Aalto University
School of Electrical
Engineering

Master's Programme in Computer, Communication and Information Sciences

FPGA-based DECT-2020 New Radio Packet Detection

Roni Fagerholm

Master's Thesis
2024

Aalto University
School of Electrical Engineering

Author	Roni Fagerholm
Title	FPGA-based DECT-2020 New Radio Packet Detection
Degree programme	Computer, Communication and Information Sciences
Major	Communications Engineering
Supervisor	D. Sc. (Tech.) Kalle Ruttik
Advisor	D. Sc. (Tech.) Kalle Ruttik
Date 30.09.2024	**Number of pages** 64 — **Language** English

Abstract

A rapid increase in number and diverse applications of connected devices has forced digital wireless protocols to specialize further for specific purposes. One of such applications is the previously unanswered market for reliable wireless low-latency communication in private self-sustained networks. As a revision of an older standard, DECT-2020 New Radio aims to provide a solution for this market.

To research different physical layer properties of DECT in real-world environments, a software-defined radio (SDR) framework would be required. Since such design already existing for IEEE 802.11, a well-utilized open-source project openwifi, an analogous solution for DECT could be realized. The goal of this thesis was to provide an implementation for the first part of this SDR framework.

The main contribution of this thesis is an FPGA-based packet detector for DECT. Written in Verilog-2001 for an AMD Zynq system-on-chip, the implemented design manages Synchronization Training Field preamble detection and timing synchronization with hardware efficient autocorrelation approach. The proposed design was evaluated using a simulation environment that integrated MATLAB and the Icarus Verilog simulation tool. Both real-world and generated data were utilized during the simulation.

Compared to a cross-correlation based detection, the implementation utilizes only two complex-valued multiplications at the expense of a higher signal-to-noise (SNR) ratio requirement. In comparison to a reference autocorrelation-based detection, the implementation requires at least 2.6 dB higher SNR depending on the utilized configuration, yet it does not suffer from co-channel interference and includes a simpler signal power estimator. The resource utilization of the proposed design is comparable to that of openwifi despite differences in communication protocols. Finally, different power scaling factors were simulated to analyze the implementation's performance. The thesis thus provides an initial foundation to a hardware-based SDR for DECT-2020 NR.

Keywords DECT-2020 New Radio, DECT-2020 NR+, SDR, FPGA, packet detection, timing synchronization, autocorrelation

Aalto-yliopisto
Sähkötekniikan korkeakoulu

Tekijä	Roni Fagerholm
Työn nimi	FPGA-pohjainen DECT-2020 New Radio paketintunnistin
Koulutusohjelma	Computer, Communication and Information Sciences
Pääaine	Communications Engineering
Työn valvoja	TkT Kalle Ruttik
Työn ohjaaja	TkT Kalle Ruttik
Päivämäärä 30.09.2024	**Sivumäärä** 64 **Kieli** englanti

Tiivistelmä

Kommunikoivien laitteiden määrän nopea kasvu sekä käyttökohteiden moninaistuminen ovat pakottaneet digitaaliset langattomat tietoliikennetekniikat erikoistumaan tarkempiin käyttökohteisiin. Yksi tällainen vielä avoin sovelluskohde on pieni viiveiset ja luotettavat yksityiset langattomat verkot omavaraisella järjestelyllä. Vanhemmasta standardista laajennettu uuden sukupolven standardi DECT-2020 New Radio pyrkii tarjoamaan ratkaisua tälle sovelluskohteelle.

Voidakseen tutkia DECT:n fyysisen kerroksen eri ominaisuuksia oikeassa maailmassa, tarvittaisiin ohjelmistoradiolle soveltuvaa kehitysympäristöä. Koska IEEE 802.11 standardille on jo luotu halutunkaltainen ja laajasti käytetty avoimen lähdekoodin projekti openwifi, vastaavanlainen toteutus DECT:lle on mahdollista. Tämän työn tarkoituksena on esittää ensimmäinen osio tälle ohjelmistoradio kehitysympäristölle.

Tämän työn tärkein tuotos on FPGA-teknologiaan perustuva paketin tunnistin DECT:lle. Kirjoitettu Verilog-2001-kielellä AMD Zynq-järjestelmäpiirille, toteutus sisältää Synchronization Training Field-synkronointikentän tunnistuksen sekä ajoituksen synkronoinnin autokorrelaatiomenetelmällä. Tunnistinta arvioitiin MATLAB ja Icarus Verilog-simulaatiotyökalu pohjaisessa ympäristössä todellisilla sekä luoduilla signaaleilla.

Verrattuna ristinkorrelatiopohjaiseen tunnistukseen, totetus käyttää vain kahta kompleksiarvoista kertolaskuoperaatiota, kuitenkin tarviten huomattavasti korkeamman signaali-kohinasuhteen. Toteutus myös vaatii on vähintään 2.6 dB korkeamman signaali-kohinasuhteen kuin verrokki autokorrelaatiomalli, mutta ei kärsi samakanavahäiriöistä ja käyttää yksinkertaisempaa tehoestimaattoria. Ehdotetun tunnistimen FPGA-resurssien käyttö on rinnastettavissa openwifi:in huolimatta radiotekniikoiden eroista. Lopuksi eri tehoestimaattorin skaalauskertoimia simuloitiin toteutuksen suorituskyvyn arvioimiseksi. Työtä voidaan käyttää alustana laitteistopohjaiselle ohjelmistoradiolle DECT-2020 NR:iin.

Avainsanat DECT-2020 NR, DECT-2020 NR+, ohjelmistoradio, FPGA, paketin tunnistus, ajoituksen synkronointi, autocorrelaatio

Preface

First, I want to thank my supervisor D.Sc. (Tech.) Kalle Ruttik for providing me this opportunity. Without him, my passion for hardware and FPGAs had never materialized in such way it is now.

Additionally, I want to thank Nicolas Malm for his never ending guidance and unrelenting support to get me to this point, Veli-Matti Rantanen for understanding my foolish questions and helping in multitude of ways, and Jari Lietzén for his aid during my studies and hobbies. I want to also thank Kalle Koskinen and Victor Nässi for aiding with my studies and work. Without them all, I would have never gotten this far.

Finally, I want to thank my family for their passionate support. And to Linda, I want to apologize for sleepless nights that I have spent working, and thank you for understanding and supporting me.

Otaniemi, 30 September 2024

Roni Fagerholm

Contents

Symbols and abbreviations

Symbols

e	Euler's number
j	complex number
R_{xy}	cross-correlation
R_{xx}	autocorrelation
I	in-phase component of a complex signal
Q	quadrature component of a complex signal
α	power scaling factor
β	Fourier transform scaling factor
μ	subcarrier scaling factor
S	time domain base sequence of STF
c_{μ}	STF signal cover sequence
N_{S}	number of samples in STF base sequence
$N_{\mathrm{REP}}^{\mathrm{STF}}$	number of time domain repetitions in STF
R_{SS}	autocorrelation of the STF base sequence
R_{-SS}	autocorrelation of the STF base sequence with inverted sign

Operators

$\otimes$	convolution
$\bigotimes$	Kronecker product
$\cdot$	multiplication
$\mathbf{A}^*$	complex conjugate of $\mathbf{A}$
$\gg$	bitwise logical right shift
$\lfloor\mathbf{A}\rfloor$	floor function of $\mathbf{A}$

Abbreviations

5G NR	5G New Radio
ADC	Analog-to-Digital Converter
ASIC	Application-Specific Integrated Circuit
AWGN	Additive White Gaussian Noise
BRAM	Block Random-Access Memory
BLE	Bluetooth Low Energy
CORDIC	Coordinate Rotation Digital Computer
DAC	Digital-to-Analog Converter
DDR	Double Data Rate
DECT	Digital Enhanced Cordless Telecommunications
DF	Data Field
DSP	Digital Signal Processor
ETSI	European Telecommunications Standards Institute
FF	Flip-Flop
FFT	Fast Fourier Transform
FPGA	Field-Programmable Gate Array
FSM	Finite-State Machine
FT	Fixed Termination point
GI	Guard Interval
GPP	General Purpose Processor
HDL	Hardware Description Language
HW	Hardware
I/O	Input/Output
IEEE	Institute of Electrical and Electronics Engineers
ISM	Industrial, Scientific, and Medical frequency
IoT	Internet of Things
L-STF	Legacy Short Training Field
LTE	Long Term Evolution
LTE-M	Long Term Evolution Machine Type Communication
LUT	Lookup Table
MAC	Medium Access Control
MIMO	Multiple-Input Multiple-Output
NB-IoT	NarrowBand-Internet of Things
NR	New Radio
OFDM	Orthogonal Frequency-Division Multiplexing
PT	Portable Termination point
RD	Radio Device
RF	Radio Frequency
SDR	Software-Defined Radio
SerDes	Serializer/Deserializer
SNR	Signal-to-Noise Ratio
SoC	System-on-Chip
STF	Synchronization Training Field
TDD	Time Domain Duplexing

1 Introduction

With increasing digitization from commercial applications to industrial machines, granting wireless access over the air between different devices has necessitated various protocols and standards. Since the inception of wireless digital communications devices, later inventions have allowed digital devices to communicate independently without requirement for human intervention [1, Chapter 1]. As the scale of these interconnected devices has increased, this phenomenon of networked devices has become known as Internet of Things (IoT).

By utilizing IoT to remotely supervise large scale systems and create autonomous networks, the range of use cases for the technology has rapidly increased in the past years. Such applications include detecting forest fires [2], automatic wall plant climate control [3], automated warehouses [4] and aiding healthcare [5]. Thus, multiple different wireless protocols have been proposed and utilized to allow machine-to-machine communications over-the-air [6, 7]. Some of these communication technologies have been specifically tailored for IoT applications, such as Bluetooth Low Energy (BLE), LoRa and Zigbee. Others technologies have been later modified from existing protocols, such as Narrowband-IoT (NB-IoT) and Long-Term Evolution Machine Type Communication (LTE-M) from 3rd Generation Partnership Project's (3GPP) cellular LTE standard, Reduced Capability (RedCap) from 3GPP's 5G New Radio (NR), and Institute of Electrical and Electronics Engineers' (IEEE) 802.11, marketed as Wi-Fi.

However, as the number of devices and the variety of operational requirements for different applications have increased, the ability for current protocols to universally meet requested demands has become more challenging. For example, a mobile industrial robot has vastly different performance needs from a static outdoor agricultural sensor. With such a wide range of IoT applications, each of the wireless communication standard has to balance between scalability, power consumption, latency, infrastructural costs, network configuration, and data rates. Hence, each of the wireless IoT technologies has begun targeting a specific range of use cases, such as local real-time device-to-device communications for BLE, long range low-energy applications in case of NB-IoT or self-organizing infrastructureless mesh networks with Zigbee [6, 7].

Even then, a market for reliable low-latency wireless IoT protocol in private self-contained network environment is still left unanswered. In the advent of industrial IoT, where user equipment requires interlinked cable-like wireless transmissions with steady bandwidth, low latency, and reliability, current IoT standards fail to achieve such demands in full [8]. Additional issues from various current standards are the infrastructure requirement or protocol complexity [9]. For example, while 3GPP's RedCap could provide a possible solution, the necessity for setting-up, operating, and administrating a 5G NR network incurs costs and additional labor. From an IoT equipment perspective, standards such as IEEE802.11 include features and requirements that needlessly increase engineering costs when simpler designs could be used. An additional challenge to provide reliability over a wireless channel has been spectrum congestion, as multiple different technologies share the same frequency range [10]. These overlapping frequency bandwidths cause signal interference, that in turn produces data errors and outages. Frequency congestion is a prevalent issue with protocols utilizing industrial, scientific, and medical frequency (ISM) bands, such as Wi-Fi, BLE, LoRa and Zigbee.

To answer the need for a protocol that meets demands for such specific market, European Telecommunications Standards Institute (ETSI) has revised an older wireless communication standard known as Digital Enhanced Cordless Telecommunications (DECT). While the protocol was originally designed for in-premise wireless audio transmissions, the newer revision of DECT has been designed for broader scope of applications. This new standard, named DECT-2020 New Radio, targets infrastructureless networking with need for sub-millisecond latency, adequate data rate, and high reliability [11, 12]. The standard achieves the performance requirements by utilizing state-of-the-art physical layer components with added error correction, a license-exempt technology exclusive frequency range, and self-contained private network architecture.

However, as the protocol is new, most commercial products for it are still in development or starting production. Thus, for academia to further research DECT-2020 NR, the only options currently available are software simulation, software-defined radios (SDR) or development of their own radio devices for the protocol. In the absence of commercial options, only SDRs enable relatively quick-to-setup measurements in real world environments by providing customizable radio interface platforms [13]. Furthermore, SDRs with access to hardware allow additional research avenues. An open-source project for Wi-Fi utilizing an SDR has enabled researchers to study time-sensitive networking, which would have been otherwise challenging to conduct with commercial solutions. This project, openwifi [14], utilizes off-the-shelf SDR with a field-programmable gate array (FPGA) and a general-purpose processor (GPP) to create a framework that could be tailored to users needs.

1.1 Thesis objectives

Creating a similar project for DECT-2020 New Radio as openwifi would provide a framework for further studies to be conducted on DECT. As the IEEE 802.11 project has allowed a multitude of different research topics, a similar template for a new wireless protocol would allow off-the-shelf solutions to be utilized to examine the physical layer properties of DECT.

Hence, this thesis proposes a packet detector implementation for ETSI's DECT-2020 New Radio protocol on an FPGA-based SDR architecture. By utilizing similar design principles as in the openwifi project and migrating them to DECT, the proposed implementation provides preamble detection and timing synchronization for the new protocol. While the thesis implementation is intended to be the initial module for the openwifi-like SDR platform for DECT, it could be further used in other self-contained hardware projects. Such projects could be radio protocol comparison, in researching the physical layer of DECT, as part of wireless traffic analysis, and as a part of physical radio receiver implementation.

In order to achieve this objective, the thesis used an autocorrelation-based packet detection approach and a hardware-description language (HDL) Verilog-2001 to develop the implementation on a target FPGA architecture. Additionally, multiple values were analyzed for a power scaling factor that is used in normalizing the output of the autocorrelation. The validity of the proposed design was achieved by verifying it against a reference modelled in MATLAB. Results of the implementation were obtained by comparing it to other correlation-based packet detection models with simulation and by comparing its resource utilization to the open-source Wi-Fi project for the target FPGA.

As the thesis uses FPGAs for its architectural realization, it needs a hardware development approach compared to software-based design. However, while the proposed implementation could be realized as an application-specific integrated circuit (ASIC), such approach is excluded from the thesis due to the increased design complexity and the process dependent approach. In addition, the following aspects were deemed out of scope for the realized thesis implementation and its evaluation: the physical realization of the proposed implementation on target SDR and support for all DECT physical layer configurations. Excluding assessment on the physical SDR platform was done due to the scope of the thesis implementing an FPGA-based design; hence, evaluation with simulation with captured over-the-air data was deemed sufficient. In case of DECT physical layer configuration, the additional complexity for supporting multiple configurations in the realized design was deemed out-of-scope in favor of a simpler implementation that could be further expanded into other configurations in future works. The thesis discusses how such expansion could be done with the proposed design.

1.2 Structure of thesis

The remainder of this thesis is organized into five chapters as follows. Chapter 2 explains the necessary theoretical background and concepts used for the proposed implementation. Chapter 3 describes the simulation environment utilized in evaluation and hardware platform targeted by the thesis design. Chapter 4 presents the main contribution of the thesis and provides the development process used. Chapter 5 analyzes the proposed implementation and its results by comparing the design against the reference model, with different power scaling factors, against other correlation models, and FPGA resource utilization against openwifi. Chapter 6 concludes the thesis by suggesting future works that could be derived from the proposed implementation.

2 Background

The theoretical and state-of-the-art background necessary for the proposed DECT-2020 New Radio packet detection implementation are as follows: autocorrelation, which is utilized for packet detection and timing synchronization; FPGA architecture, which is the hardware system used to execute the proposed implementation; hardware-based SDR, which is the target radio platform encompassing the hardware system; DECT-2020 NR, which is the radio protocol that this thesis targets; and openwifi, which is a related work with similar design objectives as the thesis on the 802.11 wireless communication standard.

2.1 Correlation

Correlation is used in signal processing and statistical analysis to identify relationships between sets of random variables. This thesis focuses on the signal processing applications of correlation, particularly in the domain of digital wireless communication. A cross-correlation of two finite discrete complex vectors x and y with length N is a convolution between x and complex conjugate of y, defined as

$$R_{xy}(m) = x_m \otimes y^* = \sum_{k=-N}^{N} x_{k+m} y_k^* = \sum_{k=-N}^{N} x_{k+m} \left(s_k + n_k\right)^*, \tag{1}$$

where s is the transmitted signal and n is noise from the transmission channel [15, Chapter 6]. Cross-correlation can be utilized to find a known sequence x from a vector y of random elements. If the correlation and its terms are constrained to a well-defined space, then the cross-correlation maximum will imply if x is within y. Additionally, the location of x can be determined by the position of maximum and the number of maxima will indicate how many times x is located within y.

If $y = x$, and therefore $s = x$, then cross-correlation becomes autocorrelation

$$R_{xx}(m) = x_m \otimes x^* = \sum_{k=-N}^{N} x_{k+m} x_k^* = \sum_{k=-N}^{N} \left(s_{k+m} + n_{k+m}\right) \left(s_k + n_k\right)^*, \tag{2}$$

where $m = 0$, the center of autocorrelation, will have maximum [15, Chapter 6]. A vector with repeating patterns can be examined with autocorrelation function to detect existence of these repetitions. A computational approach using a sliding window for the autocorrelation function additionally allows revealing locations of the repeating patterns within a observed vector x. If a defined window is smaller than the vector x and it is slid over the observed sequence, and the window is at least large enough to hold two repeating patterns, then parts of the x that repeat themselves will have maximum correlation. The drawbacks for computational approach are possibly computationally expensive sliding window operation, determining large enough window size to accommodate repeating patterns and plausible false correlation detection, as all repetitions within the window will be taken into account.

In wireless communications, both cross-correlation and autocorrelation can allow detecting existence of a data transmission. These operations are achieved by inserting a known pattern called preamble or training sequence into a time domain portion of the transmitted signal [16, Chapter 5.1.3]. The receiver then cross-correlates the sent signal with the same locally saved preamble, and detects when the data has been received once the maximum correlation is achieved. This cross-correlation operation is also known as matched filter. However, determining the known pattern from sent data with cross-correlation must be handled with care. As a transmitted signal propagates through a path as an electromagnetic radio wave, it will experience noise and both constructive and destructive effects from environmental multipath components. Furthermore, as baseband processing of a signal is typically done in digital domain, the transmitted stream of samples will experience sampling difference and carrier frequency offset between transmitter and receiver due to unavoidable physical imperfections of utilized electronics. These effects from both channel and digital devices will accumulate to timing and frequency errors, which may cause locating and decoding a preamble from transmitted signal impossible for receiver with cross-correlation. While in digital communications matched filter improves signal quality against noise [17, p. 61-63], the filter provides no benefits towards multipath components.

To circumvent multipath interference, a transmitter can send a vector v that includes a repeating pattern p. If a path that the signal propagates through is constant for the duration of the transmission, all elements in v will experience same propagation effects. The receiver may then use autocorrelation Equation (2) on the v to detect p. Such approach with repeating patterns does require following assumptions: the receiver knows how p is formed, the sequence p should have good correlation properties and the utilized channel does not change during the transmission [16, Chapter 5.1.3]. As an example for such repeating patterns, IEEE802.11 packets include short and long preamble [18, p. 2813-2814] that are utilized to detect the existence of a transmission [19]. Autocorrelation has lower signal-to-noise ratio in noisy radio channels compared to matched filter, thus combining both of them would trade computational complexity for improved signal detection [20].

An issue with correlation equations (1) and (2) is that the values of the correlations are not properly constrained. If a receiver attempts to determine the existence of a pattern in a random stream of samples, it must know when the maximum correlation has occurred. A process that limits values to well-defined constraints is known as normalization, for example to $[-1, 1]$, where 1 equals maximum correlation and -1 maximum anti-correlation. Normalizing the autocorrelation Equation (2) for finite discrete complex vector x with length N yields

$$R_{xx}(m) = \frac{\sum_{n=-N}^{N} x_{n+m}x_n^*}{\sum_{n=-N}^{N} x_n x_n^*} = \frac{E\left[XX^*\right]}{\sigma^2}, \tag{3}$$

where $E[XX^*]$ is the mean of X and its complex conjugate, and σ^2 is the variance of X. If the variance of the random variable is known, then the receiver can unbias the normalization so that the maximum correlation occurs at a predetermined value. However, in cases where variance is unknown, biased normalization must be utilized. These normalization operations are known as biased and unbiased estimators.

2.2 Field-programmable gate array

Field-programmable gate arrays are integrated circuits that consist of multiple electrically reprogrammable logic elements interconnected via a reconfigurable routing matrix [21, Chapter 1]. These logic elements, known as Configurable Logic Blocks (CLBs), consist at a minimum of a lookup table (LUT), a flip-flop (FF) and a multiplexer (MUX). A LUT outputs a Boolean value based on a truth table of its inputs. An FF is a register driven by a clock, latching on its input and outputting its currently held value. Lastly, a MUX selects whenever a LUT's output is routed into an FF's input or past it to the output of a CLB. The combination of a LUT, a FF and a MUX enable CLBs to create binary logic, or any digital logic when further augmented with reprogrammability and the reconfigurable routing matrix. In addition to CLBs, most of the FPGA's input and output (I/O) ports can also be reprogrammed, granting its users freedom in integrating the device into a physical system or reconfiguring the I/Os depending on the use case. An example architecture of a simplistic FPGA is shown in Figure 1.

