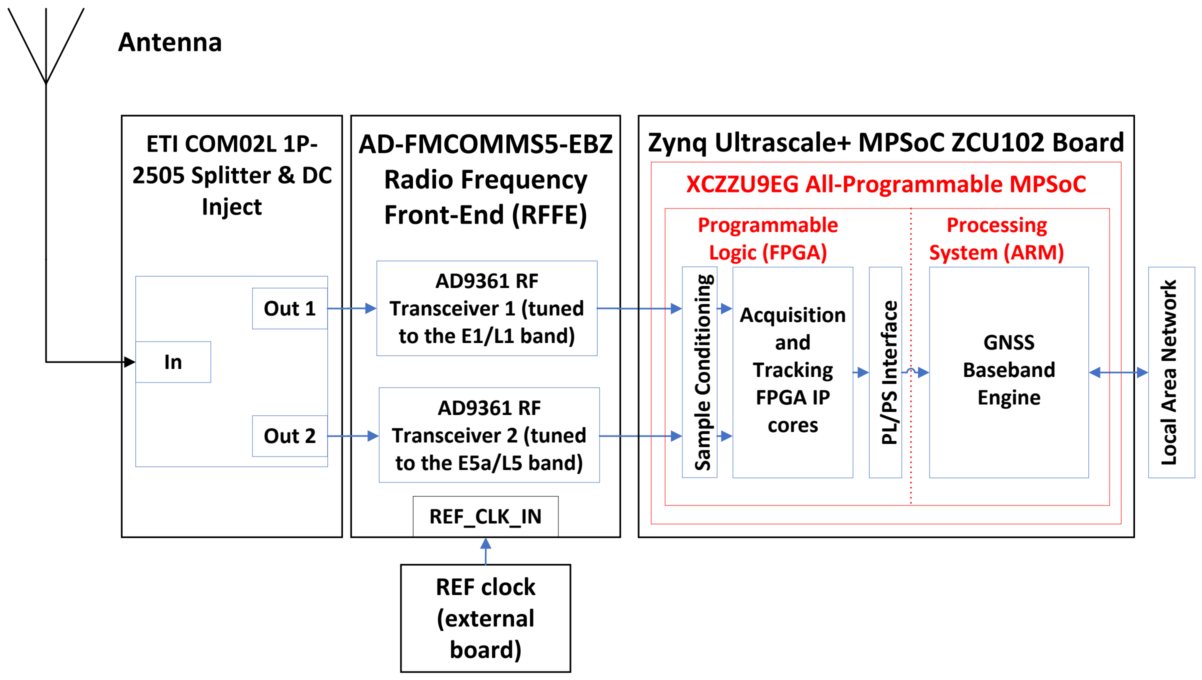
Antenna
ETI COM02L 1P-2505 Splitter & DC Inject
In
Out 1
Out 2
AD-FMCOMMS5-EBZ Radio Frequency Front-End (RFFE)
AD9361 RF Transceiver 1 (tuned to the E1/L1 band)
AD9361 RF Transceiver 2 (tuned to the E5a/L5 band)
REF_CLK_IN
REF clock (external board)
Zynq Ultrascale+ MPSoC ZCU102 Board
XCZZU9EG All-Programmable MPSoC
Programmable Logic (FPGA)
Processing System (ARM)
Sample Conditioning
Acquisition and Tracking FPGA IP cores
PL/PS Interface
GNSS Baseband Engine
Local Area Network

Figure 4.5 Spaceborne GNSS receiver block diagram using the ZCU102 development board and the AD-FMCOMMS5-EBZ Radio Frequency Front-End (RFFE).

The AD-FMCOMMS5-EBZ board is equipped with a Rakon RXO3225M IC crystal oscillator operating at 40 MHz and a frequency stability of 25 ppm. However, this level of stability is insufficient for GNSS signal processing requirements. Consequently, an external Abracon AST3TQ-50 Temperature-Compensated Crystal Oscillator (TCXO) was utilized as a reference clock, offering a frequency stability of ±50 parts per billion (ppb) across a wide operating temperature range from $-40°C$ to $+85°C$ [143], making it highly suitable for GNSS applications.

Both AD9361 RF transceivers were controlled by the reference clock. However, the relative timing of the sampling, controlled by the reference clock, was not calibrated.

Similarly to the ADRV9361-Z7035-based implementation, to alleviate the computational burden and synchronize operations between the FPGA and the embedded processor, a dedicated interrupt request signal is dispatched from each FPGA hardware accelerator to the PS. The Zynq UltraScale+ ZU9EG on the ZCU102 board incorporates a GIC-400 GIC. The PL is capable of asynchronously asserting up to 16 peripheral interrupts to the GIC as detailed in [63]. To enhance this capability, the FPGA integrates two instances of AMD's Interrupt Controller IP core [142], each equipped with 32 interrupt request lines. This configuration mirrors the setup described in Section 4.3.1.2.

The ZCU102 development board was chosen because it features a high-performance SoC FPGA with many peripherals and interfaces, enabling development and experimentation for a wide range of GNSS applications. However, utilizing the ZCU102 and AD-FMCOMMS5-EBZ development boards leads to a considerable design footprint. Approximately, the dimensions of the ZCU102 board are 23.8 cm by 24.4 cm, whereas the analog front-end spans 14 cm by 9 cm.

4.3.3 Tracking GNSS Signals in High-Dynamic Environments

The proposed spaceborne GNSS receiver implements standard scalar tracking loops using DLL, PLL, and FLL architectures for code and carrier phase tracking, as described in Section 2.3.5. During the tracking process, the receiver replicates the PRN code signal, applies Doppler correction, and estimates the signal synchronization parameters. It continuously adjusts the replica code delay and Doppler correction phase, utilizing these synchronization parameters to ensure synchronization with the incoming signal.

The receiver initiates tracking of detected signals without synchronization to the pilot's secondary code or the telemetry preambles. The synchronization parameters estimated by the acquisition process are used to initialize the tracking loops. The tracking process commences with a CI time equivalent to the duration of the PRN primary code. Once synchronization with the pilot's secondary code or the telemetry preambles is achieved, the tracking process dynamically increases the CI time based on user configuration.

Simultaneously, the receiver uses a dynamic Costas loop/PLL/DLL tracking method, where a Costas loop discriminator with a larger noise bandwidth is used before the pilot's or secondary code synchronization, to account for a potential parameter estimation error in acquisition. After synchronization, a narrower bandwidth is used to decrease the tracking jitter, dynamically replacing the Costas loop by a PLL when tracking pilot signals with known secondary code transitions. A FLL assisting the Costas loop can be enabled during the pull-in time to cope with the high-dynamic conditions.

This approach treats signal tracking as an estimation problem focusing on the signal synchronization parameters. These parameters are presumed to change slowly, remaining constant during an observation interval t_p (defined by the time range $t_p \leq t < t_p + T$) and potentially changing in the subsequent observation interval $p + 1$. Each tracking loop performs a new estimation of these synchronization parameters that are considered to remain constant during the tracking coherent integration time.

Similar to static receivers, the code loop replicates the PRN code considering the initial code phase and the chipping rate. However, to address the high dynamics in the received signals, the carrier loop not only takes into account the initial phase and the Doppler shift but also incorporates the Doppler rate, implementing a second-order NCO. The second-order NCO implementation is explained below.

The signal received from a GNSS satellite is modeled as

$$r(t) = a_0(t)e^{j\phi(t)} + w(t) = a_0[p]e^{j\sum_{m=0}^{M} b_m[p]t^m} + w(t)\ . \qquad t_p \leq t < t_p + T \tag{4.1}$$

In this equation, $a_0[p]$ represents the unknown constant amplitude within the time interval p, $b_m[p]$ for $m = \{0, .., M\}$ denotes the phase coefficients to be estimated for that period, and $w(t)$ is additive white Gaussian noise. This model represents the received signal in terms of the parameters $a_0[p]$ and $b_m[p]$, dictating the evolution of signal amplitude and carrier phase.

Based on this model, the polynomial phase term $\phi(t)$ can be approximated by its Taylor series expansion around the time instant t_n. This Taylor expansion can be expressed as

$$\phi(t) = \sum_{m=0}^{M} b_m[p]t^m \approx \sum_{m=0}^{M} \frac{1}{m!}\frac{\partial^m \phi(t_n)}{\partial t^m}(t - t_n)^m\ , \qquad t_p \leq t < t_p + T \tag{4.2}$$

where $\frac{\partial^m \phi(t_n)}{\partial t^m}$ represents the m-th order partial derivative of the function $\phi(t)$ with respect to time, evaluated at t_n.

As mentioned above, the creation of the carrier phase replica $\hat{\phi}(t)$ incorporates the initial phase, the Doppler shift and the Doppler rate. These correspond to the first three terms of the Taylor expansion. Setting $M = 2$ and evaluating $\phi(t)$ at $t = t_p$, $\phi(t)$ the evolution of the carrier phase can be approximated as

$$\phi(t) \approx \phi(t_p) + \frac{\partial \phi(t_p)}{\partial t}(t - t_p) + \frac{1}{2}\frac{\partial^2 \phi(t_p)}{\partial t^2}(t - t_p)^2\ . \tag{4.3}$$

By defining the sampling time as $t_n = t - t_p = nT_s$, we can express the phase term $\phi(t)$ in the following form:

$$\phi(t_p + nT_s) = \phi[n] \approx \phi(t_p) + \frac{\partial \phi(t_p)}{\partial t}nT_s + \frac{1}{2}\frac{\partial^2 \phi(t_p)}{\partial t^2}n^2T_s^2\ . \tag{4.4}$$

The initial phase, Doppler shift and Doppler rate synchronization parameters $\phi_0[p]$, $\omega_d[p]$, and $\alpha[p]$, can be defined as

$$\phi(t_p) \triangleq \phi_0[p] \qquad \text{Initial phase at interval } p\text{, in rad,} \tag{4.5}$$

$$\frac{\partial \phi(t_p)}{\partial t} \triangleq \omega_d[p] \qquad \text{Average Doppler shift during interval } p\text{, in rad/s,} \tag{4.6}$$

and

$$\frac{\partial^2 \phi(t_p)}{\partial t^2} \triangleq \alpha[p] \qquad \text{Average Doppler rate during interval } p\text{, in rad/s}^2. \tag{4.7}$$

The Doppler correction signal, locally generated within the receiver via the second-order NCO, is given by

$$v(t) = e^{-j\hat{\phi}(t)} \;, \tag{4.8}$$

where $\hat{\phi}(t)$ is the estimation of the phase evolution $\phi(t)$. The discrete time domain version of (4.8) using a sample period T_s, and a time origin t_p as a reference, can be represented as

$$v[n] = v(t_p + nT_s) = e^{-j\hat{\phi}(t_p+nT_s)} \;. \tag{4.9}$$

Upon integrating (4.4) into (4.9), we derive

$$v[n] = e^{-j\left(\hat{\phi}_0[p]+\hat{\omega}_d[p]nT_s+\frac{1}{2}\hat{\alpha}[p]n^2T_s^2\right)} \;, \tag{4.10}$$

where $\hat{\phi}_0[p]$, $\hat{\omega_d}[p]$ and $\hat{\alpha}[p]$ are the estimators of the corresponding true values in the interval p. These estimators are addressed in [144] and [145].

By generating the replica Doppler correction signal in this manner, the receiver is enabled to accurately track GNSS signals within high-dynamic environments, encompassing scenarios in LEO orbits.

4.3.4 Telemetry and Telecontrol Interfaces

The SoC FPGA is equipped with multiple communication interfaces to enable remote control of the receiver and retrieval of GNSS products. These interfaces include Ethernet (LAN) and UART. The embedded processor runs on the Geniux OS, which is a customization of GNU/Linux (see Section 3.2.2). This customization facilitates the implementation of standardized interfaces for reporting the receiver's status, accessing internal data, and controlling all receiver features through the LAN interface. Table 4.1 outlines the protocols and data available for interfacing the receiver for telecommand purposes, enhancing communication from the ground station to the satellite

Table 4.1 Telecommand (ground station to satellite)

Type of communication	Protocol and available data
Secure Shell (SSH)	• Secure File Transfer Protocol (SFTP) to upload updated/new receiver configuration files. • SFTP to upload acquisition assistance files – Cusotm XML files for satellite ephemeris. – Custom XML files for initial satellite position, velocity and time. • Receiver start/stop control • Remote firmware update (including FPGA Read Only Memory (ROM)) • Board reboot control

Table 4.2 displays the protocols and data available for Telemetry communication (satellite to ground station).

Table 4.2 Telemetry (satellite to ground station)

Type of communication	Protocol and available data
NMEA over Transmission Control Protocol (TCP) port	• Position • Position • Velocity • Time • Satellite visibility (Az, El) • Satellite C/N_0
RTCM/Network Transport of RTCM via Internet Protocol (NTRIP) over TCP port	• raw observables (code, pseudoranges, carrier phase, cycle slips, C/N_0) • Satellite ephemeris • Receiver time

Table 4.3 shows the protocols and data available for data logging, enabling the reporting of the internal status of the receiver's signal processing path.

Table 4.3 Embedded Data logger)

Type of communication	Protocol and available data
SSH	• Local log data files (.BIN and .MAT) – Acquisition log file – Tracking log file – Telemetry decoder log file – Observables log file – RTKLIB solver log file • NMEA log file • RTCM/NTRIP raw log file • KML Google Earth position file

4.4 Test Results

As detailed in Section 4.3, the proposed spaceborne GNSS receiver was implemented and tested on the ADRV9361-Z7035 and ZCU102 development boards. To ensure a comprehensive evaluation, some tests conducted initially with the ADRV9361-Z7035 were replicated on the ZCU102 board.

Several tests were conducted to thoroughly evaluate the receiver's capabilities and performance. These tests included its ability to capture GNSS signals and transfer the digitized signals to a remote PC in real-time, and the ability to operate in multi-frequency and multi-constellation modes in real-time. The acquisition and tracking sensitivity was also measured. Furthermore, assessments were made on the quality of observables. Finally, the power consumption was estimated, both when using the ADRV9361-Z7035 and ZCU102 development boards. The FPGA resource usage is also reported.

This section documents the testing procedures and outcomes for both boards. To streamline the discussion and focus on significant differences, only the results from the ZCU102 board are reported for tests that yielded more accurate data than those conducted with the ADRV9361-Z7035, except where power consumption and real-time performance are concerned. The receiver was tested using the GESTALT test-bed facility available at Centre Tecnològic de Telecomunicacions de Catalunya (CTTC) premises [146]. The GESTALT test-bed boasts an array of high-end equipment. This includes broadband GNSS antennas and GNSS signal generators designed for conducting controlled experiments. It is also outfitted with advanced radio-frequency front-ends, supporting simultaneous operation in three GNSS frequency bands, and featuring adjustable bandwidth, as well as options for frequency downshifting and filtering.

4.4.1 Test Results using the ADRV9361-Z7035 platform

This section summarizes the test results obtained when using the ADRV9361-Z7035 platform.

4.4.1.1 Raw Sample Storage

The raw sample storage test was conducted to verify the receiver's ability to capture I and Q samples post-ADC conversion and transfer them to a PC in real time. The receiver was connected to a signal generator, which reproduced live GNSS signals from replay files, simulating a static scenario. The receiver was also configured to capture both L1/E1 and L5/E5a signals in real time, transmitting them to a PC via TCP/IP, utilizing the LAN interfaces. The sampling rate was set to 12.5 Msps. The PC captured and stored these signals into files. The signals were recorded during 5 minutes.

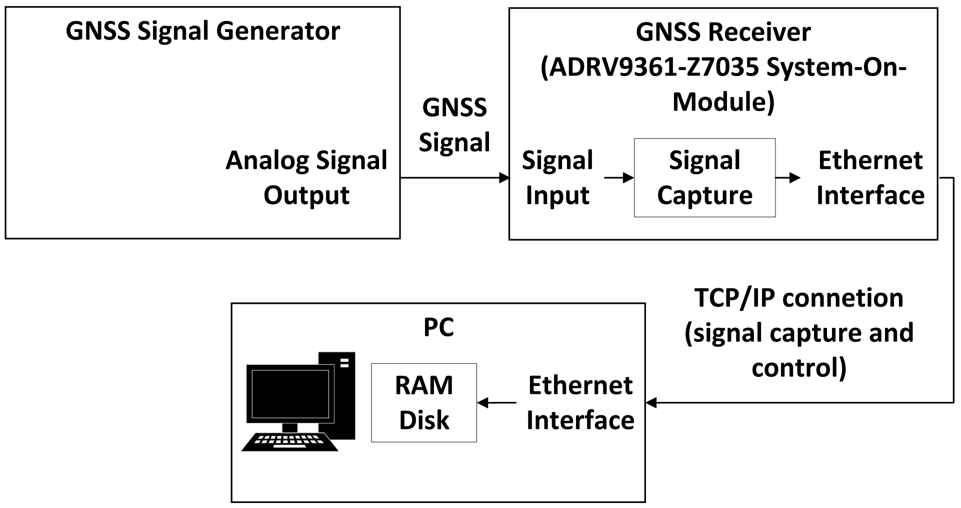

Figure 4.6 Sample capture

To confirm the correct capture of the signals, the files stored on the PC were post-processed using GNSS-SDR. GNSS-SDR effectively processed the recorded GNSS signals. It managed to simultaneously use up to 12 GPS L1 C/A channels, 12 Galileo E1 channels, 10 GPS L5 channels, and 11 Galileo E5a channels, successfully obtaining navigation solutions.

To transmit received signals in real-time and store them as files, a direct Ethernet connection between the receiver and the PC was necessary, along with the use of a RAM disk. This direct Ethernet connection was crucial to prevent signal transmission delays caused by LAN congestion. Additionally, the RAM disk was utilized to ensure fast I/O performance and prevent potential buffer overflow on the PC.

The signal capture capability is valuable for debugging and performance verification of the analog front-end. Although the need for fast throughput may complicate the real-time transmission of received signal samples to a remote location, this capability can in any case be effectively used to remotely capture brief snapshots of the received signal.

4.4.1.2 Multi-Frequency and Multi-Constellation Operating Mode

This test was conducted to assess the receiver's capability to process various satellite signals in real-time when using the ADRV9361-Z7035 platform. The receiver was connected to the rooftop antenna of the GESTALT test-bed facility [146] to enhance satellite visibility, utilizing a static scenario configuration. It was set to track GPS L1 C/A, Galileo E1b/c, GPS L5, and Galileo E5a signals simultaneously. The receiver was configured according to Appendix 4.A. In addition, it was configured to simultaneously track the same signals across multiple channels, ensuring that each channel was allocated to a satellite signal. The testing duration lasted for an hour. The largest number of channels that the receiver was able to process in real-time is shown in table 4.4.

Table 4.4 GNSS signal combinations tested in real-time using the ADRV9361-Z7035 platform

GNSS Signals	Total Number of Channels
6 GPS L1 C/A + 6 Galileo E1b/c + 6 GPS L5 + 6 Galileo E5a	24

This test was repeated using a LEO scenario generated by playing recorded files. The receiver successfully acquired and tracked up to 24 GNSS signals, delivering navigation solutions in real-time. In this case, however, the test duration was constrained to the length of the replay files (5 minutes). In any case this test demonstrates the receiver's capability to acquire and track GNSS signals in LEO orbits under real-time operations.

The number of signals the receiver could simultaneously track was restricted by the computational capabilities of the embedded processor. While the FPGA handled acquisition and tracking multi-correlations, the embedded processor managed the tracking loops, with the FPGA generating interrupt requests for this purpose. The rate of interrupt requests was directly related to the number of tracked channels, and an excessive number of interrupts could potentially overwhelm the embedded processor.

The ability to process up to 24 GNSS signals in real-time is contingent upon the computational load demanded by the PVT algorithms. This load varies depending on the satellites' geometry and the quality of the measurements. To enable stable real-time processing of up to 24 channels, the GNSS-SDR source code was temporarily modified. A software throttle was introduced to the PVT module, providing the embedded processor with additional processing time to handle pending interrupts from the hardware accelerators (FPGA). While this adjustment allowed for improved handling of processing demands, it also led to an increased duration for computing navigation solutions

Enhancing receiver performance could be achieved by adjusting the multicorrelator tracking hardware accelerators in the FPGA. By extending the coherent integration time within the FPGA beyond the duration of a single data symbol, the frequency of interrupt requests could be reduced. This is particularly beneficial when tracking Galileo E1b/c signals, which have a brief data symbol duration of 4 ms. Such an implementation would, as a result, enable the receiver to simultaneously track a greater number of GNSS signals.

4.4.1.3 Acquisition Sensitivity

Acquisition sensitivity determines the minimum signal power threshold that allows the receiver to successfully perform a cold start TTFF within a specified time frame [64].

To estimate the acquisition sensitivity, the receiver was connected to a signal generator. The signal generator played recorded GNSS signals. The receiver was configured to run in real-time and was set in cold start mode. We decreased the signal power and recorded the lowest C/N_0 for which the receiver could at some point in time successfully acquire and track any of the generated GNSS signals. The C/N_0 reported by the receiver was recorded when the receiver went from acquisition to tracking mode.

These measurements were repeated using both a static scenario and a high dynamic (LEO orbit) scenario. The results are shown in Table 4.5. The receiver was configured according to Appendix 4.A when using the static scenario, and Appendix 4.B when using the dynamic scenario.

Table 4.5 Acquisition sensitivity.

GNSS System	Acquisition Sensitivity (dB-Hz)
GPS L1 C/A (high-dynamic scenario)	38
GPS L1 C/A (static scenario)	37
Galileo E1b/c (high-dynamic scenario)	38
Galileo E1b/c (static scenario)	39
GPS L5 (high-dynamic scenario)	37
GPS L5 (static scenario)	38
Galileo E5a (high-dynamic scenario)	38
Galileo E5a (static scenario)	38

To provide context for these measurements, GNSS satellites in LEO orbits can experience C/N_0 values ranging from approximately 15 dB-Hz to 35 dB-Hz at negative elevation angles down to $-30°$, 20 dB-Hz to 45 dB-Hz at $0°$ elevation, and up to between 45 dB-Hz and 50 dB-Hz at the zenith for GPS L1 and L2 signals. [147]. In contrast, Geosynchronous Earth Orbit (GEO) applications using GPS and Galileo signals may require a sensitivity of about 29 dB-Hz to maintain nearly full-time GNSS POD availability (at least 4 satellites visible) [148]. The proposed receiver is suitable for use in LEO orbits; however, its current acquisition sensitivity is somewhat elevated. As a result, it requires additional time to acquire satellite signals with a C/N_0 below 40 dB-Hz that originate from low elevation angles. This issue stems from the CI time utilized by the current version of the spaceborne receiver's FPGA-based acquisition hardware accelerator. Specifically, this version performs integration over a single PRN code period and does not support extended coherent or non-coherent integration. It is important to note that the acquisition hardware accelerator utilized in the spaceborne receiver described in this chapter differs from the high-sensitivity acquisition hardware accelerator discussed in Chapter 6.

Nevertheless, additional tests using the GESTALT test-bed facility's rooftop GNSS antenna [146] showed that the receiver could acquire most of the visible GNSS satellites in line of sight.

The sensitivity can be improved by implementing extended coherent and/or noncoherent integration in the spaceborne receiver acquisition hardware accelerator in the FPGA.

4.4.1.4 Tracking Sensitivity

Tracking sensitivity refers to the minimum signal level that allows the receiver to maintain the tracking process in lock. The sensitivity was measured as follows: the receiver was connected to a signal generator. The signal generator was playing recorded GNSS signals. The receiver was configured to run in real-time, and it successfully acquired and tracked various satellite signals present in the playback. When the satellite signals had been acquired, the signal power was slowly decreased. The lowest C/N_0 for which the receiver could keep lock on the tracked signals was recorded. The recorded C/N_0 was reported by the receiver.

These measurements were repeated using both a static and a high dynamic (LEO orbit) scenario. The results are shown in Table 4.5. The receiver was configured according to Appendix 4.A when using the static scenario, and Appendix 4.B when using the dynamic scenario.

Table 4.6 Tracking sensitivity.

GNSS System	Tracking Sensitivity (dB-Hz)
GPS L1 C/A (high-dynamic scenario)	26
GPS L1 C/A (static scenario)	26
Galileo E1b/c (high-dynamic scenario)	27
Galileo E1b/c (static scenario)	28
GPS L5 (high-dynamic scenario)	29
GPS L5 (static scenario)	26
Galileo E5a (high-dynamic scenario)	29
Galileo E5a (static scenario)	27

The receiver currently exhibits adequate tracking sensitivity, enabling it to consistently track all visible GNSS satellites across the entire spectrum of elevation angles within its line of sight.

4.4.1.5 FPGA Resource Utilization

Table 4.7 details the FPGA resource utilization for the spaceborne receiver developed with the XC7Z035-2L device. This table reports the resource usage of the GNSS receiver features, including channel conditioning, buffering, and acquisition and tracking hardware accelerators, along with their AXI4 memory-mapped registers. The FPGA resource usage is categorized by LUTs, Lookup Table Random-Access Memorys (LUTRAMs), Flip-Flops (FFs), BRAMs, and DSP slices.

The BRAM blocks show the most significant usage. These blocks are mostly used for the implementation of the signal sample buffers, and for the temporary storage of the FFT and Inverse Fast Fourier Transform (IFFT) calculations in the acquisition hardware accelerator. The use of LUTs and FFs could be reduced by optimizing the HDL code, and by potentially reducing the internal quantization in the acquisition and tracking multicorrelator hardware accelerators. Currently the tracking multicorrelators use 32-bit accumulators, and the acquisition uses 16-bit FFTs, while the spaceborne design quantizes the GNSS signals with 4 bits per sample at the acquisition, and 2 bits/sample in the tracking multicorrelators. The FPGA-based GNSS receiver logic operates at a clock frequency of 100 MHz. This clock frequency is sufficient to operate the spaceborne receiver at the target sampling rate, but there is potential for increasing the clock frequency.

Table 4.7 FPGA resource utilization in the spaceborne receiver implemented on the XC7Z035 device.

Resource	**Utilization**	**Resources available**	**Utilization %**
LUT	141361	171900	82
LUTRAM	7662	70400	10
FF	141799	343800	41
BRAM	472	500	94
DSP	312	900	34

4.4.1.6 Power Consumption

The power consumption of the XC7Z035-2L FBG676I SoC FPGA when implementing the GNSS receiver was estimated to be 6.5 W, based on calculations from FPGA design tools. This estimate is based on an assumed average processor load of 75%, which is typical when handling a substantial number of GNSS signals, such as 24 signals. This results in a power consumption of approximately (6.5W)/(24 channels) = 0.27 W/channel.

To provide context, GNSS-SDR was tested operating in real-time, multi-band, multi-constellation mode on an 11th generation 28 W Intel Core i7-1185G7 processor clocked at 3 GHz. Processing 26 signals simultaneously resulted in a processor load ranging between 50% and 75%, leading to a power consumption of approximately (28W × 50%)/(26 channels) = 0.53 W/channel.

This outcome suggests a higher power consumption per channel for the processor setup compared to the SoC FPGA, indicating differences in power efficiency. Based on these estimations, the SoC FPGA exhibits an improvement in power consumption efficiency of approximately 49% (0.27 W/channel in the SoC FPGA vs 0.53 W/channel in the general-purpose processor). However, this figure does not account for the fact that processing up to 24 channels in real-time on the XC7Z035 SoC FPGA required throttling the computation of the PVT solution and reducing the rate at which the PVT solutions are computed from the default value of 500 ms down to 1 s. Therefore, the actual improvement in efficiency could be slightly lower than indicated.

To evaluate the average processor load during the processing of multiple signals, we connected the receivers to the rooftop GNSS antenna facility of the GESTALT testbed [146]. The receivers were configured to simultaneously track the same signals across multiple channels, ensuring that every channel tracked at least one satellite signal. We then estimated the average processor load using the operating system resource managers of the receivers.

4.4.2 Test Results using the ZCU102 platform

This section summarizes the test results obtained when using the ZCU102 platform and the Analog Devices' AD-FMCOMMS5-EBZ RFFE.

4.4.2.1 Multi-Frequency and Multi-Constellation Operating Modes

This test was conducted to assess the receiver's capability to process various satellite signals in real-time when using the ZCU102 platform, including GPS L1 C/A, Galileo E1b/c, GPS L5, and Galileo E5a. The receiver operated in selectable modes, accommodating single or dual frequencies, as well as single or dual constellations, with a potential of up to 12 channels per signal. This test was conducted with the GESTALT test-bed rooftop GNSS antenna facility [146] to enhance satellite visibility, thereby allowing the test to extend over a longer period compared to using recorded signals.

The receiver was allowed to simultaneously track the same signals across multiple channels, ensuring that each channel was allocated to a satellite signal. The testing duration lasted for an hour. The largest number of channels that the receiver was able to process in real-time is shown in table 4.4.

The receiver was configured according to a static scenario (Appendix 4.A). The sampling frequency was set to 16 Msps.

Table 4.8 GNSS signal combinations tested using the ZCU102 platform

GNSS Signals	Total Number of Channels
12 GPS L1 C/A	12
12 Galileo E1b/c	12
12 GPS L5	12
12 Galileo E5a	12
12 GPS L1 C/A + 12 Galileo E1b/c	24
12 GPS L5 + 12 Galileo E5a	24
12 GPS L1 C/A + 12 GPS L5	24
12 Galileo E1b/c + 12 Galileo E5a	24
10 GPS L1 C/A + 10 Galileo E1b/c + 10 GPS L5 + 10 Galileo E5a	40

Similar to the ADRV9361-Z7035 platform, the number of signals that the receiver could track simultaneously on the ZCU102 platform was limited by the computational capabilities of the embedded processor. However, the ZCU102 platform boasts a more powerful Zynq UltraScale+ ZU9EG SoC-FPGA, which includes a quad-core ARM Cortex-A53 processor running at 1200 MHz. This is a significant upgrade from the dual-core ARM Cortex-A9 processor at 800 MHz found in the Xilinx Zynq-7000 XC7Z035 SoC-FPGA of the ADRV9361-Z7035 platform.

The more powerful processor is the reason why the spaceborne GNSS receiver implemented on the ZCU102 board can simultaneously track up to 40 GNSS signals and obtain navigation solutions in real-time, while the same receiver implemented on the ADRV9361-Z7035 platform can only track up to 24 channels. The temporary software throttle in the PVT that was used when testing the ADRV9361-Z7035 platform was not required on the ZCU102.

4.4.2.2 Observables Quality

This test involved measuring the Root Mean Square Error (RMSE) of both the receiver's carrier phase estimation and its code phase estimation, and then comparing these measurements against the theoretical variances of the carrier and code phase estimates.

A software GNSS simulator available online under the GPL v3 license [149] was used for this test.

The RMSE of the receiver carrier phase estimation and the RMSE of the receiver code phase estimation were measured as follows: We produced synthetic GPS L1 C/A signals at various SNRs using the software simulator mentioned above. The satellite signals were generated in pairs, using identical ephemeris information and satellite locations but different PRN numbers for each pair of satellites. The synthetic signals were generated in baseband and stored in files (no RF was used). The GNSS receiver processed these files and produced the basic observable measurements (pseudorange, carrier phase, and Doppler shift). The observables were stored in RINEX files. Then, the between-satellites single difference was computed for each satellite pair. In this way, the code delay errors and the code phase estimation errors were obtained. The RMSE of the carrier phase estimations and the RMSE of the code phase estimations were computed out of the estimation errors. Finally, the RMSE measurements were compared against the theoretical variance of the carrier phase and code phase estimates.

The theoretical variance of the carrier phase delay estimates was computed as

$$\sigma_{PLL} = \frac{\lambda_{L1}}{2\pi}\sqrt{\frac{BW_{PLL}}{C/N_0}(1+\frac{1}{2T_{CI}C/N_0})}\ . \qquad [\mathrm{m}] \qquad (4.11)$$

This equation can be used to compute the thermal noise carrier tracking jitter when using an arctangent PLL discriminator [25].

Conversely, the theoretical variance of the code phase delay estimates was derived as

$$\sigma_{DLL} = \begin{cases} cT_c\sqrt{\frac{BW_{DLL}}{2C/N_0}\delta(1+\frac{2}{T_{CI}(C/N_0)(2-\delta)})} & \text{if } \delta \geq \frac{\pi R_c}{B_{fe}} \\ cT_c\sqrt{\frac{BW_{DLL}}{2C/N_0}(\frac{1}{B_{fe}T_c}+\frac{B_{fe}T_c}{\pi-1}(\delta-\frac{1}{B_{fe}T_c})^2)(1+\frac{2}{T_{CI}(C/N_0)(2-\delta)})} & \text{if } \frac{R_c}{B_{fe}} < \delta < \frac{\pi R_c}{B_{fe}} \\ cT_c\sqrt{\frac{BW_{DLL}}{2C/N_0}(\frac{1}{B_{fe}T_c})(1+\frac{1}{T_{CI}C/N_0})} & \text{if } \delta \leq \frac{R_c}{B_{fe}} \end{cases} . \quad [\text{m}] \quad (4.12)$$

The thermal noise code tracking jitter can be determined using this equation when employing a noncoherent early-late power DLL discriminator. This formula is specifically applicable to DSSS signals generated using BPSK signaling with rectangular chips, such as the GPS L1 C/A code [25].

In all cases, thermal noise is considered the exclusive source of error.

The parameters in (4.11) and (4.12) are the following: λ_{L1} is the GPS L1 carrier wavelength in m, c is the speed of light in vacuum, C/N_0 is the carrier to noise density ratio, T_{CI} is the pre-detection integration time in s, δ is the early-to-late correlator spacing in chips, BW_{DLL} is the PLL bandwidth in Hz, BW_{DLL} is the DLL bandwidth in Hz, B_{fe} is the front-end bandwidth in Hz (which in our case it is the same as the sampling frequency), and T_c is the chip period in seconds = $1/R_c$, where R_c is the chipping rate.

The theoretical variance of the code phase estimation was also computed as

$$\sigma_{DLL} = cT_c\sqrt{\frac{BW_{DLL}}{2C/N_0}\delta(1+\frac{1}{T_{CI}C/N_0})} . \quad [\text{m}] \quad (4.13)$$

This equation offers a simplified calculation applicable when $BW_{DLL}T_{CI} \ll 1$ [150]. The GNSS receiver was configured with $DLL_{BW} = 5$ Hz and $T_{CI} = 20$ ms. Therefore, since $DLL_{BW}T_i \ll 1$, (4.13) is applicable.

The receiver was configured according to Appendix 4.A, but only GPS L1 C/A channels were enabled. In this test, the receiver was configured with 16 GPS L1 C/A channels to facilitate between-satellite single-difference measurements, utilizing some of the tracking multicorrelator hardware accelerators typically assigned to Galileo E1b/c signals for GPS. The sampling frequency was set to 12.5 Msps, which is sufficient to process GPS L1 C/A signals.

The RMSE measurements were conducted using both the SoC FPGA receiver and the GNSS-SDR receiver operating on a Personal Computer (PC) in pure software mode, without the involvement of any hardware accelerators. The FPGA hardware accelerators perform calculations using fixed-point arithmetic, whereas the computations on PCs and embedded processors are carried out using floating-point arithmetic. The employment of fixed point arithmetic in the FPGA can lead to reduced performance, owing to signal and variable quantization. The results obtained using the SoC FPGA receiver and using GNSS-SDR in pure software mode were compared to estimate possible implementation losses caused by the fixed point arithmetic in the FPGA.

The accuracy of the GNSS simulator used for this test has not been verified with measurements. One method to assess it would be conducting a comparative test with a commercial receiver, although this approach might be impractical due to the limited flexibility of commercial receivers. Alternatively, the simulator's accuracy could be verified by processing the generated signals, using known carrier and code phase timings, and analyzing these signals in Matlab.

The results are explained in the subsections below.

4.4.2.2.1 RMSE of the Carrier Phase Estimation

Figure 4.7 illustrates the RMSE measured for the carrier phase, alongside the theoretical variance of the carrier phase estimates. The RMSE measured using the FPGA-SoC receiver is shown in circles. The RMSE measured when using GNSS-SDR running in software mode (no hardware accelerators involved) is shown in stars.

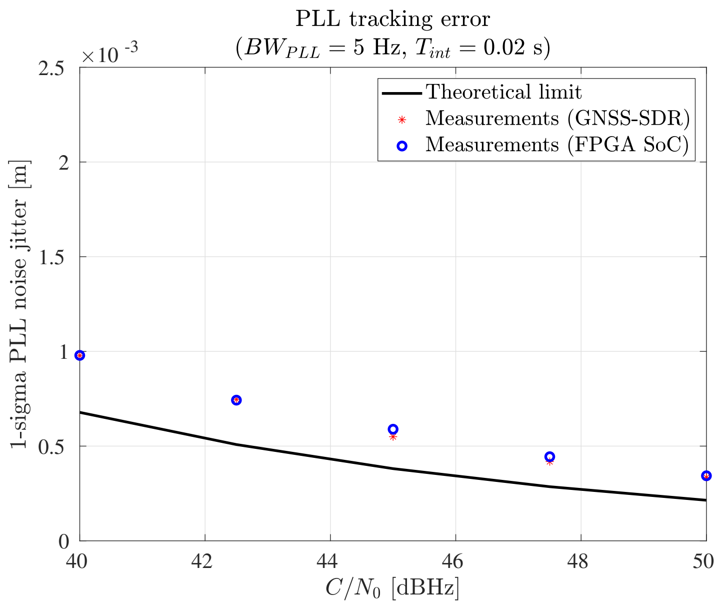

Figure 4.7 Receiver carrier phase RMSE (1-sigma) when using the FPGA-SoC (stars) and when using GNSS-SDR in software mode (circles).

Figure 4.7 indicates an implementation loss between 0.3 dB and 0.5 dB when employing the FPGA hardware accelerators with respect to GNSS-SDR running in pure software mode. This phenomenon may be attributed to the fixed point arithmetic within the FPGA or the action of the FPGA's dynamic bit selector. The tracking hardware accelerators employ two-bit signal quantization. The FPGA's dynamic bit selector dynamically adjusts the quantization of the recorded GNSS signals to match the dynamic range of the acquisition and tracking hardware multicorrelators. The aim is to achieve the most efficient quantization possible. The theoretical variance of the carrier phase estimates shown in Figure 4.7 only considers the presence of thermal noise. However, the difference between the measured values and the theoretical variance suggests that enhancements could be made to refine the estimator's variance.

4.4.2.2.2 RMSE of the Code Phase Estimation

Figure 4.8 illustrates the RMSE measured for the code phase estimation, alongside the theoretical variance of the code phase estimates. The RMSE measured using the FPGA-SoC receiver is shown in circles. The RMSE measured when using GNSS-SDR running in software mode (no hardware accelerators involved) is shown in stars. The theoretical variance of the code phase estimates was computed using (4.12). The simplified form (4.13) was also employed for comparative analysis.

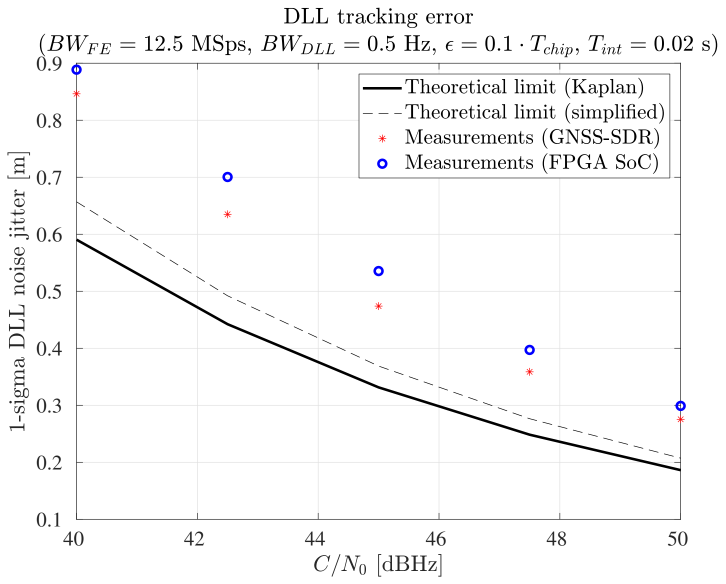

Figure 4.8 Receiver code phase RMSE (1-sigma) when using the FPGA-SoC (stars) and when using GNSS-SDR in software mode (circles).

Figure 4.8 suggests a reduction in performance by approximately 0.5 dB and 1 dB when utilizing the FPGA hardware accelerators compared to GNSS-SDR running in pure software mode. The loss may be explained by the quantization of the signals and variables that take place in the FPGA hardware accelerators. The theoretical variance of the code phase estimation shown in Figure 4.8 only considers the presence of thermal noise. However, the difference between the measured values and the theoretical variance suggests that enhancements could be made to refine the estimator's variance.

4.4.2.3 Precision of Navigation Solutions in Real-Time Mode Using a Static Scenario

This test involved measuring the precision of the navigation solutions provided by the receiver in real-time mode and using a static scenario. In this context, *precision* refers to the proximity of a solution to the mean of all obtained solutions, which indicates repeatability and the spread of the measurement. The precision of these navigation solutions was assessed using the standard positioning precision measurements detailed in Section 2.7.4 and their corresponding static confidence regions. These measurements included DRMS and CEP for 2D positioning, as well as SAS, MRSE, and SEP for 3D positioning. The DRMS and CEP measurements are detailed in Table 2.10, whereas the SAS, MRSE, and SEP measurements are detailed in Table 2.11, in Chapter 2.

