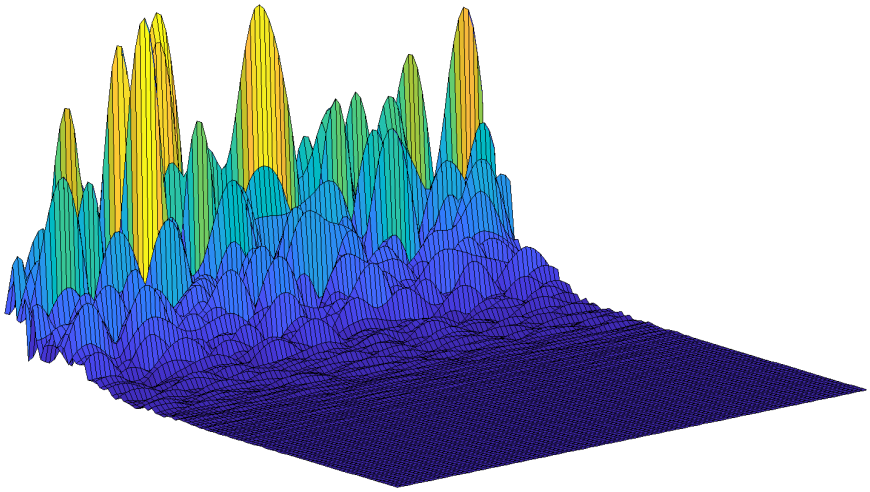
CHALMERS
UNIVERSITY OF TECHNOLOGY



Wireless Channel Emulator for Terrestrial Broadband Data Transmission

Channel Emulator for Testing of Air-to-Ground Aeronautical Communication System

Master's Thesis in Communication Engineering

SEBASTIAN YLANDER MIKKELSEN

Department of Electrical Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2019

Master's thesis 2019:EENX30

Wireless Channel Emulator for Terrestrial Broadband Data Transmission

Channel Emulator for Testing of Air-to-ground Aeronautical Communication System

SEBASTIAN YLANDER MIKKELSEN

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Electrical Engineering
Division of Communication Engineering
Chalmers University of Technology
Gothenburg, Sweden 2019

Wireless Channel Emulator for Terrestrial Broadband Data Transmission
Channel Emulator for Testing of Air-to-Ground Aeronautical Communication System
SEBASTIAN YLANDER MIKKELSEN

Supervisor: Chayan Bhar, Electrical Engineering
Examiner: Alexandre Graell i Amat, Electrical Engineering

Master's Thesis 2019:EENX30
Department of Electrical Engineering
Division of Communication Engineering
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Matlab vizualisation of multi-tap Rayleigh fading channel

Typeset in LaTeX
Gothenburg, Sweden 2019

Wireless Channel Emulator for Terrestrial Broadband Data Transmission
Channel Emulator for Testing of Air-to-Ground Aeronautical Communication System
SEBASTIAN YLANDER MIKKELSEN
Department of Electrical Engineering
Chalmers University of Technology

Abstract

This aim of this thesis is to program a channel emulator for aeronautical air-to-ground communication. The emulator will be used for testing an existing communication system developed by VITES GmbH. The emulator is implemented on an FPGA, a configurable integrated circuit.

Firstly, three separate scenarios with very different channel models are derived, which will likely be experienced in the duration of one flight: the en-route, landing/takeoff and taxi/parking scenarios. The scenarios are based on a previous Master's Thesis by S. Blandino [1]. After defining these scenarios with differing channel models, the emulator is divided into substuructures that are required in order to generate corresponding channel coefficients and fully emulate all the scenarios. A list of run time variables are derived that can be set in order to switch between the three models without resynthesizing the project.

The flexible emulator designed in this thesis is able to continuously produce random realizations of the derived channels and use this to filter the input signal. The run-time variables allows the user to adjust the model to match that of the current flight scenario. The performance of the emulator, as well as the resulting channel model produced is compared to simulation results and theoretical values. The three derived scenarios are also emulated in order to see the testing capacity of the final implementation.

Keywords: Channel emulator, Fading channel, Aeronautical communication, FPGA, HLS, Matlab

Acknowledgements

I would like to thank VITES GmbH for making this thesis possible. I would especially like to thank my supervisors Ümit Abay and Karl Anselm for taking their time to guide me through the theoretical and practical parts of the project and provide support whenever it was needed. Lastly, I would also thank my supervisor at Chalmers, Chayan Bhar and examiner Alexandre Graell i Amat for overseeing the thesis and providing me with feedback and advice.

Sebatian Ylander Mikkelsen,
Gothenburg, July 2019

Contents

List of Figures

List of Tables

List of Abbreviations

ASIC	Application Specific Integrated Circuit
AWGN	Additive White Gaussian Noise
AXI4	Advanced eXtensible Interface 4
DB	Decibel
DSP	Digital Signal Processing
FIFO	First-in First-out
FIR	Finite Impulse Response
FPGA	Field-Programmable Gate Array
HDL	Hardware Description Language
HLS	High-Level Synthesis
IIR	Infinite Impulse Response
IP	Intellectual Property
LOS	Line-of-Sight
LTE	Long-term Evolution
LUT	Look-up Tables
MAC	Multiply-Accumulate
NLOS	Non-Line-of-Sight
PDF	Probability Density Function
PDP	Power Delay Profile
PSD	Power Spectral Density
RTL	Register Transfer Level
RX	Receiver
SOC	System-on-Chip

TX	Transmitter
VHF	Very High Frequency
WGN	White Gaussian Noise
WSSUS	Wide Sense Stationary with Uncorrelated Scattering

1
Introduction

This chapter introduces aeronautical communications and motivates the need for a channel emulator for testing. The aim of the report is stated along with the scope, outlining the requirements of the final emulator and which models it will be based on.