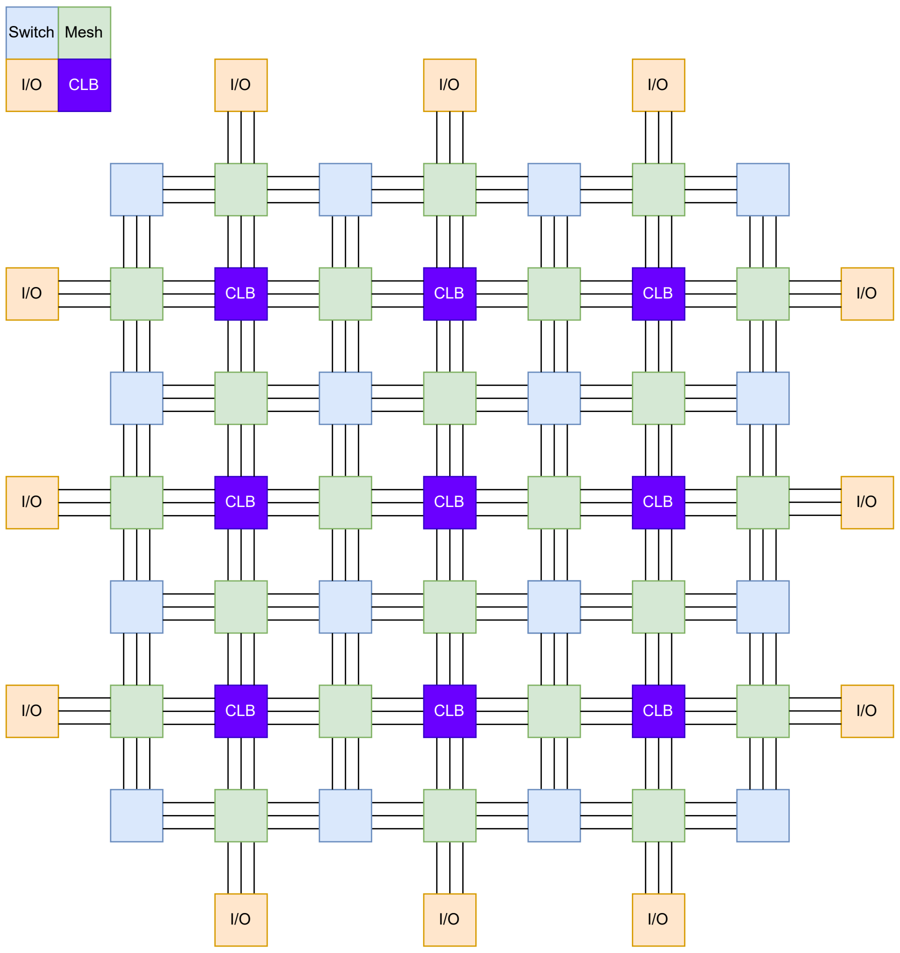


Figure 1: A simple FPGA architecture.

The main benefits for utilizing FPGAs are their ability to implement any digital logic, reconfigurability, and parallelization. With the possibility to create any digital logic, FPGA users may: hardware accelerate computationally heavy operations [22, 23, 24], prototype hardware systems, create glue logic to interface between devices, or produce digital system in small volumes without need for fabrication [25]. Such varied applications are possible due to the electrically reprogrammability of the circuit, which limits non-recurring engineering costs and allows modifications to the design even during deployment.

In addition to these benefits, the performance of FPGA architectures stems from their inherent massive parallelization, where each element operates independently. The only limiting factors for latency performance are the clock frequency utilized and number of clock cycles for a given data path, while data rate performance is limited by the available I/Os and number of hardware elements in a given FPGA device [24]. To improve data rates, dedicated hardware elements have been integrated into FPGAs to accelerate hardware-intensive processes and enhance I/O bandwidths. Examples of such improvements include digital signal processor (DSP) blocks to accelerate multiplications and large sums, block random-access memory (BRAM) modules to increase on-chip memory, and specialized I/Os such as Double Data Rate (DDR) or serializer/deserializer (SerDes) interfaces that increase bandwidths to external devices.

However, drawbacks for FPGAs include the volatility of their programming where the circuitry must be reprogrammed after each power up, limited availability of on-chip memory, a laborious and complex development process of digital logic compared to software development, and increased power consumption with worse performance in comparison to ASICs for the same task [25]. In addition, while serial tasks that include states and many branching paths can be implemented in FPGAs, such operations may not benefit from the massive parallelization of the device. Furthermore, such tasks can be more easily developed in software and executed on GPPs, which have become ubiquitous in embedded systems.

For development process of digital logic on FPGAs, hardware-description languages such as Verilog, SystemVerilog and VHDL have been the mainstay as design tools. In HDL development flow for an FPGA, as shown in Figure 2, the designer first models their given system with HDL code, before synthesizing the design into register-transfer level (RTL). The RTL is then placed-and-routed and converted to a bitstream used to program the FPGA. Each stage additionally includes intermediary steps, such as simulations, resource and timing analysis, and constraints. Additionally, designers may utilize vendor or third-party intellectual properties (IPs) to accelerate development process.

Nevertheless, due to the inherently laborious process to create and manage complex systems with HDLs, other tools have gain attention to ease development. One of such tools that has gained industry wide support is high-level synthesis (HLS), where a higher abstraction of logic is described and then converted into HDL code or directly into RTL [26, 27]. With HLS, the mental overhead of complex systems will be minimized, the system's behavioural operation simulation becomes lower effort and higher abstractions is possible to implement with minimal changes, and development time of larger designs is reduced significantly. However, the performance and resource utilization of HLS generated code can be unpredictable and sub-par in comparison to a hand-written HDL [28]. Therefore, while high-level synthesis has become more popular due to the ease of use and rapid development, hardware-description languages are still used in designs were performance and resource utilization constraints or predictability are paramount.

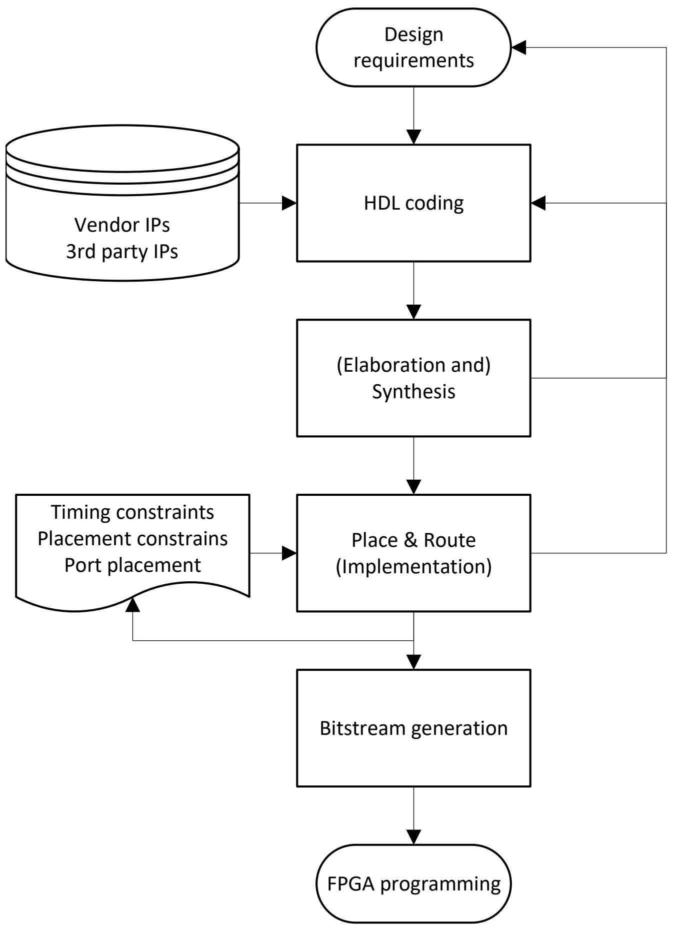


Figure 2: A FPGA design flow.

2.3 Hardware-based software-defined radio

Software-defined radios are radio platforms which allow arbitrarily modifying the digital baseband and controlling the analog radio frequency (RF) front end [29]. In applications where requirements for the analog interface are known or within well-defined range, while demands for baseband processing might change rapidly or be unknown beforehand, a SDR provides the necessary freedom and performance. Example applications that require such adaptability are research and development, electronic warfare and small-scale radio network deployments [13].

While off-the-shelf SDR platforms can support a wide range of spectra, large data rates, and high accuracy, these systems still face other drawbacks beyond cost efficiency when compared to dedicated radio terminals [13, 30]. A user of a SDR must provide the required functionality for the device, which either necessitates a development effort or the use of ready-made solutions. Additionally, while SDRs have lower power consumption and cost than multiple dedicated terminals when transmission flexibility is required, these advantages are lost when transmission requirements are fixed. Most importantly, antennas that support wide frequency ranges presents significant challenges, and in applications with high-accuracy requirement, users must account for potential impairments.

SDRs can be classified into two major categories: GPP-based or hardware (HW) based [13, 31]. In GPP-based SDR platforms, most or all of the baseband processing is operated on a GPP-host. These system typically include a secondary device that is tasked with interfacing a RF front end, in addition to handling preliminary signal processing in both the receive and the transmission chain. Examples of such systems are GNURadio [32], RTL-SDR [33], and SORA [34]. Conversely, in hardware-based systems the baseband processing is done on a dedicated SDR device interfacing its RF end. Such platforms are typically self-contained or include an expansion card for a user selected RF front end. Examples of HW SDRs are WARP [35], LOLA-SDR [36], Airblue [37] and RT-WIFI [38]. While some of these designs could be categorized as hybrid-based SDRs [13], as they utilize both a GPP and a FPGA, their system architectures rely on system-on-chip (SOC) designs that provide low-latency on-chip interfaces. This thesis will focus on the hardware-based software-defined radios due to the selected target SDR platform.

In comparison to GPP-based solutions, HW SDRs platforms provide much larger data rates, allow precise control over performance and latency, and have higher power efficiency per operation [13]. As most of the platforms utilize an FPGA, a system-on-chip (SoC) with an FPGA, or an ASIC, the massive parallelization and high throughput intrinsic to the systems enable the larger data rates, and thus both sample rates and bandwidth. Furthermore, as these system utilize hardware-based designs with logic gates and registers, SDR users can create custom designs and architectures that are tailor made for each use case. The benefit for such approach is the precise control over design parameters, which conversely translate into user-defined latency and data rates. In addition, the HW-based SDRs that are self-contained can be effortlessly utilized in field-tests without requirement for additional equipment.

However, the main drawbacks for HW SDRs are cost and development process [13]. Due to the inherent slow and laborious development of FPGAs, time-to-market or time to the prototype are much longer than in equivalent GPP-based design. Additionally, SDRs utilizing software can be operated with various free-to-use open-source tools or low-cost propriety ones, while their development can be done using plug-and-play style or various software libraries and programming languages. In hardware-based development, propriety tools are typically per device family, require expensive licences before usage, and are typically rigid, monolithic in design and have restrictive host requirements.

Furthermore, open-source tools can be limited in both availability and use cases. In conjunction with the development, while hardware-based designs can be self-contained, their unit costs to setup are typically much higher. Where as a user laptop and an existing wireless modem can create a temporary GPP-based SDR, in hardware the user must obtain the physical device before it can be utilized. In addition, HW-based SDR is only able to support the range of operations it is physically designed for and any limitations in the system require physical revision of the platform. Each of these hardware-based issues are further exacerbated with ASICs, which require additional fabrication and any faults in the design are in most of the cases non-fixable. Conversely, SDRs utilizing software can be readily scaled and expanded, as most of the designs can be swapped from one host to another with minimal or no changes required. In hardware, migration from one device to another is effortless only if both of the devices have same architecture. In most of the cases, the design requires some changes, but in the worst case the implementation can not be migrated and requires extensive work.

2.4 DECT-2020 New Radio

DECT is local area wireless communication technology by ETSI, composed of two main standards: Classic DECT and DECT-2020 New Radio. This thesis will only focus on DECT-2020 NR and thus refer to it as DECT. Approved by International Telecommunication Union Rabdiocommunication Sector (ITU-R) for International Mobile Telecommunications-2020 (IMT-2020 Standard), DECT is the first non-cellular 5G standard. The protocol aims to provide a reliable local network that allows its users to setup, scale, manage and operate without a dedicated infrastructure operator as with cellular technologies [39].

Furthermore, the standard can work independently or as a medium for both IP, IPv4 and IPv6, and non-IP data. Therefore, the key advantages which DECT offers in comparison to other IoT standards, such as, but not limited to, BLE, LoRa, Zigbee, LTE-M, NB-IoT, and RedCap, are the combination of reliability, scalability and network structure flexibility balanced with data rates and latency. By merging state-of-the-art technological advancements, a user-definable network topology, and a technology-exclusive license-exempt spectrum, the protocol's envisioned main applications are industry 4.0, public services and multimedia industry, yet supports other local networking use cases [12]. The key values of DECT are summarized in Table 1.

DECT supports two types radio transmissions: synchronous and random access [40]. In synchronous transaction mode, a coordinator allocates dedicated time slots for each of its client devices. Such approach provides predictable data rates and latency, whilst simultaneously allowing the coordinator to balance load between clients. However, transmission channels with unpredictable interference, such as ISM radio bands, such synchronization is not always possible or is too unpredictable. To aid reliability in such access, the coordinator allocates transmission slots that its clients may utilize randomly. The each of the transmission slots have a start time and each random access is limited in data size by a time window during which a single transmission must be sent.

Table 1: DECT-2020 NR key values.

Parameter	Value
Minimum bandwidth	1.728 MHz
Maximum bandwidth	221.184 MHz
Multiplexing	CP-OFDM with FDMA and TDMA
Frame length	10 ms
Slots per frame	10
Number of subslots per slot	2, 4, 8, or 16
OFDM symbols per slot	10, 20, 40, or 80
OFDM symbols per subslot	5
Duplexing	TDD
Modulation	BPSK, QPSK, 16-/64-/256-/1024-QAM
Maximum MIMO	8 receiver, 8 transmitter
CRC size	16 or 24 bits
Coding rate	1/2, 2/3, 3/4, or 5/6
Other features	Beamforming, transmit diversity, turbo code, HARQ with soft-combining, coexistence with Classic DECT

Before utilizing a transmission slot, a client must check if the channel has ongoing transaction before trying to send its own data within the allowed time window. If the allocation is already in use, the client must wait for a random period of time before trying again. As such, random access can also be beneficial in cases where data transmissions are infrequent or utilize first-come, first-served transaction style.

For radio topology, DECT allows two modes to be utilized simultaneously or one at the time: fixed termination point (FT) and portable termination point (PT) [41]. In FT mode, a radio device (RD) acts as a local coordinator that manages transmission and configurations between itself and devices in PT mode. As such, a device in portable termination may only search for, connect to and configure itself to a FT before any data transactions. To allow flexible topology, a device in FT mode may have multiple PTs connected to it and can simultaneously connect to another FT as a PT. However, a portable termination may only have a single device as its fixed termination, which means that a RD is always a FT for a PT and vice versa.

The full protocol stack of DECT, as shown in Figure 3, is partitioned into medium access control (MAC) layer and physical layer [12]. The function of MAC layer is to manage user data, a local radio device RD, and networking between other radio devices. Data from user are segmented into packets, while received data from physical layer is combined. Additionally, four different logical transmission packets are supported: paging and broadcasting (PCH/BCH), dedicated communication (DCH), and random access (RACH) [40]. Each of these are then encapsulated into either a Physical Control Channel (PCC) or a Physical Data Channel (PDC) before passed to the physical layer for transmission, or decoded from when receiving. When communicating with the user, two data modes are supported: Dedicated Traffic Channel (DTCH), for bidirectional communication with a RD, and Multicast (Broadcast) Traffic Channel (MTCH), for multicast data communications with multiple RDs simultaneously. Both of these are internally multiplexed and demultiplexed to the appropriate logical transmission packets. The MAC layer is additionally responsible for facilitating communications between user, local RD, and other radio devices by choosing an appropriate connection mode. Furthermore, selection and coordination of radio interface parameters as a FT to PTs, or correcting the parameters as a RD to the ones that its FT utilizes, are tasks of the MAC. The layer additionally handles hybrid automatic repeat request (HARQ), an error control algorithm. In HARQ, The receiver sends an acknowledgement if the packet was received correctly or included errors. The transmitter can then send a part of the packet again, which the receiver can then combine with the faulty one to get the correct packet.

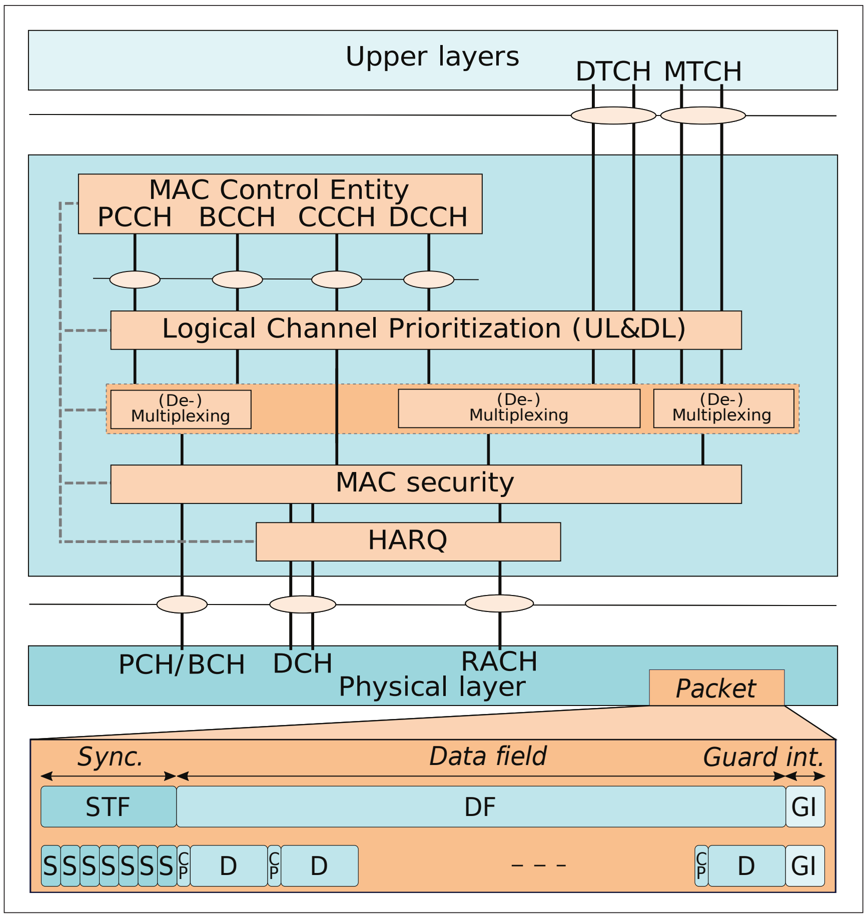


Figure 3: Full DECT-2020 NR protocol stack [12, Figure 2].

2.4.1 Physical layer

The physical layer in DECT protocol stack is responsible for connecting radio interface and MAC layer, and management of both radio and baseband data. The tasks of the physical layer include: multiple-input multiple-output (MIMO), beamforming, forward error coding and decoding, rate matching, modulation and demodulation, error detection, timing and frequency synchronization and transmission diversity [42, p. 8-9]. As such, the layer can be modeled as a stream that sends segmented data from the MAC layer to a physical radio interface and, in a vice versa manner, passes received data from the interface to the MAC. This stream, as shown in Figure 4 for a single antenna receiver, is a multistage chain of operations.

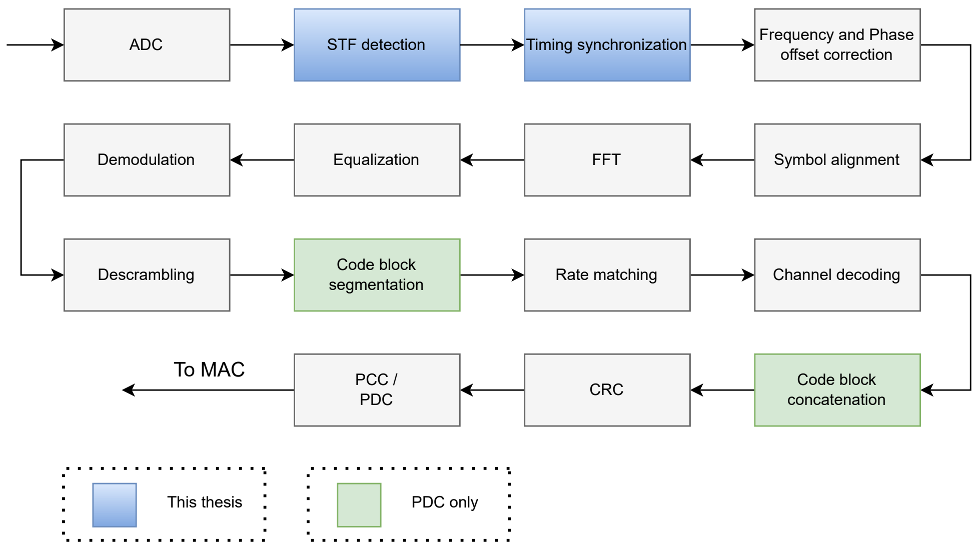


Figure 4: DECT-2020 NR single antenna receiver block diagram.

In the receiver chain, the physical layer is first tasked with receiving in-phase and quadrature (I/Q) complex samples from the radio interface, typically after an analog-digital converter (ADC) which samples the analog baseband. From this stream of I/Q-samples, the receiver must determine the existence of a packet and its location. DECT utilizes a preamble-based synchronization method, where the physical layer packet structure includes a data field to aid the receiver to timing, phase and frequency synchronize itself to the transmitter.