The latitude, longitude, and height coordinates obtained by the receiver were converted to a local ENU coordinate system. The ENU system was anchored to a selected reference point in proximity to the receiver's antenna, employing a WGS-84 reference ellipsoid. The means and standard deviations were computed as detailed in section 2.7.4.

The measurements were performed using the GESTALT test-bed rooftop antenna [146]. The receiver was configured to process the incoming signals in real-time in multi-frequency and multi-constellation mode, using GPS L1 C/A, Galileo E1b+c, GPS L5, and Galileo E5a signals. The receiver was configured according to Appendix 4.A and was set to dump the PVT solutions onto a file. The sampling frequency was set to 16 Msps.

The PVT block was configured to use single-point positioning mode. During the first 15 seconds, the navigation solutions were excluded from consideration to allow the receiver to reach a steady state. Two tests were carried out, each one lasting for 10 minutes. The 2D precision results are shown in Table 4.9, and the 3D precision results are shown in Table 4.10.

Table 4.9 2D precision results.

Measure	Results (test 1) [m]	Results (test 2) [m]	Confidence Region Probability
2DRMS	6.9	4.9	95 %
DRMS	3.4	2.4	65 %
CEP	2.8	2.0	50 %

Table 4.10 3D precision results.

Measure	Results (test 1) [m]	Results (test 2) [m]	Confidence Region Probability
99 % SAS	9.6	8.6	99 %
90 % SAS	7.1	6.4	90 %
MRSE	5.1	4.9	61 %
SEP	4.3	3.9	50 %

The 2D precision measurements are more precise than the 3D precision measurements. This is expected, as GNSS performs better in the horizontal plane than in the vertical plane. This discrepancy arises from the angle between the line of sight to different GPS satellites and the ground. More accurate results may be obtained by conducting extended measurements.

4.4.2.4 Accuracy of Navigation Solutions in Post-Processing Mode Using a LEO Scenario

This test involved measuring the accuracy of the navigation solutions provided by the receiver in post-processing mode and using a LEO scenario. The accuracy of these navigation solutions was assessed using the standard positioning accuracy measurements detailed in Section 2.7.5 and their corresponding static confidence regions. These measurements included DRMS and CEP for 2D positioning, as well as SAS, MRSE, and SEP for 3D positioning. The DRMS and CEP measurements are detailed in Table 2.12, whereas the SAS, MRSE, and SEP measurements are detailed in Table 2.13, in Chapter 2.

A signal generator produced a synthetic LEO scenario and a reference motion file. The reference motion file contained the reference positions and timings of the LEO scenario. The receiver was configured to work in high dynamics scenarios. Appendix 4.B shows the receiver configuration for the LEO scenario. The sampling frequency was set to 12.5 MSps. The receiver processed the LEO synthetic scenario in post-processing mode, using recorded files containing the GNSS signals. The recorded files had a duration of five minutes.

The PVT solutions produced by the receiver were recorded in files. These PVT solutions were compared against the solutions stored in the reference motion file. The RMSE of the PVT solutions was computed using the ECEF coordinate system, along with the mean error and the standard deviation. The results are shown in Table 4.11.

Table 4.11 PVT accuracy using a LEO scenario.

Measure	Result
3D Position RMSE	1.2 m
3D Position mean error	1.1 m
3D Position standard deviation	0.5 m
3D Velocity RMSE	0.2 m/s
3D Velocity mean error	0.2 m/s
3D Velocity standard deviation	0.1 m/s

The navigation solutions appear to be remarkably precise. The obtained 3D position RMSE was 1.2 m and the 3D Velocity RMSE was within 0.2 m/s. These results are better than the results obtained when testing the receiver in a static scenario with live signals using the rooftop GESTALT antenna (see subsection 4.4.2.3). The main reason why this might be the case is that the RFFE was not used when testing the accuracy of the LEO scenario. The AD-FMCOMMS5-EBZ RFFE contains two independent RF transceivers, each one tuned to a different frequency band (L1/E1 and L5/E5a). The RF transceivers were not calibrated for the tests using live signals in a static scenario. Thus, a minor time delay between the L1/E1 and the L5/E5a signals might be degrading the quality of the PVT solutions when using live signals.

The accuracy measurements were conducted using recorded files that lasted for 5 minutes. More accurate measurements could be obtained by conducting longer tests.

4.4.2.5 FPGA Resource Utilization

Table 4.12 details the FPGA resource utilization for the spaceborne receiver developed with the XCZU9EG device. This table reports the resource usage of the GNSS receiver features, including channel conditioning, buffering, and acquisition and tracking hardware accelerators, along with their AXI4 memory-mapped registers. The FPGA resource usage is categorized by LUTs, LUTRAMs, FFs, BRAMs, and DSP slices.

The XCZU9EG device is larger than the XC7Z035, leading to a lower FPGA occupancy percentage. Compared to the resource utilization of the XC7Z035-based design shown in Section 4.4.1.5, the XCZU9EG-based design uses slightly less LUTs, but almost twice the number of DSP slices. The reason for this is that the XCZU9EG-based design uses a less-optimized acquisition hardware accelerator, which uses a larger number of DSP slices for computations that can be done using the more abundant LUTs. The XCZU9EG belongs to a more advanced SoC FPGA family than the XCZU9EG device. This can also influence variations in the utilization report.

Equally to the XC7Z035-based design, as explained in Section 4.4.1.5, the use of LUTs and FFs could be reduced by optimizing the HDL code, and by potentially reducing the internal quantization in the acquisition and tracking multicorrelator hardware accelerators. The FPGA-based GNSS receiver logic operates at a clock frequency of 150 MHz. This clock frequency is sufficient to operate the spaceborne receiver at the target sampling rate, but there is potential for increasing the clock frequency.

Table 4.12 FPGA resource utilization in the spaceborne receiver implemented on the XCZU9EG device.

Resource	Utilization	Resources available	Utilization %
LUT	126270	274080	46
LUTRAM	6637	144000	5
FF	166510	548160	30
BRAM	477.5	912	52
DSP	732	2520	29

4.4.2.6 Power Consumption

The power consumption of the Zynq UltraScale+ XCZU9EG-FFVB1156-2-e SoC FPGA when implementing the spaceborne GNSS receiver is estimated to be 8 W using the FPGA design tools. This estimation assumes an average processor load of 75%, which is typical when handling a substantial number of GNSS signals, such as 40 signals. This results in an estimated power consumption of approximately (8 W)/(40 channels) = 0.2 W/channel.

As discussed in Section 4.4.1.6, tests conducted with an 11th-generation 28 W Intel Core i7-1185G7 processor operating at 3 GHz estimated an energy efficiency of approximately 0.53 W/channel. Based on these estimations, the SoC FPGA exhibits an increase in power consumption efficiency of approximately 62% (0.27 W/channel in the SoC FPGA vs 0.53 W/channel in the general-purpose processor).

Compared to the results detailed in Section 4.4.1.6 for the Zynq-7000 XC7Z035 device, the estimated power consumption efficiency of the Zynq UltraScale+ XCZU9EG is consistent with the expectation that Zynq UltraScale+ devices generally exhibit lower power consumption compared to Zynq-7000 devices with similar performance levels. This alignment is attributed to the more advanced process technology utilized in the Zynq UltraScale+ series.

4.5 Conclusions

This chapter has detailed the design, proof-of-concept implementation, and performance assessment of a cost-effective, software-defined spaceborne GNSS receiver. It offers practical details into the construction and functionality of a working prototype.

This design highlights the efficacy of SDR techniques for high-dynamic scenarios, underscoring the practicality of the proposed architecture and design methodology detailed in Chapter 3, which utilizes SoC FPGA technology. This approach not only proves the suitability of SDR in demanding conditions but also showcases the strategic use of SoC FPGA to create a low-power and highly customizable receiver. Capable of processing GNSS signals in real time and suited for space-like environments, this receiver serves as a pertinent application for the proposed architecture. Furthermore, the adoption of a FOSS GNSS baseband processing engine introduces a collaborative and transparent development environment, accelerating innovation and allowing a diverse range of contributors to enhance the technology. This combination of low-cost design, advanced processing capabilities, and an open-source development model represents a significant novelty in the field, opening new possibilities for satellite navigation systems.

The receiver was tested in both static and LEO scenarios, demonstrating its capability to operate in real-time dual-frequency, dual-constellation mode. It successfully acquires and tracks GNSS signals under LEO conditions, delivering highly accurate navigation solutions based on processed recorded signals in these scenarios.

Utilizing the ADRV9361-Z7035 development board, which measures 10 cm by 6.2 cm, serves as a proof of concept for the feasibility of designing receivers compact enough for integration into CubeSats. Given that CubeSats typically consist of multiples of $10 \times 10 \times 10$ cm cubic units, the compact design of this board illustrates the potential for creating such receivers that meet CubeSat size constraints.

The implementation using the ZCU102 development board and the AD-FMCOMMS5-EBZ RFFE leads to a significantly larger design, with the ZCU102 approximately measuring 23.8 cm by 24.4 cm, and the RFFE about 14 cm by 9 cm. However, the ZCU102 board encompasses various components not required for the spaceborne receiver's functionality. Thus, there's potential to minimize the receiver's overall size through the development of a specialized Printed Circuit Board (PCB). The ZCU102-based implementation showcases the design portability of the proposed architecture across various SoC-FPGA variants with different capabilities.

Potential improvements for the current prototypes are related to their acquisition sensitivity and the timing calibration of the AD-FMCOMMS5-EBZ dual RF front-end. Additionally, the receiver employing the ADRV9361-Z7035 board faces limitations in its capability to track a significant number of channels simultaneously during real-time operations.

Currently, the acquisition sensitivity of the spaceborne receiver ranges between 37 and 38 dB-Hz, varying with the specific signal acquired. This limitation stems from the use of a standard-sensitivity acquisition hardware accelerator in the spaceborne receiver, implemented as described in Section 3.1.4, which lacks the implementation of coherent integration and Post-Detection Integration (PDI) techniques. Future efforts will focus on upgrading this hardware accelerator to incorporate coherent integration and PDI techniques, aiming to enhance the acquisition sensitivity to levels below the current range.

In addition, although both RF transceivers on the AD-FMCOMMS5-EBZ board are synchronized with the reference clock, their sampling's relative timing has not been calibrated. Consequently, a slight time delay between the L1/E1 and L5/E5a signals could affect the PVT solutions' quality when operating in dual-frequency mode. Future work also involves implementing a calibration routine for dual-frequency mode.

Finally, when using the ADRV9361-Z7035 development board, the embedded processor's computational capacity limits the receiver's ability to simultaneously track more than 24 signals. Still, this performance was achieved by temporarily implementing a throttle in the GNSS-SDR's PVT block, specifically designed to lower the embedded processor's average computational load during the PVT module's operation. This was not the case when using the ZCU102 board, which was capable of tracking up to 40 signals simultaneously. This limitation in the ADRV9361-Z7035 board mainly arises from the embedded processor's limited ability to manage a large volume of interrupts originating from several tracking multicorrelator hardware accelerators at the same time. It is possible to enhance receiver performance by making adjustments to the multicorrelator tracking hardware accelerators in the FPGA. Extending the coherent integration time within the FPGA beyond that of a single data symbol would lead to a reduction in the frequency of interrupt requests, especially when tracking Galileo E1b/c signals, known for their brief data symbol duration of 4 ms. Optimizing the embedded processor software might also lead to improved performance. Implementing these strategies would enable the receiver to track more GNSS signals simultaneously.

These aspects delineate critical areas for future research and optimization to enhance the receiver's overall performance and utility.

Nevertheless, the receiver showcases the practicality of developing affordable GNSS receivers using SoC FPGAs for processing GNSS signals within LEO orbits.

The results presented in this chapter were partially published in:

- [20] M. Majoral, C. Fernández-Prades, and J. Arribas, "A Flexible System-on-Chip Field-Programmable Gate Array Architecture for Prototyping Experimental Global Navigation Satellite System Receivers," *Sensors*, vol. 23, no. 23, 2023, Art. no. 9483. doi: 10.3390/s23239483

- [22] C. Fernández-Prades, J. Arribas, M. Majoral, A. Ramos, J. Vilá-Valls, and P. Giordano, "A Software-Defined Spaceborne GNSS Receiver," in *2018 9th ESA Workshop on Satellite Navigation Technologies and European Workshop on GNSS Signals and Signal Processing (NAVITEC)*, Noordwijk, Netherlands, December 2018, pp. 1–9. doi: 10.1109/NAVITEC.2018.8642697

The work presented in this chapter was supported by the following funding sources and collaborators:

- Contract AO/2-1647/17/NL/CRS with the European Space Agency.
- Grant CPP2021-008648, titled “Receptor Abierto y Flexible de Navegación por Satélite”, funded by the Ministry of Science and Innovation.
- Grant PID2021-128373OB-I00, funded by the Spanish Ministry of Science and Innovation through MCIN/AEI/10.13039/501100011033, and by the European Regional Development Fund (ERDF) “A way of making Europe”.

Appendix 4.A Receiver Configuration for Static Scenarios

This section describes the receiver configuration used to achieve the experimental results in static scenarios. Table 4.13 shows the configuration of the acquisition block. Table 4.14 and Table 4.15 show the configuration of the Tracking blocks. Finally, Table 4.16 shows the configuration of the PVT block. By default, the sampling frequency is set to 12.5 Msps.

Table 4.13 outlines the receiver acquisition parameters. It includes the *Doppler max*, representing the highest Doppler frequency in the search space, and the *Doppler step*, which refers to the increment between frequencies within the search grid. Self-assistance to acquisition from primary to secondary band is enabled. Therefore, once the primary band is successfully acquired, the receiver predicts the Doppler frequency in the secondary band. Subsequently, it performs acquisition in the secondary band with a reduced Doppler search grid. The threshold parameter in Table 4.13, denoted as Γ in Algorithm 1 in Chapter 3, represents the decision threshold above which a signal is considered present. A downsampling filter is used in the L1/E1 band to reduce the acquisition latency. The downsampling filter is only used in the acquisition.

Table 4.13 Spaceborne receiver acquisition configuration for static scenarios.

Parameter type	Parameter	Value
Acquisition GPS L1 C/A	Doppler Max	5000 Hz
	Doppler Step	500 Hz
	Threshold	2.5
	Downsampling Factor	4
Acquisition GPS L5	Doppler Max	500 Hz
	Doppler Step	250 Hz
	Threshold	2.5
	Assistance to acquisition from primary to secondary band	Enabled
Acquisition Galileo E1b/c	Doppler Max	5000 Hz
	Doppler Step	250 Hz
	Threshold	2.5
	Downsampling Factor	4
Acquisition Galileo E5a	Doppler Max	500 Hz
	Doppler Step	125 Hz
	Threshold	2.5
	Assistance to acquisition from primary to secondary band	Enabled

Table 4.14 and Table 4.15 display the receiver tracking configuration parameters for the GPS and Galileo signals, respectively. The coherent integration time is set to 20 ms to increase the apparent signal-to-noise ratio. The early-late space chips is the spacing between the Early and Prompt, and between the Prompt and Late correlators, normalized by the chip period. The early-late narrow space chips is the spacing between the Early and Prompt, and between the

Prompt and Late correlators, normalized by the chip period, after bit synchronization. The very early-late space chips is the spacing between the Very Early and Prompt and between the Prompt and Very Late correlators, normalized by the chip period The very early late space narrow chips is the spacing between the Very Early and Prompt, and between the Prompt and Very Late correlators after removal of the secondary code and extension of the coherent integration time, normalized by the chip period. The PLL filter bandwidth is the bandwidth of the PLL low-pass filter. The PLL filter bandwidth with narrow correlator configuration is the bandwidth of the PLL low-pass filter after removal of the secondary code. The DLL filter bandwidth is the bandwidth of the DLL low pass filter. Finally, the DLL filter bandwidth with narrow correlator configuration is the bandwidth of the DLL low-pass filter after the removal of the secondary code.

Table 4.14 Spaceborne receiver GPS tracking configuration for static scenarios.

Parameter type	Parameter	Value
Tracking GPS L1 C/A	coherent integration time	20 ms
	Early-Late space chips	0.5 chips
	Early-Late space narrow chips	0.1 chips
	PLL filter bandwidth	35 Hz
	PLL filter bandwidth (narrow correlator configuration)	7.5 Hz
	DLL filter bandwidth	2 Hz
	DLL filter bandwidth (narrow correlator configuration)	0.5 Hz
Tracking GPS L5	coherent integration time	20 ms
	Early-Late space chips	0.5 chips
	Early-Late space narrow chips	0.1 chips
	PLL filter bandwidth	20 Hz
	PLL filter bandwidth (narrow correlator configuration)	7.5 Hz
	DLL filter bandwidth	1.5 Hz
	DLL filter bandwidth (narrow correlator configuration)	0.5 Hz

Table 4.15 Spaceborne receiver Galileo tracking configuration for static scenarios.

Parameter type	Parameter	Value
Tracking Galileo E1b/c	coherent integration time	20 ms
	Early-Late space chips	0.25 chips
	Very Early-Late space chips	0.5 chips
	Early-Late space narrow chips	0.15 chips
	Very Early-Late space narrow chips	0.5 chips
	PLL filter bandwidth	15 Hz
	PLL filter bandwidth (narrow correlator configuration)	7.5 Hz
	DLL filter bandwidth	0.75 Hz
	DLL filter bandwidth (narrow correlator configuration)	0.5 Hz
Tracking Galileo E5a	coherent integration time	20 ms
	Early-Late space chips	0.5 chips
	Early-Late space narrow chips	0.1 chips
	PLL filter bandwidth	20 Hz
	PLL filter bandwidth (narrow correlator configuration)	7.5 Hz
	DLL filter bandwidth	1.5 Hz
	DLL filter bandwidth (narrow correlator configuration)	0.5 Hz

Table 4.16 shows the receiver PVT configuration parameters: the positioning mode is set to single-point positioning. The Receiver Autonomous Integrity Monitoring (RAIM) Fault Detection and Exclusion (FDE) is enabled. The ionospheric correction is performed according to the broadcasted ionospheric model. A Saastamoninen tropospheric model is used [151]. The PVT output rate is set to 1 s, and the satellites marked as unhealthy are not used for the computation of the PVT solutions. On top of that, a PVT Kalman filter is used.

Table 4.16 Spaceborne receiver PVT configuration for static scenarios.

Parameter type	Parameter	Value
General	Positioning Mode	Single
	Receiver Autonomous Integrity Monitoring (RAIM) Fault Detection and Exclusion (FDE)	Enabled
	Iono Model	Broadcast
	Trop Model	Saastamoinen
	PVT Output Rate	1 s
	Use unhealthy sats	Disabled
PVT Kalman Filter	Standard deviation of the position estimations	1 m
	Standard deviation of the velocity estimations	0.1 m/s
	Standard deviation of the dynamic system model for position	2.0 m
	Standard deviation of the dynamic system model for velocity	0.5 m/s

More information regarding the GNSS-SDR configurable parameters can be found in [17].

Appendix 4.B Receiver Configuration for Low Earth Orbit (LEO) scenarios

This section provides the receiver configuration that was employed to obtain the experimental results using the LEO scenario. Table 4.17 shows the configuration of the acquisition block. Self-assistance to acquisition from primary to secondary band is enabled. Therefore, once the primary band is successfully acquired, the receiver predicts the Doppler frequency in the secondary band and performs acquisition in the secondary band with a reduced Doppler frequency space. The acquisition parameters shown in Table 4.17 have been explained in Appendix 4.A. The Doppler Max parameter is set to 50 kHz to accommodate the range of Doppler shifts encountered in Low Earth Orbit (LEO) conditions.

Table 4.17 Spaceborne receiver acquisition configuration for LEO scenarios.

Parameter type	Parameter	Value
Acquisition GPS L1 C/A	Doppler Max	50000 Hz
	Doppler Step	250 Hz
	Threshold	2.5
	Downsampling Factor	4
Acquisition GPS L5	Doppler Max	5000 Hz
	Doppler Step	250 Hz
	Threshold	2.5
	Assistance to acquisition from primary to secondary band	Enabled
Acquisition Galileo E1b/c	Doppler Max	50000 Hz
	Doppler Step	250 Hz
	Threshold	2.5
	Downsampling Factor	4
Acquisition Galileo E5a	Doppler Max	5000 Hz
	Doppler Step	250 Hz
	Threshold	2.5
	Assistance to acquisition from primary to secondary band	Enabled

Tables 4.18 and 4.19 show the configuration of the Tracking blocks for the GPS and Galileo signals, respectively. In addition to the parameters explained in Appendix 4.A, a FLL is enabled during the pull-in time. The High Dynamics flag, as shown in Tables 4.18 and 4.19, enables the operation of the high-dynamic tracking loops. These are described in Section 4.3.3 and utilize the code phase, Doppler shift, and Doppler rate for tracking. This setup is designed for enhanced performance under high-dynamic conditions.

Table 4.18 Spaceborne receiver GPS tracking configuration for LEO scenarios.

Parameter type	Parameter	Value
Tracking GPS L1 C/A	coherent integration time	20 ms
	Early-Late space chips	0.5
	Early-Late space narrow chips	0.5
	PLL filter bandwidth	35 Hz
	PLL filter bandwidth (narrow correlator configuration)	5.0 Hz
	DLL filter bandwidth	2 Hz
	DLL filter bandwidth (narrow correlator configuration)	0.5 Hz
	Enable FLL pull-in	true
	FLL filter bandwidth	10 Hz
	High Dynamics	true
Tracking GPS L5	coherent integration time	5 ms
	Early-Late space chips	0.5
	Early-Late space narrow chips	0.5
	PLL filter bandwidth	50 Hz
	PLL filter bandwidth (narrow correlator configuration)	30 Hz
	DLL filter bandwidth	4 Hz
	DLL filter bandwidth (narrow correlator configuration)	2 Hz
	Enable FLL pull-in	true
	FLL filter bandwidth	2.5 Hz
	High Dynamics	true

Table 4.19 Spaceborne receiver Galileo tracking configuration for LEO scenarios.

Parameter type	Parameter	Value
Tracking Galileo E1b/c	coherent integration time	20 ms
	Early-Late space chips	0.15
	Very Early-Late space chips	0.5
	Early-Late space narrow chips	0.15
	Very Early-Late space narrow chips	0.5
	PLL filter bandwidth	15 Hz
	PLL filter bandwidth (narrow correlator configuration)	5.0 Hz
	DLL filter bandwidth	0.75 Hz
	DLL filter bandwidth (narrow correlator configuration)	0.5 Hz
	Enable FLL pull-in	true
	FLL filter bandwidth	10 Hz
	High Dynamics	true
Tracking Galileo E5a	coherent integration time	5 ms
	Early-Late space chips	0.5
	Early-Late space narrow chips	0.5
	PLL filter bandwidth	50 Hz
	PLL filter bandwidth (narrow correlator configuration)	30 Hz
	DLL filter bandwidth	4.0 Hz
	DLL filter bandwidth (narrow correlator configuration)	2.0 Hz
	Enable FLL pull-in	true
	FLL filter bandwidth	2.5 Hz
	High Dynamics	true

Finally, Table 4.20 shows the configuration of the PVT block. The navigation solutions were computed using the least squares method.

Table 4.20 Spaceborne receiver PVT configuration for LEO scenarios.

Parameter type	Parameter	Value
General	Positioning Mode	Single
	Receiver Autonomous Integrity Monitoring (RAIM) Fault Detection and Exclusion (FDE)	Enabled
	Iono Model	Broadcast
	Trop Model	Off
	PVT Output Rate	20 ms
	Use unhealthy sats	Disabled

Chapter 5

GNSS Rebroadcaster

This chapter details the design, prototype development, and initial assessment of an economical, software-defined GNSS rebroadcaster, constructed using COTS components. This work adheres to the design principles and methodologies outlined in Chapter 3.

This design enhances the capabilities of the proposed architecture for generating GNSS signals, potentially enabling the simulation of diverse GNSS scenarios. Additionally, it showcases the architecture's flexibility for supporting research activities in the design, testing, and validation of non-standard GNSS features operating in real time, such as the rebroadcasting of GNSS signals.

The proposed rebroadcaster can generate, or receive and regenerate, GNSS signals with very low latency. It can generate and regenerate up to eight GPS L1 C/A and Galileo E1 b/c signals simultaneously. The presented rebroadcaster features a special module named telemetry symbol link. When the telemetry symbol link is enabled, the rebroadcaster can regenerate the received telemetry symbols with a latency below 30 ms. This also allows maintaining the consistency with other sensors, e.g. IMU, where available.

The underlying architecture of the proposed rebroadcaster builds upon the SoC FPGA receiver framework detailed in Chapter 3. It introduces additional FPGA hardware accelerators for GNSS signal generation and incorporates GNSS-SDR-SIM, a software-defined GNSS signal generator. This enhancement significantly enriches the system's capabilities.

The implementation of the rebroadcaster is based on the software-defined GNSS simulator referenced in [149], which is capable of generating GNSS signals in post-processing mode. In this thesis, the software was adapted to operate on an SoC-FPGA device, supporting both real-time and post-processing modes. This adaptation included the ability to generate and regenerate

GNSS signals in real time, including the regeneration of unpredictable telemetry symbols with minimal latency. Hardware accelerators were developed within the FPGA, leveraging IP cores from the spaceborne receiver discussed in Chapter 4 and creating new cores for the transmitter. The device underwent extensive testing and verification to ensure its functionality.

This chapter is organized as follows: Section 5.1 provides an introduction. Section 5.2 summarizes the objectives of the rebroadcaster implementation. Next, Section 5.3 describes the rebroadcaster architecture. This is followed by Section 5.4, which focuses on the transmitter part of the rebroadcaster and provides a detailed description of its design. Section 5.5 presents the preliminary performance assessment and experimental results, and finally, Section 5.6 wraps up some conclusions.

5.1 Introduction

SDR techniques facilitate the implementation of GNSS signal generators and regenerators for research purposes. A significant number of GNSS signal generators are discussed in academic and technical publications.

The most basic type of GNSS signal generator functions by replaying either synthetically created or previously recorded GNSS signals. The synthetic GNSS signals can be created in post-processing mode prior to their transmission. Refs. [152, 153] exemplify this type of signal generator. More complex devices can generate GNSS signals on the fly, eliminating the need for pre-processing a large volume of data. For instance, Ref. [154] describes a real-time signal generator that operates on GPUs.

Signal regenerators used for testing GNSS threat mitigation techniques operate on similar principles. The regeneration of GNSS signals can be performed in three steps: initially recording the received signals, then regenerating the signals in post-processing mode, and finally, replaying the regenerated signals in real-time mode. More sophisticated systems can perform real-time regeneration of GNSS signals, receiving and regenerating the signals simultaneously. As an example, Ref. [155] is a large and sophisticated test-bed for testing GNSS position and timing authentication methods. The regeneration of GNSS signals can employ data bit prediction to promptly retransmit the signals with accurate timing, thus minimizing the latency typically associated with symbol estimation and retransmission. However, this prevents the retransmission of any unpredictable data that might be embedded in the navigation messages. Ref. [156] introduces a test-bed with a receiver-spoofer capable of regenerating up to 14 GPS signals in real time in this manner. This receiver-spoofer is an enhanced version of the DSP-based receiver-spoofer introduced in [157]. Conversely, by utilizing improved real-time capabilities, signal regenerators can estimate and retransmit received secure codes or data symbols with low latency, albeit not zero latency, due to the time needed to estimate the received symbols. The rebroadcaster proposed in this thesis incorporates sophisticated real-time capabilities to rebroadcast the received navigation data symbols with low latency.

Practical utilities of GNSS signal regeneration include the simulation of diverse GNSS scenarios and, for instance, providing indoor GNSS positioning: Ref. [158] describes the use of multiple GNSS signal regenerators, synchronized wirelessly with a computer server, to broadcast GNSS signals received by outdoor antennas into indoor environments where GNSS signals are otherwise unavailable.

5.2 Objectives

The aim of this development is to highlight the versatility of the SoC FPGA architecture proposed in this thesis, showcasing the generation and regeneration of GNSS signals in real time. This capability could potentially enable the emulation of a wide range of GNSS scenarios. Additionally, this development aims to demonstrate the architecture's adequacy for supporting the design, testing, and real-time validation of non-standard GNSS features. The telemetry symbol link implemented in the rebroadcaster examplifies this capability, serving as a concrete instance of how the system can facilitate the exploration and integration of advanced GNSS functionalities.

The telemetry symbol link implements a fast path between the receiver tracking multicorrelators and the transmitter channels in the FPGA, enabling the fast retransmission of the telemetry symbols estimated at the receiver. This approach aims to offer greater flexibility than existing COTS signal generators, potentially enabling the real-time simulation of various scenarios with GNSS signals received from space.

The target system includes a rebroadcaster capable of operating in both real-time and post-processing modes, serving dual functions as a signal generator and a signal regenerator. The system can generate or regenerate up to 8 GPS L1 C/A and Galileo E1b/c signals in any combination. The telemetry symbol link can only be activated when regenerating Galileo E1b/c signals. Figure 5.1 shows the target system.

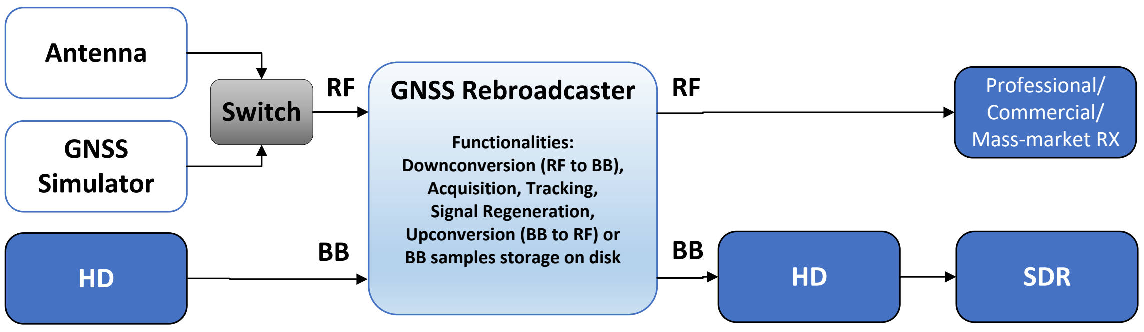
Antenna
GNSS Simulator
HD
Switch
RF
BB
GNSS Rebroadcaster
Functionalities:
Downconversion (RF to BB),
Acquisition, Tracking,
Signal Regeneration,
Upconversion (BB to RF) or
BB samples storage on disk
RF
BB
Professional/ Commercial/ Mass-market RX
HD
SDR

Figure 5.1 GNSS Signal Rebroadcaster

When operating in real-time mode as a signal regenerator, the rebroadcaster receives signals from a GNSS simulator or an antenna, and it rebroadcasts the signals to an external GNSS receiver. A GNSS signal simulator is a device designed to generate signals that mimic those produced by GNSS satellites. The external receiver, either physical or based on software-defined radio, may incorporate advanced processing techniques, such as innovative tracking loops, digital signal processing, advanced navigation message processing, or receiver protection algorithms.

The rebroadcaster can also operate in real-time mode as a signal generator. In this case, the antenna is not connected. Instead, the rebroadcaster generates GNSS signals internally, which are processed by the external GNSS Receiver.

When operating in post-processing mode as a signal regenerator, the rebroadcaster processes GNSS signals recorded on a Hard Disk (HD), regenerates these signals, and stores them back onto the HD.

Finally, when working in post-processing mode as a signal generator, the rebroadcaster generates GNSS signals internally and stores these signals in the HD.

In the initial concept demonstrator, the target retransmission delay is set to 1 s for both the retransmitted PVT solutions and the regenerated symbols. By default, the telemetry data is regenerated out of the received ephemeris data. Therefore any unpredictable data in the navigation messages is not retransmitted.

However, when the telemetry symbol link is used, the target latency for the estimation and retransmission of the received telemetry data is set to 30 milliseconds. Although the FPGA is capable of achieving lower latencies, this threshold has been chosen to expedite the initial implementation. Moreover, the telemetry symbol link regenerates any unpredictable data present in the navigation messages.

In addition, the rebroadcaster can regenerate the received satellite signals in real-time while simultaneously rebroadcasting a PVT solution that differs from the PVT fixes obtained by the receiver. In this case, a sufficient number of satellites shall be visible from the rebroadcasted position such that an external GNSS receiver can successfully obtain PVT fixes using the regenerated signals.

The software version of the regenerator (with the option not to use the FPGA hardware accelerators) can be executed in a PC. In this case, the FPGA hardware accelerator modules are replaced by software modules, which perform the equivalent functionality in software mode. When running on a PC, the signal regenerator is limited to post-processing mode.

In the current implementation, the rebroadcaster does not simulate any atmospheric effects on the regenerated signal.

5.3 System Design

The rebroadcaster features a software-defined receiver and a software-defined transmitter for proper operation. The receiver utilizes the FPGA hardware accelerators presented in Chapter 3 for real-time processing of GNSS signals. The transmitter utilizes additional FPGA hardware accelerators for the generation of the transmitted signals. The transmitter hardware accelerators were specifically developed for the design of the rebroadcaster.

To regenerate the received GNSS signals, the rebroadcaster's receiver first processes these signals to obtain PVT solutions. Subsequently, the transmitter uses the receiver's estimated signal parameters to regenerate the GNSS signals accordingly. The transmitter utilizes the receiver's ephemeris data and the rebroadcasted PVT to compute the observables corresponding to the transmitted signal. These observables include pseudoranges, carrier phase, carrier Doppler shift frequency, carrier Doppler rate, code phase, and code Doppler shift frequency.

To rebroadcast a PVT solution that differs from the receiver PVT, the transmitter recalculates the observables from the satellites that the receiver is tracking, referencing both the desired rebroadcasted PVT and the satellites' real-time positions. These observables are then used by the transmitter to determine the shape and timing of the rebroadcasted signal, which is then communicated to the FPGA. This communication between the software-defined transmitter and the FPGA occurs in the form of timed commands. The software-defined transmitter generates timed commands specifying the shape of the rebroadcasted signals. Simultaneously with the reception of the timed commands, the FPGA generates the GNSS signals. All these calculations occur in real time.

Without loss of generality, in the following we will assume that the rebroadcasted PVT is always set to the receiver PVT. The receiver PVT is rebroadcasted with a delay of 1 s due to the processing time required mainly for the PVT calculation in the receiver. By default, the telemetry data is also rebroadcasted with a delay of 1 s, because the telemetry data follows the same path as the PVT (from the software-defined receiver to the software-defined transmitter). Also by default, the transmitter pre-computes the transmitted telemetry on the fly, out of the ephemeris data obtained in the receiver. The FPGA then generates the signal sample stream for the DAC. Any secondary pilot symbols, if present in the GNSS signals, are automatically generated in the FPGA.

When the telemetry symbol link is enabled — providing a fast path within the FPGA for quickly retransmitting received telemetry data — the transmitter can rebroadcast the telemetry symbols with a latency of less than 30 ms. This low latency results from the FPGA's direct path that channels the estimated telemetry symbols from the receiver hardware accelerators to the transmitter hardware accelerators, bypassing the PS. The transmitter directly rebroadcasts the telemetry symbols obtained at the receiver, but aligns their timing with the rebroadcasted PVT instead of the receiver's signal timing. Rebroadcasting the receiver PVT still incurs a delay of 1 second, due to the time required for the receiver to estimate the PVT.

The telemetry symbol link also enables the rebroadcasting of any unpredictable symbols in the navigation data, since the telemetry symbols are estimated at the receiver and rebroadcasted at the transmitter.

5.3.1 Hardware Implementation

Similar to the spaceborne receiver discussed in Chapter 4, the proposed architecture utilizes the Analog Devices' ADRV9361-Z7035 board [159]. As described in Section 4.3.1, this board integrates the AD9361 RF Agile Transceiver [106] and the AMD Zynq XC7Z035-L2 FBG676I All-Programmable SoC [103], with a size of 100 $\times$ 62 mm. This platform facilitates the GNSS rebroadcaster's comprehensive functionality, encompassing receiver and transmitter operations from the RF output of an antenna (or signal generator), through the generation of navigation products, to the real-time regeneration of GNSS signals. The design ensures that the rebroadcaster is compact, transportable, and fully upgradeable, capable of providing end-to-end solutions on a single board.

In order to facilitate the development, debugging and testing of the GNSS receiver, the prototype makes use of a ADRV1CRR-BOB breakout carrier board, which implements suitable standard connectors for power supply, Ethernet communications, USB and JTAG programming interfaces, among other debugging features [140]

5.3.2 Signal Model

The GPS L1 C/A signals transmitted by GNSS satellites can be described by (2.6) in Chapter 2. Similarly, the GPS L1 C/A signals received at the rebroadcaster's RF front-end can be represented as

$$r(t) = \sum_{k=0}^{N} A_k c_k(t - \tau_k(t)) d_k(t - \tau_k(t)) \cos(2\pi(f_c + f_{d,k})t - \psi_k(t)) + w(t) \ , \tag{5.1}$$

where N represents the number of visible satellites; A_k, the signal amplitude; $c_k(t)$, the spreading code; $d_k(t)$, the data bit stream; $\tau_k(t)$, the code phase; f_c, the L1 carrier frequency; $f_{d,k}$, the Doppler frequency; $\psi_k(t)$, the carrier phase; and $w(t)$, the additive noise at the RF input, which is assumed to be white and Gaussian. Similarly to (2.6) in Chapter 2, the subindex k indicates that the parameter corresponds to the signal transmitted by the k-th satellite.

The regenerated GNSS signal can be represented as

$$s(t) = \sum_{k=0}^{N} A_{s,k} c_k(t - \tilde{\tau}_{s,k}(t)) \tilde{d}_k(t - \tilde{\tau}_{s,k}(t)) \cos(2\pi(f_c + \tilde{f}_{d,k})t - \psi_{s,k}(t)) + w_s(t) \tag{5.2}$$

where $A_{s,k}$ represents the generated carrier amplitude; $\tilde{d}_k$, the estimated and regenerated telemetry symbols; f_c, the carrier frequency; $\tilde{f}_{d,k}$, the estimated and regenerated Doppler frequency; $\tilde{\tau}_{s,k}(t)$, the estimated and regenerated code phase; $\psi_{s,k}(t)$, the transmitted carrier phase; and $w_s(t)$, the noise at the RF output. The generated carrier amplitude is fixed, and the transmitted carrier phase, $\psi_{s,k}(t)$, is initialized to zero at the start of rebroadcasting. This initialization also occurs during operation, when a satellite is detected, and the rebroadcaster begins its retransmission. Similar to (5.1), the subindex k indicates that the parameter corresponds to the signal transmitted by the k-th satellite.

When operating in post-processing mode and on a PC in software mode (without involving an FPGA), the rebroadcaster is capable of generating noise, $w_s(t)$, along with the output signal, based on a user-specified C/N_0. However, this option is unavailable when the system operates in real time on a SoC FPGA platform. The reason for this is that the signal generator produces noise simulating the normal distribution with floating-point calculations, which is not straightforward to implement in the FPGA using fixed-point arithmetic.

For Galileo E1b/c signals, the rebroadcaster functions in a manner akin to its processing of GPS L1 C/A signals. The Galileo E1b/c signals comprise two components: the pilot and the data signals, as outlined in Equation 2.9 in Chapter 2 [29]. When rebroadcasting Galileo signals both the pilot and the data components are regenerated in real time. By default, the data symbols are pre-computed out of the ephemeris data obtained by the receiver, and then the pre-computed values are transmitted. All this process occurs in real time. When the telemetry symbol link is enabled, the data symbols are estimated and directly retransmitted with minimal latency. The pilot's secondary code is automatically generated in the FPGA.

As shown in Equation 2.9 in Chapter 2, the Galileo E1 b/c signal uses a CBOC modulation, where both pilot and data signals are modulated using two square-waveform sub-carriers: $sc_{E1-B,a}(t)$, $sc_{E1-B,b}(t)$, $sc_{E1-C,a}(t)$, and $sc_{E1-C,b}(t)$ [29]. The rebroadcaster regenerates the $sc_{E1-B,a}(t)$ and $sc_{E1-C,a}(t)$ subcarriers, but it does not regenerate $sc_{E1-B,b}$ and $sc_{E1-C,b}$, leading to BOC(1,1) modulation being used instead of CBOC. This allows for a more efficient hardware implementation. In [160], simulations show that acquiring CBOC-modulated signals with a BOC(1,1)-modulated reference code in narrow-bandwidth receivers leads to only minimal performance loss in detection probability, compared to processing these signals with CBOC-modulated reference codes. This finding suggests that regenerating Galileo signals with BOC(1,1) modulation is similarly likely to result in small performance losses, even in narrow-bandwidth receivers. However, the specific impact was not accurately measured.

5.3.3 Rebroadcaster Architecture

Figure 5.2 shows the block diagram of the GNSS signal regenerator. It contains two main parts: the AD9361 RF transceiver, which implements the RFFE, and the Zynq-7000 XC7035 SoC FPGA.

The RFFE performs two main sets of operations on GNSS signals: first, it converts the received RF signals to baseband and then digitizes them through analog-to-digital conversion; second, it performs digital-to-analog conversion on the rebroadcasted GNSS signals and converts them from baseband back to RF.