1.1 Aeronautical Communication

The need for connectivity in avionics is increasing like in many other industries. Avionic connectivity is used for traffic control, voice communication, and to provide customers with on-board WiFi. Although satellites or Very High Frequency (VHF) ground stations have been traditionally used for this purpose, the increasing demand for data over voice services, as well as an increase in the total air traffic, has motivated the need for high bandwidth ground stations to communicate with the aircrafts [2, 3]. VITES GmbH is a company located in Germany that are currently working on developing such air-to-ground communication systems. This thesis is written in collaboration with VITES GmbH and explores a way of testing their existing modem.

1.2 Channel Emulation for System Testing

In most fields of engineering it is essential to model system behaviour to understand, analyze and design real-time systems. Likewise, in wireless communication, system models are used to analyze the effect of the channel on the signal and therefore estimate the performance of the wireless communication system. In the following paper, an emulator for Long-term Evolution (LTE) testing was implemented [4]. With an accurate model, it is also possible to reproduce the effect of the channel, and replicate this for testing purposes. Often, channels such as the wireless communication channel cannot be modelled entirely using theory from electromagnetics due to the high number of parameters and and fast-changing environments, along with a high computational costs. Therefore, statistical models can be employed, which are based on the statistics measured by channel sounding, a way of measuring channel characteristics.

1.3 Purpose

The aim of developing an emulator for wireless channels is to perform accurate and flexible testing of the underlying communication infrastructure. One approach for testing equipment is to set it up in a live environment which corresponds to the usage of the system. This is known as field testing. For instance, if the desired purpose is vehicular communication, a live scenario could be set up where a test vehicle performs measurements on the communication infrastructure.

However, performing field tests is significantly more difficult in aeronautical communication due to the involved cost concerns. Moreover, the setup time for these tests can also be relatively high. Therefore, if any changes are then made to the equipment under testing, repeating the process will augment the cost and time concerns. Furthermore, there are many components to a complete communication system such as the modem, antenna, power supply, etc. Without having all the components fully working, it is not possible to perform field test on the individual parts of the system, such as the modem, independently from the others.

A potential solution is to model the channel and use this model for performance testing.

1.4 SoC Implementation Advantages

Modern day communication systems have ever increasing demands of bit rate, latency and frequency. These require dedicated transmitter and receiver hardware to handle high data-rate transmission. Simulation based channel models will (generally) not be able to match the speed of the dedicated hardware when generating channel models and performing the filtering of the input signals, in order to produce an output.

This leads to, just as in the underlying communication technology, the need for dedicated hardware be able to real-time filter the transmitted signals according to the channel model. An System-on-Chip (SoC) containing an Field-Programmable Gate Array (FPGA) (a type of configurable integrated circuit), proves a good solution for such situations. The FPGA enables the mapping of computationally heavy algorithms to a reprogrammable architecture that will provide much higher computational speed (at the cost of flexibility). This is also a cost efficient alternative to having specifically designed hardware, or Application Specific Integrated Circuit (ASIC), as it can be reprogrammed in case of errors/changes in the model.

1.5 Scope

The aeronautical communication channel is one that greatly changes depending on the current state of the flight. Whether the flight is at cruising altitude or on its way towards the landing strip, reliable communication needs to be maintained between the airplane and the ground station.

During the development of the avionic/aeronautical communication system, it is important to have a channel emulator that allows tests to be performed on each

of the possible scenarios in order to validate the modem design. This necessitates a highly flexible emulator, where the underling model can be easily changed during run-time without having to completely redesign the hardware.

In his masters thesis, S. Blandino [1] pointed out three specific scenarios, where the communication system will experience diverse and completely dissimilar channels models. This was also illustrated by E. Haas in his paper about aeronautical channel modeling [5]. Since the transition between these scenarios is continuous, an all-purpose emulator is also required to handle the intermediate channel conditions that can occur during the flight.

The goal of this thesis is therefore to develop a flexible channel emulator that, with help of external input parameters, is able to model a majority of the channels encountered in aeronautical communication. Initially, the theoretical background for statistical channel modeling is reviewed, together with some digital signal processing theory. After this, the main function of the emulator is divided into subtasks which are implemented as blocks on the final FPGA. Finally, the results from the finished system are compared to simulations and theoretical values to validate the functionality of the design.

2
Theory

This chapter presents the theory behind wave propagation in free space, as well as common models used to estimate the effect of shadowing, free space loss, and multi-path fading for wireless channels. Some theory on discrete signal processing is explained, along with a discussion on how High-Level Synthesis (HLS) is used to program the target hardware.

2.1 Wireless Channel Modeling

Designing an accurate channel emulator requires an accurate model for understanding how the transmitted signals are affected during transmission. Ray tracing as a general solution is infeasible, due to its heavy computational requirements and large number of parameters required to accurately predict the signals behaviour. Thus, statistical models are often used in combination with a free-space loss model for modeling the behaviour of the channel.