After synchronization and correction of phase and frequency, the orthogonal frequency-division multiplexing (OFDM) symbols are aligned and the OFDM cyclic prefix (CP) is removed. The subcarriers of the OFDM are then obtained by using a discrete Fourier transform's optimized version, fast Fourier transform (FFT). Utilizing equalization and predetermined pilot subcarriers, the receiver then attempts correct any intersymbol interference (ISI). In ISI, a channel with multipath propagation or one that is bandlimited might cause OFDM symbols to overlap in time domain. The symbol that is overlapped can then have distortions in its subcarriers in frequency domain, which may cause errors in subsequent stages. Once the receiver has equalized the subcarriers, they are demodulated to extract bits, which in digital system are binary values. In addition, received samples are parallelized before FFT, and serialized again after demodulation.

With the binary stream, the receiver then performs descrambling of the data. In scrambling, vectors with multiple consecutive 0s and 1s are dispersed, which aids synchronization and reduces spurious spectral changes during the transmission. Then, if the received packet type is PDC, it is segmented. These segments, or the single binary stream in PCC case, are then individually rate matched and channel decoded with turbo codes, which attempts to correct any incorrect bits in the stream. After error correction, the segments are concatenated to obtain the final binary stream for a given received packet. Before encapsulating the received packet to either PDC- or PCC-packet type and passing it to the MAC layer, the receiver does one final error check with a cyclic redundancy check (CRC). If an error is detected, the MAC layer is informed for further actions to be taken.

The physical layer has two major configuration parameters: a Fourier transform scaling factor β and a subcarrier scaling factor μ [42]. The Fourier transform scaling factor is used to scale the size of discrete Fourier transform, which in turn defines the number of subcarriers in the OFDM. Similarly, the subcarrier scaling factor determines the OFDM subcarrier spacing, number of OFDM symbols per subslot and number of subslots in a transmission slot. Therefore, the utilized radio link bandwidth is thus the combination of both parameters, where $\mu \in [1, 2, 4, 8]$ and $\beta \in [1, 2, 4, 8, 12, 16]$. For example, the formula for nominal channel bandwidth is defined by

$$B_{DFT} = \mu \cdot 27 \text{ kHz} \cdot \beta \cdot 64.$$

Likewise, the transmission channel bandwidth is calculated as

$$B_{TX} = \mu \cdot 27 \text{ kHz} \cdot \beta \cdot (64 - \beta \cdot N_{CP} - N_{DC}),$$

where $N_{CP} = 8$ is the cyclic prefix size and $N_{DC} = 1$ is a subcarrier reserved for direct current.

These parameters are additional used to select the number of sublots and cyclix OFDM symbols, which in turn determine the data rate and transmission time. To transmit data over a physical channel, DECT utilizes a transmission packet structure illustrated in Figure 5. A packet consists, in order, of a Synchronization Training Field (STF), multiple Data Fields (DFs), and a Guard Interval (GI) [42, p. 14]. The STF enables the data-driven synchronization, while DFs are the OFDM symbols transmitted with a CP. As DECT utilizes time division duplex (TDD) for time division multiple access (TDMA), the GI reduces timing constrains between different slots. To summarize, each physical layer transmission packet contains one or more subslots, where each subslots has five OFDM symbols. The first symbol is reserved for STF, while the last symbol is reserved for GI. Hence, each transmission packet contains at minimum three DFs.

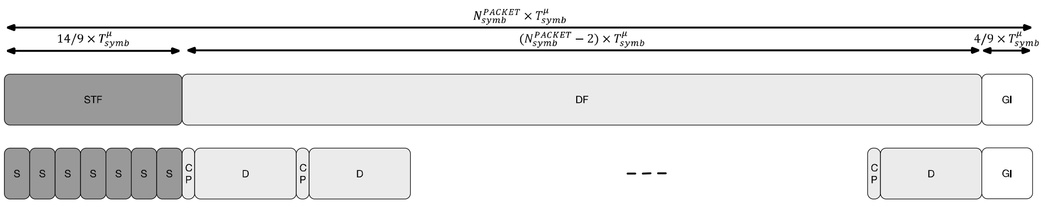

$N_{symb}^{PACKET} \times T_{symb}^{\mu}$
$14/9 \times T_{symb}^{\mu}$
$(N_{symb}^{PACKET} - 2) \times T_{symb}^{\mu}$
$4/9 \times T_{symb}^{\mu}$
STF
DF
GI
S
CP
D
GI

Figure 5: DECT physical layer transmission packet structure for $\mu = 1$ [42, Figure 5.1-1]

2.4.2 Synchronization Training Field

Each physical layer transmission packet in DECT begins with a predefined STF sequence. The function of this preamble is to allow a receiver to determine existence of a packet, time domain synchronize, steer automatic gain control, and coarse correct both frequency and phase offsets [43]. The field is formed with six predetermined pilot sequences, defined in frequency domain, with the sequence selection depending on the selected Fourier transform scaling factor β [42, p. 15-16]. For $\beta = 1$, which this thesis utilizes, the pilot pattern is:

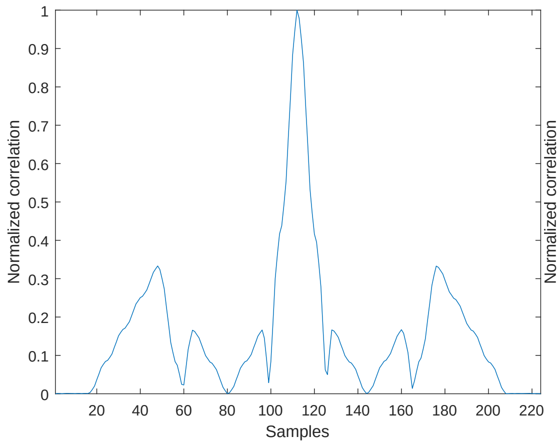
$$\mathrm{STF}_{\mathrm{pilot}}^{\beta} = e^{j\pi/4}\{1,-1,1,1,-1,1,1,-1,1,1,1,-1,-1,-1\}\,.$$

Before transmission, the pilots are transformed to time domain with an inverse Fourier transform. The resulting sequence is known as base sequence, denoted by S, and the length of the sequence is $N_S = 16 \cdot \beta$. Additionally, to both allow and improve correlation-based sequence detection, S is repeated number of times that depend of the utilized subcarrier scaling factor μ. For $\mu = 1$, the number of repetitions is seven, and nine in all other cases. For this thesis, $\mu = 1$ is utilized. As such, the time domain STF for $\{\beta = 1, \mu = 1\}$ physical layer configuration consists of seven repeating identical S patterns with length of 16.

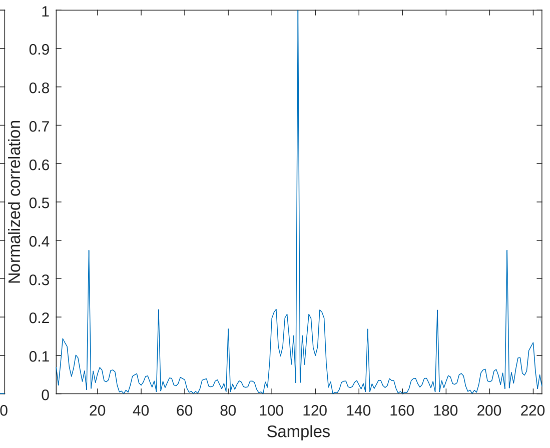
However, each of the repeated base sequences are further multiplied with an other vector called cover sequence, denoted by c_μ [42, p. 34]. This sequence has two options, depending on used μ, which are

$$c_\mu = \begin{cases} \{1,-1,1,1,-1,-1,-1\}\,, & \text{when } \mu = 1 \\ \{1,-1,1,1,-1,-1,-1,-1,-1\}\,, & \text{when } \mu \in \{2,4,8\}\,. \end{cases}$$

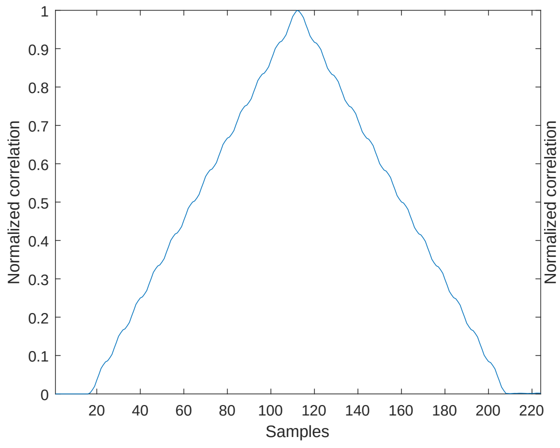
The function of c_μ is to aid accurate localization of both the end of sequence and the maximum correlation of the STF, which are required in the timing synchronization and the phase and frequency offset correction. Figure 6 illustrates the effect of c_μ in autocorrelation and cross-correlation, where cover sequence is either included or omitted and, if omitted, with correlation window sizes of the full STF sequence or only the base sequence. In cases were the cover sequence is included, Figures 6a and 6b, the maximum correlation is prominent and therefore much easier to detected. Additionally, the end of STF sequence is located at the maximum, hence both timing synchronization and offset corrections can be operated on the same sequence.



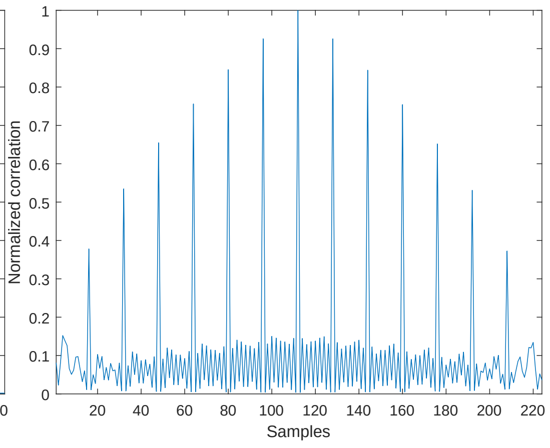
(a) Autocorrelation with cover sequence.



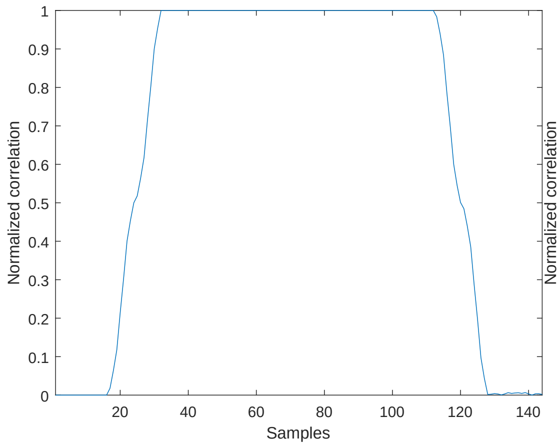
(b) Cross-correlation with cover sequence.



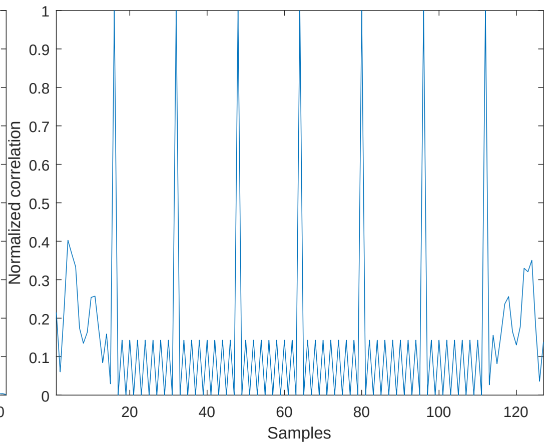
(c) Full autocorrelation without c_μ.



(d) Full cross-correlation without c_μ.



(e) S windowed autocorrelation without c_μ.



(f) S windowed cross-correlation without c_μ.

Figure 6: Illustrations of STF autocorrelation and cross-correlations with and without cover sequence c_μ for physical layer $\{\beta = 1, \mu = 1\}$ configuration. For cases without cover, window sizes for full sequence and only base sequence are additionally displayed.

Thus, the final time domain STF signal is a Kronecker product of the cover sequence c_μ and the base sequence S

$$\mathrm{STF} = c_\mu \bigotimes S = \begin{Bmatrix} c_\mu(0)\cdot S \\ \dots \\ c_\mu(N_{\mathrm{REP}}^{\mathrm{STF}}-1)\cdot S \end{Bmatrix}$$
$$= \begin{Bmatrix} c_\mu(0)\cdot S(0) & c_\mu(0)\cdot S(1) & \dots & c_\mu(0)\cdot S(N_S-1) \\ c_\mu(1)\cdot S(0) & c_\mu(1)\cdot S(1) & \dots & c_\mu(1)\cdot S(N_S-1) \\ \dots & \dots & \dots & \dots \\ c_\mu(N_{\mathrm{REP}}^{\mathrm{STF}}-1)\cdot S(0) & c_\mu(N_{\mathrm{REP}}^{\mathrm{STF}}-1)\cdot S(1) & \dots & c_\mu(N_{\mathrm{REP}}^{\mathrm{STF}}-1)\cdot S(N_S-1) \end{Bmatrix},$$

where $N_{\mathrm{REP}}^{\mathrm{STF}}$ is the nth base sequence repetition.

2.5 Openwifi

Openwifi is an open-source SDR project that implements full-stack of Wi-Fi/IEEE802.11, versions a/g/n, on a SoC [14]. The project interfaces Linux based systems with radio front ends, where the overall structure is shown in Figure 7. The main contribution of the project is an open-source physical layer implementation of the IEEE802.11 in Verilog and interfacing this implementation with Linux Kernel 802.11 MAC framework. Main benefits for openwifi are: support of off-the-shelf hardware platforms, a working template that can be expanded and modified for further uses, and a plug-and-play setup. The SDR has been used to study IEEE802.11, such as time sensitive networking and real-time communications [44, 45, 46, 47, 48, 49, 50, 51]. Although openwifi could be modified to run on any configuration with a GPP, a FPGA and a radio front end, project is aimed towards using AMD Zynq SoCs as hardware platform and Analog Devices AD9361 as radio front end.

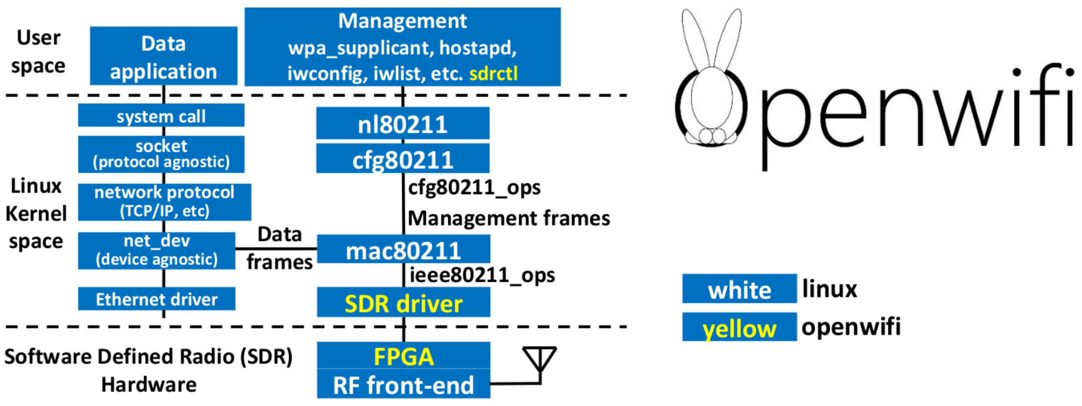


Figure 7: Openwifi overall architecture [52].

The project design can be divided into two parts: SDR driver on GPP software written in C and 802.11 physical layer on FPGA hardware written in Verilog HDL.

The software is an interface between the proctol implementation and a Linux Kernel framework that encapsulates IEEE802.11 MAC into an application programming interface (API) [53]. The SDR driver additionally controls and configures the radio front end and the 802.11 physical layer, manages transmission packets, and allows custom user logic in both software and hardware to be utilized. As an example of the customized logic, the project has a module that allows bypassing the IEEE 802.11 transmission packet structure and capturing I/Q-samples from an ADC on a custom trigger.

The physical layer implementation on FPGA is divided into five sections as shown in Figure 8. The sections are: rx_intf, tx_intf, openofdm_rx, openofdm_tx and xpu. The first section, rx_intf, is tasked with reading I/Q-samples from ADC and passing them first to OFDM demodulation before writing them as bytes to the GPP through direct memory access (DMA) interface. The section may also bypass OFDM demodulation or loopback data sent in transmission path, if configured by the user. The second section, tx_intf, is responsible for transferring bytes from GPP through DMA to OFDM modulation and finally I/Q-samples to a digital-to-analog converter (DAC). The third and fourth sections, openofdm_rx and openofdm_tx modules are tasked with de-/modulation, de-/interleaving, packet synchronization, frequency correction, symbol alignment, inverse FFT and FFT, equalization, CRC and rate matching. The fifth section, xpu, handles power detection, carrier-sense multiple access with collision avoidance (CSMA/CA), received signal strength indication (RSSI), and both transmission and receiving header parsing.

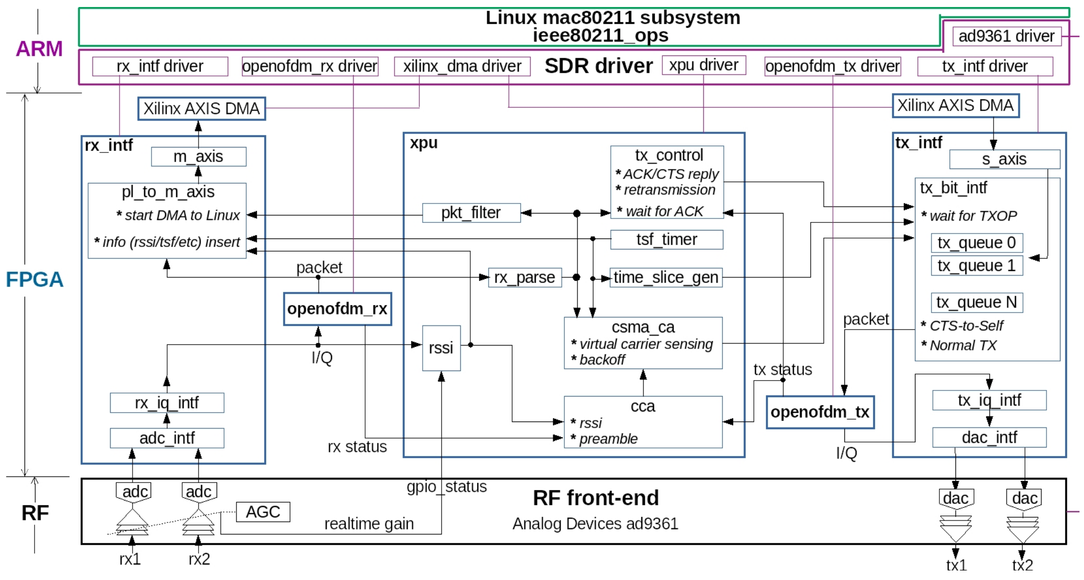


Figure 8: Openwifi design block diagram [52].

3 Simulation and hardware environment

The thesis implementation was developed on Verilog-2001 HDL using AMD Vivado, simulated during development process with Icarus Verilog [54] simulation and GTKWave [55] waveform analysis tool, and evaluated on a software simulation environment based on MATLAB. Furthermore, the proposed implementation targeted the same SDR platform as the open-source project openwifi utilized. While the platform has satisfactory performance requirement for wireless radio protocol physical layers, the main interest of the platform for this thesis is the prospect for future development in a DECT framework.

3.1 Simulation environment

The software environment utilized in this thesis can be roughly divided into two parts: algorithm evaluation and HDL development. Algorithms, which the proposed implementation were to actualize on the target FPGA, were designed and evaluated in MATLAB. During the hardware description language programming, an open-source Verilog simulation tool Icarus Verilog and an open-source waveform visualization tool GTKWave were used for simulation. AMD Vivado was the tool employed for FPGA synthesis and resource utilization evaluation.

The final system evaluation utilized a combination of MATLAB and Icarus Verilog to analyze correctness of the developed Verilog implementation against the reference, performance of the proposed design against other correlation models, and evaluation of different estimated power scaling factors. The overview of this simulation environment is shown in Figure 9.

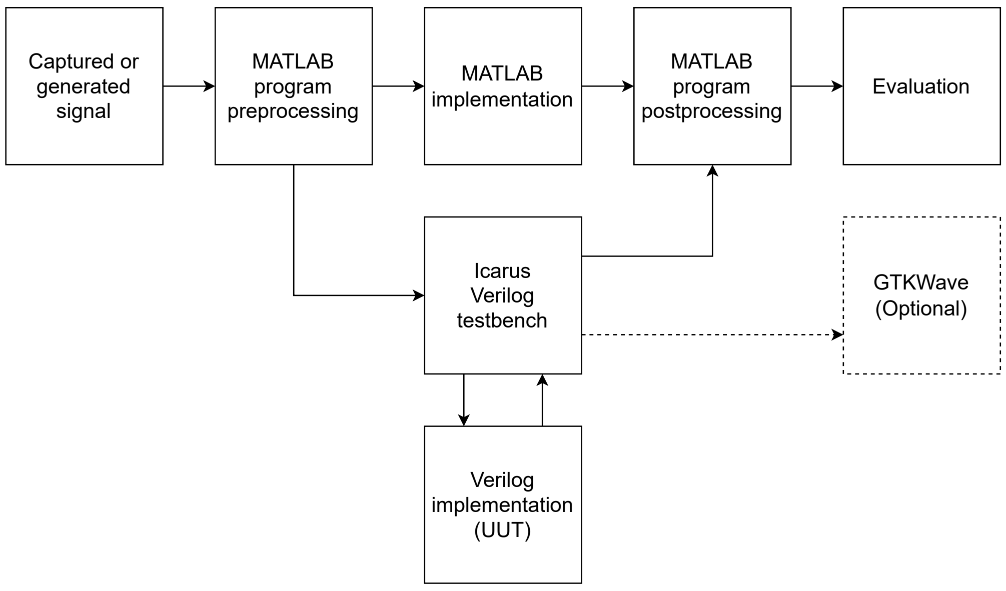

Captured or generated signal
MATLAB program preprocessing
MATLAB implementation
MATLAB program postprocessing
Evaluation
Icarus Verilog testbench
GTKWave (Optional)
Verilog implementation (UUT)

Figure 9: Block diagram of the simulation environment used to evaluate the proposed Verilog implementation.