The SoC FPGA features a software receiver and a software transmitter within the PS. Conversely, the PL is responsible for implementing receiver and transmitter hardware accelerators, as well as the telemetry symbol link.

The SoC FPGA instantiates the following GNSS Receiver modules, in line with the architecture proposed in Chapter 3:

- **FPGA receiver hardware accelerator IPs**: The FPGA implements the acquisition, tracking multicorrelators, dynamic bit selection, and other auxiliary receiver modules, following the design explained in Chapter 3. When the GNSS signal regeneration is enabled, the FPGA receives the GNSS signals from the analog front-end and executes the acquisition and the tracking multicorrelators to process the received signals.
- **GNSS-SDR software-defined receiver**: This software implements the baseband GNSS receiver engine. When the GNSS signal regeneration is enabled, GNSS-SDR performs the telemetry decoding, obtains observables, and computes PVT. Then, it communicates the PVT data and the ephemeris data to the software transmitter. This information is used by the transmitter to rebroadcast the GNSS signals promptly. When the telemetry symbol link is enabled, GNSS-SDR also sends the receiver's internal status data to the transmitter in real-time. The contents of the receiver status data is explained in Section 5.3.4.

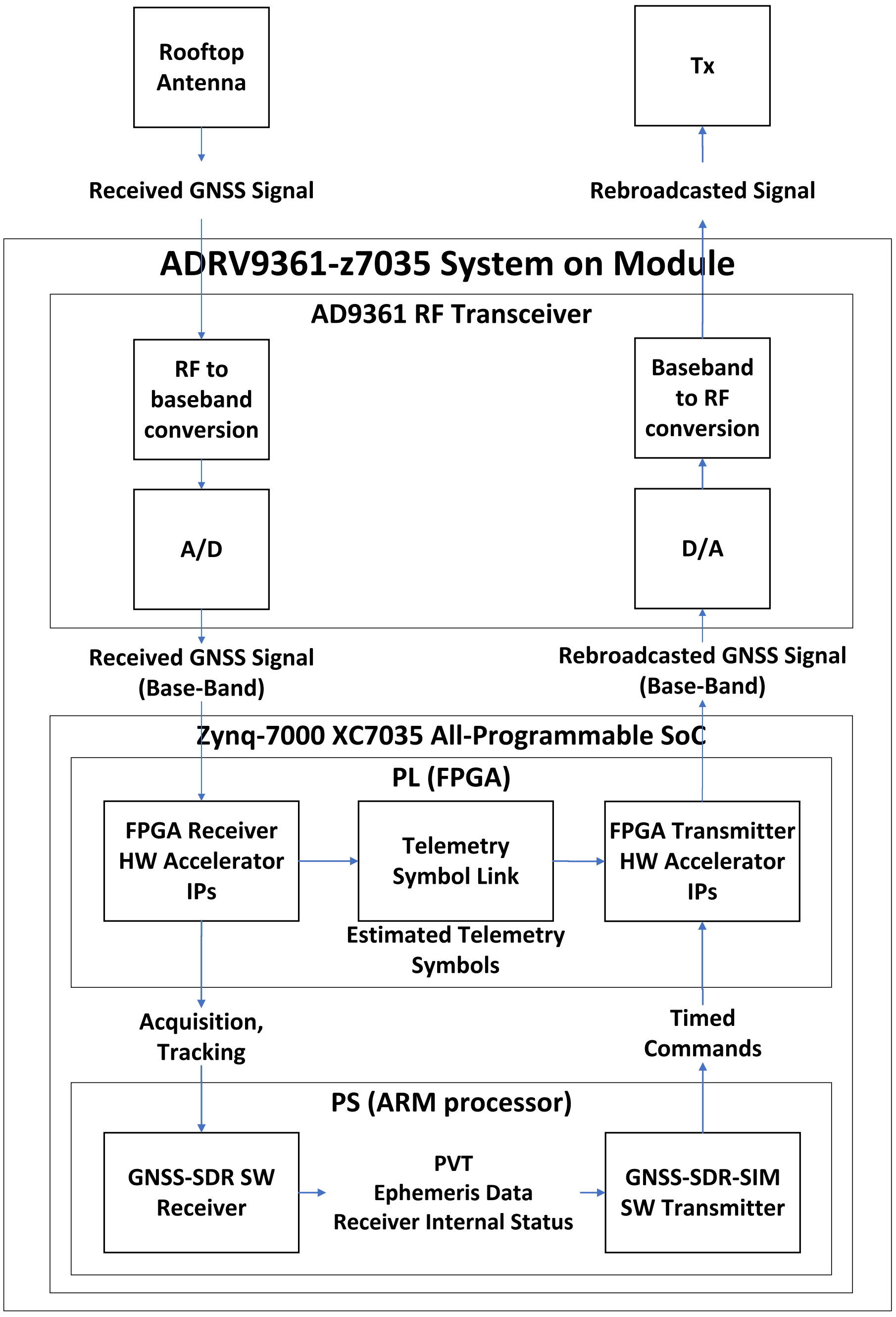
Rooftop Antenna
Tx
Received GNSS Signal
Rebroadcasted Signal
ADRV9361-z7035 System on Module
AD9361 RF Transceiver
RF to baseband conversion
Baseband to RF conversion
A/D
D/A
Received GNSS Signal (Base-Band)
Rebroadcasted GNSS Signal (Base-Band)
Zynq-7000 XC7035 All-Programmable SoC
PL (FPGA)
FPGA Receiver HW Accelerator IPs
Telemetry Symbol Link
FPGA Transmitter HW Accelerator IPs
Estimated Telemetry Symbols
Acquisition, Tracking
Timed Commands
PS (ARM processor)
GNSS-SDR SW Receiver
PVT
Ephemeris Data
Receiver Internal Status
GNSS-SDR-SIM SW Transmitter

Figure 5.2 GNSS Signal Regenerator block diagram

In addition to the receiver modules mentioned above, the SoC FPGA also instantiates the following GNSS transmitter modules:

- **GNSS-SDR-SIM software-defined transmitter**: This software computes the parameters of the transmitted signals in real time, according to the desired PVT solution. When working as a signal generator, GNSS-SDR-SIM obtains the ephemeris data and the PVT data from user input files. When working as a signal regenerator, GNSS-SDR-SIM obtains the ephemeris data and the PVT from the software receiver. It then computes the observables for the rebroadcasted satellites that are visible from the rebroadcasted PVT.

After that, it computes the shape and timing of the regenerated signal, including code phase, code phase rate, carrier phase, carrier phase rate, and Doppler frequencies, based on these observables. At the end of this process, GNSS-SDR-SIM embeds this information in the timed commands and sends the timed commands to the FPGA. When the telemetry symbol link is enabled, GNSS-SDR-SIM also uses the receiver's internal status data to assemble the timed commands. The contents of the receiver internal status data is explained in Section 5.3.4, and its role in generating the transmitted signal is explained in Section 5.4.

- **FPGA transmitter hardware accelerator IPs**: The FPGA contains hardware accelerator IPs that generate the transmitted GNSS signals in real time. When working as a signal generator, and when working as a signal regenerator with the telemetry symbol link being disabled, the FPGA generates/regenerates the signals out of the timed commands that GNSS-SDR-SIM sends to the FPGA. When functioning as a signal regenerator with the telemetry symbol link enabled, the FPGA regenerates signals using timed commands and the receiver's estimated telemetry symbols. These symbols, estimated at the receiver's tracking multicorrelators, are made available to the transmitter through the telemetry symbol link.

Finally, the SoC FPGA instantiates the telemetry symbol link, comprising various modules that facilitate a fast path for communicating the estimated telemetry symbols from the receiver to the transmitter. The telemetry symbol link includes a state machine and various First In, First Out (FIFO) queues. These queues connect the receiver's FPGA multicorrelator IPs to the transmitter's signal generator channels within the FPGA on a per-channel basis. The telemetry symbol link facilitates the retransmission of any unpredictable data embedded in the navigation messages.

When the telemetry symbol link is disabled, GNSS-SDR-SIM pre-computes the rebroadcasted telemetry symbols out of the receiver telemetry data. Thus, disabling the telemetry symbol link renders the system incapable of retransmitting any unpredictable data embedded within the navigation messages.

The PS runs on the Geniux-customized GNU/Linux OS. The methodology for configuring the embedded OS is explained in Section 3.2. Both GNSS-SDR and GNSS-SDR-SIM are compatible with Geniux and can also run on a GNU/Linux-based PC. However, when operated on a PC, they are limited to functioning in post-processing mode.

5.3.4 Receiver Internal Status

The receiver contains a sample counter that is reset when signal reception is initiated. This sample counter counts the number of samples received. Each received telemetry symbol is associated with a sample counter value and the Time of Week (TOW) corresponding to the time when the satellite sent that symbol. In addition, each symbol is associated with a ToA, indicating the time when the receiver received that symbol. As time progresses, the relationship between the received sample counter value, the TOW, and the ToA in a receiver channel changes due to the relative movement between the receiver and the satellite being tracked.

The receiver internal status data includes several parameters that are relevant for the telemetry symbol link. These parameters are the satellite channel allocation, the equivalence between the TOW values, the ToA values and the receiver sample counter values for each channel, and a flag that indicates whether the receiver PLLs are locked in 180°. When the receiver's PLL locks with a 180° phase difference, the PLL output signal is in phase opposition to the received carrier signal. This results in the inversion of demodulated data bits.

When the telemetry symbol link is enabled, GNSS-SDR-SIM uses the receiver internal status data to compute the receiver sample counter corresponding to the telemetry symbols to be rebroadcasted. GNSS-SDR-SIM also uses the receiver internal status data to determine the transmitter channel allocation. Then, GNSS-SDR-SIM embeds the sample counter values and the receiver PLL lock status (0 degrees or 180 degrees) into the timed commands. In this way, the FPGA can use the sample counter to locate the telemetry symbols to be rebroadcasted within the FIFO of the telemetry symbol link. In addition, the FPGA uses the PLL lock status to generate the symbols using the correct polarity.

Section 5.4 explains in more detail how the transmitter uses the receiver's internal status to rebroadcast the estimated telemetry symbols achieving very low latency.

5.4 Transmitter Design

5.4.1 Software Transmitter

Fig. 5.3 shows a block diagram of GNSS-SDR-SIM. It includes the following components:

- **User PVT / trajectory**: These are the PVT solutions to be rebroadcasted. When working as a signal generator, GNSS-SDR-SIM obtains the PVT data from user input files. A dynamic PVT can be specified using NMEA files or Comma-Separated Values (CSV) files containing time and positions. A dynamic PVT is automatically interpolated in the transmitter. A static PVT can be specified by setting a fixed position. Conversely, when working as a signal regenerator, GNSS-SDR-SIM obtains the PVT data from the receiver.
- **Ephemeris data**: This is the ephemeris data of the satellites to be rebroadcasted. When working as a signal generator, GNSS-SDR-SIM obtains the ephemeris data from RINEX files. When working as a signal regenerator, it obtains the ephemeris data from the receiver.
- **Receiver internal status**: This includes the receiver sample counter, Time of Week (TOW), Time of Arrival (ToA), and Phase-Locked Loop (PLL) lock status for each channel, all associated with the receiver's telemetry symbols (see Section 5.3.4). When working as a signal regenerator, and when the telemetry symbol link is enabled, GNSS-SDR sends its internal status data to the GNSS-SDR-SIM every 20 ms.
- **Orbit Engine**: The orbit engine utilizes the ephemeris data to compute the positions of rebroadcasted satellites and employs the PVT data to identify which satellites are visible from the rebroadcasted location. It then delivers the positions and velocities of these visible satellites to the observables engine.

- **Observables Engine**: The observables engine computes the observables corresponding to the transmitted signals, using the satellites'positions and velocities, and the PVT. Then, it uses the computed observables and the satellite ephemeris data to generate the timed commands. If the telemetry symbol link is enabled, it also uses the receiver's internal status to compute the timed commands. Finally, it sends the timed commands to the signal generator engine.
- **Signal Generator Engine**: The signal generator engine interprets and executes the timed commands, and generates the rebroadcasted signal. The signal generator engine is implemented in the FPGA. However, when GNSS-SDR-SIM runs on a PC, this FPGA implementation is replaced with an equivalent software implementation that operates in post-processing mode.

When the rebroadcaster is operating as a signal regenerator with the telemetry symbol link disabled, the observables engine sets a delay of 1 s in both the regenerated PVT solutions and the regenerated telemetry symbols, to account for the latency of the PVT compuation in the receiver.

Conversely, when the telemetry symbol link is enabled, the observables engine sets a delay of 1 s in the regenerated PVT solutions, but a maximum latency of only 30 ms for the regenerated telemetry symbols. This reduction in the latency of the regenerated telemetry symbols is facilitated by the telemetry symbol link. The 30-ms latency is set to account for the time it takes for the FPGA to estimate the telemetry symbols, plus the time to go through the telemetry symbol link FIFO, and to account for all temporary delays that can occur in the generation of the timed commands in the software.

In addition to the functionality mentioned above, GNSS-SDR-SIM computes the telemetry data, navigation messages, and channel coding. Since GNSS-SDR-SIM is implemented in software, doing research and testing new navigation messages or new channel coding schemes would not require an update of the FPGA code.

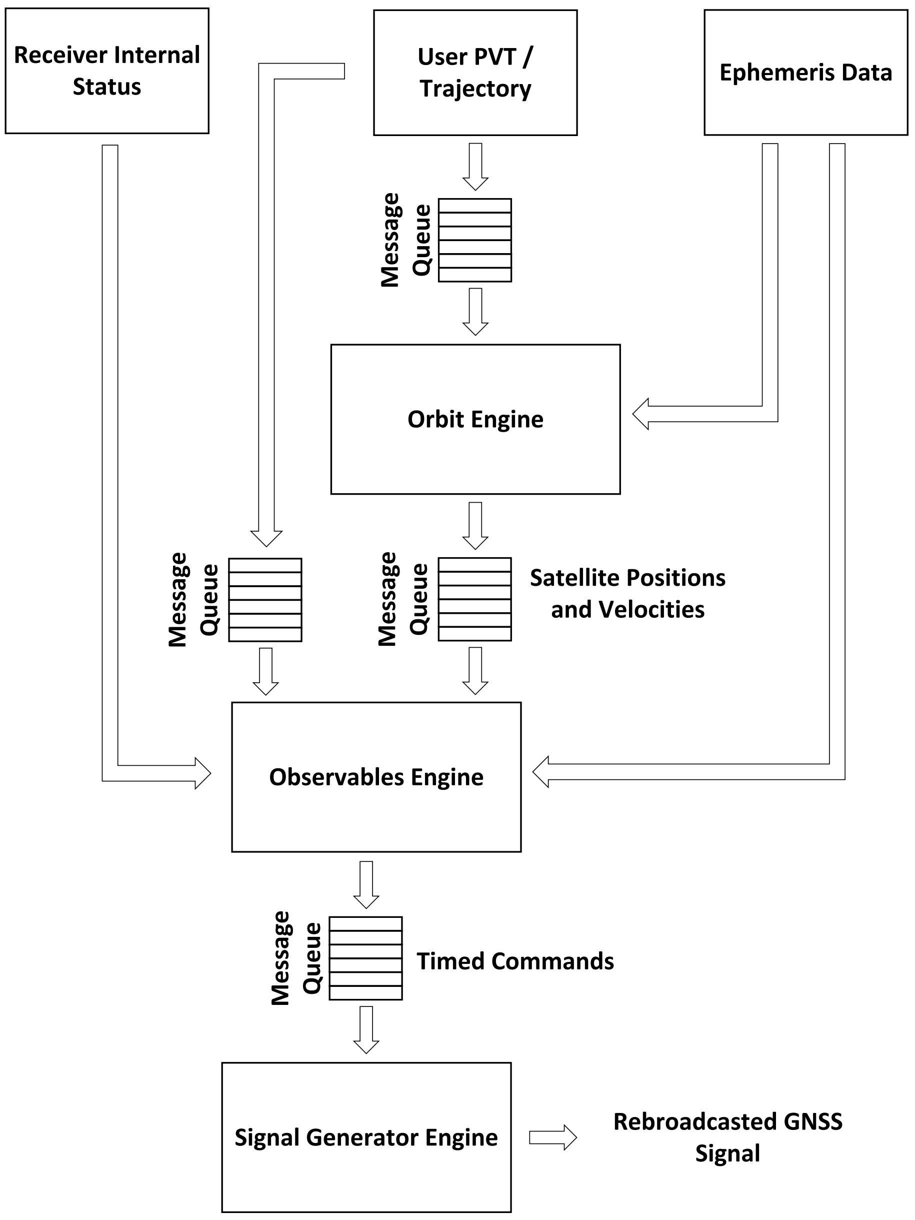

Figure 5.3 Software transmitter block diagram

5.4.2 Timed Commands

Timed commands are instructions that define the characteristics and timing of rebroadcasted signals, issued in real-time at regular intervals. The timed commands were implemented to facilitate real-time exchange of information between the software and the FPGA regarding the generation of rebroadcasted signals. The chosen timed command parameters enable the generation and regeneration of GNSS signals with sufficient accuracy for the emulation of diverse GNSS scenarios in real time. The timed commands also enable the intentional modification of the signal parameters in real time for research purposes. They include the parameters shown in Table 5.1.

Table 5.1 Timed command parameters

Parameter	**Description**
enable channel	This parameter can have the values 'true' or 'false' to enable or disable the channel specified by *channel ID*.
channel ID	Transmitter channel number.
satellite ID	This paremeter denotes the space vehicle number being rebroadcasted in channel *channel ID*.
system ID	Satellite System: GPS, or Galileo.
code phase	phase shift of the PRN code, at the *sample stamp* time instant, measured in samples, including decimal and fractional parts.
code phase rate	Rate at which the code phase is shifted per sample, measured in shifted samples per transmitted sample
carrier phase	phase of the carrier wave, at the *sample stamp* time instant, measured in radians.
carrier phase rate	Rate at which the carrier phase is adjusted, measured in radian/sample
carrier phase jerk	Rate at which the carrier phase rate is adjusted per sample, measured in radian/sample^2
telemetry symbol	data telemetry symbol to be transmitted (+1 or -1), corresponding to a phase shift of 0 or 180° in the modulated signal (only applicable when the telemetry symbol link is disabled)
symbol search sample stamp	Sample counter associated to the telemetry symbol that is to be rebroadcasted (only applicable when the telemetry symbol link is enabled).
PLL 180°	PLL lock status at the receiver. It can take the values of 0° or 180° (only applicable when the telemetry symbol link is enabled).
ibit	Telemetry symbol number within the current subframe.
sample stamp	Transmitter sample counter at which the timed command is executed.

GNSS-SDR-SIM sends timed commands to the signal generator engine within the FPGA in real-time. The signal generator engine interprets the timed commands and generates the transmitted signals. To generate the rebroadcasted signals in a timely manner, the signal generator engine has a sample counter. The FPGA executes the timed commands when the sample counter reaches the value specified by the *sample stamp* parameter in the timed commands. At that moment, the carrier phases, PRN code phases, and the respective phase rates are updated according to the timed command parameters. The transmitted telemetry symbols, and PRN secondary code symbols are updated ensuring consistency with the code phases.

GNSS-SDR-SIM sets the *enable channel* parameter to 'true' to enable the transmitter channel determined by *channel ID* in the same timed command. To disable the channel, it generates a timed command with this parameter set to ’false’.

Channel ID is the channel number that executes the timed command. *satellite ID* is the satellite PRN number assigned to *Channel ID*. The parameter *system ID* specifies whether the regenerated satellite signal is GPS L1 C/A or Galileo E1 b/c.

The shift in the code and carrier, as well as the Doppler frequency and their temporal evolution within rebroadcasted GNSS signals, are collectively determined by the code and carrier phases, their phase rates, and the carrier phase jerk.

The *telemetry symbol* parameter contains the telemetry symbol to be transmitted after the execution of the current timed command in the FPGA. The telemetry symbol is not immediately broadcasted, but only after the current symbol is completely transmitted, in line with the *code phase* parameter.

When the telemetry symbol link is enabled, the *telemetry symbol* parameter is not used. Instead, the FPGA uses the *symbol search sample stamp*. The *symbol search sample stamp* is the value of the receiver sample counter associated with the telemetry symbol that is to be rebroadcasted. The FPGA uses this parameter to locate the estimated value of the telemetry symbol in the FIFO within the telemetry symbol link.

Similarly, the *PLL* 180° parameter is only used when the telemetry symbol link is enabled. It indicates whether the receiver PLL is locked at 0° or 180° for the channel specified by *Channel ID*. If the PLL locks at 180°, the FPGA automatically inverts the estimated data symbols to maintain the correct polarity of the transmitted signal.

The *ibit* is the telemetry symbol number within the current telemetry frame. The FPGA uses this parameter when rebroadcasting Galileo E1 b/c signals, to automatically generate the PRN secondary code and align this code with the telemetry data.

5.4.3 FPGA Transmitter

The FPGA transmitter implements the signal generator channels. The FPGA has eight signal generator channels, each one receiving timed commands from the software-defined transmitter in real-time and generating the signal corresponding to one satellite (Fig. 5.4). Each signal generator channel can be assigned to a GPS L1 C/A or a Galileo E1 b/c satellite.

The signal combiner aggregates the signals from various transmitter channels. Depending on the operational mode, a switch then routes the aggregated signal either to the RFFE interface for real-time processing or to a DMA in the FPGA for storage in a file during post-processing. The switch is controlled by software.

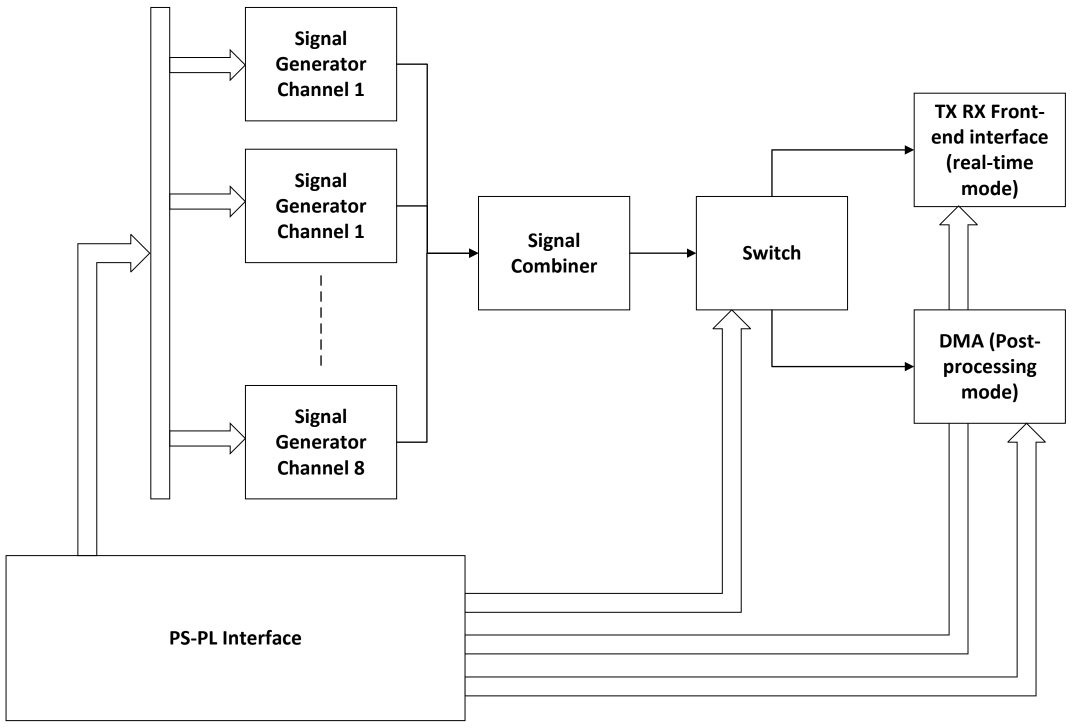
Signal Generator Channel 1
Signal Generator Channel 1
Signal Generator Channel 8
Signal Combiner
Switch
TX RX Front-end interface (real-time mode)
DMA (Post-processing mode)
PS-PL Interface

Figure 5.4 FPGA Transmitter block diagram

Each signal generator channels stores a copy of the modulated ranging codes. The software writes these codes to the FPGA when a satellite is assigned to a signal generator channel.

Modifying the characteristics of the ranging code necessitates only a software update and, if necessary, additional memory allocation in the FPGA. Thus, this architecture can potentially facilitate the testing of new ranging signals. In the current implementation, the FPGA quantizes the ranging codes using one bit per sample.

5.4.4 Telemetry Symbol Link

The telemetry symbol link comprises a set of FPGA modules that establish a connection going from each receiver multicorrelator hardware accelerator to its corresponding signal generator channel in the transmitter, on a per-channel basis. A block diagram of the telemetry symbol link is depicted in Fig. 5.5.

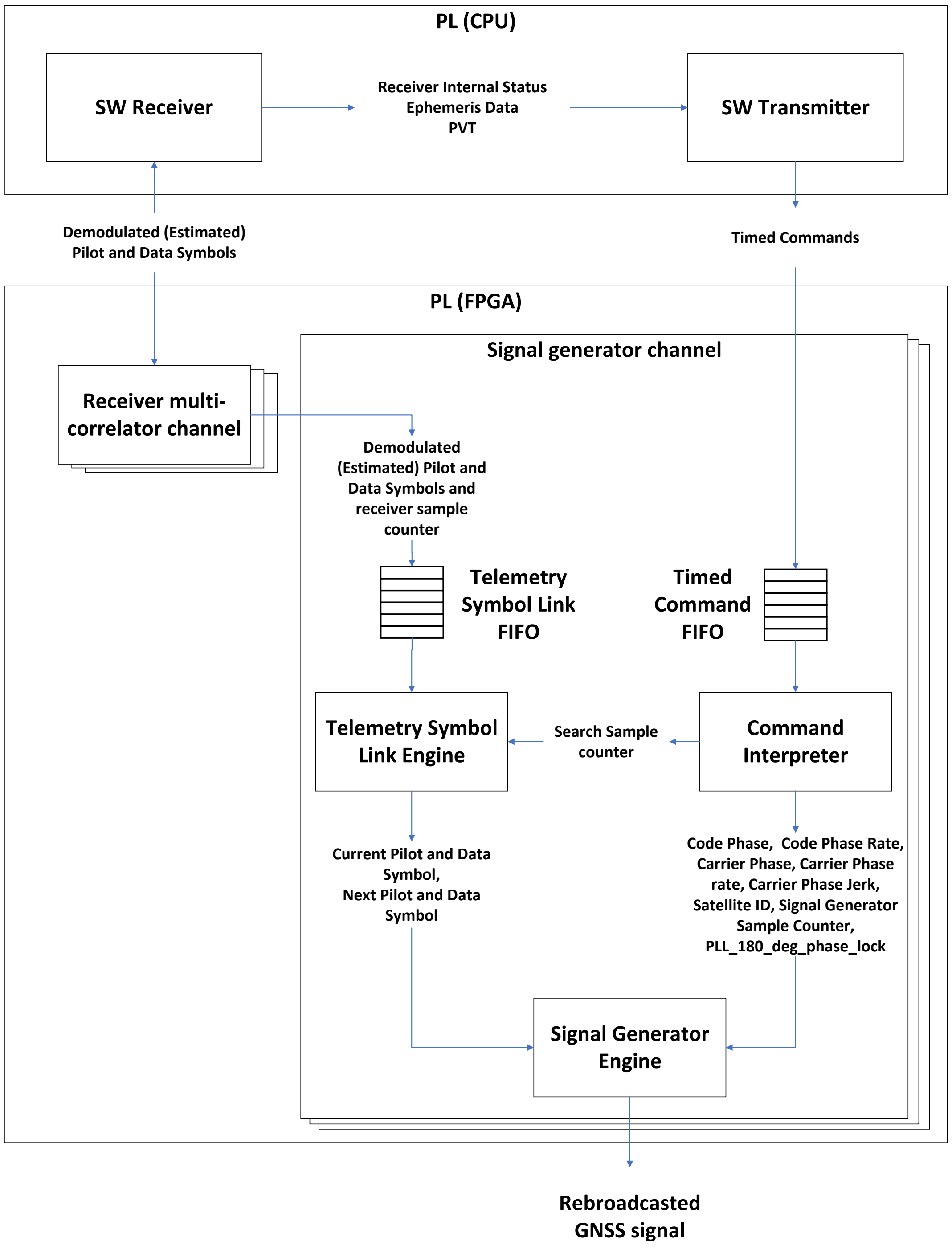
PL (CPU)
SW Receiver
Receiver Internal Status
Ephemeris Data
PVT
SW Transmitter
Demodulated (Estimated)
Pilot and Data Symbols
Timed Commands
PL (FPGA)
Signal generator channel
Receiver multi-correlator channel
Demodulated (Estimated) Pilot and Data Symbols and receiver sample counter
Telemetry Symbol Link FIFO
Timed Command FIFO
Telemetry Symbol Link Engine
Search Sample counter
Command Interpreter
Current Pilot and Data Symbol, Next Pilot and Data Symbol
Code Phase, Code Phase Rate, Carrier Phase, Carrier Phase rate, Carrier Phase Jerk, Satellite ID, Signal Generator Sample Counter, PLL_180_deg_phase_lock
Signal Generator Engine
Rebroadcasted GNSS signal

Figure 5.5 FPGA Telemetry Symbol Link block diagram

When using the telemetry symbol link, the receiver multicorrelator hardware accelerators push the demodulated data symbols into the telemetry symbol link FIFO, together with the receiver sample counter. The receiver sample counter is associated to a transmitted TOW and a ToA for each particular channel.

As explained in Section 5.3.4, the receiver communicates the relationship between the receiver sample counter, the TOW and the ToA to the software transmitter via the receiver internal status data. The software transmitter then uses uses this information to configure the timed command parameter *symbol search sample stamp* (see Table 5.1) with the receiver sample counter corresponding to the telemetry symbols to be regenerated.

A command interpreter in the signal generator channels within the FPGA interprets and executes the timed commands. When the command interpreter executes the timed commands, the telemetry symbol link module searches for the telemetry symbols whose associated receiver sample counter corresponds to the timed command *symbol search sample stamp*, in the FIFO within the telemetry symbol link. When found, the FPGA pops the symbol out of the FIFO.

The telemetry symbol link module uses the PLL 180^o lock indicator in the timed commands to disambiguate the telemetry symbol data phase.

Then, using the data symbol and the signal parameters specified in the timed commands, the signal generator engine automatically computes the transmitted PRN secondary code symbols with the correct alignment. It then regenerates and rebroadcasts the received signals.

This scheme relies on precise synchronization between the receiver and transmitter sample counters, enabling accurate control of transmission timings in the regenerated signals and resulting in very low latency for the regenerated telemetry symbols.

In the current implementation, the telemetry symbol link is limited to rebroadcasting Galileo E1 b/c signals. However, it can be adapted for use with other GNSS signals

5.5 Results

We evaluated the proposed GNSS rebroadcaster's performance across several metrics: raw sample generation, precision of the rebroadcasted GNSS signals, rebroadcaster latency, accuracy of the rebroadcasted signals, power consumption, and FPGA resource usage. The signals generated or rebroadcasted in real time were verified using commercial U-blox NEO-M8T receivers [161]. The GESTALT GNSS testbed facility [146], located at the CTTC premises, was utilized for the tests.

5.5.1 GNSS Signal Raw Sample Generator Validation

To verify the correct generation of GNSS signals, the rebroadcaster was configured to generate GPS L1 C/A and Galileo E1 b/c signals, both separately and in combination, with a duration of 10 minutes, in post-processing mode.

The rebroadcaster was set to generate the signals using a fixed static position. GNSS-SDR was set to post-process the generated signals and dump the computed PVT to a file every 20 ms. The atmospheric corrections were deactivated in GNSS-SDR because the rebroadcaster does not simulate the atmospheric effects. The RMSE of the estimated position was then computed. The RMSE was measured in two ways:

In the first measurement, GNSS-SDR was executed in the PC, using the software version of the signal generator engine (see Fig. 5.6).

In the second measurement, the software transmitter was executed in the rebroadcaster hardware, using the FPGA signal generator (see Fig. 5.7). The purpose of this test was to verify the correctness of the generated signals, and to verify that the results obtained using the software version and the FPGA version of the signal generator engine were equivalent.

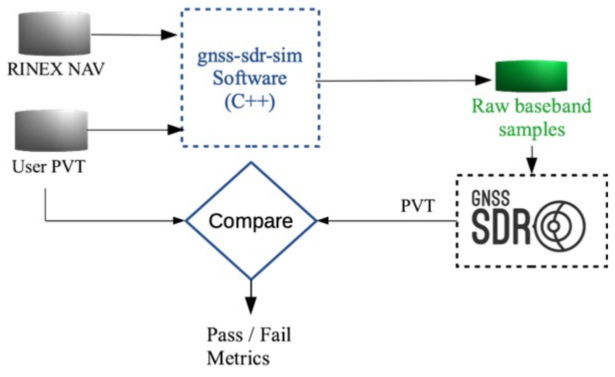

Figure 5.6 GNSS signal raw sample generator validation.

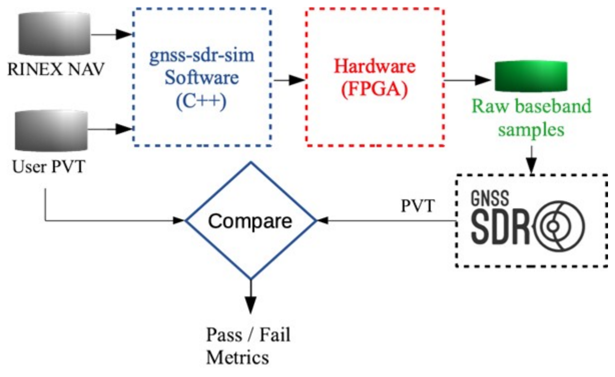

Figure 5.7 GNSS signal raw sample generator validation using the FPGA.

Both results were highly similar, and the synthetic signals were extremely accurate, confirming the correctness of the generated signals, as demonstrated in Table 5.2.

Table 5.2 Measured RMSE of position fixes

SW/FPGA	Signals	RMSE [m]
SW	GPS L1 C/A	0.16
FPGA	GPS L1 C/A	0.21
SW	Galileo E1 b/c	0.30
FPGA	Galileo E1 b/c	0.30
SW	GPS L1 C/A and Galileo E1 b/c	0.60
FPGA	GPS L1 C/A and Galileo E1 b/c	0.48

In another test, the rebroadcaster was set to generate the signal combinations shown in Table 5.2 in real-time mode, using the same static position. The rebroadcaster generated GPS L1 C/A signals and Galileo E1 b/c signals, both separately and in combination. Each combination of signals was generated for a time duration of 10 minutes The RF output was connected to a commercial receiver. The commercial receiver plotted the deviation maps for the whole duration of the generated signals.

In all cases, the PVT solution computed by the commercial receiver was stable and within 10 meters of the true solution, demonstrating the real-time signal generation capabilities of the rebroadcaster.

5.5.2 GNSS Signal Rebroadcasting

This test evaluated the accuracy of signals rebroadcasted in real-time mode by comparing how closely the rebroadcasted PVT solution aligns with the received PVT solution. The accuracy assessment was conducted in a somewhat approximate manner, utilizing graphical deviation maps generated from data collected by two commercial receivers.

The test setup, as illustrated in Figure 5.8, involved connecting the rooftop antenna of the GESTALT testbed to a splitter. This splitter then connected to the input of the rebroadcaster, with the rebroadcaster's output linked to commercial receiver 1. Additionally, the splitter provided a connection to a second commercial receiver (commercial receiver 2). Commercial receiver 1 was tasked with measuring the rebroadcasted Galileo E1b/c signals, while simultaneously, commercial receiver 2 measured the received Galileo E1 b/c signals.

The rebroadcaster was configured to regenerate up to 8 Galileo E1b/c signals in real time, utilizing the telemetry symbol link, over the course of one hour. The receiver part of the rebroadcaster was configured as specified in Appendix 5.A.

The commercial receivers generated deviation maps comparing the received signals with the rebroadcasted signals for the entire duration of the test. The deviation maps obtained with both commercial receivers were compared. The average PVT obtained by the commercial receiver connected to the splitter was considered to be the true PVT.

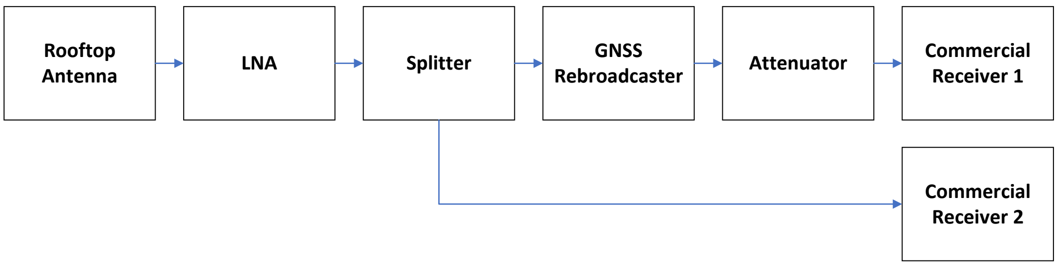

Figure 5.8 Galileo E1 b/c signal rebroadcasting: Deviation Map Test

Figure 5.9 shows the deviation map obtained with the Galileo E1 b/c signals received at the rooftop antenna. Figure 5.10 shows the deviation map obtained with the rebroadcasted Galileo E1b/c signals. This deviation map is centered at the exact coordinates corresponding to the average position obtained with the received Galileo signals in Figure 5.9 for comparison. These deviation maps were recorded over a period of 60 minutes.

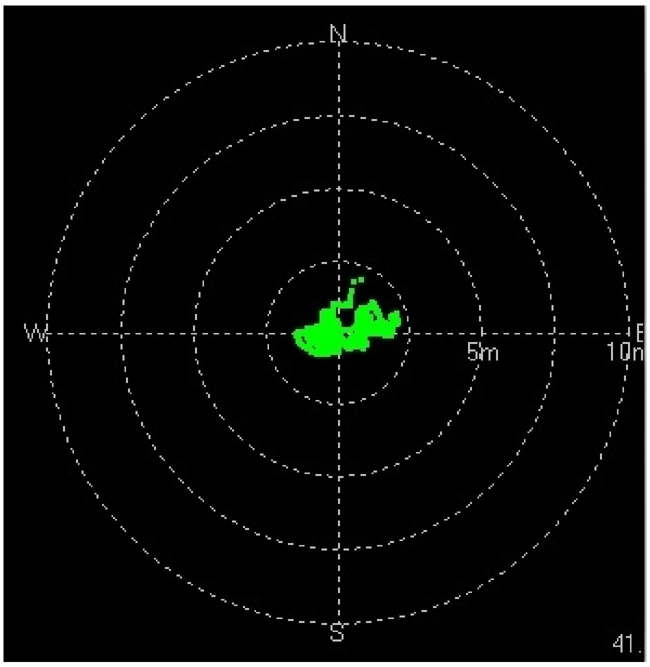

Figure 5.9 Deviation map of the received signal at the rooftop antenna.

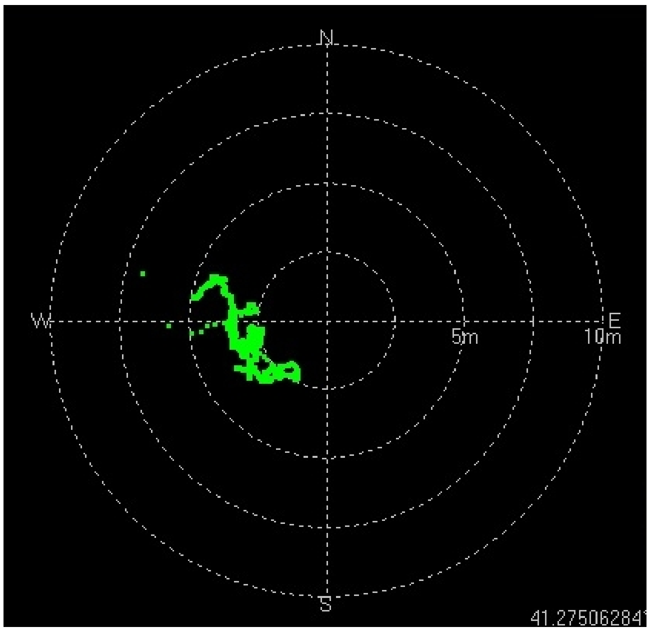

Figure 5.10 Deviation map of the rebroadcasted signal, centered on the average position derived from the received signal at the rooftop antenna, for comparison.

The figures above show that in all cases, the PVT solution obtained with the rebroadcasted signals was stable and within a radius of 5 meters of the PVT solution obtained with the received signals. An offset of approximately 3 meters was observed between both solutions. These results demonstrate the accuracy of the regenerated signals in rebroadcasting a static scenario.

5.5.3 Latency

The test setup, as shown in Figure 5.8, was utilized to measure the latency of the rebroadcasted signal. The latency was verified by comparing the time difference between the 1 Pulse Per Second (PPS) output from commercial receiver 1, which was connected to the regenerator's output, and the 1 PPS output from commercial receiver 2, connected to the rooftop antenna via a splitter. The rebroadcaster was configured to regenerate up to 8 Galileo E1 b/c signals in real time using the telemetry symbol link. The measured latency was approximately 16.7 milliseconds, as detailed in Figure 5.11, demonstrating the ability to rebroadcast the received telemetry symbols with minimal delay, thereby surpassing the target latency.