The three main factors that contribute to signal degradation in the wireless channel are path loss, shadowing and small scale fading.

2.1.1 Free Space Loss

Free space loss describes the large scale attenuation of a signal travelling though free space. This is inversely proportional to the distance the wave has traveled, where two devices communicating at a larger distance from each other are expected to have a lower average received power.

The received power due to free space loss is given by [6]

$$P_{\mathrm{r}}[\mathrm{dBm}] = P_{\mathrm{t}}[\mathrm{dBm}] + G_{\mathrm{Tx}}[\mathrm{dBi}] + G_{\mathrm{Rx}}[\mathrm{dBi}] - 20\log_{10}\left(\frac{4\pi d}{\lambda}\right), \tag{2.1}$$

where the received power P_{r} depends on the transmitted power P_{t}, the distance between the receiver and transmitter d, and the gain of the receiver and transmitter antennas, given by G_{Rx} and G_{Tx} respectively. $\lambda = c_0/f_{\mathrm{c}}$ is the wavelength of the carrier signal, which is given by the product of the speed of light c_0 and the inverse carrier frequency $1/f_{\mathrm{c}}$. Fig. 2.1 shows the average received power as a function of distance due to free space loss. More generalized path loss models exist, but require measurement data to accurately predict the attenuation due to distance traveled.

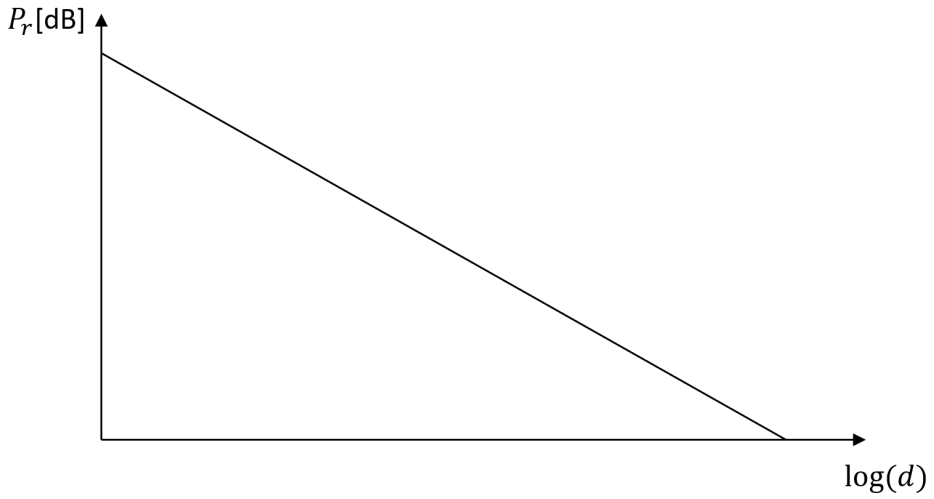


Figure 2.1: Received power as a function of distance d (in logarithms) between the transmitter and receiver.

2.1.2 Shadowing

Shadowing is another cause for signal attenuation, and relates to obstacles blocking a portion of the signal, causing reduced power at the receiver. This can usually be modeled using a log-normal Gaussian random variable, that produces random shadowing in the channel [6, Chapter 2].

The log-normal distribution for $\psi_{\mathrm{dB}} = 10\log_{10}(\psi)$ is given by

$$p(\psi_{\mathrm{dB}}) = \frac{1}{\sqrt{2\pi}\sigma_{\psi_{\mathrm{dB}}}} \exp\left(-\frac{(\psi_{\mathrm{dB}} - \mu_{\psi_{\mathrm{dB}}})^2}{2\sigma^2_{\psi_{\mathrm{dB}}}}\right) \tag{2.2}$$

where $\sigma_{\psi_{\mathrm{dB}}}$ is the standard deviation in Decibel (dB) and $\mu_{\psi_{\mathrm{dB}}} = \mathbb{E}(\psi_{\mathrm{dB}})$ denotes the mean in dB. In combination with the path loss attenuation, the received signal power is given by

$$P_{\mathrm{r}}[\mathrm{dBm}] = P_{\mathrm{t}}[\mathrm{dBm}] + G_{\mathrm{Tx}}[\mathrm{dBi}] + G_{\mathrm{Rx}}[\mathrm{dBi}] - 20\log_{10}\left(\frac{4\pi d}{\lambda}\right) - \psi_{\mathrm{dB}}. \tag{2.3}$$

The result of path loss and random shadowing can be seen in Fig. 2.2 where the average received power due to path loss is shown by the solid line and the random attenuation of shadowing is shown by the dotted line.