The main MATLAB program consists of an preprocessing, reference MATLAB implementation of the algorithm, post-processing, and evaluation of the Verilog's output. Data used for analysis was either captured from an over-the-air transmission or generated in software. The captured or generated signal was then preprocessed before executing an Icarus Verilog test bench and the reference MATLAB implementation in parallel. The testing environment in turn executed the realized Verilog implementation with the given data and wrote the outputs for post-processing. Once both the test bench and the reference had finished their operations, the post-processing read the Verilog's output and reformatted it for further analysis. In case further debugging of the HDL implementation was required, the Verilog testing environment additionally created a waveform trace to be analyzed with the GTKWave.

While such environment allows a robust simulation and analysis of an algorithm and its realization on HDL, the environment can be additionally expanded to a full radio stack simulation. In such case the HDL implementation could be analyzed in-depth as close as possible to a real-world realization, and, conversely, the upper layers of a radio protocol stack could be examined against an actual hardware implementation. Furthermore, replacing the Icarus Verilog testing environment connections with a network socket or an equivalent solution would allow real-time and real-world analysis of the realized HDL on a hardware platform. Lastly, in full stack simulation parts of the MATLAB environment can be replaced with a C or other programming language designs, which would allow designers to first simulate parts of the radio protocol in software against a real hardware implementation. The benefits for such approach is that the simulated design could be as close as possible to the same design executed in actual environment, while simultaneously allowing in-depth debugging and analysis to the design with a trivial access. In addition, the inputs to the system could be controlled, permitting simulation of rare corner cases and unit testing for both software and hardware.

However, even such approach lacks proper means to simulate issues rising from analog components or from hardware-related issues. An additional problem for software simulation is the much lower performance of the simulated system, limiting availability of fully real-time evaluations with real-world. The performance becomes especially an issue if the simulated design is complex or tools used for simulation have sub-par execution time, in which case even small steps in the simulation may take significant amount of time. Nonetheless, a full stack simulation would provide a framework which could be further modified depending on the need and as a step before actual hardware.

3.2 Hardware platform

The physical software-defined radio platform that the thesis implementation is designed towards is a combination of an Avnet's ZedBoard [56] development board and an Analog Device AD-FMCOMMS3-EBZ [57] evaluation board.

As the controlling part of the SDR platform, ZedBoard has an AMD Zynq-7000 XC7Z020 SoC at its core, an integrated circuit that includes both an ARM's Cortex-A9 general-purpose processor and an AMD Artix-7 FPGA. Table 2 lists the main features of the SoC, where most notable features are the interfaces between the processor and the FPGA. The 32-bit master and slave interfaces individually have a theoretical bandwidth of 600 MB/s per interface, while the high-performance advanced extensible interface (AXI) memories and accelerator coherency port (ACP) have theoretically a bandwidth of 1200 MB/s per interface [58]. Furthermore, as all interfaces are bidirectional, the total achievable bandwidth per interface is double of the unidirectional. While connection technologies such as DDR and SerDes have unidirectional bandwidths an order of magnitude higher, such connections require additional devices to be supported and are rarely available within integrated circuits. Additionally, the ACP interface allows FPGA logic to access GPP cores' caches in coherent manner, which enables much lower latency than off-chip solutions. As such, the SoC enables real-time applications were the lowest latency is a requirement.

Table 2: Zynq-7000 XC7Z020 SoC features.

Processor	ARM Cortex-A9 MPCore
Cores	2
Maximum frequency	667 MHz
L1 Cache	32 KB Instruction, 32 KB data per processor
L2 Cache	512 KB
On-chip Memory	256 KB
FPGA	Artix-7
Programmable Logic Cells	85000
LUTs	53200
FFs	106400
BRAMs (36 kb each)	140
DSPs	220
Processor-FPGA interfaces	2x AXI 32 bit Master 2x AXI 32 bit Slave 4x AXI 64 bit/32 bit Memory AXI 64 bit ACP 16 Interrupts

More importantly, such in-chip interfaces create opportunities to interconnect software and hardware, as in the case of the openwifi. As the FPGA implementation of project is responsible for the 802.11a/g/n physical layer, it can be supplemented with the software's ability to handle complex state machines and loops that are required in the MAC layer. In addition, analyzing and calculating different parameters in software is relatively trivial in comparison to the HDL. For example, calculating correct carrier frequency and its configuration for the radio front end's phase-locked loop occurs seldom in most of the uses cases, requires multiple branching steps and may include floating-point arithmetic for finer frequency granularity. While same operations could be realized in hardware, such approach would needlessly waste resources in a process that doesn't need immediate response with low latency. Similarly, a real-time packet detection and synchronization would require most of the GPP's processing power and thus reduce the overall performance of the system. With an FPGA, the detection and synchronization can be completely parallel to all other operations and even each others, can be done trivially done in real-time and can be efficiently designed for low resource utilization with correct algorithms.

As the radio front end part of the SDR platform, the Analog Device evaluation board utilizes AD9361 as the radio transceiver, with its key features listed in Table 3. While the maximum bandwidth of 56 MHz excludes use cases where large data rates are required, the wide spectrum support enables utilizing large selection of wireless technologies. Finally, with two independent radio chains and possibility for both TDD and frequency division duplex (FDD), the transceiver enables variety of scenarios for both single-input single-output and MIMO.

Table 3: FMCOMMS3 (AD9361) features.

TX/RX chains	2
ADC/DAC bits	12
TX spectrum	47 MHz to 6.0 GHz
RX spectrum	70 MHz to 6.0 GHz
Modes	TDD and FDD
Bandwidth	<200 kHz to 56 MHz
TX max power	9 dBm
Minimum RX noise figure	2 dB

4 Packet detector implementation

The main contribution of this thesis is an DECT-2020 New Radio packet detector and timing synchronizer written in Verilog-2001 for FPGA-based hardware architectures. The implementation uses a normalized autocorrelation-based approach for detecting STF preamble and, in conjunction with a buffering system, synchronizing user data. The autocorrelation algorithm implemented was presented in DECT Forum NR+ conference, available at [59, Webinar Series (4): Technical II].

The realized implementation, as shown in Figure 10 block diagram, is composed of two major modules: STF detection and timing synchronization. The detector has three distinct elements to detect presence of a STF preamble, which are autocorrelation's numerator $E[XX^*]$, denominator σ^2, and their comparison. For timing synchronization, a finite-state machine (FSM) and a ring buffer are used to determine a location of the autocorrelation's peak, which in turn allows locating the start position of the received packet in a sample stream.

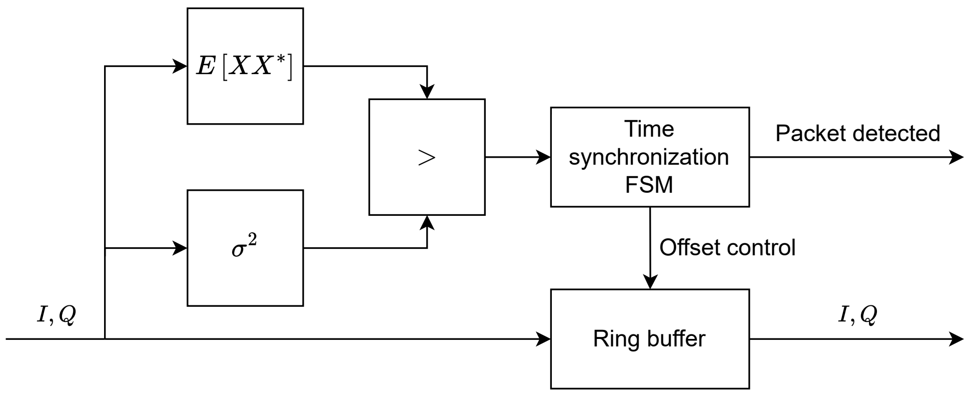


Figure 10: STF detection and timing synchronization block diagram, with numerator, denominator, and FSM abstracted.

4.1 Autocorrelation numerator

A block diagram of the implemented autocorrelation numerator $E[XX^*]$ is shown in Figure 11. The system can be divided to major three parts: complex multiplication, cumulative sum and magnitude calculation. The first part calculates complex valued multiplication between current sample and a complex conjugate delayed one. After the multiplication, the result is used to calculate the correlation with help of an auxiliary buffer for delayed complex multiplier outputs and the last iteration of the correlation output. Lastly, the magnitude of the correlation is calculated with an estimation approximation optimization, and this magnitude is outputted when a preset constant value has been exceeded by a counter since last reset.

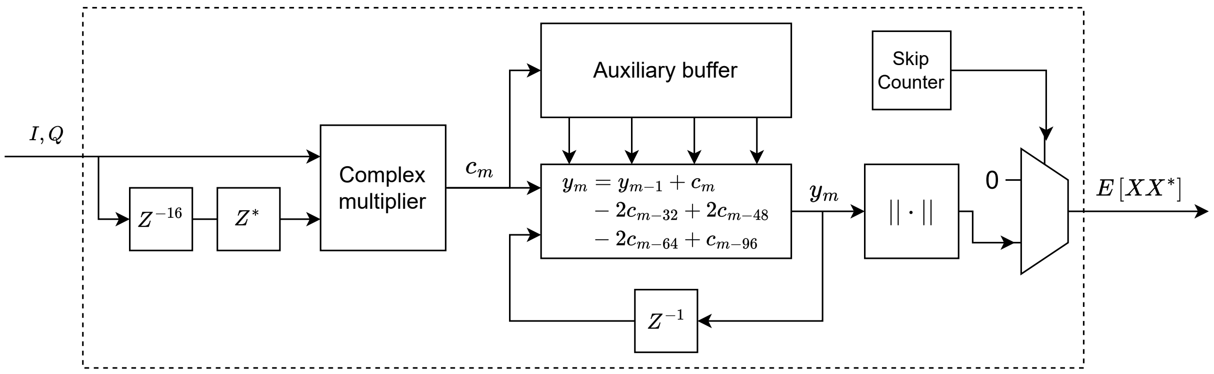


Figure 11: STF autocorrelator's numerator block diagram.

The most important part of the numerator's design is the cumulative sum, that utilizes the preamble's repeating pattern exploitation. As the STF's base sequence S repeats every $N_S = 16$ samples with $\beta = 1$ configuration, autocorrelation for a single S can be simplified by comparing sample at current step to a delayed delayed version of itself. Thus, the autocorrelation equation for a base sequence becomes

$$R_{SS}(m) = \sum_{n=0}^{N_S-1} S_{n+m} S^*_{n+m+N_S}. \tag{4}$$

However, as the cover sequence c_μ is added on top of S, this simplification must be modified. As the cover sequence only periodically changes the sign of S, then for subsequent sequences that have opposite sign with their delayed version, the Equation (4) becomes

$$R_{-SS}(m) = -R_{SS}(m) = -\sum_{n=0}^{N_S-1} S_{n+m} S^*_{n+m+N_S}. \tag{5}$$

Therefore, the autocorrelation for the full time domain STF with c_μ for $\{\beta = 1, \mu = 1\}$ can be defined with combination of equations (4) and (5) as

$$\begin{aligned} R_{xx}(m) = {} & R_{-SS}(\mathrm{STF}_{16}, \ldots, \mathrm{STF}_{31}) + R_{-SS}(\mathrm{STF}_{32}, \ldots, \mathrm{STF}_{47}) \\ & + R_{SS}(\mathrm{STF}_{48}, \ldots, \mathrm{STF}_{63}) + R_{-SS}(\mathrm{STF}_{64}, \ldots, \mathrm{STF}_{79}) \\ & + R_{SS}(\mathrm{STF}_{80}, \ldots, \mathrm{STF}_{95}) + R_{SS}(\mathrm{STF}_{96}, \ldots, \mathrm{STF}_{111}), \end{aligned} \tag{6}$$

where STF_n is the nth term of STF in time domain.

While the packet detection's numerator could be naively calculated with autocorrelation Equation (6), it requires at least six complex multiplications in parallel. Another option could be calculating one sum at a time in serial, after collecting all of the samples. The drawback for serialization is increased latency and processing delay, which requires the serializer to calculate the sums faster than the samples arrive in order to support real-time correlation.

A way to circumvent both latency and complex multiplier parallelization is to calculate the autocorrelation as a flow, which combines both serialization and parallel sum calculation. As samples arrive to the numerator as a stream due to transmission serialization, this allows unrolling the Equation (4) to a cumulative sum. In cumulative sum, each step is a sum of the result at past step and current inputs. The sum can be then calculated in an iterative manner with a running total, that is defined as

$$y(m) = y(m-1) + x_m - x_{m-Z},$$

where $y(m)$ is the rolling total at current cycle, $y(m-1)$ is the total at previous cycle, x_m is the sample at current cycle and x_{m-Z} is a sample from Z cycles ago. Substituting x with complex multiplications from autocorrelation Equation (4), the running total for single base sequence sum with length of $N_S = 16$ is

$$y(m) = y(m-1) + x_m x_{m-16}^* - x_{m-16} x_{m-32}^*. \tag{7}$$

However, the cumulative sum still requires two complex multiplications in parallel: one to compute current step $x_m x_{m-16}^*$ and other for the delayed step $x_{m-16} x_{m-32}^*$. To reduce number of complex multiplications, the $x_m x_{m-16}^*$ term can be denoted by

$$c(m) = x_m x_{m-16}^*. \tag{8}$$

This allows to rewrite complex multiplication terms in Equation (7) as

$$z(m) = c(m) - c(m-16),$$

where one may observe that the latter term is simply delayed operation of the first multiplication. Thus, the total sum Equation (7) becomes

$$y(m) = y(m-1) + z(m). \tag{9}$$

For autocorrelation R_{-SS}, where subsequent base sequences had opposite signs, the sign of $z(m)$ is similarly simply inverted.

As the autocorrelation reaches it maximum at position zero, the STF autocorrelation for $\{\beta = 1, \mu = 1\}$ will have its maximum at $m = 111$. Substituting the sums with y terms from Equation (9), the derived STF autocorrelation Equation (6) can be written as a cumulative sum. Additionally, the structure of autocorrelation allows cumulative sum to use only one running total. This term is denoted with y_{xx}. By dropping the additional $y(m-1)$ total sum terms, this yields:

$$\begin{aligned}
y_{xx}(m) &= y_{xx}(m-1) + R_{SS}(\mathrm{STF}_{96},\ldots,\mathrm{STF}_{111}) + R_{SS}(\mathrm{STF}_{80},\ldots,\mathrm{STF}_{95}) \\
&\quad + R_{-SS}(\mathrm{STF}_{64},\ldots,\mathrm{STF}_{79}) + R_{SS}(\mathrm{STF}_{48},\ldots,\mathrm{STF}_{63}) \\
&\quad + R_{-SS}(\mathrm{STF}_{32},\ldots,\mathrm{STF}_{47}) + R_{-SS}(\mathrm{STF}_{16},\ldots,\mathrm{STF}_{31}) \\
&= y_{xx}(m-1) + y_0(m) + y_1(m-16) - y_2(m-32) + y_3(m-48) \\
&\quad - y_4(m-64) - y_5(m-80) \\
&\overset{y_n=z_n}{=} y_{xx}(m-1) + z_0(m) + z_1(m-16) - z_2(m-32) + z_3(m-48) \\
&\quad - z_4(m-64) - z_5(m-80) \\
&= y_{xx}(m-1) + c(m) - c(m-16) + c(m-16) - c(m-32) \\
&\quad - c(m-32) + c(m-48) + c(m-48) - c(m-64) - c(m-64) \\
&\quad + c(m-80) - c(m-80) + c(m-96) \\
&= y_{xx}(m-1) + c(m) - 2c(m-32) + 2c(m-48) - 2c(m-64) \\
&\quad + c(m-96).
\end{aligned}$$

Thus, the cumulative sum of the time domain STF autocorrelation's numerator is

$$y_m = y_{m-1} + c_m - 2c_{m-32} + 2c_{m-48} - 2c_{m-64} + c_{m-96}. \tag{10}$$

By utilizing an auxiliary buffer to hold past values, the cumulative sum can be calculated using binary additions, subtractions and shifts. Nevertheless, the utilized architecture's memory elements must support reading values at the specific indexes, which may not be always available. For example, dedicated BRAM elements typically allow only limited access to the data with relatively large granularity, which would be insufficient in given use case. As such, look-up table-based memory should be utilized, as they efficiently support access with smaller granularity. In the proposed implementation, the buffer is implemented with a shift-register, as such design approach has lowest resource utilization on the target FPGA architecture. However, a more architecture-independent design approach would utilize a LUT-based ring buffer, due to its availability and predictable resource cost regardless of the FPGA architectures.

Additionally, the numerator requires magnitude calculation once the cumulative sum is calculated. Utilizing a magnitude estimation by approximation allows avoiding calculating square-root in magnitude equation for a complex value:

$$|z| = \sqrt{\mathrm{I}^2 + \mathrm{Q}^2}.$$

By using approximation to avoid square-root operation, the magnitude becomes

$$|z| \approx A \max(|\mathrm{I}|, |\mathrm{Q}|) + B \min(|\mathrm{I}|, |\mathrm{Q}|), \tag{11}$$

where A and B are constants between 0 and 1 [60].

However, the scaling requires multiplication with constants that are decimal numbers. To avoid using multipliers, the constants should be selected to be binary fractional numbers if possible, as such approach allows reducing the multiplication to trivial binary additions and shifts. For example, in the proposed design where the constant values are $A = 1$ and $B = 0.25$, the multiplication with 0.25 can be calculated with

$$0.25x = x \gg 2,$$

where $x \gg y$ is binary right shift that shifts bits in x by y bits to the right. However, the selected values amount to a maximum absolute error of 11.6 % due to the approximation [60].

Another approach to calculate magnitude is coordinate rotation digital computer (CORDIC) algorithm, which could be used to calculate magnitude with better accuracy and, depending on the algorithm implementation, higher clock speed [61, 62]. Additionally, CORDIC would simultaneously allow calculating the phase error of the received signal in parallel to magnitude, which is required during the phase synchronization. However, due to CORDIC requiring a number of steps to improve its accuracy, its iterative design approach adds latency with each step, while the unrolled version requires a multiplier at each step. Furthermore, phase correction is out of scope for this implementation. As such, the magnitude estimation by approximation benefits from reduced resource utilization and processing latency at the expense of accuracy.

Lastly, the autocorrelation numerator includes a counter that skips the first 96 correlation outputs after reset. The reason for this counter is to avoid corner cases where a low-power signal with some autocorrelation would cause detection after reset. As the estimated signal power is zero after reset and is used to normalize the correlation, signals with any autocorrelation properties will cause a detection to occur. Additionally, as the maximum autocorrelation for STF requires 112 samples, skipping first 96 outputs of the correlation will be sufficient.

4.2 Autocorrelation denominator

A block diagram for the implemented autocorrelation denominator σ^2 is shown in Figure 12. The main logic is divided into three parts: squared-magnitude complex calculation, a cumulative sum of signal power and scaling output with a constant factor. The squared-magnitude complex is obtained from absolute value of a current sample with complex multiplier. The output value of the multiplication is used to accumulated with a running total over the STF preamble, delayed multiplier output and last iteration of the total. Finally, the cumulative sum is scaled with α to calculate normalized autocorrelation comparison.

The most important algorithm for the denominator is the estimated signal energy, that is defined as

$$E_s = \int_{-\infty}^{\infty} |x(t)|^2 \, dt,$$

which is for finite-length discrete signals

$$E_s = \sum_{n=-N}^{N} |x(n)|^2 \,. \tag{12}$$

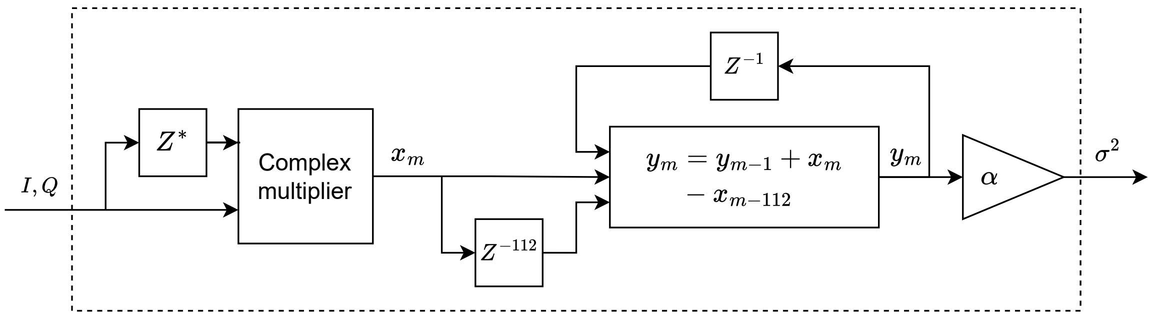


Figure 12: STF autocorrelator denominator block diagram.

When the discrete signal is complex valued, the term for discrete signal energy becomes

$$|x(n)|^2 = |\mathrm{I} + \mathrm{Q}|^2 = \sqrt{\mathrm{I}^2 + \mathrm{Q}^2}^{\,2} = \mathrm{I}^2 + \mathrm{Q}^2.$$

For complex values, the term can be further reduced to

$$|x(n)|^2 = \mathrm{I}^2 + \mathrm{Q}^2 = (\mathrm{I} + \mathrm{Q})(\mathrm{I} + \mathrm{Q})^* = (\mathrm{I} + \mathrm{Q})(\mathrm{I} - \mathrm{Q}).$$

Thus, the squared magnitude of a complex value can be calculated by multiplying the sample with a conjugate version of itself.