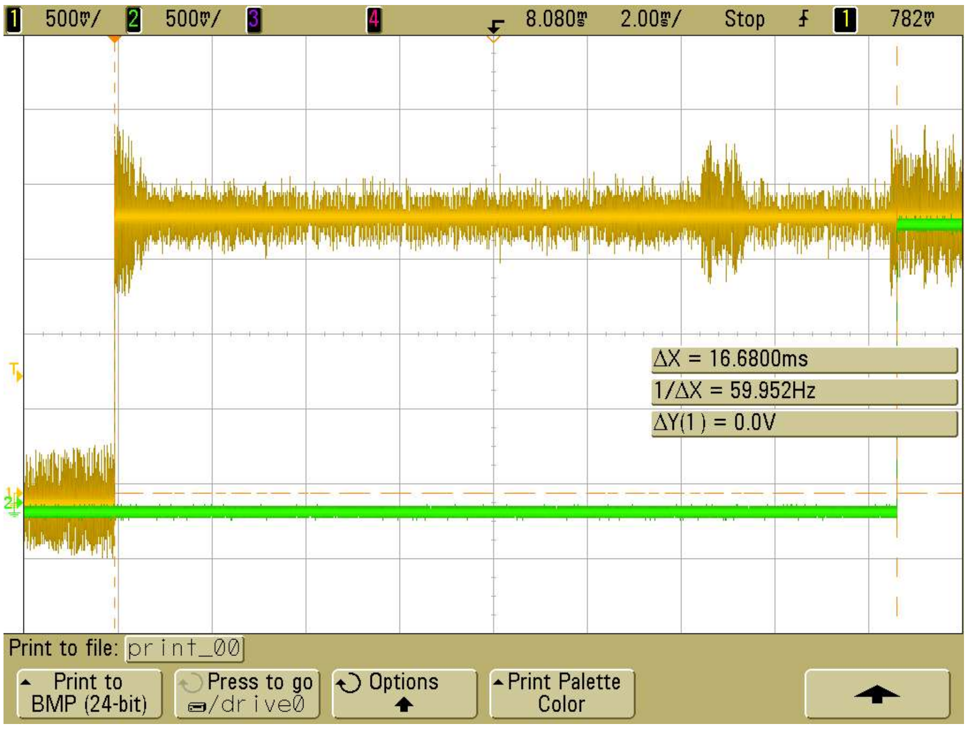

Figure 5.11 Telemetry Symbol Link latency.

5.5.4 Correctness of the Rebrodcasted Navigation Message

The correctness of the rebroadcasted navigation message was verified using the test setup illustrated in Figure 5.8. However, the commercial receivers were replaced with GNSS-SDR-based receivers for this verification.

The rebroadcaster was configured to regenerate up to 8 Galileo E1 b/c signals in real time using the telemetry symbol link.

The correctness of the rebroadcasted signal was verified by measuring the telemetry Cyclic Redundancy Check (CRC) success rate, both in the signal received at the rooftop antenna and in the rebroadcasted signal, and by comparing the results. The CRC statistics were measured over a period of 30 min.

Table 5.3 displays the CRC success rate for each satellite that was tracked by GNSS-SDR in the signal received at the rooftop antenna.

Table 5.3 CRC Statistics of the Rooftop Antenna Signal

Num CRC Tests	Successful Tests	Success Rate	Sat ID
1300	1299	0.99	E31
1826	1825	0.99	E09
1823	1821	0.99	E24
955	953	0.99	E03
1799	1797	0.99	E25
1731	1729	0.99	E05

Table 5.4 displays the CRC success rate for each satellite that was tracked by GNSS-SDR in the rebroadcasted signal.

Table 5.4 CRC Statistics of the Rebroadcasted Signal

Num CRC Tests	Successful Tests	Success Rate	Sat ID
1076	1075	0.99	E31
1628	1627	0.99	E09
1432	1431	0.99	E24
906	905	0.99	E03
774	773	0.99	E25
1624	1623	0.99	E05

The rebroadcasted signal undergoes fewer CRC tests than the received signal because the rebroadcaster begins transmitting GNSS signals only after achieving PVT lock on the received signal. Consequently, the receiver processing the rebroadcasted signal started its measurements later than the receiver connected to the rooftop antenna.

Also, the number of CRC tests differs between satellite numbers because some satellites were acquired earlier than others.

The CRC statistics obtained on the antenna signals are comparable with the CRC statistics obtained on the rebroadcasted signals. The success rate is always above 99%. The CRCs that GNSS-SDR failed to decode corresponded to the first received CRCs when transitioning from acquisition to tracking. The reason for it is that it takes a brief moment for the tracking loops in GNSS-SDR to stabilize when transitioning from acquisition to tracking. During that short period of time, the receiver may fail to validate the CRC due to bit errors in the demodulated messages.

The CRC success rate demonstrates that the rebroadcaster preserves the correctness of the GNSS signals.

5.5.5 FPGA Resource Utilization

Table 5.5 details the FPGA resource utilization for the rebroadcaster developed with the XC7Z035 device. This table reports the resource usage of the GNSS receiver and transmitter features, including channel conditioning, buffering, acquisition and tracking hardware accelerators, signal generators, along with their AXI4 memory-mapped registers, and the telemetry symbol link. The FPGA resource usage is categorized by LUTs, LUTRAMs, FFs, BRAMs, and DSP slices.

The FPGA occupancy is lower than that of the spaceborne receiver reported in Section 4.4.1.5. This is because the rebroadcaster, despite implementing both a receiver and a transmitter in the FPGA, is limited to 8 channels for each component.

Table 5.5 FPGA resource utilization in the rebroadcaster implemented on the XC7Z035 device.

Resource	Utilization	Resources available	Utilization %
LUT	61671	171900	36
LUTRAM	7936	70400	11
FF	77308	343800	22
BRAM	236	500	47
DSP	246	900	27

5.5.6 Power Consumption

The power consumption of the Zynq-7000 XC7Z035-2L FBG676I SoC FPGA, implementing the GNSS rebroadcaster, is estimated to be 5.2 W using FPGA design tools. This estimation assumes an average processor load of approximately 50%, which occurs when the rebroadcaster is receiving and regenerating 8 satellite signals simultaneously. This average processor load was estimated using the embedded GNU/Linux OS operating system resource manager.

The live generation and regeneration of GNSS signals has not been implemented in pure software. For this reason this result is not compared against the power consumption of a general-purpose processor. However, this result is consistent with the power consumption estimates for the spaceborne receiver utilizing the same SoC FPGA, which is approximately 6.5 W (refer to Section 4.4.1.6). Unlike the spaceborne receiver, which processes up to 24 channels, the rebroadcaster operates with only 8 channels. However, it integrates both receiver and transmitter functionalities within the same SoC FPGA. As a result, it consumes more power than the spaceborne receiver for an equivalent number of channels.

5.6 Conclusions

This chapter presented the design, proof-of-concept implementation and preliminary performance assessment of a low-cost, real-time, portable, low power, and small form factor GNSS rebroadcaster, providing practical details of a working prototype.

This design highlights the efficacy of SDR techniques for generating and rebroadcasting GNSS signals in real time, with the potential to modify the rebroadcasted PVT solutions with reduced latency, and simulate various GNSS scenarios. The proposed rebroadcaster enables the regeneration of any unpredictable data embedded in the navigation messages. The integration of these custom processing capabilities in a low-cost SoC FPGA design, and an open-source software-defined GNSS receiver development model constitutes a significant novelty in the field.

After detailing the system architecture, this chapter assessed the rebroadcaster's performance across various KPIs. These included the number of parallel channels processed in real time, the correct functioning of the various available operating modes and GNSS signal types, power consumption, and accuracy, referring to how close the rebroadcasted navigation solutions were to the received navigation solutions. The quality of the rebroadcasted PVT solutions was assessed by comparing them to the PVT solutions derived from live signals. Both navigation solutions were estimated using commercial receivers. The results demonstrated the correctness of the rebroadcasted signals. The rebroadcaster was tested in real time using a static scenario, demonstrating that the regenerated navigation solutions achieve an accuracy within 5 m of the received PVT.

Possible improvements considered for future work includes adding compatibility with GPS L5 and Galileo E5a signals, enabling the rebroadcasting of GNSS signals in dual-frequency mode. Also, the use of the telemetry symbol link for GPS signals would enable the fast estimation and retransmission of not only Galileo but also GPS navigation messages with a reduced latency, enabling the testing with live signals, for instance in a vehicular test campaign replicating the correct dynamic and channel impairments. Furthermore, the tests indicate potential for reducing the rebroadcasted latency. This is anticipated, as the estimation of the Galileo E1b/c telemetry symbols requires only 4 ms. Finally, the rebroadcaster has been validated using static scenarios that last one hour. Future efforts will focus on assessing its performance with a wider range of dynamic GNSS scenarios over extended periods. This includes enhancing the rebroadcaster's capability to update the ephemeris data in the retransmitted satellite signals not only at the point of acquisition but also whenever this data is updated by the satellites themselves.

The results presented in this chapter were partially published in:

- [21] M. Majoral, J. Arribas, and C. Fernández-Prades, "Implementation of a GNSS Rebroadcaster in an All-Programmable System-On-Chip Platform," in *2022 10th Workshop on Satellite Navigation Technology (NAVITEC)*, Noordwijk, Netherlands, April 2022, pp. 1–9. doi: 10.1109/NAVITEC53682.2022.9847537

The work presented in this chapter was supported by the following funding sources and collaborators:

- Contract AO/2-1647/17/NL/CRS with the European Space Agency.
- Grant RTI2018-099722-B-I00, funded by the Spanish Ministry of Science and Innovation

Appendix 5.A Receiver Configuration for Rebroadcasting Galileo E1b/c Signals

This section describes the receiver configuration used to rebroadcast Galileo E1b/c signals using the telemetry symbol link. Table 5.6 shows the configuration of the acquisition block. Table 5.7 shows the configuration of the Tracking block. Finally, Table 5.8 shows the configuration of the PVT block. The sampling frequency is set to 12.5 Msps.

Table 5.6 outlines the receiver acquisition parameters. It includes the Doppler max parameter, representing the highest Doppler frequency in the search space, and the Doppler step parameter, which refers to the increment between frequencies within the search grid. The threshold parameter in Table 4.13, denoted as Γ in Algorithm 1 in Chapter 3, represents the decision threshold above which a signal is considered present. A downsampling filter is used in the L1/E1 band to reduce the acquisition latency. The downsampling filter is only used during acquisition.

Table 5.6 Rebroadcaster acquisition configuration.

Parameter type	Parameter	Value
Acquisition Galileo E1b/c	Doppler Max	5000 Hz
	Doppler Step	125 Hz
	Threshold	2.0
	Downsampling Factor	4

Table 5.7 displays the receiver tracking configuration parameters for Galileo signals. The coherent integration time is set to 20 ms to increase the apparent signal-to-noise ratio. The early-late space chips is the spacing between the Early and Prompt, and between the Prompt and Late correlators, normalized by the chip period. The early-late narrow space chips is the spacing between the Early and Prompt, and between the Prompt and Late correlators, normalized by the chip period, after bit synchronization. The very early-late space chips is the spacing between the Very Early and Prompt and between the Prompt and Very Late correlators, normalized by the chip period The very early late space narrow chips is the spacing between the Very Early and Prompt, and between the Prompt and Very Late correlators after removal of the secondary code and extension of the coherent integration time, normalized by the chip period. The PLL filter bandwidth is the bandwidth of the PLL low-pass filter. The PLL filter bandwidth with narrow correlator configuration is the bandwidth of the PLL low-pass filter after removal of the secondary code. The DLL filter bandwidth is the bandwidth of the DLL low pass filter. Finally, the DLL filter bandwidth with narrow correlator configuration is the bandwidth of the DLL low-pass filter after the removal of the secondary code.

Table 5.7 Rebroadcaster tracking configuration.

Parameter type	Parameter	Value
Tracking Galileo E1b/c	coherent integration time	20 ms
	Early-Late space chips	0.25 chips
	Very Early-Late space chips	0.5 chips
	Early-Late space narrow chips	0.15 chips
	Very Early-Late space narrow chips	0.5 chips
	PLL filter bandwidth	15 Hz
	PLL filter bandwidth (narrow correlator configuration)	5 Hz
	DLL filter bandwidth	0.75 Hz
	DLL filter bandwidth (narrow correlator configuration)	0.2 Hz
	FLL during pull-in time	Enabled
	FLL bandwidth	10 Hz

Table 5.8 shows the receiver PVT configuration parameters: the positioning mode is set to single-point positioning. The RAIM FDE is enabled. The ionospheric correction is performed according to the broadcasted ionospheric model. A Saastamoninen tropospheric model is used [151]. The PVT output rate is set to 100 ms, and the satellites marked as unhealthy are not used for the computation of the PVT solutions. The Kalman filter was not used.

Table 5.8 Rebroadcaster PVT configuration.

Parameter type	Parameter	Value
General	Positioning Mode	Single
	Receiver Autonomous Integrity Monitoring (RAIM) Fault Detection and Exclusion (FDE)	Enabled
	Iono Model	Broadcast
	Trop Model	Saastamoinen
	PVT Output Rate	100 ms
	Use unhealthy sats	Disabled

More information regarding the GNSS-SDR configurable parameters can be found in [17].

Chapter 6

High-Sensitivity GNSS Receiver

This Chapter presents the design, proof-of-concept implementation, and preliminary performance assessment of an affordable real-time HS-GNSS receiver implemented using the architecture and design methodology presented in Chapter 3.

Specifically tailored to capture and track weak Galileo E1b/c signals, this receiver aims to support research endeavors focused on advancing GNSS signal processing algorithms, particularly in scenarios characterized by pronounced signal attenuation. Leveraging SoC-FPGA technology, this design merges the adaptability of SDR concepts with the robust hardware processing capabilities of FPGAs.

This innovative approach enhances power efficiency compared to conventional designs relying on general-purpose processors, thereby facilitating the development of embedded software-defined receivers. Furthermore, the FPGA massive parallelism, facilitates the execution of the computationally intensive algorithms required for the acquisition of weak GNSS signals in real time.

The proposed receiver's underlying architecture implements a modular GNSS baseband processing engine, offering a versatile platform for the integration of novel algorithms. The receiver undergoes testing with live signals, showcasing its capability to process GNSS signals even in challenging scenarios with a C/N_0 as low as 20 dB-Hz, while delivering navigation solutions. This work contributes to the advancement of low-cost, high-sensitivity GNSS receivers, providing a valuable tool for researchers engaged in the development, testing, and validation of experimental GNSS signal processing techniques.

The complete design and implementation of the HS-GNSS receiver was carried out in the framework of this thesis. This included developing a high-sensitivity mode for GNSS-SDR, creating a high-sensitivity acquisition hardware accelerator, and conducting verification tests.

This chapter is organized as follows: Section 6.1 provides an introduction. Section 6.2 reviews previous work investigating ways to increase the sensitivity of GNSS receivers using a combination of CI and Non-Coherent Integration (NCI), employing PDI techniques. Additionally, it discusses previous work investigating ways to implement these techniques in receivers based on FPGAs. Section 6.3 summarizes the objectives of the proposed HS-GNSS receiver. Next, Section 6.4 describes the design of the proposed receiver. This is followed by Section 6.5, which reports the performance test results, and finally, Section 6.6 presents the conclusion and directions for future work.

6.1 Introduction

GNSS technology allows us to accurately know our location in real-time and in open-sky environments. Due to the success achieved by this technology, there is an increasing demand for navigation in signal-challenged environments such as foliage canopy, urban canyons, and indoor scenarios [162]. However, these environments present great difficulties for GNSS receivers, as the received signals are severely degraded due to the presence of obstacles in the propagation path between the satellites and the receiver. Real-world usage of a GNSS receiver in indoor scenarios presents various operational challenges. The standard outdoor working conditions, under open sky, are characterized by nominal C/N_0 values of typically $\geq$ 44 dB-Hz. These conditions vary when the receiver is situated under trees or within a foliage canopy, and differ significantly from indoor environments, where signal levels decrease after penetrating roofs, walls, and windows [18,35]. Indoor scenarios lead to inferior signal detection and degrade the quality of measurements due to higher noise and multipath effects. The major degradation that the indoor receivers have to cope with is high signal attenuation [163]. The signal reception is influenced by building materials and the receiver's location, resulting in attenuation losses of approximately 10 to 20 dB in soft-indoor scenarios, 20 to 35 dB in indoor scenarios, and exceeding 35 dB in deep indoor scenarios [18]. The terms *soft indoor*, *indoor*, and *deep indoor* describe varying degrees of GNSS satellite signal obstruction caused by construction materials or vegetation. Soft indoor scenarios occur when GNSS signals are weakened by partial blockages, reflections, and diffractions as they navigate through or around buildings and other obstacles. These scenarios still provide some level of GNSS signal availability. Indoor and deep indoor scenarios involve more significant signal blockage, resulting in greatly reduced signal strength [18, 164].

GNSS-based applications are increasingly expanding their reach to encompass more demanding scenarios, including urban and light-indoor environments. In areas with weak or compromised signals, they improve the availability of satellite signals [162, 163]. This capability enhances the overall reliability and performance of GNSS receivers in diverse settings. This is significant for various applications, including location-based services (LBSs) and emergency response [165]. Operating effectively in conditions where traditional receivers may struggle, HS-GNSS receivers provide extended coverage in marginal signal conditions. The heightened need for navigation in challenging signal environments has led to a growing focus on handling faint signals [162]. This shift is motivating the development of receivers that operate in these challenging scenarios [1].

As a response to this challenge, HS techniques have been developed for GNSS receivers [1]. HS-GNSS receivers enable improved acquisition and tracking capabilities in degraded signal environments, albeit at the cost of increased complexity in terms of computational load [37]. Existing solutions that provide high accuracy and sensitivity are often constrained by their elevated energy consumption [1]. Methods to alleviate power usage encompass snapshot solutions, cloud-based receivers, and Assisted Global Navigation Satellite System (A-GNSS) receivers [1, 35].

Snapshot receivers obtain the navigation solutions by analyzing a brief segment of the received satellite signal. This involves sampling times on the order of a few milliseconds. The distinctive feature of a snapshot receiver lies in its capacity to function effectively within these brief signal sampling intervals. This characteristic makes it well-suited for a diverse range of positioning applications where the constraint of energy usage poses a significant challenge in embracing traditional GNSS solutions [166].

Cloud-based GNSS receivers leverage shared computational resources. Computational tasks typically carried out on-chip are migrated to a cloud server with the objective of enhancing the sensor's battery lifetime without compromising the performance [167, 168]. The cloud-based receiver can be set up in a manner where solely the signal capture circuitry is employed to temporarily store the digital samples. Signal processing can be deferred until the digital samples can be transmitted, without impacting the device's battery life (*e.g.*, during battery recharge), to a distant cloud server [1].

A-GNSS solutions enhance the performance of standard receivers by delivering information, using an alternate communication pathway, that the receiver would typically acquire directly from the satellites. Assistance reduces the time and information dependency on satellites for obtaining navigation solutions. Consequently, the A-GNSS receiver can swiftly make measurements from the satellites, even with weaker signals, surpassing the capabilities of an unassisted receiver [35].

HS techniques are primarily developed to capture faint signals. This is usually achieved by coherently accumulating signal samples over an extended period, performing extended CI [1]. However, several critical factors impose constraints on the maximum CI time, including the presence of unknown data bit transitions, residual frequency errors in the received signal, and the existence of phase noise due to receiver oscillator instabilities [18].

Sign reversals within the CI window may occur due to unknown data bit transitions in the received signal, potentially causing partial or complete cancellation of correlation power. The impact of these unknown bit transitions can be mitigated by using pilot signals [1]. A pilot signal is transmitted within the Galileo signals (E1, E5a, E5b, E6) and modernized GPS signals (L1C, L2C, L5) to improve the signal acquisition and tracking [18]. Both the data and the pilot signals are modulated with a unique code to distinguish them from other signals. This code is a pseudo-random noise (PRN) sequence known as a spreading code. Several GNSS signals build a long spreading sequence in a tiered manner whereby a secondary code sequence is used to modify successive repetitions of a primary code.

Residual frequency errors impact the received signal after Doppler wipe-off, potentially leading to signal cancellation if the CI exceeds a certain time duration. A smaller residual frequency shift allows for an extended CI before the SNR gain diminishes due to phase wrapping. This phenomenon introduces a trade-off between SNR gain and computational load, as minimizing the residual frequency shift necessitates a finer Doppler search, demanding increased computational operations [18].

The quality of the user receiver clock also limits the CI time. At present, TCXOs and Oven-Controlled Crystal Oscillators (OCXOs) are the most commonly utilized types of clocks. As per the simulations outlined in [37, 169, 170], the TCXO restricts the CI interval to approximately 100 ms, whereas the OCXO enables the utilization of CI times extending up to 1 s. The sensitivity of the receiver is influenced by this fact. Consumer-grade receivers commonly utilize TCXO clocks, while OCXO clocks are typically reserved for professional applications. The higher cost of OCXO clocks, as opposed to TCXO clocks, restricts their widespread adoption in mass-market GNSS receivers [1].

For practical considerations, the CI time cannot be indefinitely extended due to the reasons mentioned above. Consequently, the only recourse is to employ a combination of coherent and non-coherent integration to prolong the overall integration time and enhance receiver sensitivity [18, 37]. More specifically, the acquisition begins with coherent correlation, followed by non-coherent accumulation of outputs from multiple coherent correlations through nonlinear operations. Non-coherent accumulation is commonly achieved through the application of PDI techniques. These techniques address the constraints associated with coherent accumulation and enable the receiver to capture satellites even in conditions of extremely low C/N_0 [37]. However, the NCI time cannot be increased without bounds either, due to limited Doppler estimation accuracy, receiver clock drift instabilities, and the relative movement between the satellites and the receiver [18].

Any errors in determining the Doppler frequency can affect the baseband signal, akin to a mismatch between the chip duration of the received signal and the local code replica. If this mismatch extends over a significant integration interval, it causes blurring in the overall correlation, introducing bias in the final estimation of the code phase. The duration of the NCI also remains limited due to local clock drift instabilities, unless methods to estimate the clock dynamics are implemented. In addition, processing the received signal over the total integration time yields a single Doppler frequency and code phase estimation for each satellite. However, the accuracy of this time-delay estimation is limited by the dynamic motion of both the satellites and the receiver during the correlation time [18].

6.2 Background

Acquiring weak GNSS signals requires significant computational effort due to the need for extended integration. The most time-consuming operation during the acquisition phase is correlating the input signal with a locally generated replica. This process, explained in Section 2.3.4, involves multiplying thousands of samples every millisecond for each tested code phase, Doppler frequency, and satellite [18]. Consequently, developing a GNSS receiver with high sensitivity that works in real time presents a major challenge. Several strategies have been explored to reduce the computational demands of GNSS signal acquisition algorithms.

Several publications explore the application of CI and PDI techniques to increase the sensitivity of GNSS receivers. Ref. [169] investigates and assesses the performance of several PDI techniques, including Non-Coherent Post Detection Integration (NPDI), Differential

Post-Detection Integration (DPDI), Post-Detection Integration Truncated (GPDIT), squaring detector, and Post-Detection Integration Truncated with Squaring Detector (GPDITSD), which employs the GPDIT strategy and the squaring detector. The evaluation of these techniques involves the use of three types of clocks: a TCXO, a Chip Scale Atomic Clock (CSAC), and an OCXO. In [171], an analysis is performed to identify the most effective PDI technique in the presence of various impairments like data bit transitions and frequency offset. Additionally, [170] conducts a comprehensive investigation into current PDI techniques, specifically addressing the impact of phase noise originating from two different clocks: a TCXO and an OCXO. Ultimately, the PhD thesis referenced in [37] consolidates comprehensive insights and benchmarks the most relevant PDI techniques for acquiring GNSS signals under the challenging conditions mentioned above (phase noise, frequency offset, and the presence of data bits). According to the simulation results presented in [37], a receiver equipped with a TCXO can reliably acquire GNSS signals at a C/N_0 as low as 20 dB-Hz. This achievement is possible by employing the GPDIT strategy, using a CI time of 100 ms, and allowing for a maximum of seven non-coherent combinations. These conditions result in a total integration time of 700 ms. The GPDIT strategy is not robust against data bit transitions, necessitating that the receiver either eliminates the modulated data or utilizes pilot signals. This approach is implementable in the FPGA and facilitates the acquisition of GNSS signals in soft indoor scenarios and in outdoor scenarios with signal impairments. For this reason, the HS-GNSS receiver prototype introduced in this thesis adopts the GPDIT strategy and acquires the pilot signals.

The proposed receiver employs the PCPS algorithm to achieve CI by performing circular cross-correlation in the frequency domain between the received signal and a locally generated replica of the satellite's PRN code [38]. The PhD thesis referenced in [172] explores methods to optimize the implementation of the PCPS algorithm for HS-GNSS receivers, aiming for reduced complexity without incurring implementation losses. The detection of weak signals demands extended CI times. Consequently, the computation of the circular cross-correlations accounts for the presence of both primary and secondary codes in the pilot signals and requires the use of large FFTs and IFFTs. The thesis being referenced to investigates the implementation of this algorithm in FPGAs, utilizing typical vendor-specific FFT and IFFT IP cores. These cores typically have limitations on transform lengths, requiring them to be powers of two. As a result, the optimization of the PCPS algorithm is driven by the need to efficiently compute large FFTs using small power-of-two length FFTs and the need to minimize resource usage and algorithm latency.

Minimizing latency is crucial because excessive delays negatively affect the accuracy of parameters estimated during the acquisition process, such as PRN code phase and Doppler frequency. This degradation primarily arises from receiver clock drift and the variable nature of Doppler frequency in the received signals. This is explained in detail in Section 6.4.5. Ref. [172] also shows ways to compute an N-point FFT using a combination of two FFTs, each of $\frac{N}{2}$ points. One way to do it is by separating the input samples by parity and the output samples by section. This method can be generalized to compute an N-point FFT using P FFTs, each of $\frac{N}{P}$ points. The same method is applicable to the computation of IFFTs. The receiver presented in this paper adopts this approach. This method enables the efficient implementation of the CI in the FPGA using a multi-channel FFT, with an optimized pipeline scheme that minimizes latency, albeit with a small increase in mathematical operations compared to the computation of a single large FFT. This algorithm does not require complex branch decisions, as it relies on a streamlined and efficient process, resulting in faster execution.

Ref. [172] also illustrates that selecting the optimal implementation of an algorithm for an FPGA entails several trade-offs, including the balance between speed and FPGA resource utilization, managing power consumption while optimizing performance, and algorithm complexity versus available FPGA resources.

6.3 Objectives

The development of HS-GNSS receiver technology requires the implementation of non-standard features and a comprehensive description of the signal processing path from the antenna to the computation of the desired GNSS products. However, developing prototypes using off-the-shelf GNSS receivers poses difficulties. Researchers must contend with the increasing complexity and integration level of GNSS integrated circuits [8,117]. These systems do not offer an exact model of how the desired measurements are obtained, and they have limited reprogrammability.

In light of these challenges, this chapter aims to demonstrate the effectiveness of the SoC FPGA architecture and design methodology outlined in Chapter 3. The objective is to show how these approaches enable the creation of a programmable, adaptable, and portable HS-GNSS receiver prototype, designed specifically to advance research on experimental algorithms suited for weak signal conditions.

The proposed receiver implements two operating modes: high-sensitivity mode and normal-sensitivity mode. When operating in high-sensitivity mode, the receiver is capable of acquiring and tracking Galileo E1b/c signals with a C/N_0 down to 20 dB-Hz (equivalent C/N_0 observed at the post-correlation level), enabling the derivation of navigation solutions. On the other side, when operating in normal-sensitivity mode, the receiver processes GPS L1 C/A, Galileo E1b/c, GPS L5, and Galileo E5a signals with an acquisition sensitivity of approximately 37 dB-Hz. While the ideal scenario involves acquiring and tracking both GPS and Galileo signals in high-sensitivity mode, the initial concept demonstrator presented in this paper is designed to showcase high-sensitivity mode only for Galileo E1b/c signals.

To enhance the availability of satellite signals, the receiver is capable of processing Galileo E1 b/c signals in high-sensitivity mode, while simultaneously processing GPS L1 C/A, GPS L5, and Galileo E5a signals in normal-sensitivity mode. This approach implements a dual-band, multi-GNSS receiver centered at 1176.45 MHz and 1575.42 MHz.

When operating in high-sensitivity mode, the receiver uses assistance to speed up the acquisition of weak Galileo E1b/c signals and ultimately to decrease the time-to-first-fix (TTFF).

6.4 System Design

6.4.1 High Sensitivity GNSS Receiver Architecture

The proposed HS-GNSS receiver prototype is based on AMD's ZCU102 development board [173], featuring an AMD Zynq UltraScale+ XCZU9EG-2FFVB1156 All-Programmable Multi-Processor SoC (MPSoC) [63]. A block diagram of the concept demonstrator is shown in Figure 6.1. As explained in Chapter 4, the XCZU9EG MPSoC features PL equipped with 600k logic cells and 2520 DSP slices and a PS that houses a Quad ARM Cortex-A53 MultiProcessor Core (MPCore) running at 1.3 GHz. It also includes several peripheral interfaces, such as a LAN interface, enabling the remote control of the receiver. The receiver incorporates an Analog Devices AD-FMCOMMS5-EBZ analog front-end [119], connected to the ZCU102 evaluation board. The AD-FMCOMMS5-EBZ includes two AD9361 Radio Frequency (RF) transceivers ([106]) covering the full 6 GHz range. Specifically, one RF transceiver is tuned to the E1/L1 frequency band, and the other is tuned to the E5a/L5 frequency band. This platform enables the implementation of a Dual-Band GNSS receiver, offering end-to-end functionality from the RF output of the antenna to real-time generation of navigation products on a portable board. The receiver uses a Tallysman TW8825 antenna ([174]), which provides dual-band GPS L1/L5, GLONASS G1, Galileo E1/E5a, and BeiDou B1 coverage.

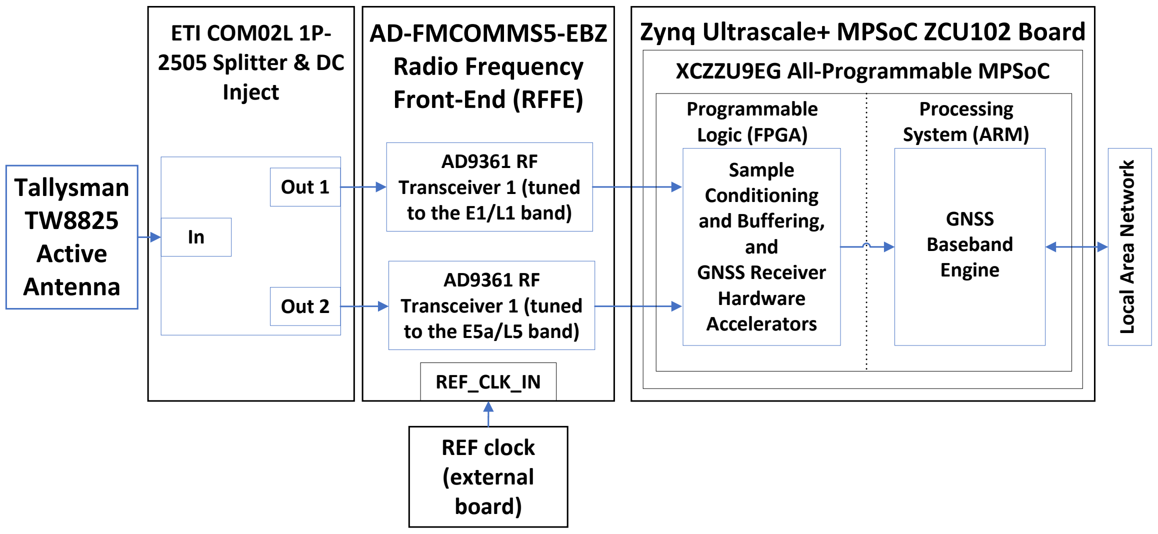
ETI COM02L 1P-2505 Splitter & DC Inject
AD-FMCOMMS5-EBZ Radio Frequency Front-End (RFFE)
Zynq Ultrascale+ MPSoC ZCU102 Board
XCZZU9EG All-Programmable MPSoC
Programmable Logic (FPGA)
Processing System (ARM)
Tallysman TW8825 Active Antenna
In
Out 1
Out 2
AD9361 RF Transceiver 1 (tuned to the E1/L1 band)
AD9361 RF Transceiver 1 (tuned to the E5a/L5 band)
Sample Conditioning and Buffering, and GNSS Receiver Hardware Accelerators
GNSS Baseband Engine
Local Area Network
REF_CLK_IN
REF clock (external board)

Figure 6.1 High-sensitivity GNSS receiver block diagram.

The local oscillator included with the AD-FMCOMMS5-EBZ board, an RXO3225M IC crystal by Rakon clocked at 40 MHz, has a stability of ±25 ppm, which is insufficient for GNSS signal processing. Therefore, an external oscillator is needed. The proposed receiver implementation was tested using two external oscillators: a TCXO and an OCXO. The selected TCXO was the AST3TQ-50 by Abracon, in its 40 MHz version [143]. This device features a frequency stability of ±50 ppb over a temperature range of −40 °C to +85 °C, making it well-suited for GNSS applications. The chosen OCXO was an ECOC-2522 by ECS [175], operating at 40 MHz, with a frequency stability of ±10 ppb.

The SoC FPGA implements the architecture detailed in Chapter 3, with Figure 6.2 depicting its various FPGA components. As explained in Section 3.1.1, the FPGA incorporates sample conditioning and buffering, as well as the high-sensitivity acquisition and tracking multicorrelator hardware accelerators. Additionally, it features an interface to the processing system (PS/PL interface).

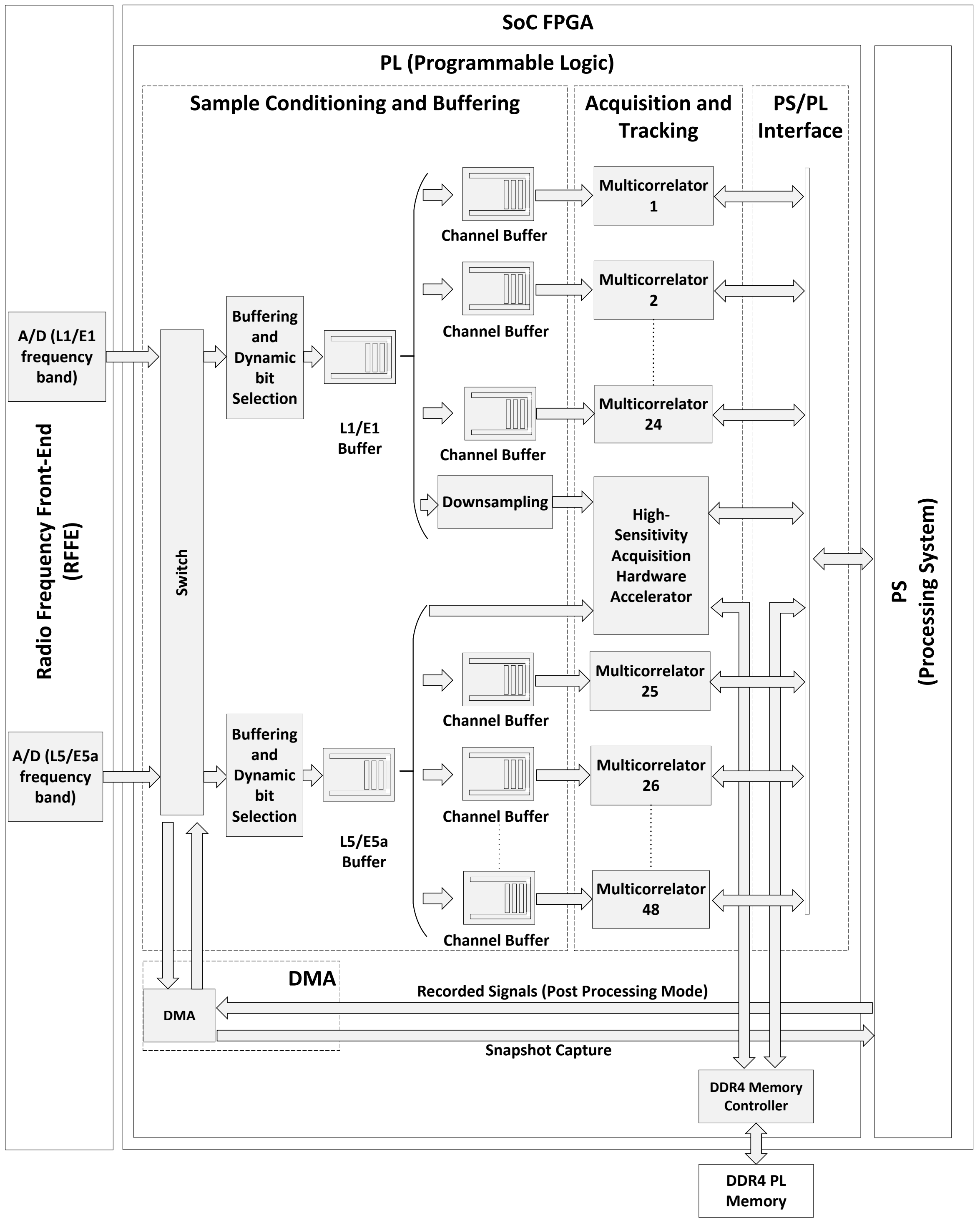

Figure 6.2 Detailed FPGA design.

The sample conditioning and buffering incorporates dynamic bit selection in each frequency band. This process dynamically selects the most significant bits from the incoming signal samples, mapping their dynamic range to the sample quantization in the hardware accelerators.

Also, as explained in Section 3.1.1, the FPGA features a downsampling filter in the L1/E1 band, in front of the acquisition hardware accelerator. In this way, the receiver utilizes a lower sampling frequency during the initial signal acquisition phase and a higher frequency for subsequent tracking. Such a design is optimized based on the fact that signal detection gains little from a bandwidth larger than needed to cover the main signal lobes—about 2 MHz for GPS L1 C/A and 4 MHz for Galileo E1b/c signals—since broader bandwidths introduce more noise [105]. Nonetheless, a wider bandwidth during the tracking phase enables more accurate carrier and code phase identification. Moreover, the initial reduction in sampling frequency streamlines the acquisition process, making it more efficient.

The hardware accelerators are implemented in the form of reusable IP cores, which can be targeted at many variants of FPGAs. The PS/PL interface uses the Advanced Microcontroller Bus Architecture (AMBA) Advanced eXtensible Interface (AXI4) protocol specification [61].

To add high-sensitivity functionality, we made significant modifications to the SoC-FPGA receiver architecture. These enhancements include the introduction of a new acquisition hardware accelerator in the FPGA and updates to the GNSS-SDR software receiver.

The new acquisition hardware accelerator, named the *high-sensitivity acquisition hardware accelerator* as illustrated in Figure 6.2, is designed to perform extended CI, enabling real-time detection of weak signals. This allows the receiver to operate in both normal and high-sensitivity modes. In contrast, the standard acquisition hardware accelerator, described in Section 3.1.4, is limited to normal sensitivity mode.

The high-sensitivity acquisition hardware accelerator necessitates temporary storage of received signal samples to compute the CI. To facilitate this, the acquisition is directly connected to a Double Data Rate 4 (DDR4) memory controller implemented in the FPGA. The memory controller, in turn, is connected to a DDR4 memory component on the ZCU102 board, which is located outside the SoC FPGA, but directly accessible by the FPGA itself. This memory component is called *PL memory* in Figure 6.2, because it is directly connected to the Programmable Logic (the FPGA). By utilizing this external memory, the acquisition hardware accelerator can efficiently store received signals temporarily and compute the CI, bypassing the PS. The PL memory is also accessible from the embedded processor, allowing the processor to read the results of the CI performed by the hardware accelerator. The PS is also connected to another DDR4 memory component supporting the embedded OS and used for program execution in the embedded processor. The memory component connected to the processing system is not illustrated in Figure 6.2.

To implement the high-sensitivity capabilities, the following modifications were applied to the GNSS-SDR software receiver: a new acquisition block was created in GNSS-SDR. This block processes the results computed by the new high-sensitivity acquisition hardware accelerator and conducts part of the acquisition algorithm in the embedded processor. GNSS-SDR was also modified to utilize assistance data for performing Doppler prediction, narrowing down the Doppler search space during acquisition. Moreover, GNSS-SDR was updated for tracking severely attenuated signals. Finally, GNSS-SDR was extended to use assistance data for obtaining navigation solutions in situations where the navigation message of weak

GNSS signals cannot be reliably demodulated due to the presence of noise. With all these enhancements, the proposed HS-GNSS receiver produces PVT solutions in the presence of weak signals and in real-time. These features are elaborated further in this chapter.

Figure 6.3 shows a picture of the HS-GNSS Receiver prototype. The receiver has been fully constructed with COTS components. Incorporating the ZCU102 and AD-FMCOMMS5-EBZ development boards results in a sizable design. Approximately, the ZCU102 board measures 23.8 cm $\times$ 24.4 cm, and the analog front-end measures 14 cm $\times$ 9 cm. Despite these dimensions, the receiver remains portable. For instance, it can be placed in backpacks to facilitate testing in soft indoor environments and in outdoor scenarios with signal impairments. The ZCU102 includes several components that are unnecessary for the implementation of the HS-GNSS receiver. Therefore, the size of the receiver could be potentially reduced by using a custom-designed PCB.