2.1.3 Fading Channel

Random, small scale variations in the power of the received signals known as fades, often occur due to interference from many scatters/reflections at the receiver, in combination with Doppler shift on each ray, caused by motion. At the receiver, the many scatters will constructively or destructively interfere with each other causing the received signal to vary over time. A common model for these small scale

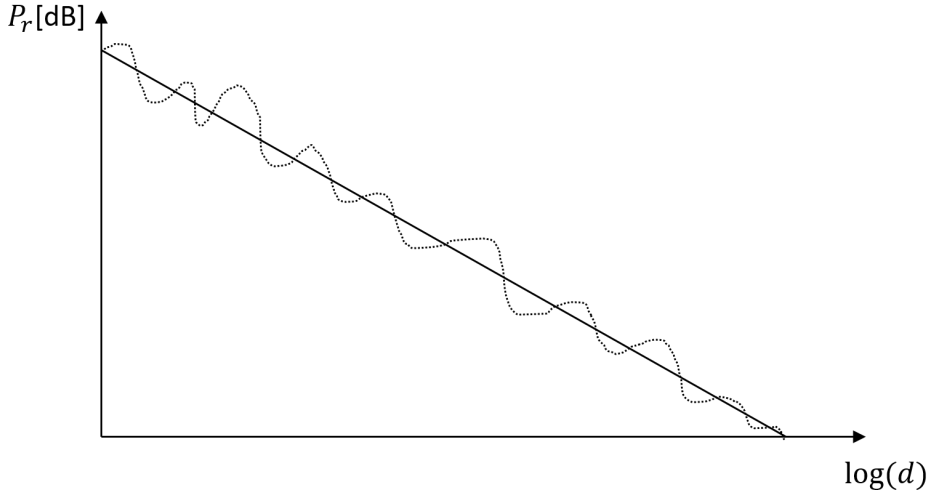


Figure 2.2: Received power as a function of distance d (in logarithms) between the transmitter and receiver due to path loss (solid line) and shadowing (dotted line).

variations is the Rayleigh fading channel model. Assuming that the scatters are uniformly distributed around the receiving antenna, (as illustrated in Fig. 2.3), gain at time t will be randomly distributed according to the Rayleigh distribution. The Probability Density Function (PDF) of the Rayleigh distribution is given by

$$f(x;\sigma) = \frac{x}{\sigma^2} e^{-x^2/(2\sigma^2)},\ x \geq 0, \tag{2.4}$$

where $2\sigma^2 = \bar{P}_\mathrm{r}$ is the average signal power at the receiver.

In presence of a Line-of-Sight (LOS) component, the received signal will have a higher average power. This is referred to as a Rician channel and the average received power is given by

$$\bar{P}_\mathrm{r} = s^2 + 2\sigma^2, \tag{2.5}$$

where s^2 is the power of the LOS component and $2\sigma^2$ is the average power of the Non-Line-of-Sight (NLOS) scatters (Rayleigh distributed). Defining

$$K = \frac{s^2}{2\sigma^2} \tag{2.6}$$

as the ratio between the power of LOS and NLOS, and

$$s^2 = \bar{P}_\mathrm{r} \frac{K}{K+1} \tag{2.7}$$

as well as

$$2\sigma^2 = \bar{P}_\mathrm{r} \frac{1}{K+1}, \tag{2.8}$$

the ratio K can be used to model varying power in the LOS component [6, Ch. 3]. Setting $K = 0$ would result in a Rayleigh fading, while $K = \infty$ would remove the impact of the fading entirely and leave a purely deterministic channel [6].

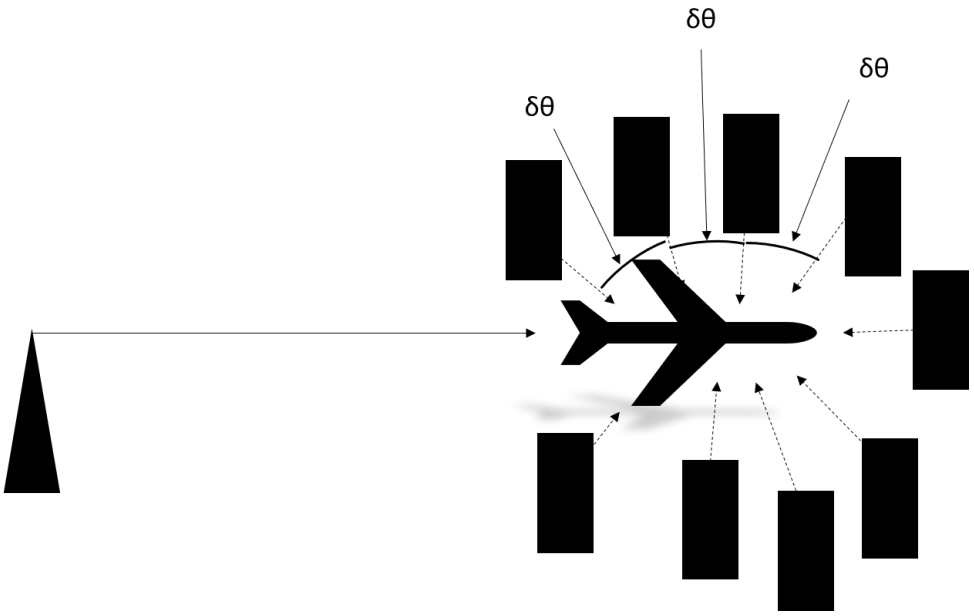


Figure 2.3: Illustration of a uniform scattering environment resulting in Rayleigh fading at receiver.

2.1.4 Jakes' Model

W.C. Jakes provides a method for generating random Rayleigh fading channels in his book Microwave Mobile Communications [7, Ch. 1]. The method takes two Gaussian noise sources, one for the in-phase component of the signal and one for the quadrature component. By filtering the noise $n_{r_\mathrm{Q}}(t)$ and $n_{r_\mathrm{I}}(t)$ through the auto correlation function $A_{r_\mathrm{I},r_\mathrm{Q}}$ corresponding to the inverse Fourier transform of the Doppler spectrum $\mathcal{F}^{-1}(S_{r_\mathrm{I},r_\mathrm{Q}})$, and adding the filtered noise out of phase will result in complex Gaussian noise with the desired auto correlation which corresponds to a Rayleigh fading channel. Fig. 2.4 shows a block diagram of how Jakes' method can be used to generate a realization of a Rayleigh channel.