As with the numerator's autocorrelation, the discerete energy sum (12) can be calculated with a cumulative sum. Due to the sum being constrained only to the STF's power in order to normalize the autocorrelation, the N term is the length of STF in time domain. For $\{\beta = 1, \mu = 1\}$ configuration, the length was 112 samples. Thus, the autocorrelation denominator's cumulative sum is

$$y_m = y_{m-1} + x_m x_m^* + x_{m-112} x_{m-112}^*. \tag{13}$$

Likewise in numerator, the $x_{m-112}x_{m-112}^*$ term can be delayed from the initial calculation, which reduces the required complex multipliers to one.

Additionally for denominator, instead of normalizing the autocorrelation function with a division, the power scaling factor α is used multiply the power estimation output in normalization process. Such approach avoid the division operation used in normalization, which is a resource or latency intensive depending on the utilized algorithm. As the maximum in normalized autocorrelation is located at value of 1, the maximum for the normalized autocorrelation, as was defined in Equation (3) in Section 2.1, can be located by calculating inequality

$$\begin{aligned} R_{xx} &\leq \frac{E[XX^*]}{\sigma^2} \\ R_{xx} \cdot \sigma^2 &\leq E[XX^*]. \end{aligned} \tag{14}$$

In optimal theoretical case, where the signal power is estimated perfectly, the power scaling factor should be $R_{xx} = 1$, which would reduce the Equation (14) to a simple equality. However, the length of estimated signal power and correlation calculations may differ because of implementation specific issues, causing a bias in the autocorrelation output. In this thesis's implementation, the estimated power of STF is calculated from its full length of 112 samples, while the correlation's calculation uses only 96 elements due to utilization of delayed autocorrelation approach. Thus, in the ideal case, the factor of $96/112 \approx 0.857$ should be chosen as it normalizes the correlation and the estimated power perfectly. In the rest of this thesis, the scaling factor of estimated signal power R_{xx} is denoted with α as it combines both the detection threshold and scaling.

Unfortunately, due to noise and other detrimental effects from channel and hardware quantization, the estimated signal power and correlation of a receiver signal fluctuate from ideal. To avoid events where the correlation fails to exceed or equal signal power because of the external factors, the scaling factor should be smaller than in the ideal case as it allows the calculated correlation to be imperfect. As a result, lowering α improves signal detecting in noisy environments by allowing detection of channel corrupted STF. However, decreasing estimated signal power can cause false positive detection events to occur in presence of interfering signals. Due to lowering the required level of correlation, such action may additionally cause unwanted signals to exceed the scaled signal power and thus result to a false detection of a packet. Signals that have repeating patterns, and thus high correlation with themselves, are especially susceptible to cause erroneous packet detection.

Lastly, scaling the estimated signal power with α requires a multiplication operation. As with magnitude calculation in Equation (11), binary operations can be utilized to avoid multiplication. For example, the computation of scaling with 0.75 would be

$$0.75x = (x \gg 1) + (x \gg 2).$$

Thus, α should be chosen so that the multiplication can be calculated using only the cheap binary operations. In this thesis, two scaling factors used are 0.75 and 0.6875, that can be calculated as

$$0.6875x = (x \gg 1) + (x \gg 3) + (x \gg 4).$$

Both of these constants have low resource requirements while providing improved signal detecting capability without causing false positive detection occurrences. While 0.75 has lower resource utilization than 0.6875, it requires received signal to have better signal-to-noise ratio.

4.3 Timing synchronization

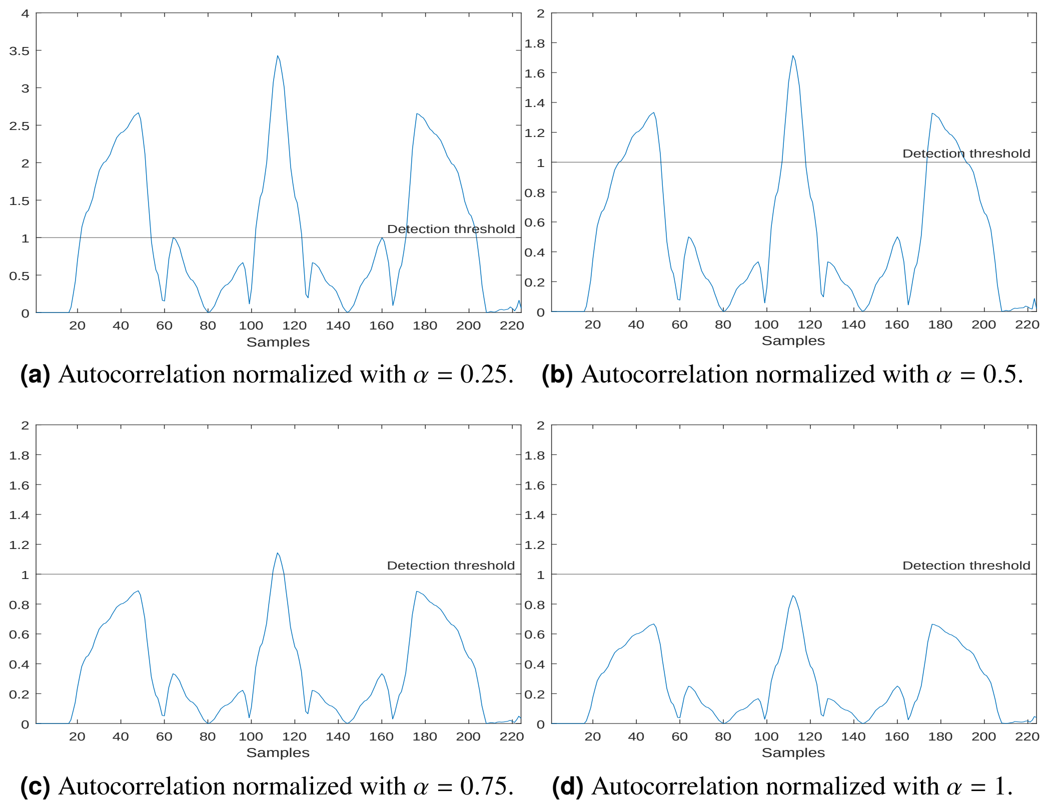
Due to the proposed implementation's autocorrelation being calculated with a comparison instead of a division, the output of the comparison is a Boolean value. As the value only indicates if the correlation exceeds estimated signal power, further logic is required to locate the maximum of the autocorrelation, also known as peak. Once this peak has been located, the received packet can be timing synchronized to its location as it indicates the start of user data. A finite-state machine is utilized to determine the peak of a possibly detected STF, while an additional ring buffer is utilized during the timing synchronization.

To analyze decision logic for packet detection and timing synchronization, the autocorrelation denominator's normalization Equation (14) can be alternatively written as

$$1 \leq \frac{E[XX^*]}{R_{xx} \cdot \sigma^2}, \tag{15}$$

where $R_{xx} = \alpha$ is the power scaling factor. In the Equation (15), the estimated power is scaled with the factor. Lowering the scaling factor increases correlation, which increases the chances that the correlation will exceed the threshold of one. Thus, the detection occurs once autocorrelation function exceeds the scaled estimated signal power, or in alternative words, normalized correlation is greater or equal to one.

The effects of α on the detection and timing synchronization are visualized in Figure 13 with an ideal STF sequence. For power scaling factors 0.25 and 0.50, both the autocorrelation's main lobe and side lobes exceed the detection threshold. As the maximum correlation is located in the main lobe's peak, the timing synchronization logic should disregard the side lobes and only consider the main lobe. Such logic is further complicated if the received presumably has low signal-to-noise ratio (SNR), in which case one or more side lobes may not exceed the threshold.



(a) Autocorrelation normalized with $\alpha = 0.25$. **(b)** Autocorrelation normalized with $\alpha = 0.5$.

(c) Autocorrelation normalized with $\alpha = 0.75$. **(d)** Autocorrelation normalized with $\alpha = 1$.

Figure 13: Normalized autocorrelation of an ideal STF with different power scaling factors.

That said, if the scaling is increased to 0.75, only the main lobe will exceed the threshold. In such case the timing synchronization becomes trivial, as the peak can be located from the center of the lobe. Regardless, if α is increased over the theoretical maximum of $96/112$ as in the case of $\alpha = 1$, then the autocorrelation will never exceed the threshold. In this thesis, only the scaling factors of 0.6875 and 0.75 are considered, which significantly simplifies timing synchronization logic as the side lobes do not exceed the threshold unless amplified by an interfering signal.

The state diagram of the peak detection and timing synchronization FSM can be observed in Figure 14. The state machine works as follows: after a reset, a positive detection counter C_o and a negative detection counter C_z are initialized to zero. The FSM then waits until a detection occurs, upon which the current position of ring buffer's write pointer is stored to W_{stored} and the C_o counter is incremented. The logic then loops, incrementing either counter each time a new I/Q-sample is received: C_o upon detection and C_z when no detection has occurred. If the negative detection counter exceeds a threshold T_z, the loop is exited and the positive detection counter is compared to a second threshold T_o. If the C_o exceeds its threshold, STF is declared to be located, its peak is calculated with

$$P_{loc} = \lfloor C_o/2 \rfloor,$$

and the ring buffer's read pointer is corrected to

$$R_{new} = W_{stored} + P_{loc}.$$

Lastly, the counters are cleared and the FSM begins to wait for a new detection.

Function of the thresholds T_o and T_z in the state machine is to avoid false positive detection from noise. If the power scaling factor α is set small and the design would not include the thresholds, then detection events caused by abrupt noise spikes would be erroneously declared as received packet. Therefore, increasing the thresholds improve design's resistance to noise. The drawback to such method is the possibility to miss the STF, as too large threshold may exceed the width of an ideal STF's peak or a smaller peak due to diminishing effects from noise.

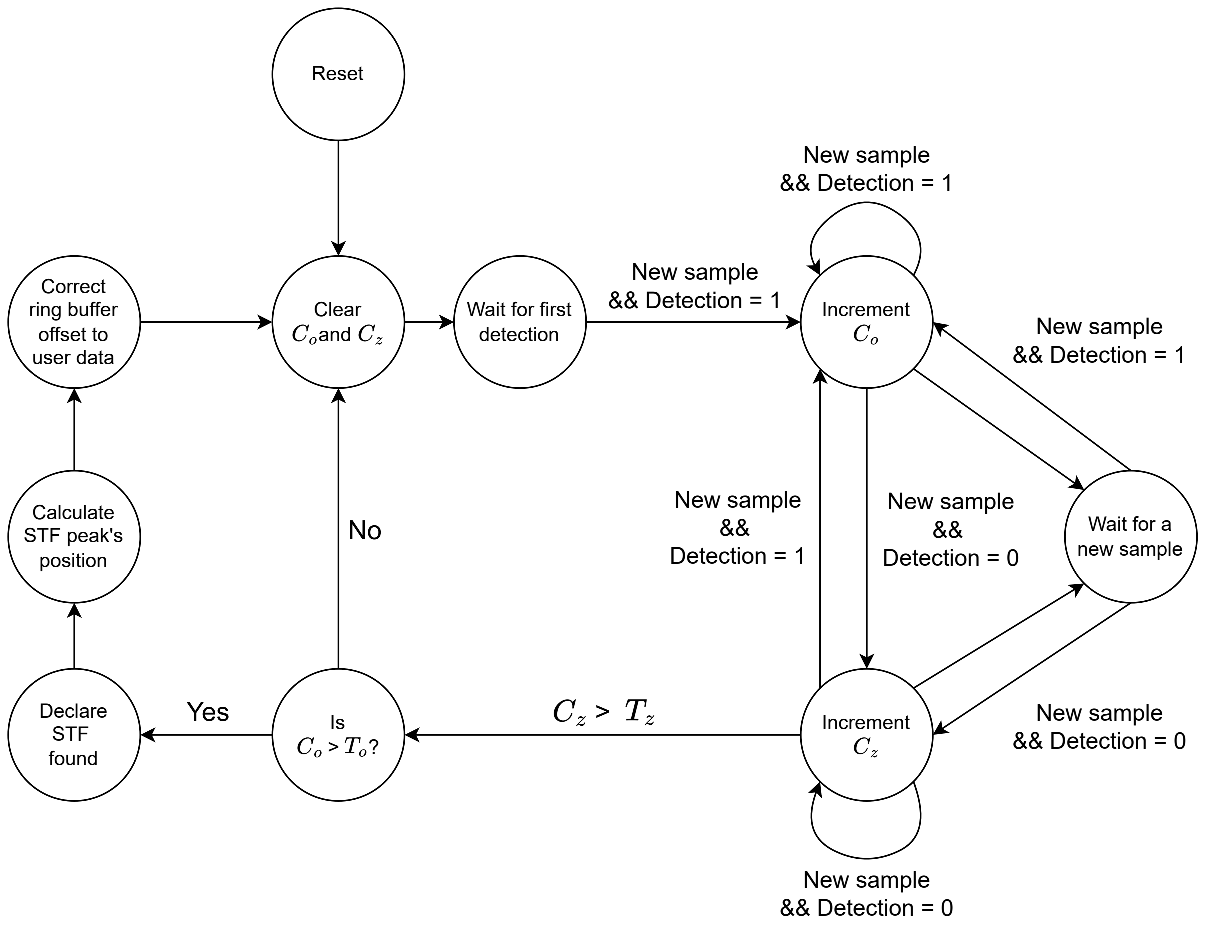

Reset
Correct ring buffer offset to user data
Clear C_o and C_z
Wait for first detection
New sample && Detection = 1
Increment C_o
New sample && Detection = 1
New sample && Detection = 1
Calculate STF peak's position
No
New sample && Detection = 1
New sample && Detection = 0
Wait for a new sample
Declare STF found
Yes
Is $C_o > T_o$?
$C_z > T_z$
Increment C_z
New sample && Detection = 0
New sample && Detection = 0

Figure 14: State diagram of the timing synchronization FSM.

4.4 Expanded β and μ support

The proposed STF packet detection and timing synchronization implementation supports only $\{\beta = 1, \mu = 1\}$ configuration of the DECT-2020 New Radio physical layer. The rigid architecture is caused by the autocorrelation numerator's cumulative sum with the fixed delay elements, which enables calculating correlation in a hardware efficient iterative manner. Likewise for the autocorrelation denominator, the fixed delay element controlling the length of running total confines the proposed system to only a one pair of β and μ values. However, this section explores how the implementation could be further expanded to function with all DECT's physical layer $\{\beta, \mu\}$ configurations.

In order for the numerator to function with other configuration values, various modifications with different design approaches are be plausible. Nonetheless, supporting different β factor values is the primary issue, as the variable determines the required delay for the correlation. This is due to the factor specifying the size for N_S term, that in turn defines the length of the time domain base sequence S. Likewise, a number of sequences that sets the needed number of summations in correlation Equation (6) is characterized by μ.

In order to achieve support for different configurations, one solution would be to expand the numerator's auxiliary buffer's size and multiplex between different delay sizes. The increased buffer is required to accommodate a larger number of samples due to the increased delay set by other configuration values. As the number of terms in Equation (4) equals size of N_S, and the number of base sequences depends on μ, the required auxiliary buffer size is then defined as

$$N_{\text{buf}} = \begin{cases} N_S \cdot 6 = 16 \cdot \beta \cdot 6, & \text{when } \mu = 1 \\ N_S \cdot 8 = 16 \cdot \beta \cdot 8, & \text{otherwise.} \end{cases}$$

Additionally, to allow dynamic change of β, the term Z in c_{m-Z} indices would have to be runtime adjustable. In hardware, a dynamic switch between different delays during operation could be achieved with multiplexing. Rewriting Equation (10) to function with any β, and thus any delay, the new numerator cumulative sum is

$$y_m = y_{m-1} + c_m - 2c_{m-32\cdot\beta} + 2c_{m-48\cdot\beta} - 2c_{m-64\cdot\beta} + c_{m-96\cdot\beta}. \tag{16}$$

As the delay element in complex multiplier output c_m depends on the length of time domain pattern S, the term can be redefined in cases of $\beta > 1$ from Equation (8) to

$$c(m) = x_m x^*_{m-N_S},$$

As the numerator delays are only integer multiples of β, a simple downsampler using decimation could be used by taking every β^{th} sample. To avoid aliasing from higher frequencies due to the downsampling, a lowpass filter is required to attenuate undesired frequencies before decimation takes place.

The numerator has rudimentary support for $\mu \neq 1$ values, as the first seven repetitions are always the same regardless of μ. However, using fewer amounts of repetitions will sacrifice autocorrelation's accuracy, may cause false positives to occur as the STF would be longer than sum's memory, and add extra logic to the sample ring buffer offset control. As the STF preamble with $\mu > 1$ have the two additional base sequences, the packet detection finite-state machine controlling the ring buffer offset would have to determine how many sequences have been sent and then skip them. To modify the implementation to correctly function with other μ values, the main issue arises from two new sums that would have to be taken into consideration.

The autocorrelation for time domain STF with c_μ for $\{\beta = 1, \mu \in \{2, 4, 8\}\}$ configuration would be

$$\begin{aligned} R_{xx}(m) = {} & R_{-SS}(\text{STF}_{16}, \ldots, \text{STF}_{31}) + R_{-SS}(\text{STF}_{32}, \ldots, \text{STF}_{47}) \\ & + R_{SS}(\text{STF}_{48}, \ldots, \text{STF}_{63}) + R_{-SS}(\text{STF}_{64}, \ldots, \text{STF}_{79}) \\ & + R_{SS}(\text{STF}_{80}, \ldots, \text{STF}_{95}) + R_{SS}(\text{STF}_{96}, \ldots, \text{STF}_{111}) \\ & + R_{SS}(\text{STF}_{112}, \ldots, \text{STF}_{127}) + R_{SS}(\text{STF}_{128}, \ldots, \text{STF}_{143}). \end{aligned}$$

For cumulative sum, this becomes

$$\begin{aligned}
y_{xx}(m) &= y_{xx}(m-1) + R_{SS}(\mathrm{STF}_{128}, \dots, \mathrm{STF}_{143}) + R_{SS}(\mathrm{STF}_{112}, \dots, \mathrm{STF}_{127}) \\
&\quad + R_{SS}(\mathrm{STF}_{96}, \dots, \mathrm{STF}_{111}) + R_{SS}(\mathrm{STF}_{80}, \dots, \mathrm{STF}_{95}) \\
&\quad + R_{-SS}(\mathrm{STF}_{64}, \dots, \mathrm{STF}_{79}) + R_{SS}(\mathrm{STF}_{48}, \dots, \mathrm{STF}_{63}) \\
&\quad + R_{-SS}(\mathrm{STF}_{32}, \dots, \mathrm{STF}_{47}) + R_{-SS}(\mathrm{STF}_{16}, \dots, \mathrm{STF}_{31}) \\
&= y_{xx}(m-1) + y_0(m) + y_1(m-16) + y_2(m-32) + y_3(m-48) \\
&\quad - y_4(m-64) + y_5(m-80) - y_6(m-96) - y_7(m-112) \\
&\overset{y_n = z_n}{=} y_{xx}(m-1) + z_0(m) + z_1(m-16) + z_2(m-32) + z_3(m-48) \\
&\quad - z_4(m-64) + z_5(m-80) - z_6(m-96) - z_7(m-112) \\
&= y_{xx}(m-1) + c(m) - c(m-16) + c(m-16) - c(m-32) \\
&\quad + c(m-32) - c(m-48) + c(m-48) - c(m-64) \\
&\quad - c(m-64) + c(m-80) + c(m-80) - c(m-96) \\
&\quad - c(m-96) + c(m-112) - c(m-112) + c(m-128) \\
&= y_{xx}(m-1) + c(m) - 2c(m-64) + 2c(m-80) - 2c(m-96) \\
&\quad + c(m-128),
\end{aligned}$$

Thus, the running total of the time domain STF autocorrelation's numerator for $\{\beta = 1, \mu \in \{2, 4, 8\}\}$ is

$$y_m = y_{m-1} + c_m - 2c_{m-64} + 2c_{m-80} - 2c_{m-96} + c_{m-128}. \tag{17}$$

If we compare the derived Equation (17) to the implementation's cumulative sum Equation (10) in Section 4.1, it can be observed that the order of constants remain unchanged while the delays increase by $2 \cdot N_S = 32$. Hence, modifying Equation (16) to work for any physical layer configuration yields

$$y_m = y_{m-1} + c_m - 2c_{m-32\cdot\beta-k} + 2c_{m-48\cdot\beta-k} - 2c_{m-64\cdot\beta-k} + c_{m-96\cdot\beta-k}, \tag{18}$$

where

$$k = \begin{cases} 0, & \text{when } \mu = 1 \\ 2N_S = 2 \cdot 16 \cdot \beta, & \text{otherwise.} \end{cases}$$

Lastly, the skip counter in numerator that is responsible for filtering out invalid correlation magnitudes must also be scaled. As the estimated signal power length will increase in conjunction with the correlation, the preset value used to determine invalid correlations should also be increased. As the preset constant depends on the length of STF and S, a dynamic variant of this preset could be calculated as follows:

$$\text{Skip threshold} = \begin{cases} 6 \cdot N_S \cdot \beta, & \text{when } \mu = 1 \\ 8 \cdot N_S \cdot \beta, & \text{otherwise.} \end{cases}$$

For the STF autocorrelator denominator, only the number of samples is a relevant numeric as the module implements a simple cumulative sum. Therefore, if the position of the last element could be modified during runtime, the module would work with any physical layer configuration when a single antenna system is considered. The rolling sum Equation (13) in Section 4.2 for the denominator could then be rewritten as

$$y_m = y_{m-1} + x_m x_m^* + x_{m-p} x_{m-p}^*, \tag{19}$$

where

$$p = \begin{cases} 7 \cdot N_S = 7 \cdot 16 \cdot \beta, & \text{when } \mu = 1 \\ 9 \cdot N_S = 9 \cdot 16 \cdot \beta, & \text{otherwise.} \end{cases}$$

For both numerator and denominator, runtime adjusting of the delays requires careful management. As both modules rely on cumulative sums to reduce amount of parallel multiplications, algorithms (18) and (19) respectively, then the rolling sums must be zeroed and delayed samples ignored when configuration changes occurs. In hardware implementations, the former operation is a simple reset, but the latter depends on the used hardware architecture. For example, resetting all memory elements during runtime would be expensive time-wise, unless supported by hardware, yet could be emulated by multiplexing read element with zeros until the memory has been filled up to the required delay. Another solution for runtime changeable delays could be implementing one or multiple first-in-first-out buffers, which use an auxiliary counter to determine when to read from their respective internal buffer.