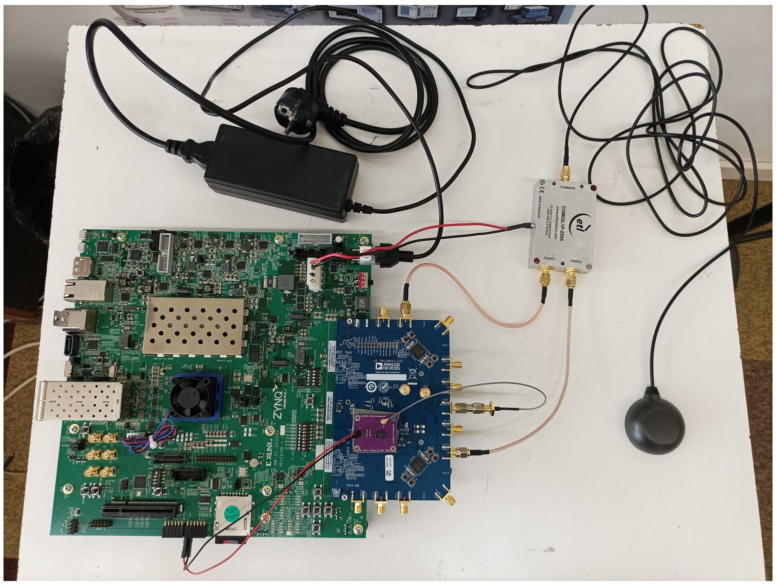

Figure 6.3 Picture of the high-sensitivity GNSS receiver prototype.

6.4.2 Receiver Operating Modes

The proposed concept demonstrator implements two operating modes: high-sensitivity mode and normal-sensitivity mode. High-sensitivity mode enables the processing of weak Galileo E1b/c signals with a C/N_0 down to 20 dB-Hz. Normal-sensitivity mode enables the processing of both Galileo signals and GPS signals received at nominal C/N_0 levels. The default configuration of the receiver is set to operate in high-sensitivity mode for Galileo E1b/c signals, and concurrently in normal-sensitivity mode for Galileo E5a, GPS L1 C/A, and GPS L5 signals, as illustrated in Table 6.1. This dual-mode setup is designed to enhance receiver availability whenever possible, given that high-sensitivity mode is currently only supported for Galileo E1b/c signals.

Table 6.1 Default receiver operating modes.

Signal	Operating Mode	Receiver Sensitivity
Galileo E1b/c	High-Sensitivity mode	20 dB-Hz
GPS L1 C/A GPS L5 Galileo E5a	Normal-sensitivity mode	approx. 37 dB-Hz

6.4.3 Assistance Data

The proposed demonstrator utilizes GNSS assistance data to process weak Galileo E1b/c signals. This assistance data is provided in various forms, including a reference date and time, a reference user location, Galileo ephemeris data, Galileo ionospheric data, and the Galileo UTC model, as detailed in Table 6.2. The receiver obtains the current date and time from the embedded GNU/Linux OS clock, retrieves the reference user location from the GNSS-SDR configuration file, and accesses the Galileo ephemeris data, ionospheric data, and UTC model from XML files. These XML files, which contain data about the visible satellites, can be generated by the receiver itself by placing it outdoors and operating it in normal-sensitivity mode.

Table 6.2 Assistance data.

Assistance Data	Source
Reference date and time	Receiver embedded OS clock
Reference user location	GNSS-SDR configuration file
Galileo ephemeris data Galileo ionospheric data Galileo UTC model	XML files

Table 6.3 outlines the purpose of the assistance data and specifies the data that is utilized for each designated task. The receiver uses the reference date and time, the reference user location, and the assistance ephemeris data to estimate the Doppler frequency of the received signals and reduce the Doppler search space during acquisition. The reference date and time are also utilized to estimate the transmitted TOW in the navigation messages. This estimation is needed because the TOW is essential for performing GNSS basic measurements. However, when tracking severely attenuated signals, the receiver may struggle to reliably demodulate the TOW field of Galileo E1b/c navigation messages due to noise-induced bit errors [176, 177]. Additionally, the assistance ephemeris data, ionospheric data and UTC model are utilized to compute the PVT when the receiver cannot reliably demodulate the navigation messages due to the presence of bit errors.

Table 6.3 Purpose of the assistance data.

Objective	Assistance Data Used
Doppler frequency estimation	Reference date and time, reference user location, Galileo ephemeris data
TOW estimation	Reference date and time
Computation of the navigation solutions	Galileo ephemeris data, ionospheric data, and UTC model

The Doppler prediction, the TOW estimation, and the computation of the navigation solutions are explained in more detail in Sections 6.4.5, 6.4.8, and 6.4.9, respectively.

6.4.4 Acquisition in High-Sensitivity Mode

6.4.4.1 Theory of Operation

The high-sensitivity acquisition process makes it possible to identify faint GNSS signals. Upon detecting a signal, the acquisition offers an estimation of both the code delay and Doppler frequency sufficiently accurate to initiate the tracking loops. As explained in Section 2.3.4, the acquisition process is a two-dimensional operation, involving the correlation of the received signal $x_{IN}[n]$ with a local replica of the transmitted signal $c[n]$ across various trial values of Doppler frequency f_d and time delay τ. The circular correlation between $x_{IN}[n]$ and $y[n]$ yields the CAF. If we assume the absence of bit transitions in the received signal, the CAF can be expressed as

$$R_{xd}(\tau, f_d) = \frac{1}{N}\sum_{n=0}^{N-1} x_{IN}[n]c[nT_s - \tau]e^{-j2\pi f_d nT_s} \ , \tag{6.1}$$

where N represents the number of samples integrated coherently, and T_s denotes the sampling period.

PDI techniques can be employed to overcome the limitation of extending CI indefinitely. These techniques involve combining multiple consecutive CAFs through a non-linear function, as depicted in

$$Z_X = f(\sum_{k=1}^{N_{nc}} R_k(\tau, f_d)) \ . \tag{6.2}$$

In (6.2), Z_X is the result of the NCI obtained using PDI techniques, R_k represents the k-th consecutive CAF, N_{nc} denotes the number of non-coherent combinations, f_d is the Doppler frequency, and τ is the time delay [37].

When operating in high-sensitivity mode, the receiver described in this paper employs the

GPDIT strategy [37] to acquire highly attenuated GNSS signals in real-time. The GPDIT strategy can be expressed as

$$Z_{\text{GPDIT}} = \sum_{k=1}^{N_{nc}} |R_k(\tau, f_d)|^2 + 2|\sum_{k=2}^{N_{nc}} R_k(\tau, f_d) R_{k-1}^*(\tau, f_d)| \; . \tag{6.3}$$

The GPDIT strategy involves determining Z_{GPDIT} as the outcome of the NCI, and it is chosen for its effectiveness, as highlighted in [37]. This technique is deemed the most suitable in the presence of a frequency offset and demonstrates reasonable performance even in the face of phase noise resulting from the use of a TCXO. According to the findings in [37], GNSS signals with a C/N_0 as low as 20 dB-Hz can be acquired, while maintaining a probability of false alarm $P_{fa} < 0.01$ and a probability of correct detection $P_d \sim 0.8$. This is achieved through CI over a time span of 100 ms and employing GPDIT with up to $N_{nc} = 7$ non-coherent combinations. These parameters ensure a reliable acquisition of weak GNSS signals. While the GPDIT technique experiences significant degradation in the presence of data bits in the received signals [37], it proves effective when applied to pilot signals devoid of data and featuring predictable pilot codes. For this reason, the proposed high-sensitivity acquisition process acquires the Galileo pilot signals.

For a stationary GNSS receiver the maximum Doppler frequency shift in the E1/L1 frequency band is around ±5 kHz. The acquisition Doppler search step is chosen considering a trade-off between computational load at the receiver and accuracy in the estimation of the received Doppler frequency. As explained in Section 2.3.4, a typical value for the Doppler search step is $f_{step} = \frac{1}{2T_{CI}}$, where T_{CI} is the CI time [37]. When T_{CI} is set to 100 ms, this formula yields a Doppler search step of 5 Hz. As a result, the acquisition must conduct a frequency sweep across all potential carrier frequencies within a ±5 kHz range, in 5 Hz increments. This process involves a vast number of combinations, leading to a significant computational burden. In practice, the use of assistance data and Doppler prediction reduces the acquisition Doppler search space and consequently the acquisition latency.

6.4.4.2 Computation of Large FFTs in the FPGA

When working in high-sensitivity mode, the acquisition hardware accelerator computes the circular cross-correlation between the received signals and a local replica of the Galileo tiered codes over a time span of 100 ms. The circular cross-correlation is computed in the frequency domain, enabling efficient implementation within parallel processing architectures. This approach is particularly advantageous for applications requiring real-time signal processing. Many HS-GNSS receivers compute the CAF in the frequency domain using FFT-based techniques for their computational efficiency [37].

Performing circular correlations over a duration of 100 ms requires the use of very large FFTs, with sizes on the order of at least several hundred thousand samples. However, FPGA FFT IP cores are limited in length, requiring the FFT length to be a power of two. For this reason, the proposed FPGA implementation computes large FFTs by combining multiple smaller FFTs with complex exponentials. This approach is extended to IFFTs, following a similar procedure. To elaborate, this approach can be explained as follows: a large N-point DFT can be computed by utilizing two smaller DFTs, each of $\frac{N}{2}$ points [172]. This is achieved by separating the input sequence into even samples (x_{2n}) and odd samples (x_{2n+1}), as illustrated in

$$\begin{aligned} X_k &= \sum_{n=0}^{N-1} x_n e^{-\frac{2\pi kn}{N}} \\ &= \sum_{n=0}^{\frac{N}{2}-1} x_{2n} e^{-\frac{j2\pi k(2n)}{N}} + \sum_{n=0}^{\frac{N}{2}-1} x_{2n+1} e^{-\frac{j2\pi k(2n+1)}{N}} \\ &= \sum_{n=0}^{\frac{N}{2}-1} x_{2n} e^{-\frac{j2\pi kn}{N/2}} + e^{-\frac{j2\pi k}{N}} \sum_{n=0}^{\frac{N}{2}-1} x_{2n+1} e^{-\frac{j2\pi kn}{N/2}} . \end{aligned} \tag{6.4}$$

In this computation, the N-point DFT is conceptually divided into two halves. The first half of the DFT corresponds to samples $k = 0$ to $k = \frac{N}{2} - 1$ and is computed as the addition of two DFTs, each of $\frac{N}{2}$ points, combined with complex exponentials, as shown in (6.4). The second half of the DFT corresponds to samples $k = \frac{N}{2}$ to $k = N-1$ and is also computed as the addition of the same two $\frac{N}{2}$-point DFTs, but combined with different complex exponentials [172]. This is illustrated in

$$\begin{aligned} X_k = X_{N/2+t} &= \sum_{n=0}^{\frac{N}{2}-1} x_{2n} e^{-\frac{j2\pi(N/2+t)n}{N/2}} + e^{-\frac{j2\pi(N/2+t)}{N}} \sum_{n=0}^{\frac{N}{2}-1} x_{2n+1} e^{-\frac{j2\pi(N/2+t)n}{N/2}} \\ &= \sum_{n=0}^{\frac{N}{2}-1} x_{2n} e^{-\frac{j2\pi tn}{N/2}} - e^{-\frac{j2\pi t}{N}} \sum_{n=0}^{\frac{N}{2}-1} x_{2n+1} e^{-\frac{j2\pi tn}{N/2}} , \end{aligned} \tag{6.5}$$

where $k = \frac{N}{2} + t, 0 \le t < \frac{N}{2}$.

In a generalization of this approach, the computation of an N-point DFT involves combining P DFTs, each of $\frac{N}{P}$ points, as illustrated in

$$X_k = \sum_{p=0}^{P-1} \left(e^{-\frac{j2\pi kp}{N}} \sum_{n=0}^{\frac{N}{P}-1} x_{(Pn+p)} e^{-\frac{j2\pi kn}{(N/P)}} \right) . \tag{6.6}$$

This method conceptualizes the computation of the N-point DFT in P sections: from samples $k = 0$ to $k = \frac{N}{P} - 1$, then from samples $k = \frac{N}{P}$ to $k = 2\frac{N}{P} - 1$, and so forth, up to the section comprising samples $k = (P-1)\frac{N}{P}$ to $k = N - 1$. Each section is computed as the summation of P identical DFTs of $\frac{N}{P}$ points, combined with different complex exponentials. This is shown by the fact that if $k = s\frac{N}{P} + t$, where $0 \le t < N/P$ and $0 \le s < P$ (the k-th sample is in the s-th section), then (6.6) can be expressed as

$$
\begin{aligned}
X_k = X_{sN/P+t} &= \sum_{p=0}^{P-1}\left(e^{-\frac{j2\pi(\frac{N}{P}s+t)p}{N}} \sum_{n=0}^{\frac{N}{P}-1} x_{(Pn+p)} e^{-\frac{j2\pi(s\frac{N}{P}+t)n}{(N/P)}}\right) \\
&= \sum_{p=0}^{P-1}\left(e^{-\frac{j2\pi(\frac{N}{P}s+t)p}{N}} \sum_{n=0}^{\frac{N}{P}-1} x_{(Pn+p)} e^{-\frac{j2\pi tn}{N/P}}\right) .
\end{aligned}
\tag{6.7}
$$

The proposed FPGA acquisition hardware accelerator uses this approach to compute large FFTs. As explained above, the FPGA computes an N-point FFT by adding the results of P FFTs, each of $\frac{N}{P}$ points, combined with different complex exponentials. This is shown in Figure 6.4. This method is subject to the restriction that $\frac{N}{P}$ must be a power of two. Each FFT of $\frac{N}{P}$ points is applied to one subset of the input samples $x_{(Pn)}$, $x_{(Pn+1)}$, $x_{(Pn+2)}$, $\ldots$, $x_{(Pn+P-1)}$ according to (6.6). In Figure 6.4, $X^1_{k\prime}$ is the output of $FFT_{\frac{N}{P}}(x_{Pn})$, $X^2_{k\prime}$ is the output of $FFT_{\frac{N}{P}}(x_{Pn+1})$, etc., and $X^P_{k\prime}$ is the output of $FFT_{\frac{N}{P}}(x_{Pn+P-1})$, where $0 \le k\prime < \frac{N}{P}$.

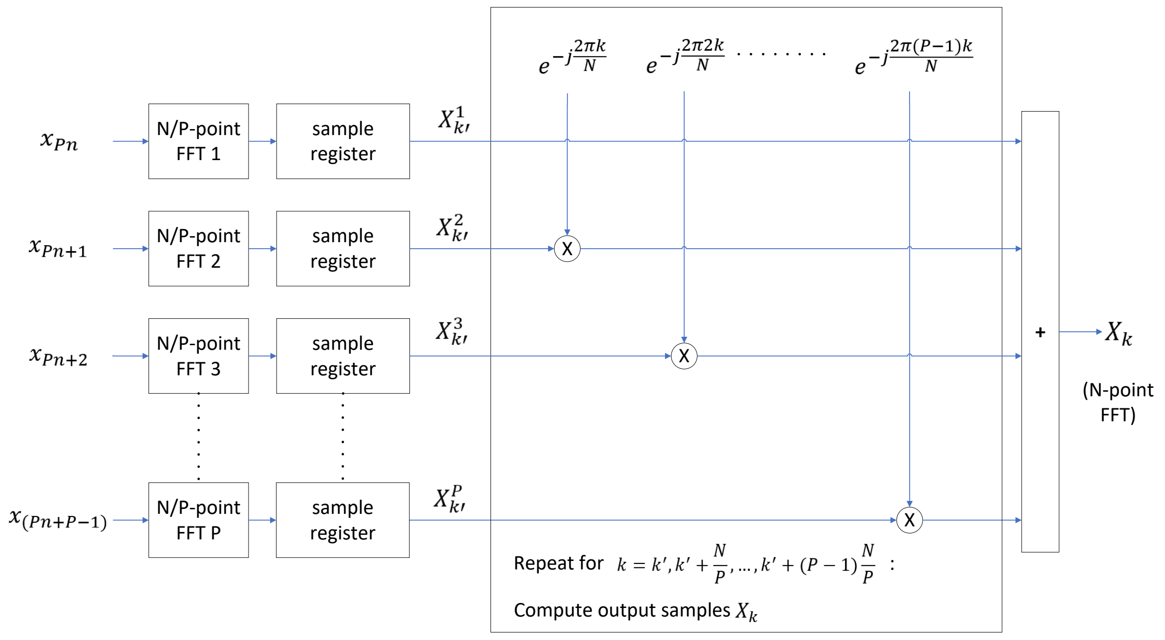

Figure 6.4 Computation of an N-point FFT as a combination of P FFTs, each of N/P points.

A state machine in the FPGA combines each set of samples produced by the $\frac{N}{P}$-point FFTs $\{X^1_{k\prime}, X^2_{k\prime}, \ldots, X^P_{k\prime}\}$, with complex exponentials, adding the results to produce P output samples of the N-point FFT. More specifically, the state machine utilizes input samples $\{X^1_{k\prime}, X^2_{k\prime}, \ldots, X^P_{k\prime}\}$ to compute the N-point FFT output samples $\{X_{k\prime}, X_{\frac{N}{P}+k\prime}, \ldots, X_{(P-1)\frac{N}{P}+k\prime}\}$). A sample register is used to hold the values $\{X^i_{k\prime}\}$, $0 \le i < P$, while the FPGA is computing the long FFT output samples $\{X_{k\prime}, X_{\frac{N}{P}+k\prime}, X_{2\frac{N}{P}+k\prime}, \ldots, X_{(P-1)\frac{N}{P}+k\prime}\}$. Because of the method employed, the N-point FFT output samples X_k are computed out-of-order.

The number of FFTs (P), and the length of each FFT (N/P) are determined as follows:

- The sampling frequency is chosen so that the number of samples N representing 100 ms is a multiple, P, of a power of two. N/P is constrained to be a power of two to ensure compatibility with FFT IP cores provided by FPGA manufacturers. We don't use zero padding because it would require a larger FFT size to accommodate the additional zero-valued samples, resulting in increased computational demands. This increase in FFT size, in turn, would increase acquisition latency and FPGA resource consumption. However, zero padding remains a potential option.
- To minimize the computation overhead in the proposed implementation, N/P is set to its maximum feasible value. This value is determined by either the maximum transform length supported by the FFT IP cores from FPGA manufacturers or a value that ensures the sampling frequency is as close as possible to the desired value.

The details regarding the selection of the sampling frequency, the number of FFTs (P), and the length of each FFT (N/P) for the receiver presented in this paper are elaborated further in Section 6.4.4.3.

6.4.4.3 Implementation

The acquisition process, when operating in high-sensitivity mode, incorporates both the PCPS algorithm [38] and the GPDIT PDI strategy [37]. The acquisition algorithm computes the CAF using a CI time of 100 ms. The CAF is computed as the circular correlation between the received signal and a local replica of the Galileo pilot codes, using various trial Doppler frequencies. The circular nature of the correlation is ensured by the fact that the Galileo tiered codes have a duration of 100 ms. The outcome of up to 7 consecutive CAFs are merged using the GPDIT technique shown in Equation (6.3), resulting in a maximum total integration time of 700 ms. The acquisition uses a Constant False Alarm Rate (CFAR) detector. Table 6.4 shows the input parameters of the acquisition algorithm.

Table 6.4 Input parameters of the high-sensitivity acquisition.

Input Parameter	**Definition**
f_{min}	Minimum tested Doppler frequency
f_{max}	Maximum tested Doppler frequency
f_{step}	Doppler search step
$C[k]$	FFT of the Galileo pilot signal to be detected, with a tiered code duration of 100 ms

Table 6.5 shows the output parameters of the acquisition algorithm for cases where a positive acquisition is obtained.

Table 6.5 Output parameters of the high-sensitivity acquisition.

Output Parameter	Definition
$f_{d_{acq}}$	Estimated Doppler frequency of the received signal
τ_{acq}	Estimated code phase

Table 6.6 shows the variables introduced in the description of the high-sensitivity algorithm.

Table 6.6 Variables introduced in the description of the high-sensitivity algorithm.

Variable	Definition
$x_{IN}[n]$	Received GNSS signal input sample stream
$\hat{P}_{IN}$	Input signal power estimation
n_{nc}	Current non-coherent combination number
N_{nc}	Maximum number of non-coherent combinations. This value is set to 7
N	Number of samples used for the coherent integration, representing 100 ms
f_d	Tested Doppler frequency
T_s	Sampling period
$R_k(\tau, f_d)$	k-th successive Cross-ambiguity function (CAF)
$Z_{\text{GPDIT}}(\tau, f_d)$	Result of the NCI using the GPDIT strategy as shown in Equation (6.3) [37]
Γ_{GLRT}	Generalized Likelihood Ratio Test (GLRT) function with normalized variance

The proposed acquisition process is detailed in Algorithm 3. In the initial step, the acquisition buffers 700 ms of samples, equivalent to $N \cdot N_{nc}$ samples, where N represents the number of samples used for CI (equivalent to 100 ms), and N_{nc} is 7, representing the maximum number of GPDIT non-coherent combinations used for detecting weak signals. For each GPDIT iteration, the algorithm takes 100 ms worth of samples (referred to as $x[n]$) from the acquisition buffer $x_{IN}[n]$ and performs a Doppler frequency search on $x[n]$, from the minimum Doppler frequency f_{min} to the maximum Doppler frequency f_{max} in f_{step} steps. For each tested Doppler frequency f_d, the acquisition conducts Doppler wipeoff and performs a circular correlation between the received signal $x[n]$ and Galileo pilot codes in the frequency domain $C[k]$, obtaining the CAF $R_k(\tau, f_d)$. The CAF is used to compute the corresponding GPDIT iteration $Z_{\text{GPDIT}}(\tau, f_d)$. The algorithm then searches the time-Doppler frequency grid for the peak value of the computed GPDIT iteration, obtaining the peak value along with an estimation of the Doppler frequency and code phase $\{S_{max}, f_i, \tau_j\}$. Finally, it evaluates a Generalized

Likelihood Ratio Test (GLRT) function Γ_{GLRT}, and determines whether the searched signal is detected. If positive, the algorithm declares successful acquisition; otherwise, it returns to the main loop to compute the next successive CAF and the next GPDIT iteration. If a total of 7 GPDIT combinations are computed and the signal is not detected, then the algorithm declares negative acquisition.

Algorithm 3 High-sensitivity acquisition

```
 1: Set the total number of GPDIT iterations $N_{nc} = 7$
 2: Buffer 700 ms worth of samples: $x_{IN}[0], x_{IN}[1], \dots, x_{IN}[N \cdot N_{nc}]$
 3: Set the current GPDIT iteration $n_{nc} = 1$
 4: While $n_{nc} \leq N_{nc}$ do
 5:     Take the next 100 ms worth of samples from the acquisition buffer:
        $x[n] = \{x_{IN}[(n_{nc}-1) \cdot N], x_{IN}[(n_{nc}-1) \cdot N + 1], \dots, x_{IN}[(n_{nc}-1) \cdot N + N - 1]\}$
 6:     For $f_d = f_{min}$ to $f_d = f_{max}$ in $f_{step}$
 7:         Perform Doppler wipe-off: $x_d[n] = x[n] \cdot e^{-j2\pi f_d n T_s}$, for $n = 0, \dots, N-1$
 8:         Compute $X_d[k] = FFT_N(x_d[n])$
 9:         Compute $Y[k] = X_d[k] \cdot C[k]$, for $k = 0, \dots, N-1$
10:         Compute $R_{n_{nc}}(\tau, f_d) = \frac{1}{N^2} IFFT_N(Y[k])$
11:         Compute $Z_{\text{GPDIT}}(\tau, f_d) = \sum_{k=1}^{n_{nc}} |R_k(\tau, f_d)|^2 + 2|\sum_{k=2}^{N_{nc}} R_k(\tau, f_d) R_{k-1}^*(\tau, f_d)|$
12:     End for
13:     Search the peak value and its indices in the search grid:
        $\{S_{max}, f_i, \tau_j\} = max_{f,\tau} Z_{\text{GPDIT}}(\tau, f_d)$
14:     Compute input signal power estimation $\hat{P}_{IN} = \frac{1}{N} \sum_{n=0}^{N-1} |x[n]|^2$
15:     Compute the GLRT function with normalized variance $\Gamma = \frac{2NS_{max}}{\hat{P}_{IN}}$
16:     If $\Gamma_{\text{GLRT}} > \gamma$ (Compare with threshold value)
17:         Declare positive acquisition and provide $f_{d_{acq}} = f_i$ and $\tau_{acq} = \tau_j$
18:         Break;
19:     Else
20:         If $n_{nc} = N_{nc} - 1$ (last GPDIT iteration)
21:             Declare negative acquisition
22:         End if
23:     End if
24: End while
```

The acquisition algorithm is executed collaboratively by the FPGA and the embedded processor, with each component handling a distinct portion of the process. The FPGA is responsible for capturing the samples from the analog front-end and computing the successive CAFs using the captured samples. Concurrently, the embedded processor executes the GPDIT iterations and the CFAR detector. Even though the FPGA manages the majority of the computational workload related to the acquisition process, the embedded processor is actively involved in the acquisition tasks.

The embedded processor can simultaneously handle the acquisition process and the other tasks required for the baseband software processing engine. These tasks include the control of the FPGA tracking multicorrelator hardware accelerators, the execution of the telemetry decoders, the computation of GNSS basic measurements, and the derivation of the navigation solutions. The Zynq UltraScale+ XCZU9EG All-Programmable MPSoC [63] used to demonstrate the proposed design, features a quad-core embedded processor. Therefore, from a computing power perspective, dedicating one processor core to the acquisition process leaves the other three cores available for additional tasks, thus facilitating the real-time processing of GNSS signals. Regardless, the embedded processor operates in an SMP configuration.

Figure 6.5 shows a detailed view of the FPGA acquisition hardware accelerator depicted in Figure 6.2. The PL DDR4 memory component shown in Figure 6.5 is the same as the one shown in Figure 6.2. The acquisition hardware accelerator comprises two modules: the Sample Capture module and the CAF computation module. When performing acquisition in high-sensitivity mode, the Sample Capture module collects 700 ms worth of samples from the analog front-end and stores the samples into the PL DDR4 memory component. Once the first 100 ms worth of samples has been captured, concurrently with the remaining part of the sample capture process, the CAF computation module initiates the calculation of the successive CAFs. By overlapping the CAF computation and the sample capture, the acquisition latency is reduced. The FPGA writes the computed CAFs to the PL memory. Concurrently, the embedded processor reads the computed CAFs from the PL memory component and executes the GPDIT iterations. The embedded processor executes the GPDIT iterations simultaneously to the FPGA calculating the CAFs, to further reduce the acquisition latency. In accordance with the architecture presented in Chapter 3, the high-sensitivity acquisition hardware accelerator exposes memory-mapped registers to the embedded processor for operation control. Additionally, it is equipped to issue interrupt requests to the embedded processor, facilitating the efficient synchronization of operations.

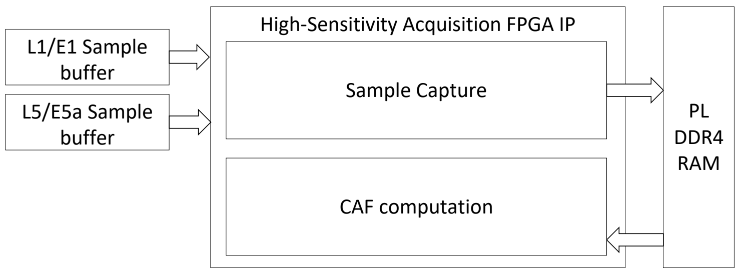

Figure 6.5 High-sensitivity acquisition FPGA hardware accelerator block diagram.

Figure 6.6 presents a timing diagram for the acquisition process, illustrating the concurrent computation of successive CAFs ($R_k(\tau, f_d)$) and GPDIT iterations ($Z_{\text{GPDIT}_k}(\tau, f_d)$) within the FPGA and the embedded processor, respectively. In the figure, time progresses from left to right. The FPGA and the embedded processor synchronize their operations by means of interrupt requests (marked as “Int.” in Figure 6.6), going from the FPGA to the PS. This process is explained below in more detail.

The embedded processor commences the acquisition process, as indicated by 'Initiate Acquisition Process' in Figure 6.6, which includes directing the FPGA to capture samples, a step referred to as 'Initiate Sample Capture' in the same figure. This involves the FPGA collecting 700 ms worth of samples in 100 ms increments. After capturing the first segment, the FPGA generates an interrupt request to the embedded processor. The processor, in response, instructs the FPGA to start the computation of the first CAF at the initial tested Doppler frequency ($R_k(\tau, f_{min})$). This instruction from the processor to the FPGA is denoted as 'Initiate CAF

Computation' in Figure 6.6. Following this, the FPGA and embedded processor work in sequence to compute the CAFs and GPDIT iterations for each Doppler frequency.

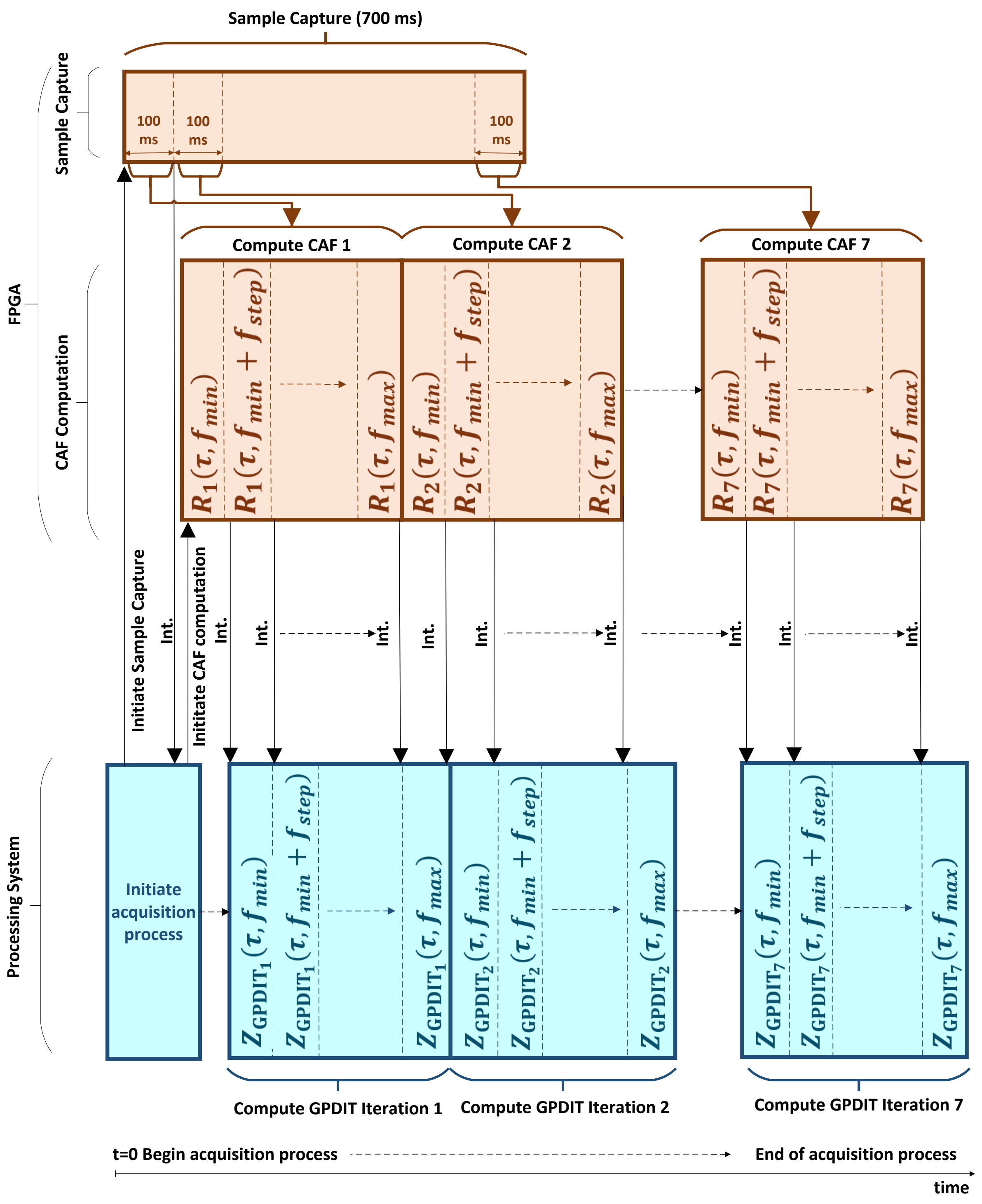
Sample Capture (700 ms)
Sample Capture
FPGA
CAF Computation
100 ms
100 ms
100 ms
Compute CAF 1
Compute CAF 2
Compute CAF 7
$R_1(\tau, f_{min})$
$R_1(\tau, f_{min} + f_{step})$
$R_1(\tau, f_{max})$
$R_2(\tau, f_{min})$
$R_2(\tau, f_{min} + f_{step})$
$R_2(\tau, f_{max})$
$R_7(\tau, f_{min})$
$R_7(\tau, f_{min} + f_{step})$
$R_7(\tau, f_{max})$
Initiate Sample Capture
Inititate CAF computation
Int.
Processing System
Initiate acquisition process
$Z_{GPDIT_1}(\tau, f_{min})$
$Z_{GPDIT_1}(\tau, f_{min} + f_{step})$
$Z_{GPDIT_1}(\tau, f_{max})$
$Z_{GPDIT_2}(\tau, f_{min})$
$Z_{GPDIT_2}(\tau, f_{min} + f_{step})$
$Z_{GPDIT_2}(\tau, f_{max})$
$Z_{GPDIT_7}(\tau, f_{min})$
$Z_{GPDIT_7}(\tau, f_{min} + f_{step})$
$Z_{GPDIT_7}(\tau, f_{max})$
Compute GPDIT Iteration 1
Compute GPDIT Iteration 2
Compute GPDIT Iteration 7
t=0 Begin acquisition process
End of acquisition process
time

Figure 6.6 Acquisition timing diagram.

For each frequency, the FPGA performs circular cross-correlation between the received signal and the local replica. After completing this computation, it issues an interrupt request to the embedded processor. Upon receiving the interrupt, the processor initiates the corresponding GPDIT iteration for that Doppler frequency. Simultaneously, it commands the FPGA to calculate the CAF for the next frequency. This command is not shown in 6.6 to maintain clarity. This procedure is systematically repeated across all Doppler frequencies, progressing from $f_d = f_{min}$ to $f_d = f_{max}$ in increments defined by f_{step}. Through this orchestrated approach, the FPGA consistently stays ahead by computing the CAF for the next frequency while the embedded processor concurrently handles the GPDIT iteration for the current frequency. Specifically, the FPGA computes the CAF for the $(n+1)$-th Doppler frequency $(R_k(\tau, f_{min} + n \cdot f_{step}))$, while the embedded processor executes the GPDIT iteration for the n-th frequency $(Z_{\text{GPDIT}k}(\tau, f_{min} + (n-1) \cdot f_{step}))$.

With this approach, upon completion of the computation of the n_{nc}-th CAF, the FPGA initiates the computation of the subsequent $(n_{nc}+1)$-th CAF in parallel with the embedded processor finishing the execution of the n_{nc}-th GPDIT iteration. Importantly, this parallel computation is speculative, meaning the results of the $(n_{nc}+1)$-th CAF might not be utilized if the CFAR detector signals a positive detection at the conclusion of the n_{nc}-th GPDIT iteration. Adopting this strategy minimizes acquisition latency through the simultaneous computation of CAF and GPDIT iterations.

Figure 6.7 shows a detailed block diagram of the CAF computation module shown in Figure 6.5, as it is implemented in the FPGA. The CAF computation module performs Doppler wipeoff and computes the circular correlation between the received signals and the local replica of the pilot tiered code in the frequency domain. It has four memory interfaces implemented using the AMBA AXI4-Lite protocol specification [61]: AXI4-Lite interface 1 retrieves the FFT of the local replica of the pilot-tiered code from the PL memory component. AXI4-Lite interface 2 fetches the captured samples from the analog front-end, which are also stored in the PL DDR4 memory component. AXI4-Lite interface 3 stores the computed CAF in the PL DDR4 memory. These three memory interfaces perform read and write transactions to and from the PL DDR4 simultaneously, thus decreasing the acquisition latency. AXI4-Lite interface 4 exposes a bank of memory-mapped registers to the embedded processor. The embedded processor uses these registers to configure and control the acquisition process. Furthermore, the CAF computation module also incorporates the following units, as depicted in Figure 6.7: a Doppler wipeoff unit, a unit for computing large FFTs and IFFTs (the large FFT/IFFT block), a unit for multiplying the FFT output with the local replica of the pilot tiered code (the Enable/Disable Local Code Mult block), and a unit for reporting a scaling factor to the embedded processor. This scaling factor is automatically applied to the FFT/IFFT output to prevent saturation, and it is referred to as the block exponent in Figure 6.7. Finally, a state machine controls the execution of the CAF computation, which occurs in two steps:

1. In the first step, the CAF computation module computes the frequency domain representation of the circular correlation: the module fetches 100 ms of samples from the PL memory component, performs Doppler wipeoff, computes the FFT of the Doppler-corrected signal, and multiplies the results by the FFT of the local replica of the pilot tiered code. Finally, the result of this multiplication is stored back into the PL memory component. Storing intermediate results in the PL memory component reduces FPGA memory usage. During this first step, the Enable/Disable Doppler wipeoff block and the Enable/Disable FFT Multiplication block shown in Figure 6.7 are enabled. The local carrier used for Doppler wipeoff is efficiently implemented using the Coordinate Rotation Digital Computer (CORDIC) algorithm [178].

2. In the second step, the CAF computation module calculates the time-domain representa-

tion of the circular cross-correlation. It retrieves the results of the multiplication of the FFT by the local replica obtained in step 1, performs the IFFT, and then stores the results back in the PL memory component.

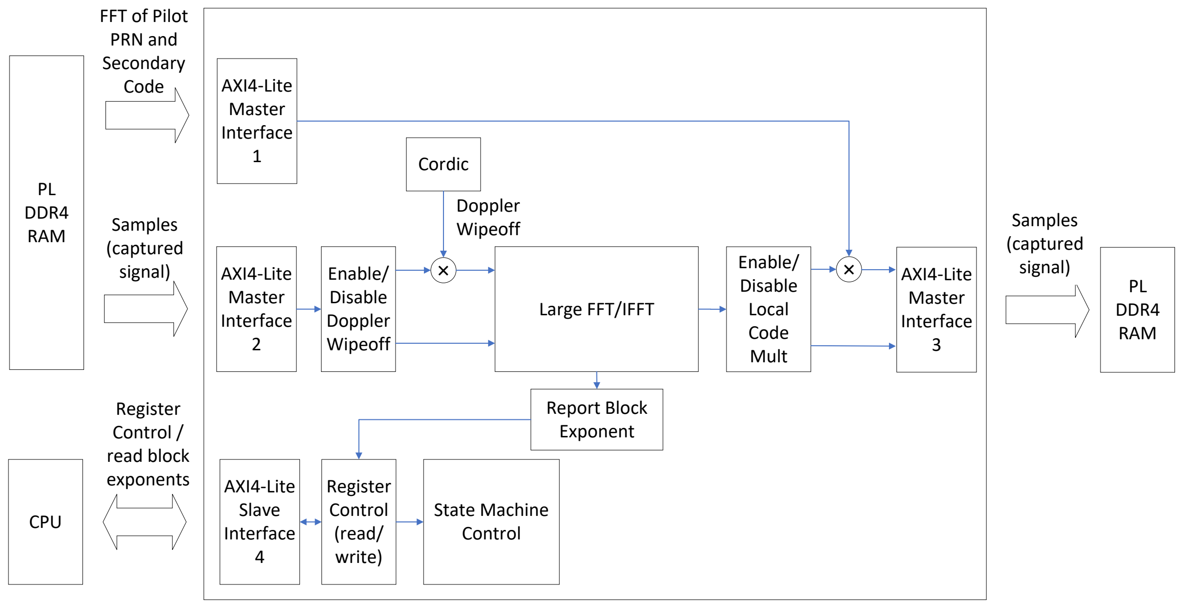
FFT of Pilot PRN and Secondary Code
PL DDR4 RAM
Samples (captured signal)
Register Control / read block exponents
CPU
AXI4-Lite Master Interface 1
Cordic
Doppler Wipeoff
AXI4-Lite Master Interface 2
Enable/ Disable Doppler Wipeoff
Large FFT/IFFT
Enable/ Disable Local Code Mult
AXI4-Lite Master Interface 3
Report Block Exponent
AXI4-Lite Slave Interface 4
Register Control (read/ write)
State Machine Control
Samples (captured signal)
PL DDR4 RAM

Figure 6.7 CAF computation module.