The auto correlation of a given Rayleigh fading channel for the in-phase and quadrature components is

$$A_{r_\mathrm{I},r_\mathrm{Q}}(\tau) = P_\mathrm{r}\mathrm{J}_0(2\pi f_\mathrm{D}\tau) \tag{2.9}$$

assuming that the scattering environment is uniform over $[-\pi, \pi]$, with an infinite number of scatters [6, Ch. 3]. In Eq. 2.9, J_0 denotes the 0th order Bessel function. Thereby, $f_\mathrm{D} = \frac{v}{\lambda}$ represents the maximum Doppler shift, where v is the velocity of the receiver relative to the transmitter.

The Doppler Power Spectral Density (PSD) for the channel is given by

$$S_{r_\mathrm{I},r_\mathrm{Q}}(f) = \begin{cases} \frac{2P_\mathrm{r}}{\pi f_\mathrm{D}} \frac{1}{\sqrt{1-(f/f_\mathrm{D})^2}} & |f| \leq f_\mathrm{D} \\ 0 \end{cases} \tag{2.10}$$

where $P_\mathrm{r} = 1$ if the received power is normalized. The auto correlation function and PSD can be seen in Fig. 2.5 and 2.6.

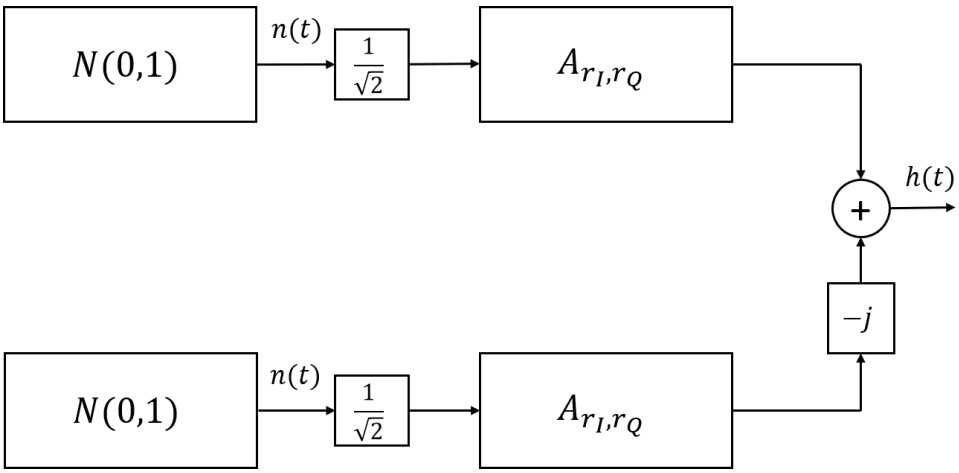


Figure 2.4: The Jakes' model passes normal Gaussian noise through two instances of the auto correlation function (in-phase and quadrature) to create a realization of the Rayleigh fading channel.

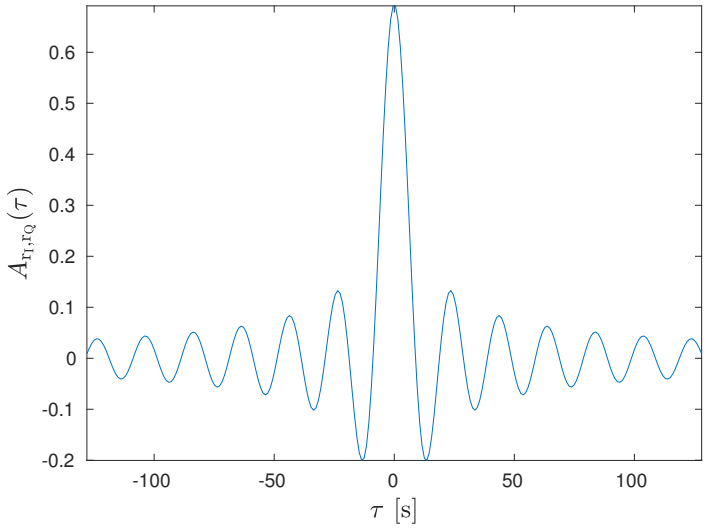


Figure 2.5: Rayleigh fading auto correlation function $A_{r_\mathrm{I},r_\mathrm{Q}}(\tau)$ used to filter noise and produce the Rayleigh fading channel realization.

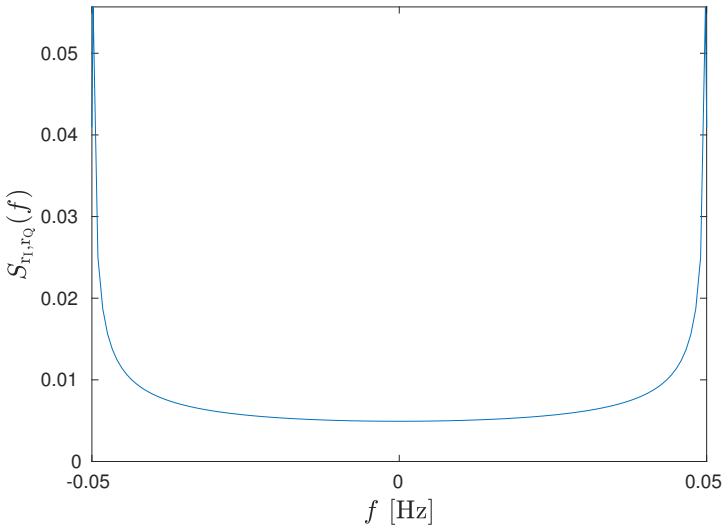


Figure 2.6: Power spectral density of a channel resulting from Jakes' channel generation.