A major issue with the proposed modifications to the implemented STF packet detection design is how the receiver is aware of the selected configuration. In the optimal case, receiver and transmitter synchronize the used β and μ values between each other before or during transmissions. However, as the DECT-2020 New Radio is designed towards network topology where new clients for a given access point can be arbitrarily added and removed, such assumption will not hold if various physical layer values are allowed within the network. To avoid conflicts with invalid configurations, the receiver would have to blindly search for all possible combinations and select the appropriate β and μ value for a given decoded STF. As the proposed implementation does not support such functionality without multiple parallel instances, the upper layers of the protocol must handle cases where physical layer configurations between transmission participants do not match.

5 Results

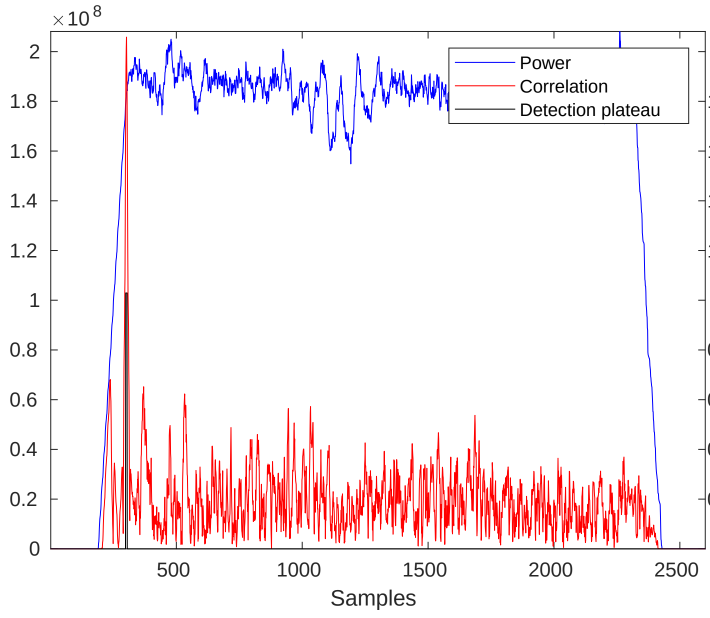
Evaluation of the proposed thesis implementation was conducted with simulation in software. The packet detection design was examined against a MATLAB reference for logic assessment with a captured DECT-2020 New Radio physical layer packet. Additionally, due to the proposed implementation using a fixed power scaling factor α to scale estimated signal power for autocorrelation normalization, different values were tested in three cases to validate the correctness of the selected scaling factors. The implementation was additionally evaluated against other correlation designs in additive white Gaussian noise (AWGN) with different SNR. Lastly, the thesis design's resource utilization was compared with openwifi project in targeted hardware architecture.

5.1 MATLAB reference comparison

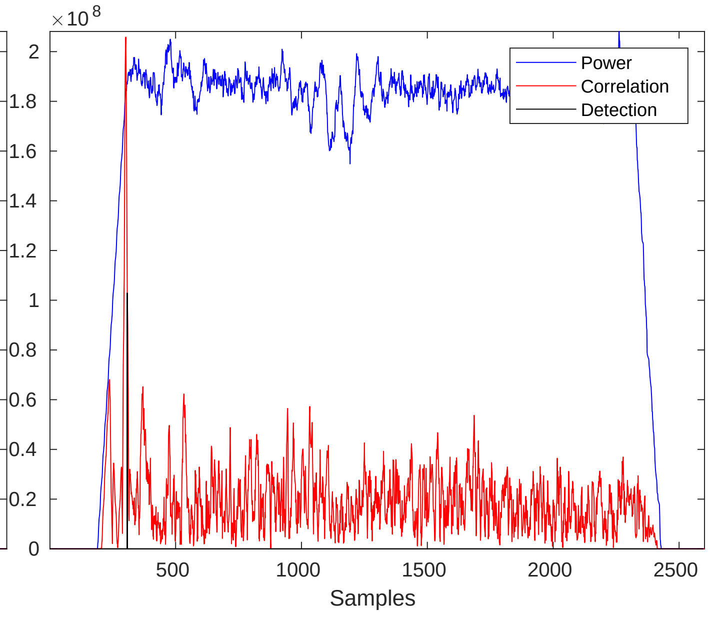
The comparison with MATLAB reference used a captured over-the-air DECT-2020 New Radio transmission with $\{\beta = 1, \mu = 1\}$ configuration. The packet was sent utilizing a commercial Nordic Semiconductor nRF9161 DK development board [63] and recorded with a USRP B210 SDR [64]. The signal begins with a short sample of noise, followed by the desired STF preamble, then by user data and finally ending in noise again.

Both the MATLAB model and proposed implementation calculated signal power and correlation, before trying to determine the STF sequence location. While power and correlation operations are similarly in both reference and implementation, the packet detection algorithm differ slightly. In case of MATLAB, the packet detection causes a plateau due to simplified decision logic: if correlation exceeds the power, a packet is declared as being detected. For the proposed implementation, the STF preamble detection declaration occurs only once and after a delay, as explained in the Section 4.3. The outputs are represented in the Figures 15a and 15b, for MATLAB and thesis implementation respectively. With visual inspection, both figures are identical: the signal power and correlation are of a similar shape and magnitude, while detection occurs on the maximum correlation.

However, closer inspection around the correlation maximum reveals differences of the detection algorithms. For MATLAB reference in Figure 16a, the detection occurs around the correlation peak and forms a plateau due to the simplified decision logic. In contrast, as the proposed implementation's detection declaration is delayed by the decision logic, the detection is delayed from the actual correlation's peak as can be observed in Figure 16b. While the packet occurrence notification has delay compared to MATLAB reference, the implementation corrects this lag with the ring buffer and offset control. With this correction, the timing synchronized outputs for the MATLAB reference and the proposed design are identical for same input, which can be confirmed in Figure 17.

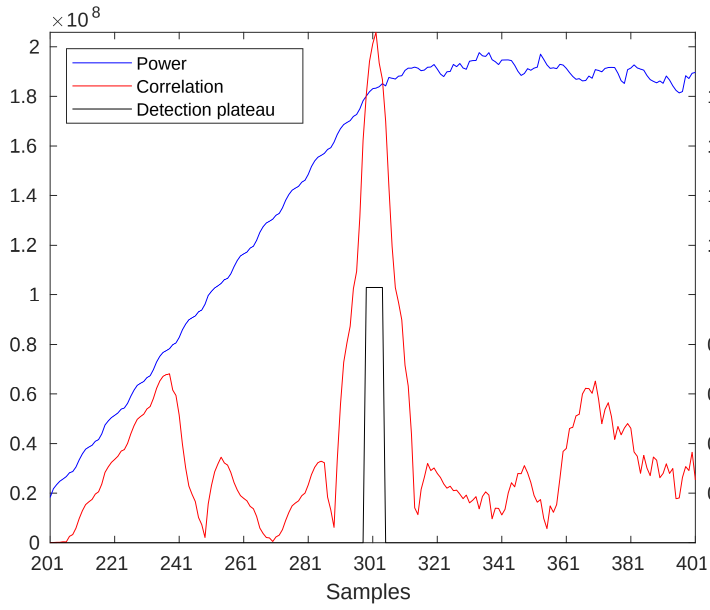


(a) MATLAB reference autocorrelation.

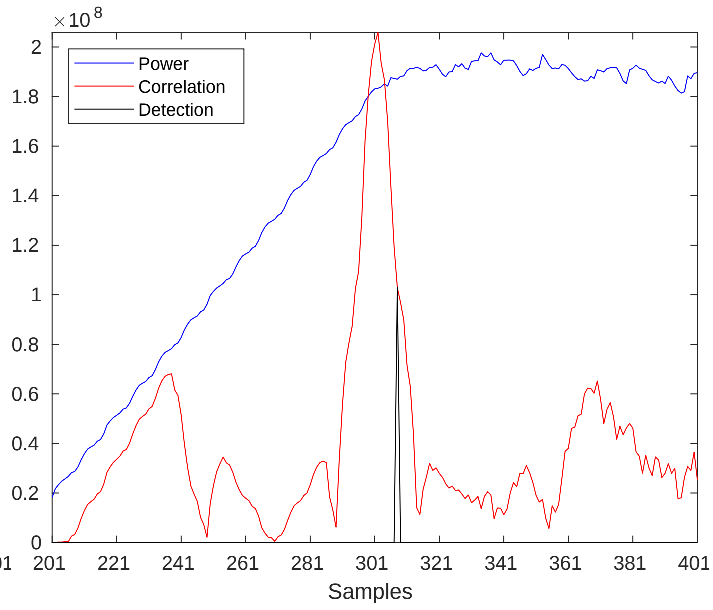


(b) Implemented Verilog autocorrelation.

Figure 15: Comparison of estimated signal power, correlation and preamble detection between MATLAB reference and proposed implemented with a captured over-the-air DECT packet.



(a) MATLAB autocorrelation peak plateau.



(b) Implemented Verilog autocorrelation peak.

Figure 16: Comparison of signal power, correlation and packet detection between MATLAB reference and implemented thesis for a captured over-the-air DECT-2020 New Radio packet centered around maximum correlation.

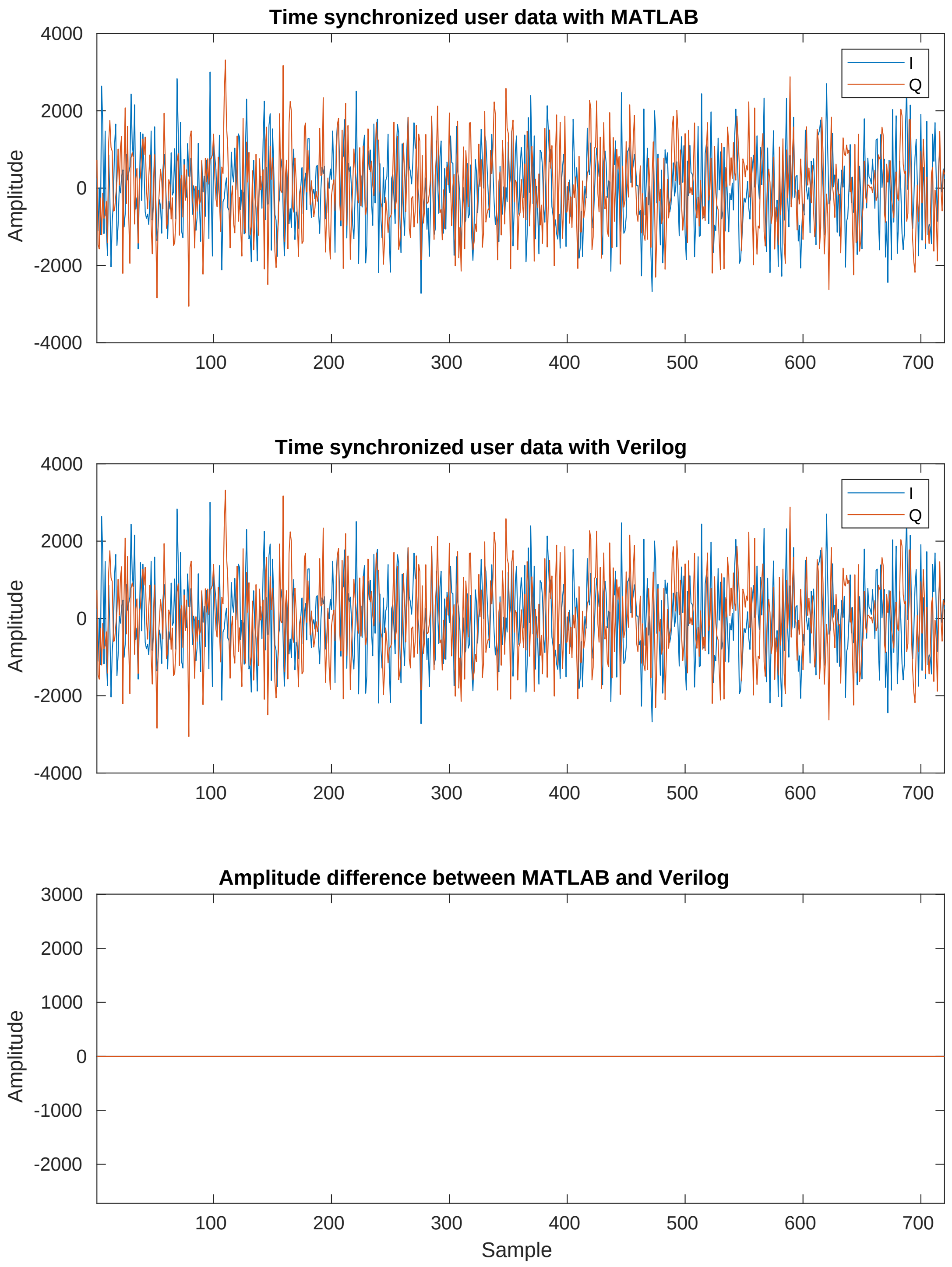


Figure 17: Comparison for a section of captured over-the-air DECT-2020 New Radio packet after timing synchronization in MATLAB reference and thesis Verilog implementation.

5.2 Power scaling factor comparison

The proposed implementation includes a power scaling factor α that is used the control the receiver sensitivity of the packet detection process. Before the comparison between autocorrelation's numerator $E[XX^*]$ and denominator σ^2, the signal power estimation σ^2 is scaled with α factor. The function of this scaling operation is to aid determining if a preamble is present, as the detection is declared once numerator exceeds denominator and lowering signal power reduces SNR requirement for STF premable. However, if the transmission channel has interfering signals that are autocorrelated, lowering the power scaling factor may cause the detector to falsely flag these interfering signals as detected packets. Thus, lowering α improves STF discovery for signals with lower SNR in AWGN channels at the expense of increased false positives in channels with interference.

The power scaling factor comparisons were done over various signal SNRs in AWGN channel for different values of α, that can be calculated with at maximum of three operands using only binary additions and shifts, in three test cases:

- packet detection rate with a ideal STF,
- false positive detection with co-channel interference from constant complex signal, and
- false positive detection with co-channel interference from 802.11 legacy short training field (L-STF).

The range of possible power scaling factor values was limited during the analysis in order to keep the thesis implementation's hardware resource utilization as similar as possible between different factors. As the scaling of the estimated signal power is done using a multiplication, only a limited subset of scaling factor constants allow calculating the multiplication in hardware efficient manner by utilizing only binary addition and bit-shifts.

The packet detection rate for various α values over received signal SNRs can be observed in Figure 18. As expected, the rate improvement correlates with decreasing power scaling factor: 90 % packet rate for $\alpha = 0.75$ requires $\approx$ 10 dB SNR, while factor of 0.5 requires only $\approx$ 2.6 dB for similar rate. However, the figure contains three points-of-interest that should be further analyzed: the perfect, or substantially better, detection rate of small scaling factors, the almost linear improvements between factors around $\alpha = 0.5$, and the almost exponential increase of required SNR for same rate at factors after $\alpha = 0.6875$.

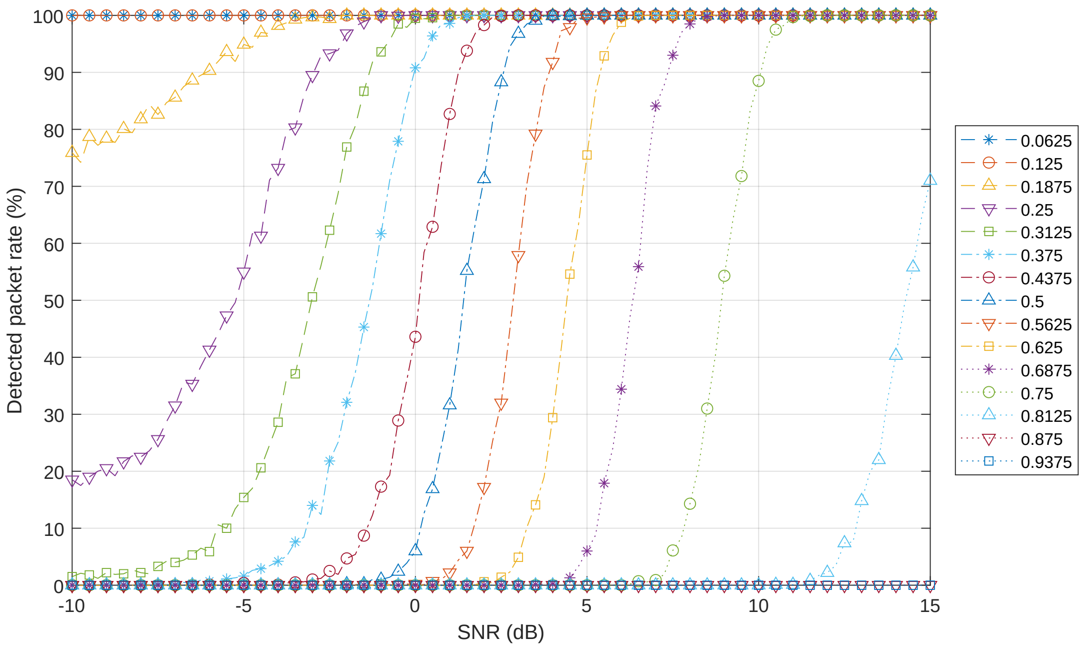


Figure 18: Comparison of the proposed implementation's packet detection rate for ideal generated STF sequence in an AWGN channel with various α over preamble's SNR values.

In case of perfect, or substantially better, packet detection rates for power scaling factors lower than 0.3125, these values cause the estimated signal power to fall close to noise floor. In these cases even white noise may exceed the detection threshold, causing the detector to falsely indicate presence of preamble in channel with only noise, as the Figure 19 indicates. As such, while some of these values would improve packet detection for attenuated signals in noisy channels, they will intrinsically introduce false positive in channels without any intended transmissions. However, the factor of 0.25 could be utilized in applications where $\approx 18\,\%$ false detection rate due to noise is an acceptable trade-off to improve packet detection. Therefore, α smaller than 0.25 should be avoided due to their high false detection rate, while factor of 0.25 could be utilized where false detection rate due to noise is allowed.

The other interesting observation from the Figure 18 is the improvement in detection capability for similar rate increases almost linearly for α from 0.3125 to 0.6875. For example, lowering the α by one factor improves SNR value by $\approx 1.2\,\text{dB}$ for detection rate value at $90\,\%$. On the other hand, for fixed SNR value, change of α by one step improves or worsens detection by approximately $40\,\%$ on average. In comparison for same miss rate, the change from factor 0.75 to 0.6875 lowers preamble's SNR requirement by $\approx 2.9\,\text{dB}$. Furthermore, for factors over 0.6875 the SNR requirement increases rapidly, as $\alpha = 0.75$ has perfect detection rate at 11 dB while factor of 0.8125 is unable to detect any preambles at same SNR. As such, power scaling factor 0.3125 would provide best performance when only the AWGN channel is considered.