The large FFT/IFFT module depicted in Figure 6.7 computes the FFT and the IFFT, applying a scaling factor to the input signals to prevent overflow in the calculations. The scaling factor applied is made available to the embedded processor through the Report Block Exponent block. The embedded processor normalizes the application of the scaling factor to ensure uniform scaling across all Doppler frequencies and GPDIT iterations.

Figure 6.8 shows a block diagram of the large FFT/IFFT module depicted in Figure 6.7. This module computes large FFTs and IFFTs by combining multiple small FFTs and IFFTs with complex exponentials, extending the transform size. This approach is explained in Section 6.4.4.2 and illustrated in Equation (6.6), where an N-point DFT is computed as a weighted combination of P DFTs, each of length N/P. The same principle applies for the computation of the IFFT.

The 64k-FFT/IFFT module, shown in Figure 6.8, is implemented using AMD's FFT LogiCORE IP v9.1 [179]. This FFT core imposes a maximum transform size limit, restricting it to $2^{16} = 64k$ samples. To accommodate this limitation, the receiver's sampling frequency is deliberately chosen to ensure that the computation of a 100 ms circular cross-correlation in the frequency domain results in an FFT size that is a multiple of 64k samples. Consistent with this, the sampling frequency is set to 15.728640 Mega samples per second (Msps), which provides sufficient capability to track GNSS signals in both the L1/E1 and L5/E5a bands [139]. As shown in Figure 6.2, the acquisition hardware accelerator incorporates a downsampling filter in the L1/E1 band, reducing the sampling frequency by a factor of 4 to 3.932160 Msps. Consequently, to compute the CAF using a CI time of 100 ms, FFTs/IFFTs with a transform size of 393,216 samples are required. This transform size is equivalent to 6× 64k samples.

To efficiently handle this calculation, the FPGA computes FFTs of 393, 216 samples by combining six FFTs, each of 64k samples, following the approach depicted in Figure 6.4.

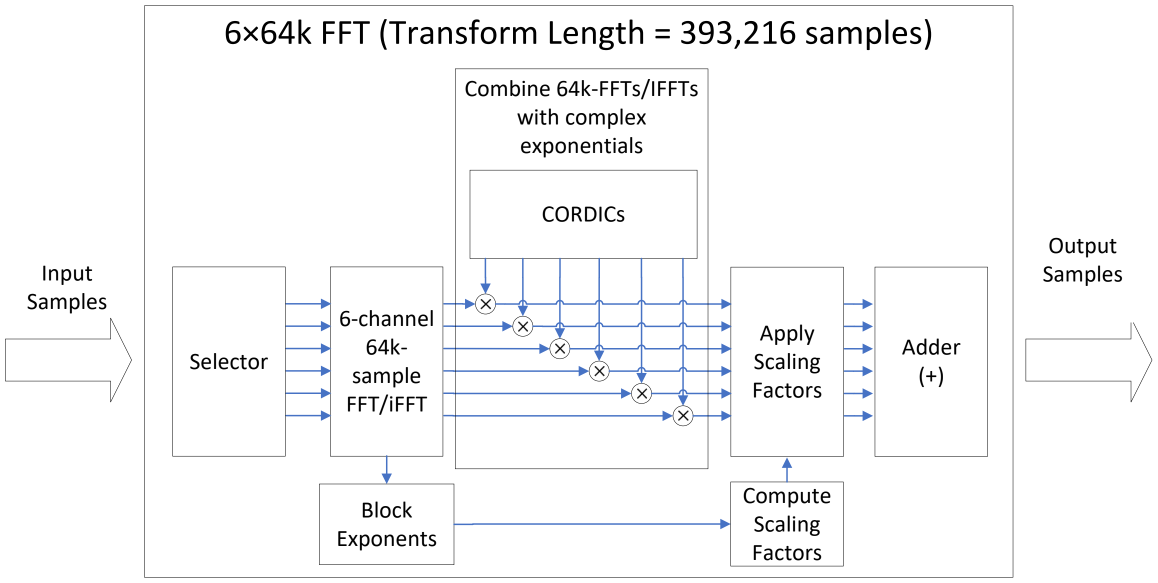
6×64k FFT (Transform Length = 393,216 samples)
Combine 64k-FFTs/IFFTs with complex exponentials
CORDICs
Input Samples
Selector
6-channel 64k-sample FFT/iFFT
Apply Scaling Factors
Adder (+)
Output Samples
Block Exponents
Compute Scaling Factors

Figure 6.8 Large FFT/IFFT module implemented in the high-sensitivity acquisition.

The FFT LogiCORE is configured to implement a 6-channel 64k-FFT on the incoming samples. An input sample selector in the 64k-FFT/IFFT module cyclically distributes the incoming samples to the FFT channels sequentially, assigning the 1st sample to the 1st channel, the 2nd sample to the 2nd channel, and so forth. In the complex exponential combining block, a state machine combines the outputs of the 64k-FFT with the complex exponentials $e^{-j2\pi \frac{k}{N}s}$, as illustrated in Equation (6.6). The complex exponentials are implemented using the CORDIC algorithm. TheFFT LogiCORE automatically applies a scaling factor (block exponent) to each channel to prevent overflow. The Apply Scaling Factors module in Figure 6.8 dynamically equalizes the various FFT channels by ensuring that the same scaling factor is uniformly applied to all of them. The scaling factor used is reported to the embedded processor. Finally, the 64k-FFTs, combined with the complex exponentials, are added to obtain the 393, 216-point FFT results. The 393, 216-point FFT and IFFT output samples are produced out of order. However, these samples are automatically reordered during the storage process to memory, minimizing latency.

6.4.5 Doppler Prediction

The high-sensitivity acquisition process parallelizes the code phase search by conducting a circular cross-correlation between the received GNSS signal and a local replica of the signal's tiered PRN code. However, this correlation has to be performed once for each possible trial Doppler frequency [38]. In cases where a static GNSS receiver lacks prior knowledge of the received Doppler frequencies, it is obligated to conduct a search spanning from −5 kHz to +5 kHz. As shown in Section 6.4.4.1, when operating in high-sensitivity mode, the use of a CI time of 100 ms yields a Doppler search step of 5 Hz. Consequently, the acquisition process involves testing 2001 Doppler frequencies, conducting a frequency sweep across all possible frequencies within ±5 kHz of the nominal carrier frequency in increments of 5 Hz. This leads to a significant acquisition latency.

When the receiver is processing signals in real time, the acquisition latency cannot be arbitrarily large. The key factors limiting the maximum acceptable latency are the maximum allowable TTFF and the degradation over time of the parameters estimated during acquisition, such as timing synchronization and Doppler frequency. This degradation is caused by the receiver clock drift and the continuously changing Doppler frequency in the received signals. A slow acquisition process can result in the receiver beginning to track the detected signal only after changes have occurred in the Doppler frequency, affecting both the carrier signal and the code phase. This, coupled with local crystal oscillator inaccuracies, diminishes the accuracy of the Doppler frequency and code phase estimated during the acquisition process.

The receiver employs the assistance data to predict the Doppler frequency of the incoming signals, leading to a reduction in the Doppler search space during acquisition. The Doppler search is performed considering potential inaccuracies in the Doppler prediction, and the presence of a Carrier Frequency Offset (CFO) in the RFFE, originating from deviations in the accuracy of the local crystal oscillator.

Even with Doppler prediction, the search range for the Doppler frequency can remain large, due to the need to consider the inaccuracies of Doppler prediction and the effect of CFO in the RFFE. This situation can result in significant acquisition latency, thereby diminishing the chances of successfully tracking detected signals. To address this problem, the receiver offers an option to execute the acquisition process in two stages. The first stage performs a potentially large Doppler search around the predicted Doppler frequency to account for inaccuracies in Doppler predictions and CFO mentioned above. Then, in the second stage, the acquisition process is repeated with new samples from the analog front-end, using a much smaller Doppler search space centered on the Doppler frequency identified in the first stage. The Doppler search space in stage 2 can be significantly smaller because CFO and potential inaccuracies in the Doppler prediction have already been accounted for in stage 1. The execution of stage 2 is much faster than the execution of stage 1 because of the smaller Doppler search space. While this two-stage execution increases overall acquisition latency and may influence the probability of detection, stage 2 reduces the latency from sample capture to the initiation of the tracking process, thereby enhancing the accuracy of the parameters estimated during acquisition. The two-step acquisition process is shown in Table 6.7.

The method outlined in Table 6.7 is effective provided that, during the execution of stage 1, the change in the Doppler frequency of the received signal remains within the Doppler search range specified for stage 2. Otherwise, the signal detected in stage 1 may not be detected in stage 2. The probability of detection may also be affected by the need to detect the same signal twice.

Table 6.7 Acquisition process in two steps.

Stage	Algorithm
Stage 1	Capture samples from the analog front-end. Perform acquisition using a reduced Doppler search space around the predicted Doppler frequency (the Doppler search space is large enough to account for any CFO and Doppler prediction inaccuracy).
Stage 2	Capture samples from the analog front-end. Perform acquisition using a much smaller Doppler search space around the Doppler frequency estimated during stage 1. As a result, when executing stage 2, the system quickly transitions from capturing samples to starting the tracking process.

6.4.6 Acquisition in Normal-Sensitivity Mode

When working in normal-sensitivity mode, the acquisition process detects GPS L1 C/A, GPS L5, Galileo E1b/c, and Galileo E5a signals received at nominal C/N_0 levels. Additionally, the acquisition process can be configured to detect either the data component or the pilot component of the received signals. Much like the procedure in high-sensitivity mode, the acquisition process involves conducting a cross-correlation between the received signal and a local replica of the satellite PRN code. However, this correlation is performed using a short CI time, equal to the length of the PRN code (1 ms for GPS L1 C/A, GPS L5, and Galileo E5a; 4 ms for Galileo E1b+c) or a small multiple of this length. The results from several consecutive CAFs $R_k(\tau, f_D)$ can be effectively combined using the NPDI technique, illustrated in

$$Z_{\text{NPDI}} = \sum_{k=1}^{N_{nc}} |R_k(\tau, f_D)|^2 \,, \tag{6.8}$$

where Z_{NPDI} is the result of the NCI. Throughout this process, a CFAR detector is employed.

When operating in normal-sensitivity mode, the high-sensitivity acquisition hardware accelerator is responsible for capturing samples from the analog front-end. Subsequently, the embedded processor takes charge of all calculations, including Doppler wipe-off, the circular cross-correlation, and the computation and the non-linear combinations of consecutive CAFs. The rationale behind this arrangement lies in the current design of the high-sensitivity acquisition hardware accelerator in the FPGA. Specifically tailored for acquiring weak signals using long CI times, the current version lacks the flexibility required to compute the CAF in normal-sensitivity mode. Nonetheless, the Ultrascale+ XCZU9EG MPSoC's embedded processor has ample computing power, enabling it to execute the acquisition in normal-sensitivity mode in real-time. Additionally, it can manage other assigned tasks, including operating the FPGA tracking multicorrelators, running the telemetry decoders, computing the observables, and determining the PVT.

6.4.7 Tracking

As explained in Section 2.3.5, the role of a tracking algorithm is to follow the evolution of the signal synchronization parameters: code phase $\tau(t)$, Doppler frequency $f_d(t)$ and carrier phase $\phi(t)$. The proposed receiver implements tracking loops to continuously monitor and adjust to the code and carrier parameters of the incoming signal. Specifically, a Delay Lock Loop (DLL) is employed to track the signal's code delay, a Phase Lock Loop (PLL) is dedicated to monitoring and adjusting to the signal's phase, and a Frequency Locked Loop (FLL) can be enabled to monitor the signal's frequency.

The receiver implements the tracking multicorrelator hardware accelerators presented in Section 3.1.5. These multicorrelators perform the Doppler wipe-off and the multicorrelation of the incoming signal with the local replica of the PRN codes. The embedded processor executes the PLL, the FLL, and the DLL. The PLL and the FLL employ a four-quadrant arctangent discriminator. The DLL employs a noncoherent Very Early Minus Late Power (VEMLP) normalized discriminator when tracking Galileo E1b/c signals, and a noncoherent Early Minus Late envelope-normalized discriminator when tracking Galileo E5a, GPS L1 C/A, and GPS L5 signals.

The tracking multicorrelator hardware accelerators are configured as shown in Table 3.3 in Chapter 3. When tracking GPS L1 C/A signals, the receiver follows the data component of the received signals. For other signals, the receiver tracks the pilot component, utilizing an additional correlator dedicated to demodulating the data component.

6.4.7.1 Tracking in High-Sensitivity Mode

When tracking Galileo E1b/c signals in high-sensitivity mode, the receiver initiates signal tracking with synchronization to the pilot's secondary code. This synchronization is achieved through the acquisition process, which computes the CAF using a circular cross-correlation with the complete Galileo pilot tiered code, including both the primary and secondary codes.

Consequently, the tracking process can commence with an extended CI time that surpasses the duration of the PRN code. The receiver is configured to track the pilot signals, which have predictable tiered codes. This configuration enables the utilization of CI times that exceed the duration of a single data bit in the tracking process. These capabilities are essential for tracking weak signals, addressing the challenges posed by potential bit errors in the demodulated data due to low $C/N0$ conditions [176, 177].

To effectively track weak signals, it is necessary to employ an extended CI time. However, due to the fundamental differences between tracking and acquisition in signal processing, the tracking process can utilize a shorter CI time compared to acquisition. Tracking is an estimation problem that benefits from prior knowledge about the code phase and the carrier phase. This information not only enhances the sensitivity of the tracking loops but also allows for refined estimations of these parameters. In contrast, acquisition is primarily a detection problem, focused on initially identifying the signal's presence without the benefit of prior accurate information. In line with this, the receiver was tested using various extended CI times between 20 ms and 100 ms. When using an OCXO, we could track signals with a C/N_0 down or very close to 20 dB-Hz with a tracking CI time of 20 ms. However, tracking sensitivity appeared to be at its limit. Minor changes in the C/N_0 of the received signals caused the tracking loops to lose lock. To enhance tracking sensitivity and accommodate small fluctuations in received signal power during tracking, the receiver was set to use an extended CI time of 40 ms for weak signals. Therefore, the tests reported in Section 6.5 utilized a tracking CI time of 40 ms for high-sensitivity mode, shorter than the CI time used in acquisition (100 ms). For more details on the configuration of the tracking loops, refer to Appendix 6.A.

The current implementation of the tracking algorithm does not include any type of multipath mitigation. The high sensitivity tracking mode is mainly designed to track weak GNSS signals.

6.4.7.2 Tracking in Normal-Sensitivity Mode

In normal-sensitivity mode, the receiver initiates tracking of detected signals without synchronization to the pilot's secondary code or the telemetry preambles. The tracking process commences with a CI time equivalent to the duration of the PRN primary code. Once synchronization with the pilot's secondary code or the telemetry preambles is achieved, the tracking process dynamically increases the CI time based on user configuration.

6.4.8 Telemetry Decoding

When tracking very weak Galileo E1b/c signals, the receiver may not be able to correctly demodulate the telemetry messages due to the presence of bit errors [176, 177]. However, demodulating the received telemetry messages is needed to obtain the TOW of the received messages. The TOW is used to compute the GNSS basic measurements and the navigation solutions. For this reason, the GNSS-SDR software telemetry decoder was upgraded to estimate the TOW of the received Galileo E1b/c messages using the ToA of the telemetry synchronization patterns and the receiver's local clock. The receiver performs TOW estimation in three steps: sync frame detection, sync frame confirmation, and TOW estimation. These steps are explained below.

6.4.8.1 Sync Frame Detection

The Galileo telemetry decoder initiates its process by identifying the I/NAV synchronization patterns within the received telemetry frames. To address the potential presence of bit errors, the receiver maintains a list of sample stamps corresponding to the most recently detected synchronization patterns. These sample stamps represent the receiver sample counter associated with those patterns. For each identified synchronization pattern, the receiver verifies if the timing relative to any of the most recently detected patterns is a multiple of the Galileo E1 I/NAV synchronization pattern period. If this condition is met, then the telemetry decoder assumes sync frame detection and proceeds to the second step, which involves sync frame confirmation.

6.4.8.2 Sync Frame Confirmation

The telemetry decoder checks that at least a certain percentage of preambles is accurately detected at the expected timings. If the detection of the telemetry sync patterns is confirmed, then the receiver assumes correct sync frame detection and proceeds to the third step, which involves TOW estimation. Otherwise, the receiver assumes a synchronization issue and goes back to the sync frame detection.

6.4.8.3 TOW Estimation

The receiver estimates the TOW using the ToA of the telemetry synchronization patterns and the local embedded OS clock. The TOW of the Galileo synchronization patterns corresponds to the exact timing of the telemetry page frame boundaries [29]. The receiver takes advantage of the fact that, in the Galileo E1b/c signals, the synchronization patterns are transmitted in sync with the second. The estimation of the TOW involves measuring the exact ToA of the telemetry sync patterns and rounding this time to the nearest second. The ToA is computed in Galileo System Time (GST) as follows: In the first step, the local time is converted to UTC considering the local time zone. In the second step, the computed UTC time is converted to GST considering the leap seconds between UTC and GST. Assuming that the GNSS signal travel time, plus the error induced by delays in the receiver detecting the preambles, added to the receiver's local clock offset is in total below ±500 ms, the computed TOW estimation is deemed correct. This is because rounding the ToA to the nearest second provides the TOW. Achieving the required accuracy in the local OS clock is performed by synchronizing the receiver's OS clock to the true local time using the Network Time Protocol (NTP) [180]. The NTP protocol offers a nominal accuracy of tens of milliseconds on Wide Area Networks (WANs) [180], which is sufficient for the proposed TOW estimation process.

The need for precise clock accuracy becomes less critical in this receiver when exclusively tracking Galileo E1b/c signals. In such cases, even with a slight clock offset, the PVT solution may still converge, indicating a clock bias. However, when tracking Galileo E1b/c alongside other GNSS signal types that do not perform TOW estimation in the same manner, precise clock synchronization is essential. Without it, relative timing delays could arise between observables of different GNSS signal types, significantly compromising the PVT solution's accuracy.

When in the TOW estimation state, the telemetry decoder continuously verifies that a minimum percentage of preambles is accurately detected at the expected timings. If the receiver fails to detect the telemetry preambles, it assumes a synchronization issue and reverts to the frame sync detection state. This allows the receiver to resynchronize with the received preambles in case of an incorrect sync frame detection.

When working in post-processing mode using recorded GNSS signals, the receiver local OS clock time does not correspond to the recorded signal time. Therefore, the receiver embedded OS clock cannot be used to estimate the ToA of the telemetry sync patterns. In this case, the telemetry decoder estimates the TOW of the telemetry messages using the receiver sample counter and a predetermined relationship between the sample counter and the recorded signal time that shall be provided by the user. The sample counter counts up the number of processed samples and is automatically reset when the user starts the GNSS-SDR software receiver.

6.4.9 Computation of the Navigation Solutions

The embedded processor computes navigation solutions using the GNSS basic measurements: pseudorange, carrier phase, and Doppler shift. The computation of the navigation solutions is implemented using the RTKLIB open-source package for standard and precise positioning with GNSS [181].

The estimated receiver positions might include invalid solutions due to unmodeled measurement errors. The processor performs a series of validation tests on the estimated receiver positions, including a residuals test and a Geometric Dilution of Precision (GDOP) Test [181]. If any of these tests fails, the solution is rejected as an outlier. In addition, the software implements Receiver Autonomous Integrity Monitoring Fault Detection and Exclusion (RAIM-FDE) [182]. RAIM-FDE attempts to exclude invalid measurements due to satellite malfunction, receiver fault, or large multipath by iteratively estimating the receiver's position while excluding one visible satellite at a time.

When operating in high-sensitivity mode, the receiver utilizes assistance ephemeris data to determine PVT. This is necessary because the receiver may encounter challenges in correctly demodulating the telemetry messages of weak Galileo E1b/c signals, primarily due to the presence of bit errors [176, 177]. If the telemetry decoder successfully demodulates the telemetry messages during operation, the receiver then updates the assistance ephemeris data with the latest information received from the satellites. Assistance ephemeris data can be optionally employed for the GPS and the Galileo E5a signals as well.

6.4.10 GNSS Output Products

The proposed receiver outputs GNSS signal products in standard open formats as detailed in Section 2.4.3, such as RINEX, RTCM (for sensor integration), and GPX, while also generating PVT solutions in WGS-84, tailored for various applications. It offers precise satellite measurements—pseudorange, Doppler shift, signal strength—and navigation and timing data (GPS Time, Galileo Time, UTC time), enabling thorough satellite tracking and analysis.

6.5 Results

The performance of the proposed receiver was assessed based on several KPIs. These included acquisition latency, the receiver's ability to process Galileo E1b/c signals in high-sensitivity mode, and its capability to simultaneously process Galileo E1b/c signals in high-sensitivity mode alongside Galileo E5a and GPS signals in normal-sensitivity mode. The precision of navigation solutions and the power consumption were also evaluated. Additionally, FPGA resource usage is also reported.

The receiver's ability to process Galileo E1b/c signals in high-sensitivity mode was tested using three different TCXOs and an OCXO. Two TCXOs caused a CFO in the received signals larger than expected, requiring the user to perform a manual estimation and correction of the CFO before using the receiver in high-sensitivity mode. This procedure was not required when using an OCXO, which offers better frequency stability. For this reason, we switched to using an OCXO for the remaining tests. However, improving the acquisition algorithm to eliminate the need for manual CFO compensation remains as future work. A detailed explanation is provided in Section 6.5.2.

Appendix 6.A provides the receiver configuration used for processing the Galileo E1b/c signals in high-sensitivity mode, while Appendix 6.B offers the configuration for processing the Galileo E5a and GPS signals in normal-sensitivity mode. Appendix 6.C details the receiver's PVT configuration.

6.5.1 Test Setup

Figure 6.9 illustrates the block diagram of the test setup used to apply live signals to the proposed receiver. The antenna captures GNSS signals from the sky. A variable attenuator is manually adjusted to decrease the C/N_0 of the received signals down to 20 dB-Hz. Achieving a uniform C/N_0 of 20 dB-Hz for all signals proved to be challenging due to variations in the received signal power from different satellites. Splitter 1 divides the received signals into two paths: one leading to the device under test and the other to a commercial receiver, which is used to validate the C/N_0 of the received signals. The proposed HS-GNSS receiver uses splitter 2 to divide the incoming signal internally, distributing it between the two RF inputs of the AD-FMCOMMS5-EBZ analog front-end. RF1 is tuned to the E1/L1 frequency band, while RF2 is tuned to the E5a/L5 band.

The test setup involves two splitters in the signal path of the high-sensitivity GNSS receiver (splitter 1 and splitter 2). Similarly, the signal path of the commercial receiver incorporates splitter 1 and splitter 3. Each splitter introduces a 3-dB attenuation in the received signal. This arrangement allows us to maintain equal signal conditions, as detailed in the following explanation.

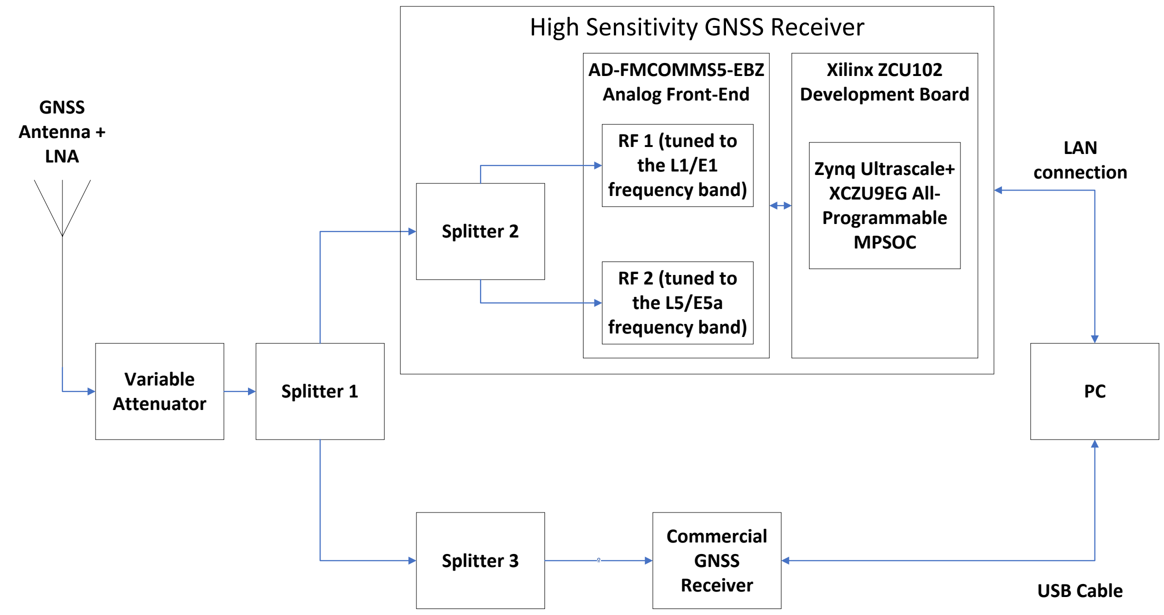
High Sensitivity GNSS Receiver
AD-FMCOMMS5-EBZ Analog Front-End
Xilinx ZCU102 Development Board
GNSS Antenna + LNA
RF 1 (tuned to the L1/E1 frequency band)
Zynq Ultrascale+ XCZU9EG All-Programmable MPSOC
LAN connection
Splitter 2
RF 2 (tuned to the L5/E5a frequency band)
Variable Attenuator
Splitter 1
PC
Splitter 3
Commercial GNSS Receiver
USB Cable

Figure 6.9 Test setup for the high-sensitivity GNSS receiver.

The C/N_0 observed by both receivers can be computed using the Friis formulas for noise [35]. When the variable attenuator is not used, the C/N_0 observed by the receivers is primarily influenced by factors such as received signal strength, antenna noise temperature, and the noise figure of the LNA in the active antenna. The splitters in the signal path, including splitters 1, 2, and 3, each introduce an insertion loss of 3 dB, because they divide the received signal power between their two output ports. However, these losses have negligible effects on the C/N_0 due to the compensating presence of the LNA, as explained in Section 2.3.2. However, despite the presence of the LNA, when the variable attenuator introduces substantial attenuation, the C/N_0 observed by the receivers becomes primarily determined by the total attenuation from the antenna to the analog front-end. This total attenuation encompasses both the variable attenuator and the attenuation introduced by the splitters. Splitter 3 introduces an attenuation of 3 dB, ensuring that the commercial receiver perceives the same C/N_0 as the receiver under test when the variable attenuator introduces significant attenuation. One of the output ports of splitter 3 is connected to the commercial GNSS receiver. The other output port is left open.

The C/N_0 of received Galileo E1b/c signals was validated using a U-blox NEO-M8T receiver [161]. The variable attenuator was adjusted to decrease the carrier-to-noise density ratio (C/N_0) of the received signals. This adjustment aimed to reduce the C/N_0 of the weakest Galileo E1b/c signals to as low as 20 dB-Hz, while maintaining consistent attenuation levels across other signals.

The rooftop GNSS antenna facility of the GESTALT testbed [146] was utilized to enhance satellite visibility during the tests. Additionally, the tests evaluating the receiver's ability to process Galileo E1b/c, Galileo E5a, GPS L1 C/A, and GPS L5 signals (Section 6.5.3) were conducted using both the rooftop GNSS antenna facility and the receiver's TW8825 active antenna.

6.5.2 High-Sensitivity Acquisition Latency

The latency of the two-stage acquisition scheme was evaluated using both a TCXO and an OCXO. The evaluations were carried out following the test setup described in Section 6.5.1 and the procedure outlined in Table 6.8. The variable attenuator was used to decrease the C/N_0 of the received signals to as low as 20 dB-Hz. The software was temporarily modified to measure the worst-case acquisition latency as follows: the acquisition process consistently performed 7 GPDIT iterations, with the receiver recording the acquisition latency during each execution. The latency was measured as the time it takes for the receiver to execute each acquisition stage.

Table 6.8 Acquisition latency: test procedure.

Step	Description
Step 1	Configure the Doppler search space of the two-stage acquisition process.
Step 2	Apply weak Galileo E1b/c signals to the receiver, aiming for a C/N_0 ratio that is approximately 20 dB-Hz. Verify the C/N_0 using the commercial receiver.
Step 3	Confirm that the receiver acquires and initiates tracking of the weak signals, successfully detecting the telemetry preambles. Measure the acquisition latency.

6.5.2.1 Tests Results Using the TCXO

When using the TCXO, a significant CFO was observed in the received signals. To determine the required Doppler search space for acquisition, we used the receiver to perform a coarse estimation of the CFO. To perform this estimation, the receiver operated in normal-sensitivity mode to predict the Doppler frequency of the tracked satellites. It then compared this prediction to the measured Doppler frequency, and averaged the error. Three different instances of TCXOs were tested. Each TCXO was tested multiple times over the course of a week. Table 6.9 shows the minimum and maximum estimated CFO values in the E1 frequency band for each TCXO over that week.

Table 6.9 CFO in the E1 frequency band when using a TXCO.

TCXO	Measured CFO in the E1 Frequency Band
TCXO 1	−61 Hz to 89 Hz
TCXO 2	109 Hz to 156 Hz
TCXO 2	−273 Hz to −185 Hz

The measurements shown in Table 6.9 indicate a frequency stability of approx. 56 ppb, 99 ppb, and 170 ppb for TCXO 1, TCXO 2, and TCXO 3, respectively. A certain variation in the detected CFO is expected, given that the stability of the TCXO can be influenced by environmental factors including temperature, voltage and load variations, as well as long term frequency drift [183]. However, the CFOs induced by TCXOs 2 and 3 were higher than expected. The TCXOs were mounted on a custom-made PCB, which was attached to the RFFE and exposed to the environment. The exploration of the underlying reasons behind the observed frequency stability fluctuations in the TCXOs lies beyond the scope of this thesis.

The large CFO observed in TCXO 2 and TCXO 3 required the use of a large Doppler search space in the acquisition. The receiver could in principle be capable of handling this Doppler search space at the expense of increased computational load, memory usage and TTFF. In the current design, that might require optimizing memory usage. Testing the receiver with a very large number of trial Doppler frequencies could lead to insufficient OS system memory. This issue stemmed from the need to simultaneously store the temporary results of non-coherent integrations for all these frequencies. For the sake of simplicity and to prevent a large TTFF, a crude two-step CFO estimation and correction process was implemented in the software to mitigate the CFO. The first step required taking the receiver outdoors and using the receiver in normal-sensitivity mode. The receiver estimated the CFO during tracking using the method described above. In the second step, the receiver was tested in high-sensitivity mode, and the estimated CFO was corrected by adjusting the tuning frequency in the E1/L1 and the E5a/L5 frequency bands accordingly.

When using the TCXO, the two-stage acquisition process was set as follows: the Doppler search space for the first acquisition stage was configured to accommodate Doppler prediction inaccuracies and the CFO observed in the L1/E1 frequency band, but it was limited to ±100 Hz to limit memory usage and to prevent a large TTFF. The Doppler search space for the second stage was set to be small enough to minimize the latency from sample capture to the beginning of the tracking process, ensuring successful tracking of the detected signals. Additionally, the following constraint was considered: in the worst-case scenario, the change in the Doppler frequency of the received signal during the time it takes to go from stage 1 to stage 2 should not exceed the Doppler search space set for the stage 2. As explained in Section 6.4.5, failure to adhere to this constraint could result in the inability of the second stage to identify the signal detected during the first stage. In line with this, the acquisition was configured as shown in Table 6.10.

Table 6.10 Acquisition latency using a TCXO.

Stage	Doppler Search Space	Measured Latency
Stage 1	±100 Hz around the Doppler frequency predicted using assistance data, in steps of 5 Hz	initial part of sample capture: 100 ms + 7 non-coherent combinations: 14 s Total: 14.1 s
Stage 2	±15 Hz around the Doppler frequency estimated in step 1, in steps of 5 Hz	initial part of sample capture: 100 ms + 7 non-coherent iterations: 2 s Total: 2.1 s

When utilizing a TCXO with the configuration outlined in Table 6.10, the acquisition process takes a maximum of 14.1 s to transition from stage 1 to stage 2 in the worst-case scenario. Considering that the worst-case Doppler shift rate experienced by a static receiver on the Earth's surface is approximately 1 Hz/s, as explained in Section 2.2.4, the Doppler frequency of the received signal may change by 14.1 Hz during the transition from stage 1 to stage 2. The Doppler search space used in the second stage is ±15 Hz, which is larger than 14.1 Hz. Therefore, any signal detected in the first stage can be detected in the second stage.

The limitation of the Doppler search space to ±100 Hz necessitates the use of CFO compensation and correction, due to the large CFO observed in TCXO 2 and TCXO 3. For this reason, the receiver was tested using the two-stage acquisition process configured according to Table 6.10, along with the CFO estimation and correction procedure. The receiver successfully acquired the weak GNSS signals listed in Table 6.11, initiated tracking of them, and detected their telemetry preambles. The satellites acquired by each TCXO differ because the results were obtained at different times. The advantage of utilizing the second acquisition stage became evident, as the receiver failed to track the detected signals if the second stage was disabled. When using the TCXOs, some channels experienced occasional loss of lock during tracking. This issue was related to the configuration of the tracking loops (see Appendix 6.A) and the measured TCXO frequency stability shown in Table 6.9. This problem did not occur when using the OCXO (see Section 6.5.2.2).

Table 6.11 List of signals acquired and tracked when utilizing a TCXO.

TCXO	E1b/c Signals	C/N_0 in dB-Hz
TCXO 1	E1	33
	E4	27
	E13	30
	E15	20
	E21	33
	E26	27
TCXO 2	E2	31
	E11	28
	E18	27
	E24	21
	E25	31
	E36	36
TCXO 3	E10	29
	E12	25
	E24	26
	E25	23
	E33	22

6.5.2.2 Tests Results Using the OCXO

The acquisition was also tested using an OCXO. The OCXO did not require any CFO correction. The OCXO was accurately tuned using a trimmer when used for the first time, and required no further adjustments. When using an OCXO, the receiver was configured according to Table 6.12, with a Doppler search space of ±50 Hz. This Doppler search space accounts for possible inaccuracies in the estimation of the Doppler frequency and in the OCXO calibration.

Table 6.12 Acquisition latency using an OCXO.

Stage	**Doppler Search Space**	**Measured Latency**
Stage 1	±50 Hz around the Doppler frequency predicted using assistance data, in steps of 5 Hz	Initial part of sample capture: 100 ms + 7 non-coherent iterations: 7 s Total: 7.1 s
Stage 2	±10 Hz around the Doppler frequency estimated in step 1, in steps of 5 Hz	Initial part of sample capture: 100 ms + 7 non-coherent iterations: 1.4 s Total: 1.5 s

As shown in Table 6.12, when using the OCXO, stage 1 takes 7.1 s to complete. Considering that the largest Doppler shift rate experienced by a static receiver is about 1 Hz/s [26], the Doppler shift in the received signal may vary by up to 7.1 Hz when transitioning from stage 1 to stage 2. This change is larger than acquisition Doppler search step, which is set to 5 Hz. Consequently, if the second acquisition stage is not utilized, the Doppler frequency of the received signal might not align with the Doppler frequency estimated during acquisition upon initiation of the tracking process.

To mitigate this issue, the two-stage acquisition process was employed also for the OCXO. As explained in Section 6.4.5, stage 2 captures new samples from the analog front-end, increasing the total acquisition latency, but reducing the latency from sample capture to the initiation of the tracking process. In this case, stage 2 was configured to perform a Doppler search of ±10 Hz around the Doppler frequency detected in Stage 1. In this way, stage 2 reduces the time required to transition from sample capture to tracking down to 1.5 s. This ensures that when the receiver begins tracking the detected signals, the Doppler frequency of the detected signals remains close to the Doppler frequency estimated during acquisition. This approach is expected to increase the probability of successfully tracking the detected signals.

The receiver was tested using the configuration shown in Table 6.12, successfully acquiring and tracking weak Galileo E1b/c signals and detecting the telemetry preambles. However, we did not perform tests to quantify the impact of enabling or disabling the second acquisition stage when using the OCXO. Detailed lists of weak signals that were successfully acquired and tracked by the receiver, utilizing an OCXO, are provided in Section 6.5.3.

Due to the manual estimation and correction of the CFO induced by the TCXOs being inconvenient, we switched to using an OCXO, configuring the receiver as outlined in Table 6.12 for subsequent tests. Nevertheless, it would be desirable to improve the acquisition algorithm to eliminate the need for manual CFO compensation. Achieving this would make the use of TCXOs practical in the proposed receiver, as TCXOs enable the detection of weak GNSS signals with a C/N_0 as low as 20 dB-Hz

6.5.3 Acquisition and Tracking of GNSS Signals in Real Time

The objective of the acquisition and tracking tests was to confirm the receiver's capability to process GNSS signals and derive navigation solutions in real-time. The receiver underwent testing using the test setup described in Section 6.5.1. The performance of the receiver was assessed both using signals received at nominal power and using weak signals with a C/N_0 down to 20 dB-Hz. The OCXO was used for these tests. The test results presented in this section were obtained using the GESTALT testbed rooftop GNSS antenna facility [146]. Similar results could be obtained using the Tallysman TW8825 active antenna and a variable attenuator in outdoor conditions. Each test was conducted over a period of 10 min.

The test procedure is outlined in Table 6.13. Steps 1 and 2 verify the receiver's ability to process Galileo E1b/c signals in high-sensitivity mode concurrently with Galileo E5a and GPS signals in normal-sensitivity mode. Steps 3 and 4 confirm the receiver's ability to process Galileo E1b/c signals at a C/N_0 down to 20 dB-Hz. The OCXO was used for these tests.

Table 6.13 Test procedure.

Step	Description
Step 1	Use the receiver in normal-sensitivity mode to obtain the ephemeris data, the ionospheric data, and the UTC model for the visible Galileo satellites.
Step 2	Use the receiver in high-sensitivity mode with assistance data, including the data obtained in step 1, to receive GNSS signals at nominal power and obtain navigation solutions
Step 3	Stop the receiver and increase the signal attenuation until the C/N_0 of some Galileo E1b/c signals is down to 20 dB-Hz
Step 4	Use the receiver in high-sensitivity mode, to acquire and track weak GNSS signals and obtain navigation solutions.

For each test, we report the GNSS signals that were successfully acquired and tracked, and the time it took for the receiver to obtain a position. When using weak signals, the C/N_0 of the signals that were successfully tracked is also reported.

More accurate TTFF measurements could be obtained by measuring and averaging the time required for the receiver to determine its position over a period of 24 hours, thereby reducing the impact of geometric effects.

6.5.3.1 Tests Results Using GPS and Galileo Signals at Nominal Power Levels

Table 6.14 shows the test results obtained using signals received at nominal power. The receiver successfully acquired and tracked Galileo E1b/c, Galileo E5a, GPS L1 C/A, and GPS L5 signals, achieving navigation solutions with TTFF values of about 1 minute.

Table 6.14 Processing GNSS signals at nominal power levels.

Test	Signals Tracked	TTFF
Test 1	8 Galileo E1b/c + 8 Galileo E5a + 7 GPS L1 C/A + 5 GPS L5	1 min 03 s
Test 2	8 Galileo E1b/c + 8 Galileo E5a + 6 GPS L1 C/A + 4 GPS L5	57 s
Test 3	8 Galileo E1b/c + 8 Galileo E5a + 7 GPS L1 c/a + 4 GPS L5	1 min 12 s

6.5.3.2 Tests Results Using Weak Galileo E1b/c Signals

The test results using weak signals are summarized in Table 6.15. The receiver successfully acquired and tracked weak Galileo E1b/c signals, including those with a C/N_0 down to 20 dB-Hz, obtaining navigation solutions with TTFF values ranging from 1 to 2 minutes. The receiver used the observations obtained from signals with a C/N_0 at or close to 20 dB-Hz to compute the navigation solutions.

Table 6.15 Processing weak Galileo E1b/c signals.

Test	**E1b/c Signals Tracked and C/N_0 in dB-Hz**	**TTFF**
Test 1	E2: 27 dB-Hz E8 : 20 dB-Hz E10: 20 dB-Hz E25: 25 dB-Hz E36: 23 dB-Hz	1 min 40 s
Test 2	E13: 23 dB-Hz E15: 28 dB-Hz E21: 20 dB-Hz E27: 28 dB-Hz E30: 26 dB-Hz E34: 26 dB-Hz	53 s
Test 3	E3: 21 dB-Hz E8: 22 dB-Hz E13: 22 dB-Hz E15: 21 dB-Hz	1 min 15 s

The results demonstrate that the receiver can detect weak Galileo E1b/c signals with a C/N_0 as low as 20 dB-Hz in real-time and obtain navigation solutions. However, while many measurements yielded a TTFF between 1 and 2 min, there were instances where the receiver faced challenges in determining position using the observables derived from weak signals.

6.5.4 Precision of the Navigation Solutions

The quality of navigation solutions obtained by the receiver was evaluated in high-sensitivity mode through two tests. In the first test, the receiver processed Galileo E1b/c signals in high-sensitivity mode along with Galileo E5a and GPS signals in normal-sensitivity mode, all signals received at nominal power levels. In the second test, the receiver processed weak Galileo E1b/c signals with a C/N_0 of approximately 20 dB-Hz.