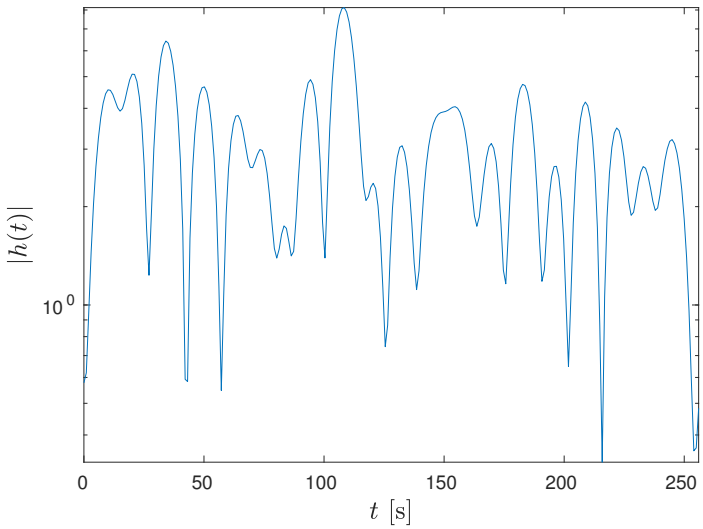


Figure 2.7: Fading envelope of Rayleigh channel realization $|h(t)|$.

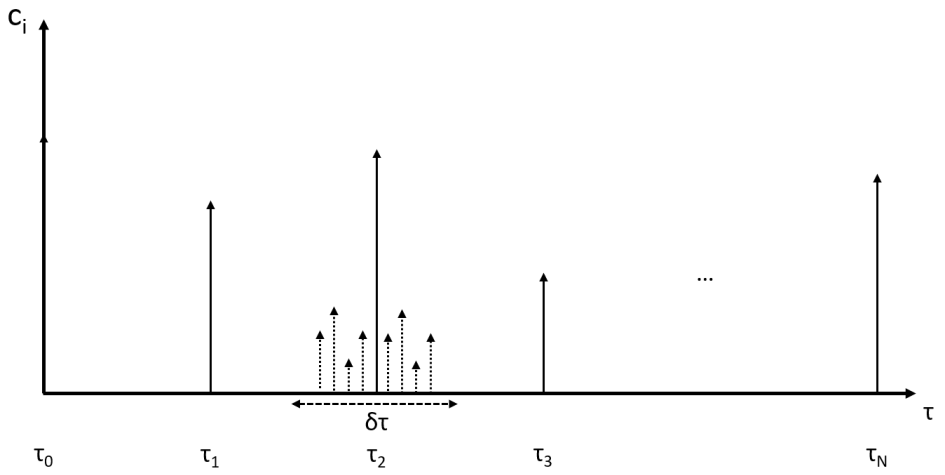


Figure 2.8: Impulse response for N taps of the tapped delay line model, where scatters within $\delta\tau$ second of each tap impacts the gain of the tap.

By filtering White Gaussian Noise (WGN) through the auto correlation function, this produces a realization of a random Rayleigh fading channel with the frequency characteristics of the PSD. Fig. 2.7 shows a realization of a Rayleigh fading channel.

2.2 Tapped Delay Line Model

A tapped delay line is a simplified model of the impulse response of a channel that is commonly used for simulating channels in discrete time, and assumes a finite number of scatters evenly spaced in time [8] [6, Chapter 3]. The fixed number of scatters can be a result of the closest $\delta\tau$ scatters nearby, that arrive approximately at the same time but are unresolvable. An example of a tapped delay line can be seen in Fig. 2.8, where each spike along the τ axis represents a reflection (or cluster of scatters) with complex gain $c_i(t)$ and delay τ_i.

The result of a discrete signal $x(t)$ passing through such a channel would be given by

$$y(t) = \sum_{i=0}^{N} c_i(t)x(t - \tau_i). \tag{2.11}$$

N is the fixed number of taps, and $c_i(t)$ corresponds to the complex gain of the ith tap at time t.

In his paper [9], G.L. Turin states that each path can be seen as an uncorrelated Rayleigh fading channel. In other words, the coefficient $c_i(t)$ for tap i in the impulse response can be modeled as a independent Rayleigh fading channel realization. This simplifies the channel generation process as the taps $c_i(t)$ can be generated using

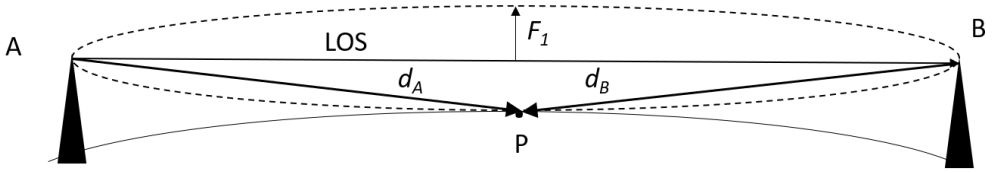


Figure 2.9: First Freznel zone formed between LOS path between transmitter A and receiver B.

the Jakes' method from section 2.1.4, where a Power Delay Profile (PDP) is used to scale the impulse responses for each tap to match that of the scattering environment.