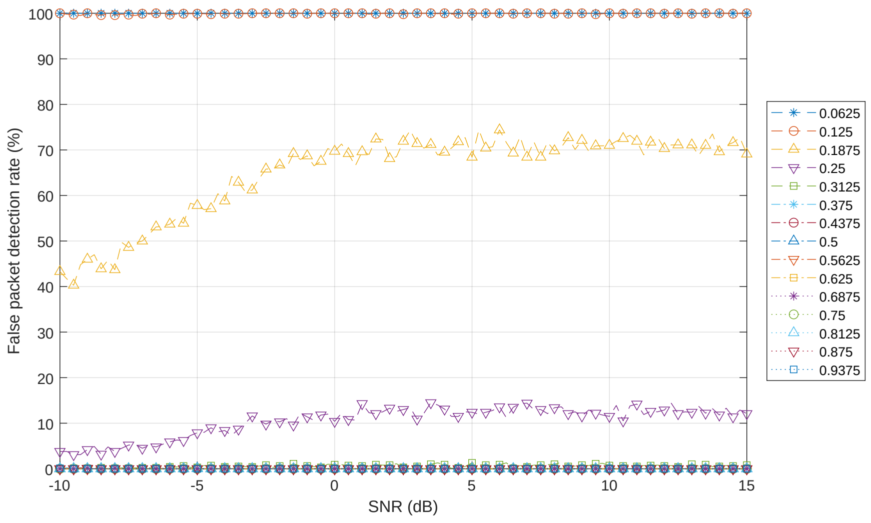
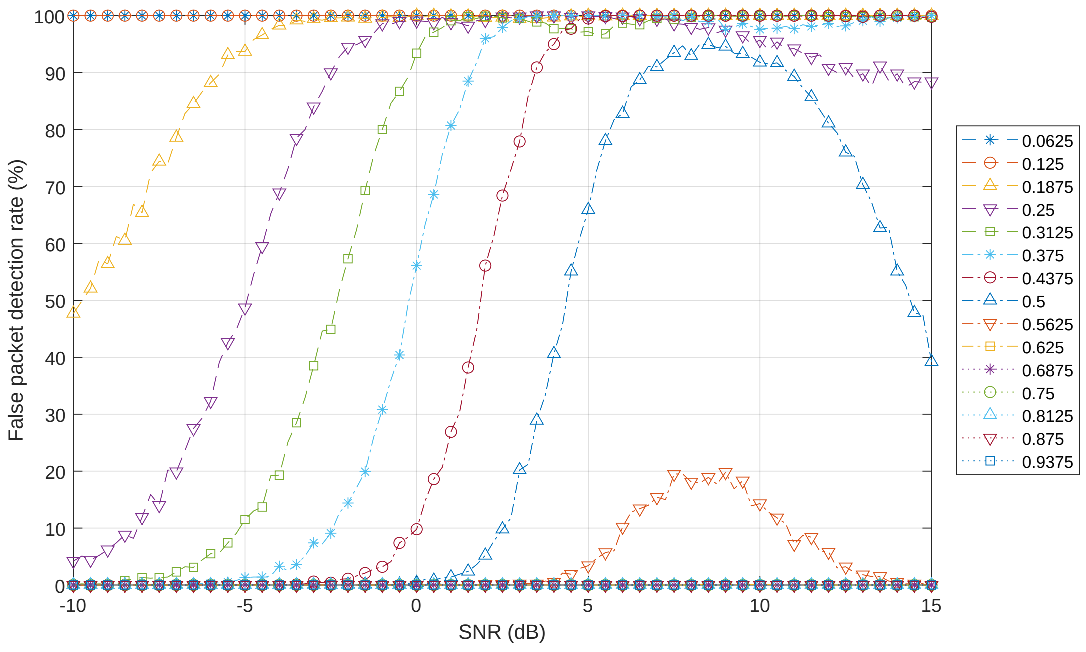


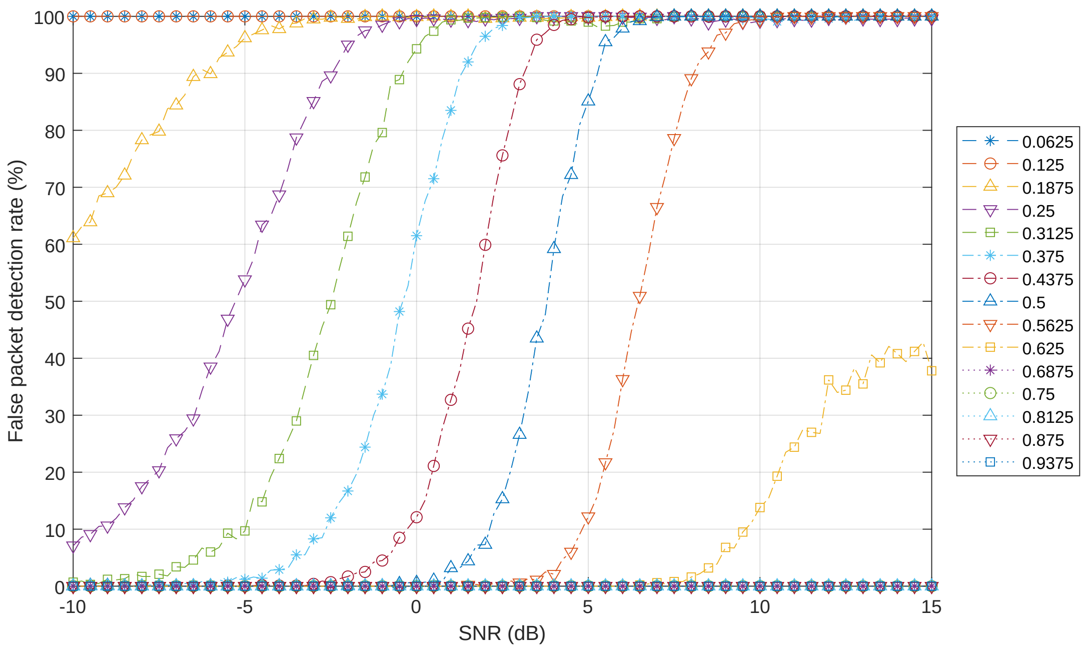
Figure 19: Comparison of the proposed implementation's false packet detection rate for a random noisy signal in an AWGN channel with various α over signal's SNR values.

However, α smaller than 0.6875 requires more complex timing synchronization, as was discussed in Section 4.3. As the detection threshold is lowered, the side lobes in autocorrelation function will also cause correlation to exceed power estimation, causing possibly three packet detection events to occur: two for the side lobes and one for the main lobe. Therefore, to correctly synchronize a given packet to the correlation's maximum peak at the main lobe, a more complex finite state-machine would be needed. Not only would the FSM have to obtain the location of the peak in the presence of side lobes, but additionally in cases where one or both of the side lobes may not exceed estimated signal power. As such, an improved timing synchronization logic that takes the autocorrelation's side lobes into consideration would greatly enhance the autocorrelation-based packet detector's capabilities to detect signals with lower SNR.

The outcome of test cases with co-channel interference signals are shown in Figures 20a, for constant complex signal, and 20b, for 802.11 L-STF. In both cases, power scaling factors larger than 0.625 do not cause any false detection, while factors 0.0625 to 0.3125 have comparable detection rate for both interference and actual STF preamble. In case of $\alpha > 0.625$, addition of the cover sequence c_μ modifies the STF pattern to require sign switching between base sequences, which introduces a restricted form of cross-correlation to the autocorrelation [65]. Hence, sequences that do not follow the restricted form of cross-correlation pattern, such as the L-STF, are slightly less correlated and therefore require lower threshold to cause a false detection. This result can be observed with $0.3125 \leq \alpha \leq 0.625$, which have similar detection rate trend when detecting STF or co-channel interference. However, with interfering signals the false detection rate increase is not a steep and the signals require higher SNR. For example, power scaling factor 0.5 has 90 % STF detection rate at 2.6 dB SNR and same false rate for L-STF at 5.3 dB.



(a) An interfering complex constant signal.



(b) An interfering 802.11 L-STF.

Figure 20: Comparison of the proposed implementation's false packet detection rate for two different co-channel interference signals and various α values in an AWGN channel over interfering signal's SNRs.

Additionally, factors 0.5 and 0.5625 peak at 8.5 dB for the interfering complex constant signal, their false detection rate increasing before the peak and decreasing after the peak when the signal's SNR is increased. This is due to the autocorrelation function of the constant signal achieving its maximum at the peak and remaining constant thereafter, while the estimated signal power increases further in conjunction with the signal's SNR.

To evaluate if a scaling factor has sufficient properties, a quality metric Q_m is proposed. The metric is calculated by

$$Q_m = (R_t) \cdot \left(1 - R_{fc}\right) \cdot \left(1 - R_{fw}\right), \tag{20}$$

where R_t is a packet detection rate for ideal STF, R_{fc} is a false detection rate with constant complex signal, and R_{fw} is the false packet detection rate with L-STF. Hence, Q_m indicates a rate at which given threshold experiences either packet misses or false detection events. A value of 1 indicates that all preambles are detected and no false positives are detected, while value of 0 indicates that all preambles are missed, interfering signal will always cause erroneous detection, or both.

By combining the detection rates for all three test cases to calculate the quality metric Q_m with Equation (20), Figure 21 display the results over each respective signal's SNR. As the false packet detection rates were almost similar for both cases, so are the quality factors. As can be observed from the figures, power scaling factors $[0.375, 0.8125]$ gain Q_m values over 0.2. Additionally, range of α at $0.375 - 0.5625$ have similar structure by forming a downward U-shaped curve. However, only scaling factors of 0.6825 and 0.75 have increasing Q_m that remains at maximum quality, due to their lack of false positive detection. In addition, 0.625 receives the maximum quality, yet falls to 0.6. The U-shaped curves and the 0.625 decreasing are due to their sensitivity to co-channel interference. In channels without any interference, such as the DECT's technology exclusive spectrum, only the STF detection rate in the first test case can be considered a valid metric.

Regardless, 0.6875 and 0.75 were deemed to provide a sufficient balance for α between receiver sensitivity and false positive detection avoidance, in addition for their for ability to utilize simplified peak locating in timing synchronization. Furthermore, as the implementation aims to be utilized in a SDR platform, co-channel interference in such cases can be an issue as the channel spectrum can be unknown before use. However, if the spectrum is fixed to a channel without co-channel interference or false positive detection events are allowed, then 0.3125 provides the best performance. Unfortunately for the proposed implementation, the timing synchronization logic will fail if $\alpha < 0.6875$ are utilized.

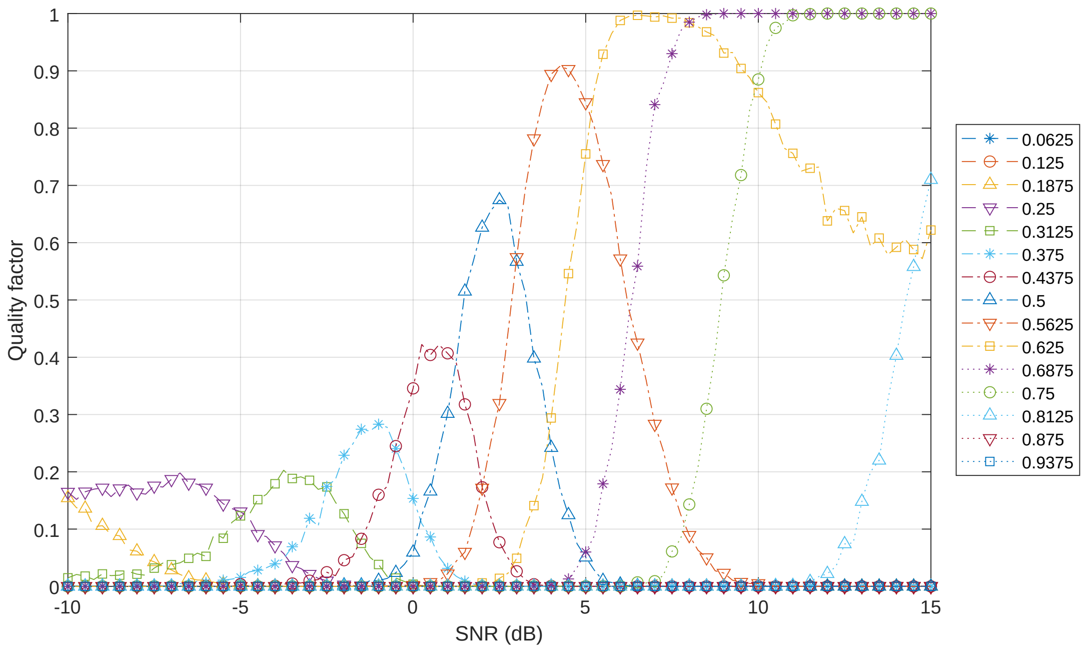


Figure 21: Comparison of the proposed implementation's quality factor for various α values in an AWGN channel over different signals' SNRs.

5.3 Packet detection algorithm comparison

The detection rate of thesis implementation was compared against a theoretical cross-correlation, which was defined as Equation (1) in Section 2.1, and the reference autocorrelation model presented in [59, Webinar Series (4): Technical II] for various SNRs transmitted over simulated AWGN channel. The performance evaluation comparison metrics were a packet detection rate with a ideal generated STF preamble and a false positive detection rate with an interfering 802.11 L-STF preamble, where both signals were additionally corrupted by white-additive Gaussian noise. For the analysis, detection thresholds of 0.6875 and 0.75 were utilized for comparable results.

The cross-correlation was calculated by taking correlation between a saved version of the ideal STF sequence's complex conjugate and a sliding window over the received signal. The window is a portion of the received signal with a size of STF. If the normalized correlation between the saved sequence and the sliding window exceeded a set threshold, the preamble was declared to be found.

The autocorrelation reference, from which the proposed implementation was derived, is defined as

$$R_{xx} = \frac{\gamma}{\phi},$$

where γ is

$$\gamma(m) = \sum_{k=0}^{N_R-1} r(m+k)r^*(m+k+Q_D)u(\lfloor\frac{k}{Q_D}\rfloor),$$

and ϕ is

$$\phi(m) = \frac{1}{2} \sum_{k=0}^{N_R-1} |r(m+k)|^2 + |r(m+k+Q_D)|^2 \,, \tag{21}$$

where N_R is the autocorrelation window size, Q_D is the delay and u is the nth cover sequence's sign, is defined as

$$u(p) = c(p) \cdot c(p+1),$$

where the p is nth cover sequence μ_c such that for $\mu = 1$, $p = 0, 1, \ldots, 5$. During the simulation, delay $Q_D = 16$ and window size $N_R = N - Q_D = 96$ were utilized. While the proposed implementation's autocorrelation was derived from the reference, the thesis deviates in calculating the estimated signal power in Equation (21).

As a comparison, the reference calculates estimation from both the last 96 samples and same sized portion delayed by 16 samples, before taking half of the resulting sum for the two portions. Such approach halves the estimated signal power for the first and last STF base sequence. The benefit in doing so is the perfect normalization with the autocorrelation function: with an ideal STF, the maximum normalized correlation result is 1 for reference and $96/112 \approx 0.8571$ for the proposed implementation without power scaling factor α. Hence, the reference normalization is implemented as an unbiased estimator, while the proposed implementation utilizes a biased estimator. As the reference's estimator in theoretical case estimates the variance perfectly, it allows a wider dynamic range to be utilized with the detection threshold.

The drawback for using the unbiased estimator is the additional operand for the sum, which increases logic utilization. Thus, the thesis implementation can be consider a balance between accuracy and resource utilization for the autocorrelation. However, an additional benefit for using the biased estimator is the channel calculation: in DECT's random access, a radio device must listen to the transmission channel and determine if the channel is free to use before sending its own packets. As such, due to the packet detector always sampling and thus calculating channel estimations, the biased estimator can be used to obtain channel utilization.

For the evaluation, as a logic for the detection declaration in the autocorrelation reference was not established, the preamble search algorithm was simplified to declare STF found if the normalized autocorrelation exceeded given detection threshold. For the thesis implementation, four iterations were used with different guard bands and detection thresholds: no guard band with $\alpha = 0.6875$, guard band size of one sample with $\alpha = 0.6875$, no guard band with $\alpha = 0.75$, and guard band size of one with $\alpha = 0.75$.

As can be observed from Figure 22, cross-correlation has the best packet detection rate for a given threshold regardless of signal-to-noise ratio. This is an expected result for an AWGN channel, where cross-correlation is the optimal solution for detecting existence of a known sequence.

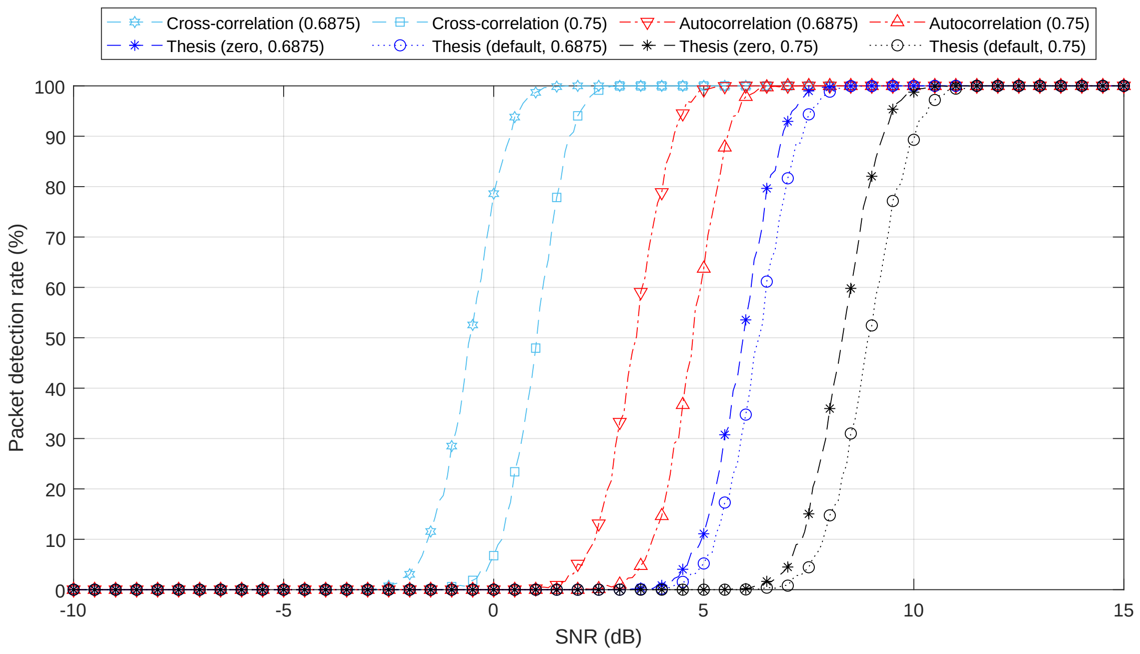


Figure 22: Comparison of packet detection rates for different correlations, detection thresholds and SNR values with an ideal generated STF preamble in AWGN channel.

In case of the reference autocorrelation, it requires signal to have approximately 4 dB higher SNR than cross-correlation for equivalent packet detection rate with same threshold. For example, while autocorrelation achieves 90 % packet detection rate at 4.3 dB SNR for 0.6875 threshold, cross-correlation has same rate at 0.4 dB SNR.

Nonetheless, the reference model achieves better detection rate than proposed implementation for same SNR regardless of selected guard band or scaling factor. For example, the thesis implemention requires signal to have 2.6 dB higher SNR to achieve 90 % detection rate with 0.6875 detection threshold than the reference. Due to biased estimator in the thesis in comparison to the reference's unbiased, the result was as expected. Regardless, in the presence of interfering signal such as 802.11 L-STF, the reference autocorrelation is only implementation that has false positives, as can be observed from Figure 23. However, the interference must have relatively high SNR than DECT's STF to reliably cause false detection.

As such, in channels with relatively low-powered or no interference, the proposed implementation sacrifices detection capability for much lower computational complexity due to avoiding division or numerous complex multiplications. In case of strong co-channel interference, such as in ISM with other radio traffic, any signal with high autocorrelation may cause false positive detection events for the reference design. In such cases, the proposed implementation is a viable alternative to the reference for given detection thresholds unless proper filtering is added before or after packet detection to reject erroneous signals.

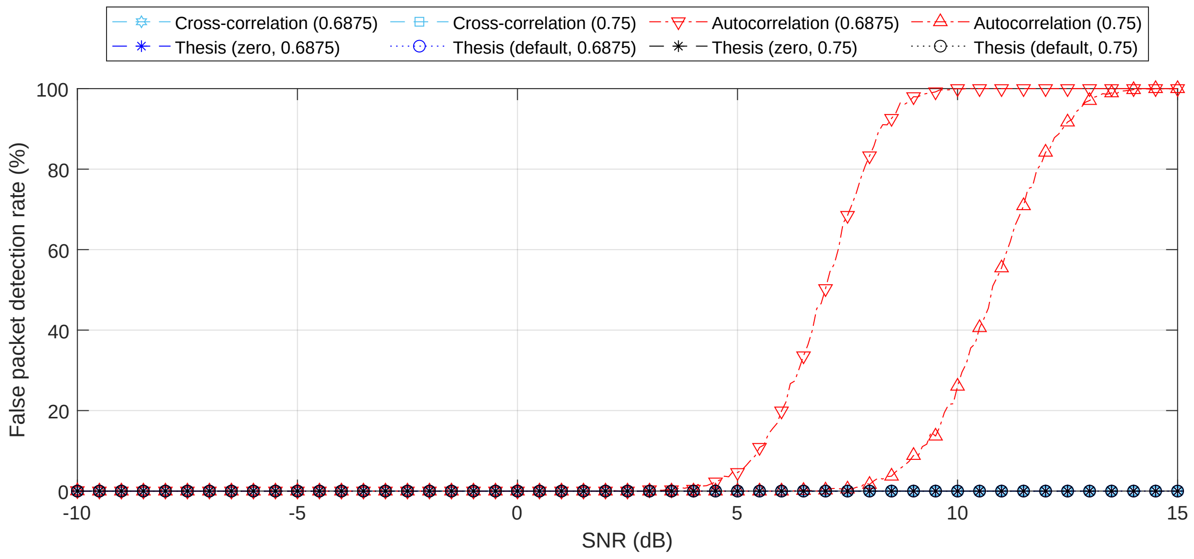


Figure 23: False positive detection rate for an interfering 802.11 L-STF preamble with different correlations and SNR values in AWGN channel. Each correlation includes two detection threshold versions.

5.4 Resource utilization comparison

For the target FPGA on AMD Zynq-7000 SoC hardware, the logic usage of both thesis' STF detection and openwifi's 802.11 preamble are shown in Table 4.

Table 4: Comparison of thesis and openwifi logic utilizations in target FPGA.

Implementation	Resource	Amount	From total
Thesis	LUT	1307	2.5 %
`STF_detection`	FF	1153	1.1 %
Power scaling factor ($\alpha = 0.75$)	DSP	5	2.3 %
	BRAM	0	0 %
Thesis	LUT	1311	2.5 %
`STF_detection`	FF	1153	1.1 %
Power scaling factor ($\alpha = 0.6875$)	DSP	5	2.3 %
	BRAM	0	0 %
openwifi	LUT	667	1.3 %
`sync_short`	FF	943	0.9 %
Only L-STF	DSP	6	2.7 %
detection	BRAM	2.5	1.8 %
openwifi	LUT	2901	5.5 %
`sync_short` +	FF	3641	3.4 %
`sync_long`	DSP	42	19.1 %
Both synchronization fields	BRAM	3	2.1 %

The explanations for different logic usage between thesis and openwifi implementations stems from preambles used in their respective radio technologies. For DECT-2020 New Radio, synchronization, channel estimation, and frequency and phase corrections are done by using the STF, as detailed in Section 2.4.1. In 802.11, the legacy synchronization sequence used in openwifi is divided into two parts: L-STF and legacy long training field (L-LTF).