As explained in Section 2.7.4, the concept of *precision* denotes how closely a given solution aligns with the average of all solutions gathered, reflecting the repeatability and distribution of the measurements. The precision of the navigation solutions was evaluated by employing the standard positioning precision metrics outlined in Section 2.7.4, along with their respective static confidence regions. These measurements included DRMS and CEP for 2D positioning, as well as SAS, MRSE, and SEP for 3D positioning. The DRMS and CEP measurements are detailed in Table 2.10, whereas the SAS, MRSE, and SEP measurements are detailed in Table 2.11, in Chapter 2.

The latitude, longitude, and height coordinates obtained by the receiver were converted to a local ENU coordinate system. The ENU system was anchored to a selected reference point in proximity to the receiver's antenna, employing a WGS-84 reference ellipsoid. The means and standard deviations were computed as detailed in Section 2.7.4.

The measurements were conducted using the GESTALT testbed rooftop antenna [146]. The experimental scenario described in Section 6.5.1 was employed, along with the test setup depicted in Figure 6.9. The OCXO was used for these tests.

The receiver, configured for single-point positioning mode, processed GNSS signals in real time and dumped PVT solutions to a file. Each test was conducted over a period of 10 min.

6.5.4.1 Tests Results Using GPS and Galileo Signals at Nominal Power Levels

The 2D precision results are presented in Table 6.16, and the 3D precision results are provided in Table 6.17. The receiver successfully processed 7 Galileo E1b/c, 7 Galileo E5a, 7 GPS L1 C/A, and 4 GPS L5 signals. The navigation solutions were obtained using up to 11 observations. The satellites broadcasting the unhealthy flag were not used for the computation of the PVT.

Table 6.16 The 2D precision results.

Measure	Results [m]	Confidence Region Probability
2DRMS	4.3	95%
DRMS	2.1	65%
CEP	1.8	50%

Table 6.17 The 3D precision results.

Measure	Results [m]	Confidence Region Probability
99% SAS	8.5	99%
90% SAS	6.3	90%
MRSE	5.0	61%
SEP	3.8	50%

Significantly, 2D precision measurements demonstrate greater accuracy compared to their 3D counterparts, as expected due to the superior GNSS accuracy in the horizontal plane when contrasted with the vertical plane. This variation is influenced by the angle between the line of sight to various satellites and the Earth's surface.

6.5.4.2 Tests Results Using Weak Galileo E1b/c Signals

The accuracy of the navigation solutions was evaluated using weak signals. These measurements yielded variable outcomes. The test results shown below are an example of the performance that could be achieved.

The C/N_0 of the received Galileo E1b/c signals was decreased to the levels shown in Table 6.18 using the variable attenuator. The receiver processed the received signals and obtained the navigation solutions using four observations. The 2D precision results are presented in Table 6.19, and the 3D precision results are provided in Table 6.20.

Table 6.18 Galileo satellites tracked and C/N_0.

Satellite	**C/N_0 in dB-Hz**
E3	20 dB-Hz
E8	23 dB-Hz
E13	22 dB-Hz
E15	20 dB-Hz

Table 6.19 The 2D precision results using weak Galileo E1b/c signals.

Measure	**Results [m]**	**Confidence Region Probability**
2DRMS	7.5	95%
DRMS	3.8	65%
CEP	3.1	50%

Table 6.20 The 3D precision results using weak Galileo E1b/c signals.

Measure	**Results [m]**	**Confidence Region Probability**
99% SAS	18.0	99%
90% SAS	13.4	90%
MRSE	11.4	61%
SEP	8.9	50%

The precision of navigation solutions can significantly decrease when tracking weak GNSS signals, and the quality of these solutions becomes highly variable, while the results in Tables 4.9 and 6.20 are achievable, errors can increase depending on satellite geometry and signal strength. Implementing algorithms to enhance the accuracy of PVT solution when tracking severely degraded signals is a focus for future work.

6.5.5 FPGA Resource Utilization

Table 6.21 details the FPGA resource utilization for the HS-GNSS receiver developed with the XCZU9EG device. This table reports the resource usage of the GNSS receiver features, including channel conditioning, buffering, and acquisition and tracking hardware accelerators, along with their AXI4 memory-mapped registers. The FPGA resource usage is categorized by LUTs, LUTRAMs, FFs, BRAMs, and DSP slices.

Compared to the resource utilization of the XCZU9EG-based spaceborne receiver shown in Section 4.4.2.5, this design consumes a larger number of resources in the FPGA. This increase in resource usage is associated with the high-sensitivity acquisition hardware accelerator, which utilizes a large, multi-channel FFT, resulting in increased memory consumption.

Table 6.21 FPGA resource utilization in the high-sensitivity GNSS receiver implemented on the XCZU9EG device.

Resource	Utilization	Resources available	Utilization %
LUT	146000	274080	53
LUTRAM	4028	144000	3
FF	232544	548160	42
BRAM	823	912	90
DSP	885	2520	35

6.5.6 Estimated Power Consumption

The estimated power consumption of the XCZU9EG-FFVB1156-2-e SoC-FPGA, when operating a high-sensitivity GNSS receiver, is 12 W. This estimate, provided by FPGA design tools, assumes an average processor load of 75%. Such a load occurs when the receiver processes a large volume of GNSS signals simultaneously, including 10 signals each from Galileo E1, GPS L1 C/A, Galileo E5a, and GPS L5. This figure results in a power consumption of approximately (12 W)/(40 channels) = 0.3 W/channel.

To evaluate the average processor load during the processing of multiple signals, we followed a similar approach as described in Section 4.4.1.6 for the spaceborne receiver. The receiver was connected to the rooftop GNSS antenna facility of the GESTALT testbed [146], allowing it to simultaneously track the same signals across multiple channels. This ensured that every channel tracked at least one satellite signal. The average processor load was estimated using the OS tools available on the receiver.

The rise in power consumption, compared to the spaceborne receiver implemented on the same board as discussed in Section 4.4.2.6, is primarily attributed to the activity of the high-sensitivity acquisition hardware accelerator. This accelerator executes computationally demanding algorithms and accesses an additional external memory (referred to as the PL memory in Section 6.4) through the FPGA pinout. Consequently, this leads to increased I/O operations and requires additional circuitry within the FPGA to support the memory interface. It is assumed that the high-sensitivity acquisition hardware accelerator continuously searches for visible satellites as long as there are available channels in the receiver.

The real-time acquisition of weak signals in high-sensitivity mode has not been implemented purely in software. However, as detailed in Section 4.4.1.6, the power consumption of the GNSS-SDR software receiver is estimated to be approximately 0.53 W per channel when operating in normal-sensitivity mode on a PC equipped with a 28 W, 11th generation Intel Core i7-1185G7 processor running at 3 GHz. This comparison suggests an improvement in power efficiency in the SoC-FPGA of at least 43% in terms of power consumption per channel, compared to the general purpose processor.

6.6 Conclusions

This chapter reported the design, proof-of-concept implementation, and preliminary performance assessment of a HS-GNSS receiver, providing practical details of a working prototype.

This design showcases the effectiveness of SDR techniques for the acquisition and tracking of severely degraded GNSS signals in real-time, thereby facilitating experimentation and the development of high-sensitivity GNSS receiver algorithms. Leveraging the SoC-FPGA GNSS receiver architecture proposed in Chapter 3, it combines the massive parallelism and energy efficiency of FPGAs with the flexibility of embedded processors to offer a balanced solution for implementing experimental HS-GNSS receivers that work in real time. Furthermore, the integration of a FOSS software-defined GNSS receiver with hardware accelerators in the FPGA facilitates experimentation of innovative high-sensitivity algorithms.

After presenting the system architecture, the receiver was tested. It demonstrated the ability to acquire and track weak Galileo E1b/c signals in real-time. The receiver processes Galileo E1b/c signals with a carrier-to-noise ratio (C/N_0) as low as 20 dB-Hz, successfully obtaining navigation solutions. Furthermore, it can simultaneously handle Galileo E5a, GPS L1 C/A, and GPS L5 signals at their nominal power levels, enhancing the availability of satellite signals.

The estimated power consumption of the SoC-FPGA implementing the proposed receiver is approximately 12 W, making it suitable for operation in battery-powered devices. The physical dimensions of the receiver prototype are relatively large, measuring approximately 23.8 cm by 24.4 cm, with the analog front-end being 14 cm by 9 cm. Despite its size, the receiver remains portable and can be conveniently placed in backpacks, for instance, to facilitate testing in various indoor environments.

To enhance the proposed receiver's performance, the following potential improvements were identified:

The quality of navigation solutions can deteriorate when tracking GNSS signals in moderate indoor scenarios, due to variations in received signal strength, satellite geometry, and potential signal reflections. The proposed receiver is designed to facilitate research and testing of experimental multipath mitigation algorithms, which could significantly enhance these navigation solutions. Although it is capable of acquiring and tracking weak GNSS signals, it currently does not incorporate such algorithms.

In addition, receiver availability in moderate indoor scenarios (the proportion of time the software receiver is in a functioning condition) may be significantly improved by acquiring and tracking GPS and Galileo E5a signals in high-sensitivity mode. This would improve the success rate of positioning and the quality of the navigation solutions by receiving more satellite signals in harsh environments.

Finally, incorporating a TCXO into the receiver necessitates a manual process for estimating and correcting the CFO, which arises from clock instability. As an alternative, optimizing memory usage could allow the receiver to manage the increased CFO, but would lead to greater acquisition latency and a longer TTFF. To improve usability, it's desirable to enhance the receiver's algorithms, allowing the practical application of TCXO without the need for manual CFO estimation and correction, and without a significant increase in TTFF. This is particularly important as TCXOs are capable of detecting and tracking GNSS signals down to 20 dB-Hz and are widely used in mass-market receivers.

Future work includes implementing multipath mitigation algorithms extending the high-sensitivity mode for GPS and Galileo E5a signals, thereby enhancing the accuracy of navigation solutions. Additionally, it is planned to improve the resilience of the acquisition algorithm to CFO and to integrate it with the proposed receiver that utilizes a TCXO.

The results presented in this chapter were published in

- [19] M. Majoral, J. Arribas, and C. Fernández-Prades, "Implementation of a High-Sensitivity Global Navigation Satellite System Receiver on a System-on-Chip Field-Programmable Gate Array Platform," *Sensors*, vol. 24, no. 5, 2024, Art. no. 1416. doi: 10.3390/s24051416

The work presented in this section was supported by the following funding sources and collaborators:

- Space Agency contract 4000133576/20/NL/CRS with SENER Aeroespacial S.A., with the CTTC serving as the subcontractor.
- Grant CPP2021-008648, titled "Receptor Abierto y Flexible de Investigación por Satélite", funded by the Ministry of Science and Innovation.
- Grant PID2021-128373OB-I00, funded by the Spanish Ministry of Science and Innovation through MCIN/AEI/10.13039/501100011033, and by the European Regional Development Fund (ERDF) "A way of making Europe".

Appendix 6.A Receiver configuration for Galileo E1 b/c signals in high-sensitivity mode

Table 6.22 presents the acquisition parameters for the Galileo E1b/c signals when operating in high-sensitivity mode. The *Doppler max* represents the maximum Doppler frequency, and the *Doppler step* is the frequency step in the search grid. Doppler prediction is enabled to narrow down the Doppler frequency search space. The proposed two-step acquisition scheme is used. In the first step, a large Doppler search space of ±50 Hz is performed. In the second step, a very small Doppler search space of ±10 Hz is used around the Doppler frequency detected in the first step.

The sampling frequency of the base-band processing engine is set to 15.728640 Msps. Additionally, a downsampling filter is specifically applied in the L1/E1 band to reduce the acquisition latency. This filter is exclusively utilized during the acquisition process and reduces the sampling frequency used to acquire the Galileo E1b/c signals from 15.728640 Msps to 3.932160 Msps, hence facilitating the computation of long FFTs in the acquisition.

Table 6.22 High-sensitivity receiver Galileo E1b/c acquisition configuration.

Parameter type	Parameter	Value
Acquisition Galileo E1b/c	CI time	100 ms
	Max NCI combinations	7
	Doppler prediction	Enabled
	1st stage acq: Doppler max	50 Hz
	1st stage acq: Doppler step	5 Hz
	2nd stage acq: Doppler max	10 Hz
	2nd stage acq: Doppler step	5 Hz
	PFA	0.01
	Downsampling factor	4

Table 6.23 displays the receiver's tracking configuration parameters for the Galileo E1b/c signals when operating in high-sensitivity mode. The CI time is specifically set to 40 ms to enhance the apparent signal-to-noise ratio. In high-sensitivity mode, upon initiating tracking for the detected signals, the receiver is already synchronized with the secondary code of the received signals, enabling for the utilization of a long CI time right from the start.

In Table 6.23, *early-late narrow space chips* refers to the spacing between the Early and Prompt correlators, and between the Prompt and Late correlators. Similarly, *very early late space narrow chips* indicates the spacing between the Very-Early and Prompt correlators, and between the Prompt and Very-Late correlators. Both spacings are normalized by the chip period. The *PLL filter bandwidth* with a narrow correlator configuration represents the bandwidth of the PLL low-pass filter. Finally, the *DLL filter bandwidth* with a narrow correlator configuration denotes the bandwidth of the DLL low-pass filter.

Table 6.23 High-sensitivity receiver Galileo E1 b/c tracking configuration.

Parameter type	**Parameter**	**Value**
Tracking Galileo E1b/c	CI time	40 ms
	Early-Late space narrow chips	0.15 chips
	Very Early-Late space narrow chips	0.5 chips
	PLL filter bandwidth (narrow correlator configuration)	5 Hz
	DLL filter bandwidth (narrow correlator configuration)	1.0 Hz

Appendix 6.B Receiver configuration for Galileo E5a and GPS signals in normal-sensitivity mode

In Table 6.24, the receiver configuration for the acquisition of the Galileo E5a and the GPS signals is detailed. Doppler prediction is not utilized. However, self-assistance to acquisition, from the primary to secondary band, is enabled. Therefore, once the primary band is successfully acquired, the receiver estimates the Doppler frequency in the secondary band. Subsequently, it initiates acquisition in the secondary band with a reduced Doppler search grid. For instance, a Doppler search space of ±5000 Hz is applied for GPS L1 C/A signals, while a reduced Doppler search space of ±500 Hz around the estimated Doppler frequency in the secondary band is employed for the GPS L5 and the Galileo E5a signals.

The sampling frequency of the base-band processing engine is set to 15.728640 Msps. Additionally, a downsampling filter is specifically applied in the L1/E1 band to reduce the acquisition latency. This filter is exclusively utilized during the acquisition process and reduces the sampling frequency used to acquire the GPS L1 C/A signals from 15.728640 Msps to 3.932160 Msps.

Table 6.24 High-sensitivity receiver GPS and Galileo E5a acquisition configuration (normal-sensitivity mode).

Parameter type	Parameter	Value
Acquisition GPS L1 C/A	CI time	1 ms
	Max NCI combinations	4
	Doppler Max	5000 Hz
	Doppler Step	250 Hz
	PFA	0.1
	Downsampling Factor	4
Acquisition GPS L5	CI time	1 ms
	Max NCI combinations	4
	Assistance to secondary band	Enabled
	Doppler Max	500 Hz
	Doppler Step	250 Hz
	PFA	0.1
Acquisition Galileo E5a	CI time	1 ms
	Max NCI combinations	4
	Assistance to secondary band	Enabled
	Doppler Max	500 Hz
	Doppler Step	250 Hz
	PFA	0.1

Table 6.25 presents the tracking configuration parameters for the GPS signals, while Table 6.26 presents the tracking configuration parameters for the Galileo E5a signals. Initially, the tracking process operates without synchronization to telemetry preambles and/or the pilot's secondary code, utilizing a CI time of 1 ms, equivalent to the duration of the primary PRN code.

After achieving synchronization, the CI time is set to 20 ms to enhance the apparent signal-to-noise ratio. The *early–late space chips* represents the spacing between the Early and Prompt, and between the Prompt and Late correlators before synchronization, while the *early–late narrow space chips* denotes the spacing between the Early and Prompt, and between the Prompt and Late correlators after synchronization. Both parameters are normalized by the chip period. Additionally, the PLL filter bandwidth and the DLL filter bandwidth, along with their configurations with narrow correlator settings, indicate the bandwidths of the PLL and DLL low-pass filters before and after achieving synchronization, respectively.

Table 6.25 High-sensitivity receiver GPS tracking configuration (normal-sensitivity mode).

Parameter type	Parameter	Value
Tracking GPS L1 C/A	CI time	20 ms
	Early-Late space chips	0.5 chips
	Early-Late space narrow chips	0.1 chips
	PLL filter bandwidth	35 Hz
	PLL filter bandwidth (narrow correlator configuration)	7.5 Hz
	DLL filter bandwidth	2 Hz
	DLL filter bandwidth (narrow correlator configuration)	0.5 Hz
Tracking GPS L5	CI time	20 ms
	Early-Late space chips	0.5 chips
	Early-Late space narrow chips	0.1 chips
	PLL filter bandwidth	20 Hz
	PLL filter bandwidth (narrow correlator configuration)	7.5 Hz
	DLL filter bandwidth	1.5 Hz
	DLL filter bandwidth (narrow correlator configuration)	0.5 Hz

Table 6.26 High-sensitivity receiver Galileo E5a tracking configuration (normal-sensitivity mode).

Parameter type	Parameter	Value
Tracking Galileo E5a	CI time	20 ms
	Early-Late space chips	0.5 chips
	Early-Late space narrow chips	0.1 chips
	PLL filter bandwidth	20 Hz
	PLL filter bandwidth (narrow correlator configuration)	7.5 Hz
	DLL filter bandwidth	1.5 Hz
	DLL filter bandwidth (narrow correlator configuration)	0.5 Hz

Appendix 6.C Receiver configuration for PVT

Table 6.27 displays the receiver PVT configuration parameters, with the positioning mode set to single-point positioning. The receiver incorporates RAIM FDE, follows the broadcasted ionospheric model for ionospheric correction, and employs the Saastamoinen tropospheric model [151]. The PVT output rate is configured at 1 s, and satellites broadcasting the unhealthy bit are excluded from the computation of the PVT solution. Additionally, a Kalman filter is applied to the PVT solutions.

Table 6.27 High-sensitivity receiver PVT configuration.

Parameter type	Parameter	Value
General	Positioning Mode	Single
	RAIM FDE	Enabled
	Iono Model	Broadcast
	Trop Model	Saastamoinen
	PVT Output Rate	1 s
	Use Unhealthy sats	Disabled
PVT Kalman Filter	Standard deviation of the position estimations	1 m
	Std. dev. of the velocity estimations	0.1 m
	Std. dev. of the dynamic system model for pos.	2 m
	Std. dev. of the dynamic system model for vel.	0.5 m

Chapter 7

Conclusions and Directions for Future Work

This chapter consolidates the conclusions and contributions derived from the research discussed in the previous chapters. Furthermore, it suggests a list of potential topics for future investigation.

This dissertation focused on the design and development of a low-cost SoC FPGA architecture for prototyping experimental GNSS receivers. The proposed architecture not only overcomes the flexibility limitations of commercial receivers but also improves energy consumption compared to software-defined receivers operating on general-purpose processors. Combining the adaptability of SDR with the parallel processing capabilities and energy efficiency of reprogrammable hardware, it presents a synergistic solution that emphasizes customization, flexibility, and reprogrammability. Furthermore, this architecture is based on a FOSS software-defined GNSS receiver engine, fostering collaborative development and facilitating both the inspection and modification of the signal processing path. Such an approach facilitates the creation of compact, portable, multi-band, and multi-constellation GNSS receivers for research and field testing.

This work began with a background on GNSS systems in Chapter 2. The operation of GNSS receivers was introduced, providing an overview of basic positioning techniques. Additionally, we discussed the GNSS-SDR software-defined receiver, which plays an integral role in the SoC FPGA architecture proposed in this thesis. Subsequently, we explored the intricacies of FPGA and SoC FPGA technologies, focusing on their principal characteristics and design flows. An FPGA is a programmable, multi-purpose digital chip comprising a matrix of CLBs connected through programmable interconnects. This makes FPGAs particularly suitable for applications that benefit from massive parallelism, such as those requiring high computing performance with minimal decision branching. However, designing with FPGAs presents several challenges, including the use of complex design tools, the need to master HDL languages, and engaging in extensive design processes like simulation, synthesis, and implementation, consequently extending the design cycle and potentially increasing development times. Therefore, FPGAs are most effectively utilized in tasks that leverage their strengths in parallel processing and simultaneously require infrequent updates. After exploring FPGA and SoC FPGA technologies, the background section concluded with an overview of design forces and KPIs relevant to GNSS devices.

Following the background section, Chapter 3 delved into the proposed SoC FPGA receiver architecture and its design methodology. To maximize the synergy between software and hardware and to merge their benefits, we adopted a two-dimensional approach to software-hardware partitioning, aligning with the FPGA strengths explored in Chapter 2. This approach is based on the nature of the tasks and their maintenance requirements: Signal processing tasks demanding significant computing power, benefiting from parallelism, and requiring infrequent updates are allocated to the FPGA. Generally, in GNSS receivers, these tasks are those that process the incoming samples from the RFFE at the base-band sampling rate. In line with this, in the proposed architecture, the FPGA implements sample conditioning, buffering, and the execution of acquisition and tracking multicorrelator hardware accelerators for signal processing. Detailed descriptions were provided for the implementation of these hardware accelerators within the FPGA, covering the specific acquisition and tracking algorithms used. Conversely, tasks with lower processing demands, ranging from managing tracking loops to computing PVT solutions, are allocated to the embedded processor.

The proposed architecture implements a scalable platform, enabling the unrestricted incorporation of additional signals and algorithms. Key highlights of the architecture are its extensive testing framework, detailed application-level logging, and adaptable configuration system. The FPGA hardware accelerators are not necessarily subject to FOSS licenses. This provides an opportunity to monetize research while also increasing research impact and reputation. Nevertheless, accessibility to the signal processing details can be facilitated through the implementation of equivalent algorithms in GNSS-SDR and the documentation of FPGA IP cores.

After discussing the architecture we moved to the design methodology. In the proposed design methodology, experimental algorithms are initially tested in software, using GNSS-SDR. Subsequently, these algorithms are ported to a SoC FPGA platform for field testing and validation on portable devices. The modifications proposed to the GNSS-SDR software receiver designed to facilitate the offloading of computationally intensive tasks to the FPGA were also discussed. Communication between the embedded processor and the FPGA is facilitated by memory-mapped registers and interrupt lines, ensuring fast data exchange and synchronized operations with minimal latency.

The introduction of a SoC FPGA architecture and a generic design methodology leveraging a FOSS core GNSS engine opens new possibilities for prototyping satellite navigation systems for various applications requiring precise positioning and timing solutions, with low power consumption. The adoption of a FOSS GNSS baseband processing engine accelerates innovation by allowing a diverse range of contributors to enhance the technology. Collectively, these innovations represent a step forward, making advanced GNSS technologies more accessible and adaptable to research needs.

The main challenges for the proposed architecture in supporting cost-effective prototyping of GNSS receivers within the research community stem from the intricacies associated with FPGA design and its portability. These challenges are compounded by the fact that major FPGA manufacturers offer specialized proprietary software tools designed specifically for their hardware, driven by the intricacies of FPGA architecture, the limited availability of open documentation, and the need for intellectual property protection. The concept demonstrators proposed in this thesis have been developed using AMD's SoC FPGAs, which span from low-power and cost-efficient designs to high-performance models suited for complex applications, offering flexibility across a broad spectrum of hardware options. However, porting these designs to SoC FPGAs from other manufacturers poses considerable challenges due to significant hardware differences between the FPGAs of various producers. These challenges can be mitigated by employing platform-independent HDL coding practices as much as possible and embracing modular designs. Still, portability across manufacturers remains an obstacle. In addition to this, adding new functionalities or addressing fixes in the FPGA often requires more time compared to software modifications because of the inherent complexities in FPGA design and implementation. A viable approach to enhancing maintainability would be to establish a continuous integration system for the FPGA code that includes automated builds and testing. This continuous integration system could automatically instantiate newly committed FPGA code on a development board, integrate it with the embedded OS and GNSS-SDR, and then automatically run a series of tests. This process would not only verify the system but also further streamline the development process.

Following our discussion on the proposed architecture and design methodology, Chapter 4 presented the design and proof-of-concept implementation of the first concept demonstrator—a spaceborne GNSS receiver based on the proposed framework. It also included a preliminary performance assessment of this demonstrator. This receiver was tested in both static and LEO scenarios, proving its real-time capability in dual-frequency, dual-constellation mode. Tests with recorded signals demonstrated the system's ability to deliver highly accurate navigation solutions. The receiver was implemented using two development boards. One board, featuring a SFF with dimensions of 10 cm by 6.2 cm, served as a proof of concept to demonstrate the feasibility of designing compact receivers suitable for CubeSat integration. The other board, which targets a more powerful SoC FPGA, demonstrated the scalability of the proposed architecture, adapting to increased computational resources, potentially enabling the support of more complex processing tasks.

The spaceborne receiver exemplifies the effectiveness of SDR techniques in handling high-dynamic scenarios, underscoring the practicality of the proposed architecture and design methodology. It not only validated the suitability of SDR techniques for challenging environments but also illustrated how SoC FPGA technology can be employed to develop low-power, highly customizable receivers. Designed to process GNSS signals in real time and suitable for space-like conditions, this receiver effectively demonstrated the application potential of the architecture.

Chapter 5 detailed the design, proof-of-concept implementation, and initial performance evaluation of the second concept demonstrator: a GNSS rebroadcaster. Specifically designed for low-latency rebroadcasting of incoming GNSS signals, this device not only reproduces any unpredictable data embedded in navigation messages accurately but can also perform real-time modifications to the rebroadcasted PVT solution. Its performance was assessed across multiple metrics: ability to process multiple parallel channels in real-time, functionality across diverse operating modes and GNSS signal types, power efficiency, and the accuracy of rebroadcasted navigation solutions compared to the original received signals.

The test results confirmed the rebroadcaster's efficacy in accurately reproducing GNSS signals. This design underscores the potential of SDR techniques to efficiently generate and rebroadcast GNSS signals, offering opportunities for real-time PVT solution adjustments and various GNSS scenario simulations. The incorporation of advanced processing features within a cost-effective SoC FPGA framework, combined with an open-source software-defined GNSS receiver model, marks a significant innovation in the domain.

Finally, Chapter 6 reported the design, proof-of-concept implementation, and preliminary performance assessment of the third concept demonstrator: a HS-GNSS receiver for processing weak GNSS signals. The proposed concept demonstrator implements two operating modes: normal sensitivity mode and high-sensitivity mode, with an acquisition and tracking sensitivity of about 37 dB-Hz for normal sensitivity mode, and as low as 20 dB-Hz for high-sensitivity mode. The initial version of the concept demonstrator was specifically designed to showcase high-sensitivity mode for Galileo E1 b/c signals. To maximize signal availability whenever possible, the receiver can be configured to process Galileo E1 b/c signals in high-sensitivity mode while simultaneously handling GPS signals and Galileo E5a signals in normal-sensitivity mode.

The HS-GNSS receiver was tested against several KPIs, demonstrating the receiver's ability to process Galileo E1b/c signals with a carrier-to-noise ratio (C/N_0) as low as 20 dB-Hz, successfully obtaining navigation solutions, and to simultaneously handle Galileo E5a, GPS L1 C/A, and GPS L5 signals at their nominal power levels, enhancing the availability of satellite signals. This design showcases the effectiveness of SDR techniques for the acquisition and tracking of severely degraded GNSS signals in real-time, thereby facilitating experimentation and the development of high-sensitivity GNSS receiver algorithms. Potential improvements include implementing a high-sensitivity mode for GPS and Galileo E5a signals, along with the testing and development of multipath mitigation algorithms for indoor environments.

Potential areas for improvement among the concept demonstrators showcased in this thesis include increasing processing capacity for simultaneous GNSS channels—especially when operating on low-power SFF development boards—and enhancing the quality of navigation solutions when operating with dual RF transceivers.

Specifically, the spaceborne GNSS receiver deployed on the ADRV9361-Z7035 development board could only process up to 24 satellite signals simultaneously in real-time. The receiver achieved this capability by temporarily throttling the PVT block to reduce the processor load during operation. This limitation mainly arises from the elevated rate of interrupts requests generated by the tacking multicorrelator hardware accelerators in the FPGA, whose maximum coherent integration time is currently limited to the duration of one data symbol, meaning that longer integrations have to be accumulated in software. By extending the coherent integration time within the FPGA beyond that, the frequency of interrupt requests would be reduced. This would be particularly beneficial when tracking Galileo E1b/c signals and GPS L5 signals, which have brief data symbol durations of 4 ms and 10 ms respectively. Such an implementation would enable the receiver to simultaneously track a greater number of GNSS signals.

Furthermore, when utilizing the ZCU102 development board and the AD-FMCOMMS5-EBZ

RFFE in multi-frequency mode, there is potential for improvement in the quality of navigation solutions. This challenge arises from the RF transceivers in the AD-FMCOMMS5-EBZ not being calibrated for dual-frequency operation, which may result in delays between frequency bands.

Future work includes addressing these potential areas for improvement, increasing the number of channels that can be simultaneously processed in real time, and developing calibration procedures for the RFFE. Furthermore, we intend to implement a continuous integration system to effectively manage and improve the efficiency of the development process of the FPGA source code. This will facilitate the collaborative utilization of the proposed architecture for the implementation, testing, and validation of novel GNSS signal processing algorithms.

The continuous improvement and integration of new features will progressively enhance the proposed architecture's value, elevating its flexibility and optimizing the development process for experimental GNSS receiver prototypes. This initiative marks a significant advancement for the research community, streamlining the creation and deployment of experimental GNSS receivers, and facilitating the thorough validation and field-testing of groundbreaking GNSS signals and systems.

Bibliography

[1] D. Egea-Roca, M. Arizabaleta-Diez, T. Pany, F. Antreich, J. A. López-Salcedo, M. Paonni, and G. Seco-Granados, "GNSS User Technology: State-of-the-Art and Future Trends," *IEEE Access*, vol. 10, pp. 39 939–39 968, 2022. doi: 10.1109/ACCESS.2022.3165594

[2] European GNSS Agency, *GNSS user technology report. Issue 3.* Luxembourg: Publications Office of the European Union, 2020. doi: doi/10.2878/565013

[3] C. A. Ogaja, "Augmented Reality: A GNSS Use Case." in *Introduction to GNSS Geodesy: Foundations of Precise Positioning Using Global Navigation Satellite Systems*, 1st ed. Springer International Publishing: Cham, Switzerland, 2022, pp. 3–12. doi: 10.1007/978-3-030-91821-7

[4] X. Xia, Z. Meng, X. Han, H. Li, T. Tsukiji, R. Xu, Z. Zheng, and J. Ma, "An automated driving systems data acquisition and analytics platform," *Transportation Research Part C: Emerging Technologies*, vol. 151, p. 104120, 2023. doi: 10.1016/j.trc.2023.104120

[5] M. Navarro, J. Arribas, J. Vilà-Valls, J. Casademont, A. Calveras, and M. Catalán, "Hybrid GNSS/INS/UWB Positioning for Live Demonstration Assisted Driving," in *Proceedings of the 2019 IEEE Intelligent Transportation Systems Conference (ITSC)*, Auckland, New Zealand, October 2019, pp. 3294–3301. doi: 10.1109/ITSC.2019.8917435

[6] G. De Angelis, G. Baruffa, and S. Cacopardi, "GNSS/Cellular Hybrid Positioning System for Mobile Users in Urban Scenarios," *IEEE Transactions on Intelligent Transportation Systems*, vol. 14, no. 1, pp. 313–321, March 2013. doi: 10.1109/TITS.2012.2215855

[7] T. Janssen, A. Koppert, R. Berkvens, and M. Weyn, "A Survey on IoT Positioning Leveraging LPWAN, GNSS, and LEO-PNT," *IEEE Internet of Things Journal*, vol. 10, no. 13, pp. 11 135–11 159, 2023. doi: 10.1109/JIOT.2023.3243207

[8] C. Fernández-Prades, J. Arribas, M. Majoral, J. Vilà-Valls, A. García-Rigo, and M. Hernández-Pajares, "An Open Path from the Antenna to Scientific-grade GNSS Products," in *6th ESA Intl. Colloquium on Scientific and Fundamental Aspects of GNSS / Galileo*, Valencia, Spain, October 2017.

[9] G. E. Moore, "Cramming more components onto integrated circuits, Reprinted from Electronics, volume 38, number 8, April 19, 1965, pp.114 ff." *IEEE Solid-State Circuits Society Newsletter*, vol. 11, no. 3, pp. 33–35, September 2006. doi: 10.1109/NSSC.2006.4785860

[10] R. H. Dennard, F. H. Gaensslen, H.-N. Yu, V. L. Rideout, E. Bassous, and A. R. LeBlanc, “Design of ion-implanted MOSFETs with very small physical dimensions,” *IEEE Journal of Solid-State Circuits*, vol. 9, no. 5, pp. 256–268, October 1974. doi: 10.1109/JSSC.1974.1050511

[11] A. A. Chien and V. Karamcheti, “Moore’s Law: The First Ending and a New Beginning,” *Computer*, vol. 46, no. 12, pp. 48–53, December 2013. doi: 10.1109/MC.2013.431

[12] M. Horowitz, “1.1 Computing’s energy problem (and what we can do about it),” in *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, San Francisco, CA, USA, February 2014, pp. 10–14. doi: 10.1109/ISSCC.2014.6757323

[13] *Versal: The First Adaptative Compute Acceleration Platform (ACAP)*, Advanced Micro Devices Inc., Santa Clara, CA, USA, September 2020, WP505 (v1.1.1).

[14] R. K. Snider, *Advanced Digital System Design using SoC FPGAs* , 1st ed. Springer Cham, Switzerland, 2023. doi: 10.1007/978-3-031-15416-4

[15] L. H. Crockett, D. Northcote, C. Ramsay, F. D. Robinson, and R. W. Stewart, *Exploring Zynq MPSoC: With PYNQ and Machine Learning Applications*. Glasgow, UK: Strathclyde Academic Media, 2019. ISBN 9780992978754

[16] C. Fernández-Prades, J. Arribas, P. Closas, C. Aviles, and E. Luis, “GNSS-SDR: An Open Source Tool for Researchers and Developers,” in *24th International Technical Meeting of the Satellite Division of The Institute of Navigation (ION GNSS 2011)*, Portland, OR, USA, September 2011, pp. 780–794.

[17] “GNSS-SDR An open-source Global Navigation Satellite Systems Software-Defined Receiver.” [Online]. Available: https://gnss-sdr.org/ (Accessed March 19, 2024).

[18] G. Seco-Granados, J. López-Salcedo, D. Jiménez-Baños, and G. López-Risueño, “Challenges in Indoor Global Navigation Satellite Systems: Unveiling its core features in signal processing,” *IEEE Signal Processing Magazine*, vol. 29, no. 2, pp. 108–131, 2012. doi: 10.1109/MSP.2011.943410

[19] M. Majoral, J. Arribas, and C. Fernández-Prades, “Implementation of a High-Sensitivity Global Navigation Satellite System Receiver on a System-on-Chip Field-Programmable Gate Array Platform,” *Sensors*, vol. 24, no. 5, 2024, Art. no. 1416. doi: 10.3390/s24051416

[20] M. Majoral, C. Fernández-Prades, and J. Arribas, “A Flexible System-on-Chip Field-Programmable Gate Array Architecture for Prototyping Experimental Global Navigation Satellite System Receivers,” *Sensors*, vol. 23, no. 23, 2023, Art. no. 9483. doi: 10.3390/s23239483

[21] M. Majoral, J. Arribas, and C. Fernández-Prades, “Implementation of a GNSS Rebroadcaster in an All-Programmable System-On-Chip Platform,” in *2022 10th Workshop on Satellite Navigation Technology (NAVITEC)*, Noordwijk, Netherlands, April 2022, pp. 1–9. doi: 10.1109/NAVITEC53682.2022.9847537

[22] C. Fernández-Prades, J. Arribas, M. Majoral, A. Ramos, J. Vilá-Valls, and P. Giordano, "A Software-Defined Spaceborne GNSS Receiver," in *2018 9th ESA Workshop on Satellite Navigation Technologies and European Workshop on GNSS Signals and Signal Processing (NAVITEC)*, Noordwijk, Netherlands, December 2018, pp. 1–9. doi: 10.1109/NAVITEC.2018.8642697

[23] M. Majoral, C. Fernández-Prades, and J. Arribas, "Implementation of GNSS Receiver Hardware Accelerators in All-programmable System-On-Chip Platforms," in *Proceedings of the 31st International Technical Meeting of the Satellite Division of The Institute of Navigation (ION GNSS+ 2018)*, Miami, FL, USA, September 2018, pp. 4215–4230. doi: 10.33012/2018.16082

[24] C. Fernández-Prades, J. Arribas, M. Majoral, G. Araujo, A. Arnold, C. Avilés, M. Branzanti, Cebrián-Juan, A. Cecilia-Luque, L. Esteve, F. Fabra, D. Fehr, P. Gupta, G. LaMountain, M. Lenhart, J. Melton, D. Miralles, M. Molina, R. Muñoz, C. O'Driscoll, I. Pääkkönen, I. Pérez Riega, D. Pubill, A. Ramos, E. Shin, J. Schindehette, W. Silberman, L. Tonetto, and S. van der Linden, "GNSS-SDR," January 2024. doi: 10.5281/zenodo.10579595. [Online]. Available: https://github.com/gnss-sdr/gnss-sdr (Accessed March 19, 2024).

[25] E. Kaplan and C. Hegarty, *Understanding GPS/GNSS: Principles and Applications*, 3rd ed. Norwood, MA, USA: Artech House, Inc., 2017. ISBN 9781630810580

[26] J. B.-Y. Tsui, *Fundamentals of Global Positioning System Receivers: A Software Approach*, 2nd ed. Hoboken, NJ, USA: John Wiley & Sons, Inc, 2004. doi: 10.1002/0471712582

[27] *Interface Specification IS-GPS-200: Navstar GPS Space Segment/Navigation User Interfaces*, Global Positioning System Directorate, August 2022, Revision N. [Online]. Available: https://www.gps.gov/technical/icwg/IS-GPS-200N.pdf (Accessed March 19, 2024).

[28] *Interface Specification IS-GPS-705: Navstar GPS Space Segment/User Segment L5 Interfaces*, Global Positioning System Directorate, August 2022, Revision J. [Online]. Available: https://www.gps.gov/technical/icwg/IS-GPS-705J.pdf (Accessed March 19, 2024).

[29] E. Union, *Galileo Open Service Signal-in-Space Interface Control Document*. Luxembourg: Publications Office of the European Union, 2023. doi: 10.2878/39727

[30] J. Leclère, R. Landry, and C. Botteron, "Comparison of L1 and L5 Bands GNSS Signals Acquisition," *Sensors*, vol. 18, no. 9, 2018, Art. no. 2779. doi: 10.3390/s18092779

[31] J. Svatoň and F. Vejražka, "Joint Acquisition Estimator of Modern GNSS Tiered Signals Using Block Pre-Correlation Processing of Secondary Code," *Sensors*, vol. 20, no. 10, 2020, Art. no. 2965. doi: 10.3390/s20102965

[32] L. Lestarquit, G. Artaud, and J.-L. Issler, "AltBOC for Dummies or Everything You Always Wanted To Know About AltBOC," in *Proceedings of the 21st International Technical Meeting of the Satellite Division of The Institute of Navigation (ION GNSS 2008)*, Savannah, GA, September 2008, pp. 961–970.

[33] E. Simona Lohan, "Analytical performance of CBOC-modulated Galileo E1 signal using sine BOC(1,1) receiver for mass-market applications," in *Proceedings of the IEEE/ION Position, Location and Navigation Symposium*, Indian Wells, CA, USA, May 2010, pp. 245–253. doi: 10.1109/PLANS.2010.5507207

[34] B. Hofmann-Wellenhof, H. Lichtenegger, and E. Wasle, *GNSS – Global Navigation Satellite Systems*, 1st ed. Springer Vienna, Austria, 2007. doi: 10.1007/978-3-211-73017-1

[35] F. S. T. Van Diggelen, *A-GPS: Assisted GPS, GNSS, and SBAS*, 1st ed. Norwood, MA, USA: Artech House, Inc., March 2009. ISBN 9781596933743

[36] F. Bastide, D. Akos, C. Macabiau, and B. Roturier, "Automatic Gain Control (AGC) as an Interference Assessment Tool," in *Proceedings of the 16th International Technical Meeting of the Satellite Division of The Institute of Navigation (ION GPS/GNSS 2003)*, Portland, OR, USA, September 2003, pp. 2042–2053.

[37] D. Gómez-Casco, "Non-Coherent Acquisition Techniques for High-Sensitivity GNSS Receivers," Ph.D. dissertation, Universitat Autònoma de Barcelona, Barcelona, Spain. ISBN 9788449083198 November 2018. [Online]. Available: https://www.tdx.cat/handle/10803/665404#page=1 (Accessed March 19, 2024).