2.3 Fresnel Zone

The Fresnel zone is defined as the ellipsoidal shaped area centered around the LOS ray with focal points at the transmitter and receiver [10]. An illustration of the Fresnel zone is shown in Fig. 2.9.

When a wave is transmitted with LOS, the Fresnel zone describes the area where interference may occur at the receiver, causing reduced or increased attenuation of the main LOS path. If an object is located within this zone, it will cause reflections at receiver that may be in- or out of phase with the direct LOS. However, if the Fresnel zone is obstruction free, pure LOS can be assumed with no additional attenuation.

The radius of the first Fresnel zone is computed as

$$F_1 = \sqrt{\left(\frac{\lambda d_A d_B}{d_A + d_B}\right)}, \; d_A, d_B \gg \lambda. \tag{2.12}$$

d_A and d_B describe the distance from transmitter A and receiver B to the obstruction point P.

2.4 Discrete Signal Processing

To analyze a signal passing through a certain channel, convolution is often used to analytically calculate the output in continuous time. This usually involves non-causal impulse responses infinite in time. However, when the filtering tasks are implemented in hardware, some modifications and simplifications have to be made on the sampled signals in order for the operations to be realizable.

2.4.1 Window Function

Representing an Infinite Impulse Response (IIR) in discrete time, a window function is needed to truncate the (possibly) infinite signal into a finite time span. Provided a IIR $c(t)$, the sampled, finite representation of this, or Finite Impulse Response (FIR) is given by

$$c(nT_s) = c(t)w(t) * \delta(nT_s). \tag{2.13}$$

$w(t)$ corresponds to the filter function and T_s is the sampling interval. E.g. the rectangular window is given by

$$w(t) = \begin{cases} 1 & 0 \leq t \leq T_s \\ 0 & \text{else} \end{cases}. \tag{2.14}$$

Sampling an IIR with finite time and samples will always distort the signal, however by carefully choosing the windowing function, the unwanted effects in the resulting filter can be reduced.

For the rest of this report, discrete signals will be written in terms of time unit t, but note that this is a discrete time unit, with the exception of Eq. 2.13.

2.4.2 Interpolation

Some applications in digital signal processing may require a larger number of Multiply-Accumulate (MAC) operations per sample. Interpolation can be exploited in order to output the required samples per time unit at a relatively low computational cost. Given a pair of discrete data points $x(0)$ and $x(1)$, the interpolator will compute a number of intermediate samples according to the chosen interpolation method. If, for example linear interpolation is used, the R_i intermediate samples $x(t), t \in [0, 1]$, where $1/R_i$ represents the interpolation rate, lie on the straight line that is formed between the input samples. Given a pair of sequential samples $x(0)$ and $x(1)$, the intermediate points $x(t)$ are computed as

$$x(t) = (1 - t)x(0) + tx(1) \tag{2.15}$$

with $t \in (0, 1)$. Fig. 2.10 shows an example of a linear interpolation of a sunusoidal signal.

However, linear interpolation can cause large sidelobes in the frequency spectrum of the signal. An example of this is shown in Fig. 2.11, where the PSD of signal $x(t)$ and interpolated version $x_{\text{int}}(t)$ are shown. The reason for this distortion is that a linear interpolation is equivalent to a convolution with a rectangular filter, with a sinc frequency response, that will cause these aliases [11]. In the case where this is not possible due to system specifications, a cubic spline interpolator can be used, which instead estimates a cubic curve between the data points, on which the intermediate samples lie. Fig. 2.12 shows a sinusoidal signal interpolated with the cubic spline interpolator.

Cubic interpolators do not produce as strong sidelobes as the linear interpolator, however, it is more computationally complex. The resulting spectrum of a cubic interpolation on a sinusoid is seen in Fig. 2.13.

Another effect of the interpolation is that creating more samples of the same signal, as the interpolator does, while keeping a constant sampling rate results in a frequency scaling of the interpolated signal. If a sampled signal