The function of short field is initial timing synchronization, steer automatic gain control, aid antenna diversity selection and coarse correction of phase and frequency offsets. Such functionality is achieved by repeating same pattern 10 times, which allows the receiver to divide the different operations into discrete serialized steps. In openwifi's implementation, only the first pattern is used to detect the packet and timing synchronize, which significantly reduces logic utilization.

Nevertheless, L-LTF is used to fine tune both phase and frequency offsets, symbol timing, and channel estimation. As the field has 160 samples allocated to it, openwifi optimizes the calculation by only using it partially. Even then with only partial synchronization, the logic utilization for the full legacy preamble in openwifi exceeds that of the thesis. However, such comparison is not completely valid, as the `sync_long` module in openwifi includes the aforementioned phase and frequency synchronization that are missing from the proposed implementation. Therefore, a full satisfactory resource utilization analysis between implementations and wireless protocols would require a more complete design of the thesis implementation.

Regardless, an important note is that while the thesis implementation uses more LUTs and FFs resources than openwifi's `synch_short` for packet detection and timing synchronization, the proposed design doesn't require any BRAM memory units. This is lack of BRAMs is achieved by using architecture-specific shift-registers for numerator's auxiliary buffer and denominator's delay element, which are implemented efficiently with only LUTs and FFs. However, such elements are not readily available across different FPGA architectures, and as such the logic utilization might be higher on other architectures. Hence, a ring buffer-based design approach would be be better if portability of the implementation is required, due to it being readily available regardless of architecture. The drawback for ring buffer is the requirement of pointers for each index to access the memory, which becomes an issue for the numerator's auxiliary where five pointers are required. As pointers are typically designed as counters, each additional pointer increases resource utilization. Nevertheless, the required logic to access the memory with ring buffer can be reduced with fixed pointers.

For design clock speeds in the target FPGA, Table 5 summarizes performance that openwifi and thesis implementations have achieved. The recorded values are obtained from AMD Vivado's post-implementation timing report with the modules being synthesized and implemented as out-of-context. Thus, actual values for the implementations may change due to different placing and routing when integrated as part of a larger design.

As with logic utilization, maximum frequency gap between thesis and openwifi stems from difference radio technology preamble structure, which changes design approach for a given correlation and signal power implementation. In the proposed thesis design, the STF autocorrelation numerator's calculation requires multiple samples with different delays to be calculated in same clock cycle. In comparison, the simplified structure of L-STF in 802.11 allows openwifi to spread the calculation over multiple clock cycles. This spreading is known as pipelining, which shortens the achievable clock frequency defining critical path in synchronous digital hardware at the expense of additional processing latency. Even when the openwifi design includes both training fields, the pipelining approach keep the achieved clock frequency relatively high in comparison to the thesis implementation. Hence, utilizing same approach for the thesis implementation would improve maximum frequency, as the autocorrelation normalization comparison is the critical path. Furthermore, it is important to note that the maximum sampling rate supported by both implementations is the same as their maximum clock frequency.

Table 5: Comparison of thesis and openwifi achieved clock speeds in target FPGA.

Implementation	Total delay (ns)	Maximum frequency (MHz)
Thesis $\alpha = 0.75$	8.75	114.3
Thesis $\alpha = 0.6875$	8.75	114.3
openwifi (short preamble detection)	6.8	147.1
openwifi (short + long)	7.7	129.9

6 Conclusions

This thesis proposed an implementation for DECT-2020 New Radio physical layer STF preamble detection and timing synchronization on an FPGA. The purpose of this implementation was to be the foundation for a hardware-based SDR framework for DECT, similar to a state-of-the-art IEEE 802.11 open-source project named openwifi. The packet detector utilized an autocorrelation-based approach for detection and timing synchronization, where correlation threshold and normalization were combined to a power scaling factor α. To support verification with real-world captured data and minimize design complexity, the implementation was designed for a fixed physical layer configuration of $\{\beta = 1, \mu = 1\}$.

The proposed implementation was verified to be identical with its MATLAB reference design in simulation with an over-the-air captured signal. However, the design's detection performance was lower than other correlation-based for same SNR and detection threshold. and the achieved performance was further downgraded when the implementation's timing synchronization guard bands were increased. Nonetheless, the proposed implementation had lower resource utilization, which was comparable to the openwifi's 802.11 design when protocol differences were considered. Furthermore, various estimated signal power scaling factor α values were observed to analyze the implementation's performance over different detection thresholds.

Best to the author's knowledge, the proposed implementation is the first fully FPGA-based research topic for DECT-2020 NR. As such, the provided implementation can be further utilized in other FPGA- and hardware-based use cases, such as FPGA-based SDRs. While the thesis implementation has limitations that may restrict its use cases, each of them have been addressed with possible solutions. The current limitations of the implementation are as follows: insufficient support for timing synchronization with power scaling factors smaller than 0.6875, missing support for run-time power scaling factor selection, lacking support for configurations beyond $\{\beta = 1, \mu = 1\}$, lack of parametrization for selecting biased or unbiased signal power estimation, lack of support for selecting the timing-synchronized data to start from STF or user data during run-time, missing support to select run-time guard band sizes, architecture-based buffer optimization that increases logic utilization between different FPGA architectures, and inadequate maximum clock frequency in comparison to openwifi.

For future works, other than fixing the listed limitations, the thesis implementation could be integrated as part of a SDR framework for DECT-2020 New Radio, or in other hardware-based projects. Additionally, the simulation environment utilized during this thesis could be expanded to encompass full DECT stack, creating a possibility for software simulation of hardware-based DECT designs and research.

References

[1] M. Miller, *The Internet of things: how smart TVs, smart Cars, smart homes, and smart cities are changing the world*, 1st ed. Indianapolic, Indiana: Que, 2015.

[2] U. Dampage, L. Bandaranayake, R. Wanasinghe, K. Kottahachchi, and B. Jayasanka, "Forest fire detection system using wireless sensor networks and machine learning," *Scientific Reports*, vol. 12, no. 1, pp. 1–11, July 2022.

[3] Y. Liu, K. Akram Hassan, M. Karlsson, O. Weister, and S. Gong, "Active Plant Wall for Green Indoor Climate Based on Cloud and Internet of Things," *IEEE Access*, vol. 6, pp. 33 631–33 644, 2018.

[4] IEEE Spectrum, "Amazon Shows Off Impressive New Warehouse Robots," [Online]. Available from: https://spectrum.ieee.org/amazon-warehouse-robots, [Accessed 2024-09-30].

[5] G. Yang, L. Xie, M. Mäntysalo, X. Zhou, Z. Pang, L. D. Xu, S. Kao-Walter, Q. Chen, and L.-R. Zheng, "A Health-IoT Platform Based on the Integration of Intelligent Packaging, Unobtrusive Bio-Sensor, and Intelligent Medicine Box," *IEEE Transactions on Industrial Informatics*, vol. 10, no. 4, pp. 2180–2191, Nov 2014.

[6] L. Chettri and R. Bera, "A Comprehensive Survey on Internet of Things (IoT) Toward 5G Wireless Systems," *IEEE Internet of Things Journal*, vol. 7, no. 1, pp. 16–32, Jan 2020.

[7] M. N. Bhuiyan, M. M. Rahman, M. M. Billah, and D. Saha, "Internet of Things (IoT): A Review of Its Enabling Technologies in Healthcare Applications, Standards Protocols, Security, and Market Opportunities," *IEEE Internet of Things Journal*, vol. 8, no. 13, pp. 10 474–10 498, July 2021.

[8] A. Fuller, Z. Fan, C. Day, and C. Barlow, "Digital Twin: Enabling Technologies, Challenges and Open Research," *IEEE Access*, vol. 8, pp. 108 952–108 971, 2020.

[9] E. Sisinni, A. Saifullah, S. Han, U. Jennehag, and M. Gidlund, "Industrial Internet of Things: Challenges, Opportunities, and Directions," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 11, pp. 4724–4734, Nov 2018.

[10] B. Chen, J. Wan, L. Shu, P. Li, M. Mukherjee, and B. Yin, "Smart Factory of Industry 4.0: Key Technologies, Application Case, and Challenges," *IEEE Access*, vol. 6, pp. 6505–6519, 2018.

[11] European Telecommunications Standards Institute, "Digital Enhanced Cordless Telecommunications (DECT)," [Online]. Available from: https://www.etsi.org/technologies/dect, [Accessed 2024-09-30].

[12] R. Kovalchukov, D. Moltchanov, J. Pirskanen, J. Säe, J. Numminen, Y. Koucheryavy, and M. Valkama, "DECT-2020 New Radio: The Next Step toward 5G Massive Machine-Type Communications," *IEEE Communications Magazine*, vol. 60, no. 6, pp. 58–64, June 2022.

[13] R. Akeela and B. Dezfouli, "Software-defined Radios: Architecture, state-of-the-art, and challenges," *Computer Communications*, vol. 128, pp. 106–125, 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0140366418302937

[14] X. Jiao, W. Liu, M. Mehari, M. Aslam, and I. Moerman, "openwifi: a free and open-source IEEE802.11 SDR implementation on SoC," in *2020 IEEE 91st Vehicular Technology Conference (VTC2020-Spring)*, May 2020, pp. 1–2.

[15] S. M. S.-M. Kuo, B. H. Lee, and W. Tian, *Real-time digital signal processing : fundamentals, implementations and applications*, 3rd ed. Chichester, West Sussex: Wiley, 2013.

[16] R. Heath, *Introduction to Wireless Digital Communication: A Signal Processing Perspective*, 1st ed. Pearson, 2017.

[17] H.-C. Yang, *Introduction to digital wireless communications*, ser. IET Telecommunications Series ; 72. London, England: The Institution of Engineering and Technology, 2017 - 2017.

[18] "IEEE Standard for Information Technology–Telecommunications and Information Exchange between Systems - Local and Metropolitan Area Networks–Specific Requirements - Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications," *IEEE Std 802.11-2020 (Revision of IEEE Std 802.11-2016)*, pp. 1–4379, Feb 2021.

[19] E. Sourour, H. El-Ghoroury, and D. McNeill, "Frequency offset estimation and correction in the IEEE 802.11a WLAN," in *IEEE 60th Vehicular Technology Conference, 2004. VTC2004-Fall. 2004*, vol. 7, Sep. 2004, pp. 4923–4927 Vol. 7.

[20] Z. Xiao, D. Jin, and N. Ge, "Matched Filter-Autocorrelation (MF-AC) for Packet Detection in Multipath Channels," *IEEE Communications Letters*, vol. 17, no. 8, pp. 1608–1611, August 2013.

[21] R. Wood and R. Woods, *FPGA-based implementation of signal processing systems*, 1st ed. Chichester, United Kingdom: John Wiley & Sons, 2008.

[22] P. Possa, D. Schaillie, and C. Valderrama, "FPGA-based hardware acceleration: A CPU/accelerator interface exploration," in *2011 18th IEEE International Conference on Electronics, Circuits, and Systems*, Dec 2011, pp. 374–377.

[23] J. Fowers, J.-Y. Kim, D. Burger, and S. Hauck, “A scalable high-bandwidth architecture for lossless compression on fpgas,” in *The 23rd IEEE International Symposium on Field-Programmable Custom Computing Machines*. IEEE – Institute of Electrical and Electronics Engineers, May 2015. [Online]. Available: https://www.microsoft.com/en-us/research/publication/a-scalable-high-bandwidth-architecture-for-lossless-compression-on-fpgas/

[24] A. Nechi, L. Groth, S. Mulhem, F. Merchant, R. Buchty, and M. Berekovic, “FPGA-based Deep Learning Inference Accelerators: Where Are We Standing?” *ACM Trans. Reconfigurable Technol. Syst.*, vol. 16, no. 4, Oct. 2023. [Online]. Available: https://doi.org/10.1145/3613963

[25] P. Alfke, “20 years of FPGA evolution from glue logic to major system component,” in *2007 IEEE Hot Chips 19 Symposium (HCS)*, Aug 2007, pp. 1–19.

[26] P. Coussy, D. D. Gajski, M. Meredith, and A. Takach, “An Introduction to High-Level Synthesis,” *IEEE Design & Test of Computers*, vol. 26, no. 4, pp. 8–17, July 2009.

[27] R. Nane, V.-M. Sima, C. Pilato, J. Choi, B. Fort, A. Canis, Y. T. Chen, H. Hsiao, S. Brown, F. Ferrandi, J. Anderson, and K. Bertels, “A Survey and Evaluation of FPGA High-Level Synthesis Tools,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 35, no. 10, pp. 1591–1604, Oct 2016.

[28] S. Lahti, P. Sjövall, J. Vanne, and T. D. Hämäläinen, “Are We There Yet? A Study on the State of High-Level Synthesis,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 38, no. 5, pp. 898–911, May 2019.

[29] J. Mitola, “Software radios: Survey, critical evaluation and future directions,” *IEEE Aerospace and Electronic Systems Magazine*, vol. 8, no. 4, pp. 25–36, April 1993.

[30] D. Sinha, A. K. Verma, and S. Kumar, “Software defined radio: Operation, challenges and possible solutions,” in *2016 10th International Conference on Intelligent Systems and Control (ISCO)*, Jan 2016, pp. 1–5.

[31] D. M. Molla, H. Badis, L. George, and M. Berbineau, “Software Defined Radio Platforms for Wireless Technologies,” *IEEE Access*, vol. 10, pp. 26 203–26 229, 2022.

[32] GNU Radio project, “GNURadio,” [Online]. Available from: https://www.gnuradio.org/, [Accessed 2024-09-30].

[33] R. W. Stewart, K. W. Barlee, D. S. W. Atkinson, and L. H. Crockett, *Software Defined Radio using MATLAB & Simulink and the RTL-SDR*. Glasgow, 2015.

[34] K. Tan, H. Liu, J. Zhang, Y. Zhang, J. Fang, and G. M. Voelker, “Sora: high-performance software radio using general-purpose multi-core processors,” *Commun. ACM*, vol. 54, no. 1, p. 99–107, Jan 2011. [Online]. Available: https://doi.org/10.1145/1866739.1866760

[35] A. Khattab, J. Camp, C. Hunter, P. Murphy, A. Sabharwal, and E. W. Knightly, “WARP: a flexible platform for clean-slate wireless medium access protocol design,” *SIGMOBILE Mob. Comput. Commun. Rev.*, vol. 12, no. 1, p. 56–58, jan 2008. [Online]. Available: https://doi.org/10.1145/1374512.1374532

[36] O. Etrillard, R. Gerzaguet, L. Feichter, M. Mabon, A. Courtay, and O. Berder, “LOLA SDR: Low Power Low Latency Software Defined Radio for Broadcast Audio Applications,” *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 70, no. 2, pp. 481–485, February 2023.

[37] M. C. Ng, K. E. Fleming, M. Vutukuru, S. Gross, Arvind, and H. Balakrishnan, “Airblue: a system for cross-layer wireless protocol development,” in *Proceedings of the 6th ACM/IEEE Symposium on Architectures for Networking and Communications Systems*, ser. ANCS ’10. New York, NY, USA: Association for Computing Machinery, 2010. [Online]. Available: https://doi.org/10.1145/1872007.1872013

[38] Z. Yun, P. Wu, S. Zhou, A. K. Mok, M. Nixon, and S. Han, “RT-WiFi on Software-Defined Radio: Design and Implementation,” in *2022 IEEE 28th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, May 2022, pp. 254–266.

[39] M. D. Perez-Guirao, T. Weisshaupt, and A. Wilzeck, “DECT NR+:Unveiling the Essentials of a new non-cellular 5G Standard for Verticals,” in *Mobile Communication - Technologies and Applications; 26th ITG-Symposium*, May 2022, pp. 1–6.

[40] ETSI TS 103 636-4, *DECT-2020 New Radio (NR); Part 4: MAC layer; Release 1*, European Telecommunications Standards Institute Std., 2023. [Online]. Available: https://www.etsi.org

[41] ETSI TS 103 636-1, *DECT-2020 New Radio (NR); Part 1: Overview; Release 1*, European Telecommunications Standards Institute Std., 2023. [Online]. Available: https://www.etsi.org

[42] ETSI TS 103 636-3, *DECT-2020 New Radio (NR); Part 3: Physical layer; Release 1*, European Telecommunications Standards Institute Std., 2023. [Online]. Available: https://www.etsi.org

[43] T. Schmidl and D. Cox, “Robust frequency and timing synchronization for OFDM,” *IEEE Transactions on Communications*, vol. 45, no. 12, pp. 1613–1621, Dec 1997.

[44] J. Haxhibeqiri, X. Jiao, M. Aslam, I. Moerman, and J. Hoebeke, "Enabling TSN over IEEE 802.11: Low-overhead Time Synchronization for Wi-Fi Clients," in *2021 22nd IEEE International Conference on Industrial Technology (ICIT)*, vol. 1, March 2021, pp. 1068–1073.

[45] M. Aslam, W. Liu, X. Jiao, J. Haxhibeqiri, J. Hoebeke, I. Moerman, E. Municio, P. Isolani, G. Miranda, and J. Marquez-Barja, "High Precision Time Synchronization on Wi-Fi based Multi-Hop Network," in *IEEE INFOCOM 2021 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, May 2021, pp. 1–2.

[46] M. Aslam, W. Liu, X. Jiao, J. Haxhibeqiri, G. Miranda, J. Hoebeke, J. Marquez-Barja, and I. Moerman, "Hardware Efficient Clock Synchronization Across Wi-Fi and Ethernet-Based Network Using PTP," *IEEE Transactions on Industrial Informatics*, vol. 18, no. 6, pp. 3808–3819, June 2022.

[47] P. Avila-Campos, J. Haxhibeqiri, I. Moerman, and J. Hoebeke, "Impactless Beacon-Based Wireless TSN Association Procedure," in *2022 IEEE 18th International Conference on Factory Communication Systems (WFCS)*, April 2022, pp. 1–8.

[48] L. Baldesi, F. Restuccia, and T. Melodia, "ChARM: NextG Spectrum Sharing Through Data-Driven Real-Time O-RAN Dynamic Control," in *IEEE INFOCOM 2022 - IEEE Conference on Computer Communications*, May 2022, pp. 240–249.

[49] J. Haxhibeqiri, P. A. Campos, I. Moerman, and J. Hoebeke, "Safety-related Applications over Wireless Time-Sensitive Networks," in *2022 IEEE 27th International Conference on Emerging Technologies and Factory Automation (ETFA)*, Sep. 2022, pp. 1–8.

[50] L. Zhang, S. C. Liew, and H. Chen, "A Just-in-Time Networking Framework for Minimizing Request-Response Latency of Wireless Time-Sensitive Applications," *IEEE Internet of Things Journal*, vol. 10, no. 8, pp. 7126–7142, April 2023.

[51] G. Miranda, J. Haxhibeqiri, N. Slamnik-kriještorac, X. Jiao, J. Hoebeke, I. Moerman, D. F. Macedo, and J. M. Marquez-Barja, "The Quality-Aware and Vertical-Tailored Management of Wireless Time-Sensitive Networks," *IEEE Internet of Things Magazine*, vol. 5, no. 4, pp. 142–148, December 2022.

[52] X. Jiao, W. Liu, M. Mehari, H. Thijs, and A. Muhammad. open-source IEEE802.11/Wi-Fi baseband chip/FPGA design. [Online]. Available from: https://github.com/open-sdr/openwifi. [Accessed 2024-09-30].

[53] The kernel development community, "mac80211 subsystem," [Online]. Available from: https://docs.kernel.org/driver-api/80211/mac80211.html, [Accessed 2024-09-30].

[54] Stephen Williams, “Icarus Verilog,” [Online]. Available from: https://steveicarus.github.io/iverilog/, [Accessed 2024-09-30].

[55] Tony Bybell, “GTKWave,” [Online]. Available from: https://gtkwave.sourceforge.net/, [Accessed 2024-09-30].

[56] Advanced Micro Devices, Inc., “Avnet ZedBoard,” [Online]. Available from: https://www.xilinx.com/products/boards-and-kits/1-8dyf-11.html, [Accessed 2024-09-30].

[57] Analog Devices, Inc., “AD-FMCOMMS3-EBZ,” [Online]. Available from: https://www.analog.com/en/resources/evaluation-hardware-and-software/evaluation-boards-kits/EVAL-AD-FMCOMMS3-EBZ.html, [Accessed 2024-09-30].

[58] Advanced Micro Devices, Inc., *Zynq 7000 SoC Technical Reference Manual (UG585)*, 06 2023. [Online]. Available: https://docs.amd.com/r/en-US/ug585-zynq-7000-SoC-TRM/Zynq-7000-SoC-Technical-Reference-Manual

[59] DECT Forum, “NR+ Lower Layers,” [Online]. Available from: https://www.dect.org/nrplus, [Accessed 2024-09-30].

[60] R. G. Lyons, *Understanding Digital Signal Processing*, 3rd ed. Place of publication not identified: Prentice Hall, 2011.

[61] J. E. Volder, “The CORDIC Trigonometric Computing Technique,” *IRE Transactions on Electronic Computers*, vol. EC-8, no. 3, pp. 330–334, Sep. 1959.

[62] R. Andraka, “A survey of CORDIC algorithms for FPGA based computers,” in *Proceedings of the 1998 ACM/SIGDA Sixth International Symposium on Field Programmable Gate Arrays*, ser. FPGA ’98. New York, NY, USA: Association for Computing Machinery, 1998, p. 191–200. [Online]. Available: https://doi.org/10.1145/275107.275139

[63] Nordic Semiconductor ASA, “nRF9161 DK,” [Online]. Available from: https://www.nordicsemi.com/Products/Development-hardware/nrf9161-dk, [Accessed 2024-09-30].

[64] Ettus Research LLC., “USRP B210,” [Online]. Available from: https://www.ettus.com/all-products/ub210-kit/, [Accessed 2024-09-30].

[65] A. B. Awoseyila, C. Kasparis, and B. G. Evans, “Improved preamble-aided timing estimation for OFDM systems,” *IEEE Communications Letters*, vol. 12, no. 11, pp. 825–827, November 2008.