[38] K. Borre, D. Akos, N. Bertelsen, P. Rinder, and S. Jensen, *A Software-Defined GPS and Galileo Receiver: A Single-Frequency Approach*, 1st ed. Birkhäuser Boston, MA, USA, 2007. doi: 10.1007/978-0-8176-4540-3

[39] C. Fernández-Prades, J. Arribas, L. Esteve, D. Pubill, and P. Closas, "An open source Galileo E1 software receiver," in *Proceedings of the 2012 6th ESA Workshop on Satellite Navigation Technologies (NAVITEC 2012) & European Workshop on GNSS Signals and Signal Processing*, Noordwijk, The Netherlands, December 2012, pp. 1–8. doi: 10.1109/NAVITEC.2012.6423057

[40] B. N. Vu and M. Andrle, "The code and carrier tracking loops for GPS signal," in *Proceedings of the 16th International Conference on Mechatronics - Mechatronika 2014*, Brno, Czech Republic, December 2014, pp. 569–574. doi: 10.1109/MECHATRONIKA.2014.7018322

[41] "GNU Radio, a Free & Open-Source Software Development Toolkit." [Online]. Available: http://gnuradio.org/ (Accessed March 19, 2024).

[42] G. Kahn and D. B. MacQueen, "Coroutines and networks of parallel processes," in *Information Processing, Proceedings of the 7th IFIP Congress 1977*, B. Gilchrist, Ed. Toronto, Canada: North-Holland, August 1977, pp. 993–998.

[43] *OGC KML 2.3 Standard*, Open Geospatial Consortium, Arlington, VA, USA, August 2015, OGC 12-007r2. [Online]. Available: https://docs.ogc.org/is/12-007r2/12-007r2.html (Accessed March 19, 2024).

[44] H. Butler, M. Daly, A. Doyle, S. Gillies, T. Schaub, and S. Hagen, "The GeoJSON Format," August 2016. doi: 10.17487/RFC7946. [Online]. Available: https://www.rfc-editor.org/info/rfc7946 (Accessed March 19, 2024).

[45] *NMEA 0183 Standard*, National Marine Electronics Association, Severna Park, MD, USA, Novermber 2018, v4.11.

[46] *GPX 1.1 Schema Documentation*, Topografix, Stow, MA, USA, August 2004. [Online]. Available: https://www.topografix.com/GPX/1/1/ (Accessed March 19, 2024).

[47] *RTCM 10403.4, Differential GNSS (Global Navigation Satellite Systems) Services*, Radio Technical Commission for Maritime Services, Washington DC, USA, December 2023, version 3.

[48] *The Receiver Independent Exchange Format*, International GNSS Service (IGS), December 2021, Version 4.00. [Online]. Available: https://files.igs.org/pub/data/format/rinex_4.00.pdf (Accessed March 19, 2024).

[49] D. Kusswurm, "X86-64 Core Architecture," in *Modern X86 Assembly Language Programming*, 2nd ed. Berkeley, CA: Apress, 2018, pp. 491–502. doi: 10.1007/978-1-4842-0064-3_17

[50] K. C. Wang, "ARM Architecture and Programming," in *Embedded and Real-Time Operating Systems* , 1st ed. Springer International Publishing: Cham, Switzerland, 2017, pp. 7–46. doi: 10.1007/978-3-319-51517-5_2

[51] O. Salvador and D. Angolini, *Embedded Linux Development Using Yocto Projects - Second Edition: Learn to Leverage the Power of Yocto Project to Build Efficient Linux-Based Products*, 2nd ed. Birmingham, UK: Packt Publishing, November 2017. ISBN 9781788470469

[52] C. Fernández-Prades, "meta-gnss-sdr." [Online]. Available: https://github.com/carlesfernandez/meta-gnss-sdr (Accessed March 19, 2024).

[53] C. Fernández-Prades, P. Balister, and T. Flynn, "oe-gnss-sdr-manifest," February 2024. doi: 10.5281/zenodo.10653162. [Online]. Available: https://github.com/carlesfernandez/oe-gnss-sdr-manifest (Accessed March 19, 2024).

[54] "The Multiple Git Repository Tool." [Online]. Available: https://gerrit.googlesource.com/git-repo/ (Accessed March 19, 2024).

[55] C. Fernández-Prades, "yocto-geniux v2.0," August 2021. doi: 10.5281/zenodo.5244379. [Online]. Available: https://github.com/carlesfernandez/yocto-geniux (Accessed March 19, 2024).

[56] C. Fernández-Prades, "docker-petalinux2 v1.0," May 2021. doi: 10.5281/zenodo.4744322. [Online]. Available: https://github.com/carlesfernandez/docker-petalinux2 (Accessed March 19, 2024).

[57] C. Maxfield, *The Design Warrior's Guide to FPGAs: Devices, Tools and Flows*, 1st ed. USA: Newnes, 2004. ISBN 0750676043

[58] "IEEE Standard for IP-XACT, Standard Structure for Packaging, Integrating, and Reusing IP within Tool Flows," *IEEE Std 1685-2022 (Revision of IEEE Std 1685-2014)*, pp. 1–750, February 2023. doi: 10.1109/IEEESTD.2023.10054520

[59] S. Kilts, *Advanced FPGA Design: Architecture, Implementation, and Optimization*, 1st ed. Hoboken, NJ, USA: John Wiley & Sons, 2007. ISBN 9780470054376

[60] *Vivado Design Suite User Guide: Design Flows Overview*, Advanced Micro Devices Inc., Santa Clara, CA, USA, October 2023, UG892 (v2023.2).

[61] *AMBA AXI Protocol Specification*, ARM Limited, Cambridge, UK, September 2023, ARM IHI 0022 (Issue K).

[62] L. H. Crockett, R. A. Elliot, M. A. Enderwitz, and R. W. Stewart, *The Zynq Book: Embedded Processing with the Arm Cortex-A9 on the Xilinx Zynq-7000 All Programmable SoC*. Glasgow, UK: Strathclyde Academic Media, 2014. ISBN 9780992978709

[63] *Zynq Ultrascale+ Device Technical Reference Manual*, Advanced Micro Devices Inc., Santa Clara, CA, USA, January 2023, UG1085 (v2.3.1).

[64] C. Fernández-Prades, J. Arribas, and P. Closas, "Assessment of software-defined GNSS receivers," in *Proceedings of the 2016 8th ESA Workshop on Satellite Navigation Technologies and European Workshop on GNSS Signals and Signal Processing (NAVITEC)*, Noordwijk, The Netherlands, December 2016, pp. 1–9. doi: 10.1109/NAVITEC.2016.7931740

[65] *GPS Position Accuracy Measures*, NovAtel Inc., Calgary, AB, Canada, December 2003, APN-029 (Rev. 1).

[66] C. Fernández-Prades, J. Arribas, and P. Closas, "Turning a Television Into a GNSS Receiver," in *Proceedings of the 26th International Technical Meeting of the Satellite Division of The Institute of Navigation (ION GNSS+ 2013)*, Nashville, TN, USA, September 2013, pp. 1492–1507.

[67] S. True, "Planning the future of the World Geodetic System 1984," in *PLANS 2004. Position Location and Navigation Symposium (IEEE Cat. No.04CH37556)*, Monterey, CA, USA, April 2004, pp. 639–648. doi: 10.1109/PLANS.2004.1309054

[68] D. Doberstein, *Fundamentals of GPS Receivers: A Hardware Approach*, 1st ed. Springer New York, NY, 2012. doi: 10.1007/978-1-4614-0409-5

[69] Y. Lu, *BDS/GPS Dual-Mode Software Receiver: Principles and Implementation Technology*, 1st ed. Springer Singapore, 2021. doi: 10.1007/978-981-16-1075-2

[70] K. Borre, I. Fernández-Hernández, J. A. López-Salcedo, and M. Z. H. Bhuiyan, *GNSS Software Receivers*, 1st ed. Cambridge, UK: Cambridge University Press, 2022. doi: 10.1017/9781108934176

[71] K. Krumvieda, P. Madhani, C. Cloman, E. Olson, J. Thomas, P. Axelrad, and W. Kober, "A Complete IF Software GPS Receiver: A Tutorial about the Details," in *Proceedings of the 14th International Technical Meeting of the Satellite Division of The Institute of Navigation (ION GPS 2001)*, Salt Lake City, UT, USA, September 2001, pp. 789–829.

[72] V. Chakravarthy, J. Tsui, D. Lin, and J. Schamus, "Software GPS Receiver," *GPS Solutions*, vol. 5, pp. 63–70, October 2001. doi: 10.1007/PL00012887

[73] F. Principe, G. Bacci, F. Giannetti, and M. Luise, “Software-Defined Radio Technologies for GNSS Receivers: A Tutorial Approach to a Simple Design and Implementation,” *International Journal of Navigation and Observation*, vol. 2011, May 2011. doi: 10.1155/2011/979815

[74] T. Pany, D. Akos, J. Arribas, M. Z. H. Bhuiyan, P. Closas, F. Dovis, I. Fernández-Hernández, C. Fernández–Prades, S. Gunawardena, T. Humphreys, Z. M. Kassas, J. A. L. Salcedo, M. Nicola, M. L. Psiaki, A. Rügamer, Y.-J. Song, and J.-H. Won, “GNSS Software-Defined Radio: History, Current Developments, and Standardization Efforts,” *NAVIGATION: Journal of the Institute of Navigation*, vol. 71, no. 1, March 2024. doi: 10.33012/navi.628

[75] D. M. Akos, “A Software Radio Approach to Global Navigation Satellites System Receiver Design,” PhD thesis, Ohio University, OH, USA, 1997. [Online]. Available: https://etd.ohiolink.edu/acprod/odb_etd/etd/r/1501/10?clear=10&p10_accession_num=ohiou1174615606 (Accessed March 19, 2024).

[76] B. Ledvina, S. Powell, K. P.M., and M. Psiaki, “A 12–Channel Real–Time GPS L1 Software Receiver,” in *Proceedings of the 2003 National Technical Meeting of The Institute of Navigation*, Anaheim, CA, January 2003, pp. 767–782.

[77] M. Fantino, A. Molino, and M. Nicola, “N-Gene GNSS Receiver: Benefits of Software Radio in Navigation,” in *Proceedings of the The European Navigation Conference (ENC 2009)*, Napoli, Italia, May 2009.

[78] C. Stöber, M. Anghileri, A. Sicramaz Ayaz, D. Dötterböck, I. Krämer, V. Kropp, J.-H. Won, B. Eissfeller, D. S. Güixens, and T. Pany, “ipexSR: A real-time multi-frequency software GNSS receiver,” in *Proceedings ELMAR-2010*, Zadar, Croatia, September 2010, pp. 407–416.

[79] M. T. Gamba, M. Nicola, and E. Falletti, “Performance assessment of an ARM-based dual-constellation GNSS software receiver,” in *Proceedings of the 2015 International Conference on Localization and GNSS (ICL-GNSS)*, Gothenburg, Sweden, June 2015, pp. 1–6. doi: 10.1109/ICL-GNSS.2015.7217143

[80] Y. Pei, H. Chen, and B. Pei, “Implementation of GPS Software Receiver Based on GNU Radio,” in *Proceedings of the 2018 Cross Strait Quad-Regional Radio Science and Wireless Technology Conference (CSQRWC)*, Xuzhou, China, July 2018, pp. 1–3. doi: 10.1109/CSQRWC.2018.8455475

[81] M. Petovello, C. O’Driscoll, G. Lachapelle, D. Borio, and H. Murtaza, “Architecture and Benefits of an Advanced GNSS Software Receiver,” *Journal of Global Positioning Systems*, vol. 7, pp. 156–168, December 2008. doi: 10.5081/jgps.7.2.156

[82] T. Hobiger, T. Gotoh, J. Amagai, Y. Koyama, and T. Kondo, “A GPU based real-time GPS software receiver,” *GPS Solutions*, vol. 14, pp. 207–216, March 2010. doi: 10.1007/s10291-009-0135-2

[83] B. W. Tolman, R. B. Harris, T. Gaussiran, D. Munton, L. Jon, R. Mach, S. Nelsen, B. Renfro, and D. Schlossberg, “The GPS Toolkit … Open Source GPS Software,” in *Proceedings of the 17th International Technical Meeting of the Satellite Division of The*

Institute of Navigation (ION GNSS 2004), Long Beach, CA, USA, September 2004, pp. 2044 – 2053.

[84] P. Kovar and F. Vejrazka, "Software radio and its applications in GNSS," in *Proceedings of the Elmar-2004. 46th International Symposium on Electronics in Marine*, Zadar, Croatia, June 2004, pp. 16–21.

[85] A. Soghoyan, A. Suleiman, and D. Akopian, "A Development and Testing Instrumentation for GPS Software Defined Radio With Fast FPGA Prototyping Support," *IEEE Transactions on Instrumentation and Measurement*, vol. 63, no. 8, pp. 2001 – 2012, 2014. doi: 10.1109/TIM.2014.2304352

[86] F. Garzia, A. Rügamer, R. Koch, P. Neumaier, E. Serezhkina, M. Overbeck, and G. Rohmer, "Experimental multi-FPGA GNSS receiver platform," in *Proceedings of the 2014 24th International Conference on Field Programmable Logic and Applications (FPL)*, Munich, Germany, September 2014, pp. 1–4. doi: 10.1109/FPL.2014.6927399

[87] F. S. Larosa, "Design, Simulation, Implementation and Testing of Search and Tracking Modules for a FPGA-Based GPS Receiver," in *Proceedings of the 2019 X Southern Conference on Programmable Logic (SPL)*, Buenos Aires, Argentina, April 2019, pp. 33–38. doi: 10.1109/SPL.2019.8714299

[88] V. Keshihaa Rudra Gana Dev and G. Ranjani, "An Optimized FPGA Architecture for GPS Signal Acquisition," in *Proceedings of the 2022 IEEE 3rd Global Conference for Advancement in Technology (GCAT)*, Bangalore, India, October 2022, pp. 1–6. doi: 10.1109/GCAT55367.2022.9972099

[89] M. Spelat, F. Dovis, G. Girau, and P. Mulassano, "A Flexible FPGA/DSP Board for GNSS Receivers Design," in *Proceedings of the 2006 Ph.D. Research in Microelectronics and Electronics*, Otranto, Italy, June 2006, pp. 77–80. doi: 10.1109/RME.2006.1689900

[90] M. Grondin, M. Belasic, L. Ries, J.-L. Issler, P. Bataille, L. Jobey, and G. Richard, "A new operational low cost GNSS software receiver for microsatellites," in *Proceedings of the 2010 5th ESA Workshop on Satellite Navigation Technologies and European Workshop on GNSS Signals and Signal Processing (NAVITEC)*, Noordwijk, The Netherlands, December 2010, pp. 1–5. doi: 10.1109/NAVITEC.2010.5707991

[91] A. Avanzi, P. Tortora, and A. Garcia-Rodriguez, "Design and Implementation of a Novel Multi-constellation FPGA-based Dual Frequency GNSS Receiver for Space Applications," in *Proceedings of the 24th International Technical Meeting of the Satellite Division of The Institute of Navigation (ION GNSS 2011)*, Portland, OR, USA, September 2011, pp. 746–752.

[92] S. Fantinato, L. Foglia, P. Iacone, D. Rovelli, C. Facchinetti, and A. Tuozzi, "PEGASUS –GNSS receiver platform for safety of life user segment," in *Proceedings of the 2012 6th ESA Workshop on Satellite Navigation Technologies (NAVITEC 2012) & European Workshop on GNSS Signals and Signal Processing*, Noordwijk, The Netherlands, December 2012, pp. 1–8. doi: 10.1109/NAVITEC.2012.6423062

[93] G. Kappen, C. Haettich, and M. Meurer, "Towards a robust multi-antenna mass market GNSS receiver," in *Proceedings of the 2012 IEEE/ION Position, Location and Navigation Symposium*, Myrtle Beach, SC, USA, April 2012, pp. 291–300. doi: 10.1109/PLANS.2012.6236894

[94] B.-S. Wang, S.-W. Iv, D.-K. Yang, and Q.-S. Zhang, "A New Solution to GNSS Receiver Baseband Signal Processing SOC Platform Based on OpenRISC Processor," in *Proceedings of the 2011 7th International Conference on Wireless Communications, Networking and Mobile Computing*, Wuhan, China, September 2011, pp. 1–4. doi: 10.1109/wicom.2011.6040266

[95] J. Raasakka, H. Hurskainen, and J. Nurmi, "GNSS baseband processing in a multi-core platform," in *Proceedings of the 2011 International Conference on Localization and GNSS (ICL-GNSS)*, Tampere, Finland, June 2011, pp. 42–46. doi: 10.1109/ICL-GNSS.2011.5955263

[96] A. Fridman and S. Semenov, "System-on-Chip FPGA-based GNSS receiver," in *Proceedings of the East-West Design & Test Symposium (EWDTS 2013)*, Rostov on Don, Russia, September 2013, pp. 1–7. doi: 10.1109/EWDTS.2013.6673192

[97] J. Buttgereit, T. Schwarte, and G. C. Kappen, "Design and Implementation of a Software Defined Radio GNSS Receiver Based on OpenCL," in *Proceedings of the 2020 IEEE/ION Position, Location and Navigation Symposium (PLANS)*, Portland, OR, USA, April 2020, pp. 1237–1246. doi: 10.1109/PLANS46316.2020.9110191

[98] B. Priot, A. Dion, G. Beaugendre, and R. Kasaraneni, "Accurate Events Synchronization in a System-on-Chip Navigation Receiver," in *Proceedings of the International Conference on Localization and GNSS*, Nuremberg, Germany, June 2019, pp. 1–5.

[99] K.-Y. Huang, J.-C. Juang, Y.-F. Tsai, and C.-T. Lin, "Efficient FPGA Implementation of a Dual-Frequency GNSS Receiver with Robust Inter-Frequency Aiding," *Sensors*, vol. 21, no. 14, 2021, Art. no. 4634. doi: 10.3390/s21144634

[100] S. Guruprasad, "FPGA-Based GNSS Receiver Design for Reflectometry Applications," Ph.D. dissertation, York University, Toronto, ON, Canada, October 2022. [Online]. Available: https://yorkspace.library.yorku.ca/items/6a6ea19f-f261-4c48-a1c2-c595f6acc5ba (Accessed March 19, 2024).

[101] A. Grenier, E. S. Lohan, A. Ometov, and J. Nurmi, "A Survey on Low-Power GNSS," *IEEE Communications Surveys & Tutorials*, vol. 25, no. 3, pp. 1482–1509, 2023. doi: 10.1109/COMST.2023.3265841

[102] *Zynq-7000 All Programmable SoC Technical Reference Manual*, Advanced Micro Devices Inc., Santa Clara, CA, USA, June 2023, UG585 (v1.14).

[103] *Zynq-7000 SoC Product Selection Guide*, Advanced Micro Devices Inc., Santa Clara, CA, USA, 2019, XMP097 (v1.3.2).

[104] *Zynq Ultrascale+ MPSoC Product Tables and Product Selection Guide*, Advanced Micro Devices Inc., Santa Clara, CA, USA, 2022, XMP104 (v2.6).

[105] J. Arribas, “GNSS Array-Based Acquisition: Theory and Implementation,” Ph.D. dissertation, Universitat Politecnica de Catalunya (UPC), Barcelona, Spain, June 2012. [Online]. Available: https://www.tdx.cat/handle/10803/125031#page=1 (Accessed March 19, 2024).

[106] *AD9361 Reference Manual*, Analog Devices Inc., Wilmington, MA, USA, 2015, UG-570 (Rev. A).

[107] *PetaLinux Tools Documentation Reference Guide*, Advanced Micro Devices Inc., Santa Clara, CA, USA, May 2023, UG1399 (v2023.1).

[108] J. Turnbull, *The Docker Book: Containerization Is the New Virtualization.* Navi Mumbai, India: Shroff Publishers, 2019. ISBN 9789352138913

[109] *Vivado Design Suite User Guide: Release Notes, Installation, and Licensing*, Advanced Micro Devices Inc., Santa Clara, CA, USA, May 2022, UG973 (v2022.2).

[110] *Vitis High-Level Synthesis User Guide*, Advanced Micro Devices Inc., Santa Clara, CA, USA, July 2023, UG1399 (v2023.1).

[111] “AMD University Program,” Advanced Micro Devices, Inc. [Online]. Available: https://www.amd.com/en/corporate/university-program.html/#faq (Accessed March 19, 2024).

[112] “HDL Reference Designs,” Analog Devices Inc. [Online]. Available: https://wiki.analog.com/resources/fpga/docs/arch (Accessed March 19, 2024).

[113] S. Venkateswaran, *Essential Linux Device Drivers*, 1st ed. Prentice Hall, Upper Saddle River, NJ, USA, 2008. ISBN 9780132396554

[114] C. Fernández-Prades, C. Avilés, L. Estove, J. Arribas, and P. Closas, “Design patterns for GNSS software receivers,” in *Proceedings of the 2010 5th ESA Workshop on Satellite Navigation Technologies and European Workshop on GNSS Signals and Signal Processing (NAVITEC)*, Noordwijk, The Netherlands, December 2010, pp. 1–8. doi: 10.1109/NAVITEC.2010.5707981

[115] H. Myeong, *Googletest in Practice: Unit Testing Guide for C++ Programmers*. Independently Published, 2021. ISBN 9798767335619

[116] “Google logging library (google-glog). C++ implementation of the Google logging module.” [Online]. Available: https://github.com/google/glog (Accessed March 19, 2024).

[117] C. Fernández-Prades, J. Vilà-Valls, J. Arribas, and A. Ramos, “Continuous Reproducibility in GNSS Signal Processing,” *IEEE Access*, vol. 6, pp. 20 451–20 463, 2018. doi: 10.1109/ACCESS.2018.2822835

[118] G. K. Adam, N. Petrellis, and L. T. Doulos, “Performance Assessment of Linux Kernels with PREEMPT_RT on ARM-Based Embedded Devices,” *Electronics*, vol. 10, no. 11, 2021, Art. no. 1331. doi: 10.3390/electronics10111331

[119] Analog Devices Inc., “AD-FMCOMMS5-EBZ User Guide.” [Online]. Available: https://wiki.analog.com/resources/eval/user-guides/ad-fmcomms5-ebz (Accessed March 19, 2024).

[120] J. Roselló, P. Silvestrin, G. L. Risueño, R. Weigand, J. V. Perelló, J. Heim, and I. Tejerina, “AGGA-4: Core device for GNSS space receivers of this decade,” in *Proceedings of the 2010 5th ESA Workshop on Satellite Navigation Technologies and European Workshop on GNSS Signals and Signal Processing (NAVITEC)*, Noordwijk, The Netherlands, December 2010, pp. 1–8. doi: 10.1109/NAVITEC.2010.5708068

[121] J. Roselló, P. Silvestrin, R. Weigand, S. d’Addio, A. G. Rodríguez, and G. L. Risueño, “Next generation of ESA’s GNSS receivers for Earth Observation satellites,” in *Proceedings of the 2012 6th ESA Workshop on Satellite Navigation Technologies (NAVITEC 2012) & European Workshop on GNSS Signals and Signal Processing*, Noordwijk, The Netherlands, December 2012, pp. 1–8. doi: 10.1109/NAVITEC.2012.6423104

[122] *AGGA-4 User Manual*, Airbus DS GmbH, November 2018, AGGA-ADSO-UM-1000485489 (Issue 1). [Online]. Available: https://ww1.microchip.com/downloads/aemDocuments/documents/AERO/ProductDocuments/DataSheets/AGGA4_User_Manual_1.pdf (Accessed March 19, 2024).

[123] M. Schütz, S. Zehetmayer, K. Zajac, J. von Borany, M. Laabs, F. Zangerl, R. Zangl, D. Plettemeier, and M. Sust, “Spaceborne GNSS-Receiver Heritage Leveraged on New Space,” in *Proceedings of the 2020 European Navigation Conference (ENC)*, Dresden, Germany, November 2020, pp. 1–10. doi: 10.23919/ENC48637.2020.9317497

[124] N. Ibellaatti, E. Lepape, A. Kilic, K. Akyel, K. Chouayakh, F. Ferrandi, C. Barone, S. Curzel, M. Fiorito, G. Gozzi, M. Masmano, A. R. Navarro, M. Muñioz, V. N. Gallego, P. L. Cueva, J.-N. Letrillard, and F. Wartel, “HERMES: qualification of High pErformance pRogrammable Microprocessor and dEvelopment of Software ecosystem,” in *2023 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, Antwerp, Belgium, April 2023, pp. 1–5. doi: 10.23919/DATE56975.2023.10136921

[125] M. Amrbar, F. Irom, S. M. Guertin, and G. Allen, “Heavy Ion Single Event Effects Measurements of Xilinx Zynq-7000 FPGA,” in *Proceedings of the 2015 IEEE Radiation Effects Data Workshop (REDW)*, Boston, Massachusetts, USA, July 2015, pp. 1–4. doi: 10.1109/REDW.2015.7336714

[126] C. Spindeldreier, B. Ustaoglu, U. Kulau, and J. Rust, “Performance Evaluation of Space-Grade FPGA Architectures,” in *Proceedings of the 2023 European Data Handling & Data Processing Conference (EDHPC)*, Juan-Les-Pins, France, October 2023, pp. 1–5. doi: 10.23919/EDHPC59100.2023.10396163

[127] *Radiation-Hardened, Space-Grade Virtex-5QV FPGA Data Sheet: DC and AC Switching Characteristics*, Advanced Micro Devices Inc., Santa Clara, CA, USA, January 2015, DS692 (v1.3.1).

[128] *Radiation Tolerant Kintex UltraScale XQRKU060 FPGA Data Sheet*, Advanced Micro Devices Inc., Santa Clara, CA, USA, April 2022, DS882 (v1.3).

[129] *Radiation-Tolerant FPGAs*, Microchip Technology Inc., 2355 W. Chandler Blvd. Chandler AZ, 85224-6199, 2022, DS00003023D.

[130] E. Danard and A. Comolet-Tirman, “OBC-Ultra, the rad-hard NG-Ultra-based On Board Computer for future applications,” in *Proceedings of the 2023 European Data Handling and Data Processing Conference (EDHPC)*, Juan-Les-Pins, France, October 2023, pp. 1–4. doi: 10.23919/EDHPC59100.2023.10396378

[131] A. Grillenberger, R. Rivas, M. Markgraf, P. Mumford, K. Parkinson, and C. Rizos, “The Namuru Receiver as Development Platform for Spaceborne GNSS Applications,” in *Proceedings of the 2008 4th ESA Workshop on Satellite Navigation User Equipment Technologies (NAVITEC 2008)*, December 2008.

[132] T. Grelier, L. Ries, P. Bataille, C. Perrot, and G. Richard, “A new operational low cost GNSS Software receiver for Microsatellites,” in *Proceedings of the 2012 6th ESA Workshop on Satellite Navigation Technologies (NAVITEC 2012) & European Workshop on GNSS Signals and Signal Processing*, Noordwijk, The Netherlands, December 2012, pp. 1–5. doi: 10.1109/NAVITEC.2012.6423051

[133] V. VS, A. K. Thampi, B. VS, A. S. OT, and C. Radhakrishna Pillai, “Design of a Dual Processor Based Single Board GNSS Receiver Embedded Navigation Computer for Low Cost Launch Vehicles,” in *Proceedings of the 2022 IEEE International Conference on Signal Processing, Informatics, Communication and Energy Systems (SPICES)*, vol. 1, Thiruvananthapuram, India, March 2022, pp. 33–37. doi: 10.1109/SPICES52834.2022.9774188

[134] J. M. Palomo Gutiérrez, P. D’angelo, P. Silva, A. Fernández, P. Giordano, P. Zoccarato, J. Tegedor, O. Oerpen, L. Hansen, C. Hill, and T. Moore, “Space GNSS Receiver Performance Results With Precise Real-Time On-board Orbit Determination (P2OD) in LEO Missions,” in *Proceedings of the 32nd International Technical Meeting of the Satellite Division of The Institute of Navigation (ION GNSS+ 2019)*, Miami, FL, USA, October 2019, pp. 1172–1186. doi: 10.33012/2019.17082

[135] Z. Chenggong, C. Xi, and H. Zhen, “A comprehensive analysis on Doppler frequency and Doppler frequency rate characterization for GNSS receivers,” in *Proceedings of the 2016 2nd IEEE International Conference on Computer and Communications (ICCC)*, Chengdu, China, October 2016, pp. 2606–2610. doi: 10.1109/CompComm.2016.7925169

[136] M. Kelley, *The Earth’s Ionosphere: Plasma Physics and Electrodynamics.* Academic Press, 1989. doi: 10.1016/B978-0-12-404013-7.X5001-1

[137] K. Xi and X. Wang, “Higher order ionospheric error correction in BDS precise orbit determination,” *Advances in Space Research*, vol. 67, no. 12, pp. 4054–4065, June 2021. doi: 10.1016/j.asr.2021.02.002

[138] H. Abdulkader, D. Roviras, R. Chaggara, W. Vigneau, and F. Castanie, “Mitigation of Multipath Influence on Tracking Errors in LEO Navigation Applications,” in *The 2006 IEEE International Joint Conference on Neural Network Proceedings*, Vancouver, BC, Canada, July 2006, pp. 2673–2678. doi: 10.1109/IJCNN.2006.247148

[139] H. Hurskainen, E.-S. Lohan, J. Nurmi, S. Sand, C. Mensing, and M. Detratti, “Optimal dual frequency combination for Galileo mass market receiver baseband,” in *Proceedings of the 2009 IEEE Workshop on Signal Processing Systems*, Tampere, Finland, October 2009, pp. 261–266. doi: 10.1109/SIPS.2009.5336262

[140] Analog Devices Inc., “ADRV1CRR-BOB Breakout Carrier.” [Online]. Available: https://wiki.analog.com/resources/eval/user-guides/pzsdr/carriers/brk (Accessed March 19, 2024).

[141] *SMD Temperatured Compensated Crystal Oscillators 3.2×2.5×1.0 mm 7Q Series*, TXC Corporation. [Online]. Available: http://txccrystal.com/images/pdf/7q-gps.pdf (Accessed March 19, 2024).

[142] *AXI Interrupt Controller (INTC) v4.1 LogiCORE IP Product Guide*, Advanced Micro Devices Inc, Santa Clara, CA, USA, May 2017, PG099.

[143] *Abracon AST3TQ-T-40.000MHZ-50-C Datasheet*, Abracon LLC, Austin, TX, USA, 2015, Rev. 04.02.15.

[144] N. Sokolova, D. Borio, B. Forssell, and G. Lachapelle, “Doppler rate measurements in standard and high sensitivity GPS receivers: Theoretical analysis and comparison,” in *Proceedings of the 2010 International Conference on Indoor Positioning and Indoor Navigation*, Zurich, Switzerland, November 2010, pp. 1–9. doi: 10.1109/IPIN.2010.5647595

[145] D. Borio, N. Sokolova, and G. Lachapelle, “Doppler measurement accuracy in standard and high–sensitivity global navigation satellite system receivers,” *IET Radar, Sonar & Navigation*, vol. 5, no. 6, pp. 657–665, July 2011. doi: 10.1049/iet-rsn.2010.0249

[146] J. Arribas, C. Fernández-Prades, and P. Closas, “GESTALT: A Testbed For Experimentation And Validation of GNSS Software Receivers,” in *Proceedings of the 28th International Technical Meeting of the Satellite Division of The Institute of Navigation (ION GNSS+ 2015)*, Tampa, FL, USA, September 2015, pp. 3222–3234.

[147] O. Montenbruck, A. Hauschild, R. B. Langley, and C. Siemes, “CASSIOPE orbit and attitude determination using commercial off-the-shelf GPS receivers,” *GPS Solutions*, vol. 23, no. 4, 2019, Art. no. 114. doi: 10.1007/s10291-019-0907-2

[148] A. Dion, V. Calmettes, M. Bousquet, and E. Boutillon, “Performances of a GNSS receiver for space-based applications,” in *Proceedings of Toulouse Space Show 2010*, Marseille, France, June 2010.

[149] “GNSS-SIMULATOR, A modular multichannel GNSS simulator written in C++.” [Online]. Available: https://bitbucket.org/jarribas/gnss-simulator/src/master/ (Accessed March 19, 2024).

[150] A. Van Dierendonck, P. Fenton, and T. Ford, “Theory and Performance of Narrow Correlator Spacing in a GPS Receiver,” *NAVIGATION: Journal of The Institute of Navigation*, vol. 39, no. 3, pp. 265–284, Fall 1992.

[151] J. Saastamoinen, “Contributions to the theory of atmospheric refraction,” *Bulletin Géodésique (1946-1975)*, vol. 105, no. 1, pp. 279–298, 1972. doi: 10.1007/BF02521844

[152] J. Magiera, “Design and implementation of GPS signal simulator,” in *Proceedings of the 2012 International Conference on Localization and GNSS*, Starnberg, Germany, June 2012, pp. 1–4. doi: 10.1109/ICL-GNSS.2012.6253117

[153] T. T.-T. Nguyen, B. Motella, D. Margaria, and T. H. Ta, “Design and validation of a flexible software-based generator of realistic GNSS signals,” in *Proceedings of the 2017 European Navigation Conference (ENC)*, Lausanne, Switzerland, May 2017, pp. 128–134. doi: 10.1109/EURONAV.2017.7954201

[154] Z. Bo, L. Guang-bin, L. Dong, and F. Zhi-liang, “Real-time software GNSS signal simulator accelerated by CUDA,” in *2010 2nd International Conference on Future Computer and Communication*, vol. 1, Wuhan, China, May 2010, pp. V1–100–V1–104. doi: 10.1109/ICFCC.2010.5497829

[155] O. Pozzobon, C. Sarto, A. Dalla Chiara, A. Pozzobon, G. Gamba, M. Crisci, and R. Ioannides, “Developing a GNSS Position and Timing Authenticaion Testbed,” *Inside GNSS*, vol. 8, pp. 45–53, January 2013.

[156] T. Humphreys, J. Bhatti, D. Shepard, and K. Wesson, “A Testbed for Developing and Evaluating GNSS Signal Authentication Techniques,” in *Proceedings of the International Symposium on Certification of GNSS Systems & Services (CERGAL)*, Dresden, Germany, July 2014, pp. 1–15. [Online]. Available: https://radionavlab.ae.utexas.edu/images/stories/files/papers/tb.pdf (Accessed March 19, 2024).

[157] T. Humphreys, B. Ledvina, M. Psiaki, B. O’Hanlon, and J. Kintner, “Assessing the Spoofing Threat: Development of a Portable GPS Civilian Spoofer,” in *Proceedings of the 21st International Technical Meeting of the Satellite Division of the Institute of Navigation*, Savannah, GA, USA, September 2008, pp. 2314–2325.

[158] W. Kim and J. Seo, “Low-Cost GNSS Simulators With Wireless Clock Synchronization for Indoor Positioning,” *IEEE Access*, vol. 11, pp. 55 861–55 874, 2023. doi: 10.1109/ACCESS.2023.3282786

[159] Analog Devices Inc., “ADRV936x System on Module (SOM) SDR User Guide.” [Online]. Available: https://wiki.analog.com/resources/eval/user-guides/adrv936x_rfsom (Accessed March 19, 2024).

[160] E. S. Lohan, “Limited Bandwidths and Correlation Ambiguities: Do They Co-Exist in Galileo Receivers,” *Positioning*, vol. 2, no. 1, pp. 14–21, February 2011. doi: 10.4236/pos.2011.21002

[161] *Product Summary NEO/LEA-M8T series*, U-blox Holding AG, Thalwil, Switzerland, 2013, UBX-16000801-R06.

[162] T. Ren and M. G. Petovello, “A Stand-Alone Approach for High-Sensitivity GNSS Receivers in Signal-Challenged Environment,” *IEEE Transactions on Aerospace and Electronic Systems*, vol. 53, no. 5, pp. 2438–2448, 2017. doi: 10.1109/TAES.2017.2699539

[163] T. Marathe, A. Broumandan, A. Pirsiavash, and G. Lachapelle, “Characterization of Range and Time Performance of Indoor GNSS Signals,” in *Proceedings of the 2018*

European Navigation Conference (ENC), Gothenburg, Sweden, May 2018, pp. 27–37. doi: 10.1109/EURONAV.2018.8433236

[164] V. Renaudin, O. Yalak, and P. Tomé, "Hybridization of MEMS and Assisted GPS for Pedestrian Navigation," *Inside GNSS*, pp. 34–42, January 2007.

[165] S.-H. Kong, "High Sensitivity and Fast Acquisition Signal Processing Techniques for GNSS Receivers: From fundamentals to state-of-the-art GNSS acquisition technologies," *IEEE Signal Processing Magazine*, vol. 34, no. 5, pp. 59–71, 2017. doi: 10.1109/MSP.2017.2714201

[166] D. Jiménez-Baños, N. Blanco-Delgado, G. López-Risueño, G. Seco-Granados, and A. Garcia-Rodríguez, "Innovative Techniques for GPS Indoor Positioning Using a Snapshot Receiver," in *Proceedings of the 19th International Technical Meeting of the Satellite Division of The Institute of Navigation (ION GNSS 2006)*, Fort Worth, TX, USA, September 2006, pp. 2944–2955.

[167] V. Lucas Sabola, G. Seco Granados, J. Lopez Salcedo, and J. García-Molina, "GNSS IoT Positioning: From Conventional Sensors to a Cloud-Based Solution," *Inside GNSS*, pp. 53–62, June 2018.

[168] European GNSS Agency, *GNSS user technology report. Issue 1.* Luxembourg: Publications Office of the European Union, 2016. doi: doi/10.2878/760803

[169] E. Domínguez, A. Pousinho, P. Boto, D. Gómez-Casco, S. Locubiche-Serra, G. Seco-Granados, J. A. López-Salcedo, H. Fragner, F. Zangerl, O. Peña, and D. Jiménez-Baños, "Performance Evaluation of High Sensitivity GNSS Techniques in Indoor, Urban and Space Environments," in *Proceedings of the 29th International Technical Meeting of the Satellite Division of The Institute of Navigation (ION GNSS+ 2016)*, Portland, OR, USA, September 2016, pp. 373–393. doi: 10.33012/2016.14677

[170] D. Gómez-Casco, J. A. López-Salcedo, and G. Seco-Granados, "Generalized integration techniques for high-sensitivity GNSS receivers affected by oscillator phase noise," in *Proceedings of the 2016 IEEE Statistical Signal Processing Workshop (SSP)*, Palma de Mallorca, Spain, June 2016, pp. 1–5. doi: 10.1109/SSP.2016.7551809

[171] D. Gómez-Casco, J. A. López-Salcedo, and G. Seco-Granados, "Optimal fractional non-coherent detector for high-sensitivity GNSS receivers robust against residual frequency offset and unknown bits," in *Proceedings of the 2017 14th Workshop on Positioning, Navigation and Communications (WPNC)*, Bremen, Germany, October 2017, pp. 1–5. doi: 10.1109/WPNC.2017.8250055

[172] J. Leclère, "Resource-Efficient Parallel Acquisition Architectures for Modernized GNSS Signals," Ph.D. dissertation, École Polytechnique Fédérale de Lausanne, Lausanne, Switzerland, June 2014. [Online]. Available: https://infoscience.epfl.ch/record/199547 (Accessed March 19, 2024).

[173] *ZCU102 Evaluation Board User Guide*, Advanced Micro Devices Inc., Santa Clara, CA, USA, February 2023, UG1182 (v1.7).

[174] *TW8825 Datasheet*, Tallysman Wireless Inc., Kanata, ON, Canada, 2019, Rev. 1.2.

[175] *ECOC-2522 SMD OCXO*, ECS Inc, Lenexa, KS, USA, 2016, Rev. 2016.

[176] A. J. Garcia Peña, C. Macabiau, A.-C. Escher, M.-L. Boucheret, and L. Ries, "Comparison between the future GPS L1C and GALILEO E1 OS signals data message performance," in *Proceedings of the ION ITM 2009, International Technical Meeting of The Institute of Navigation*, Anaheim, CA, USA, January 2009, pp. 324 – 334.

[177] A. J. Garcia Peña, M.-L. Boucheret, C. Macabiau, A.-C. Escher, and L. Ries, "Demodulation performance of Galileo E1 OS and GPS L1C messages in a mobile environment," in *Proceedings of the GNSS 2009, 22nd International Technical Meeting of The Satellite Division of the Institute of Navigation*, Savannah, GA, USA, September 2009, pp. 2875 – 2889.

[178] *CORDIC v6.0 LogiCORE IP Product Guide*, Advanced Micro Devices Inc, Santa Clara, CA, USA, August 2021, PG105.

[179] *Fast Fourier Transform v9.1 LogiCORE IP Product Guide*, Advanced Micro Devices Inc, Santa Clara, CA, USA, May 2022, PG109.

[180] D. Mills, *Computer Network Time Synchronization: The Network Time Protocol*, 1st ed. Boca Raton, FL, USA: CRC Press, 2006. ISBN 9781420006155

[181] T. Takasu, *RTKLIB Ver. 2.4.2 Manual*, April 2013. [Online]. Available: http://www.rtklib.com/prog/manual_2.4.2.pdf (Accessed March 19, 2024).

[182] J. Wang and P. B. Ober, "On the Availability of Fault Detection and Exclusion in GNSS Receiver Autonomous Integrity Monitoring," *Journal of Navigation*, vol. 62, no. 2, pp. 251–261, 2009. doi: 10.1017/S0373463308005158

[183] M. E. Frerking, "Oscillator Frequency Stability," in *Crystal Oscillator Design and Temperature Compensation*, 1st ed. Springer Dordrecht, The Nethernlands, 1978, pp. 14–19. doi: 10.1007/978-94-011-6056-8_4