$$x(nT_s) = x(t)\delta(nT_s),\ n = 0, 1, 2, ... \tag{2.16}$$

is interpolated with rate $1/R_i$, the resulting signal will be

$$x^{'}(nT_s^{'}) = x(t)\delta(nT_s^{'}),\ n = 0, 1, 2, ... \tag{2.17}$$

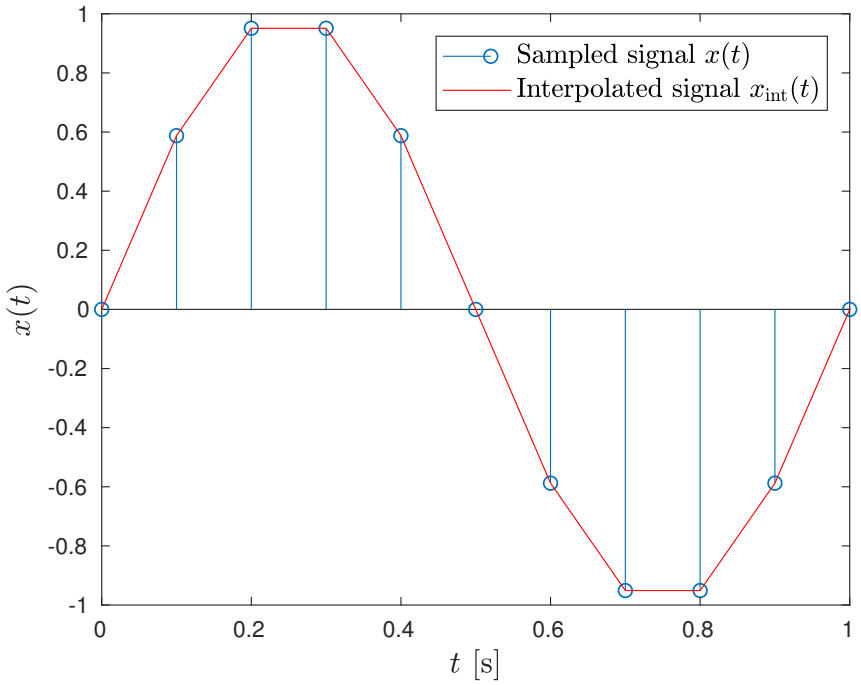


Figure 2.10: Linear interpolation of sinusoidal signal $x(t)$.

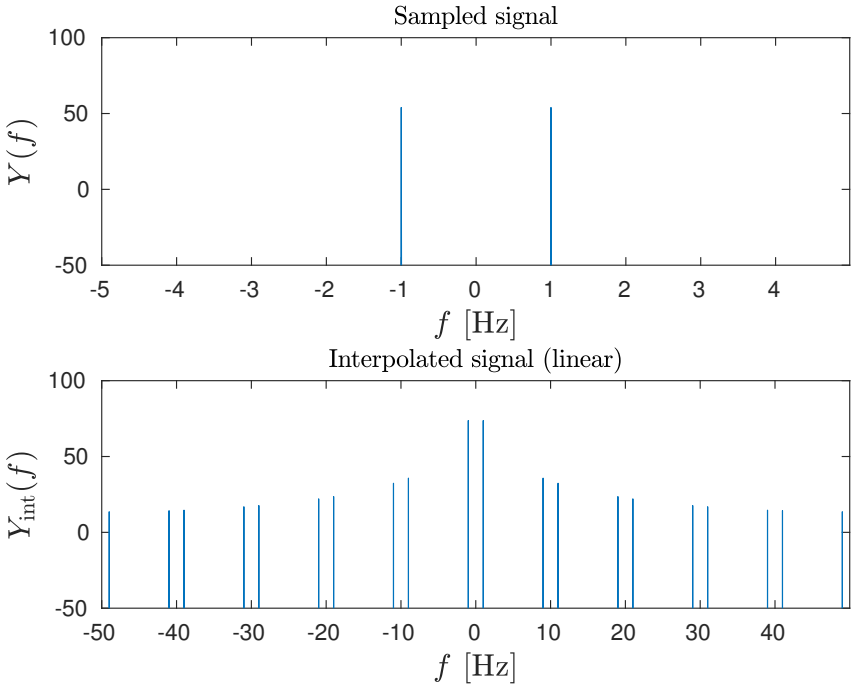


Figure 2.11: Spectrum of linearly interpolated signal $x(t)$.

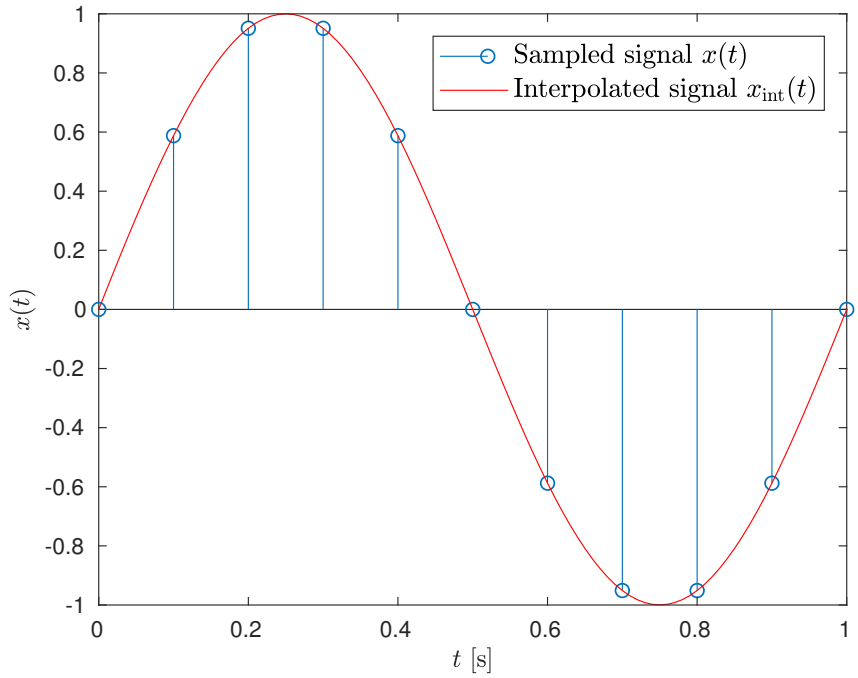


Figure 2.12: Cubic interpolation of sinusoidal signal $x(t)$.

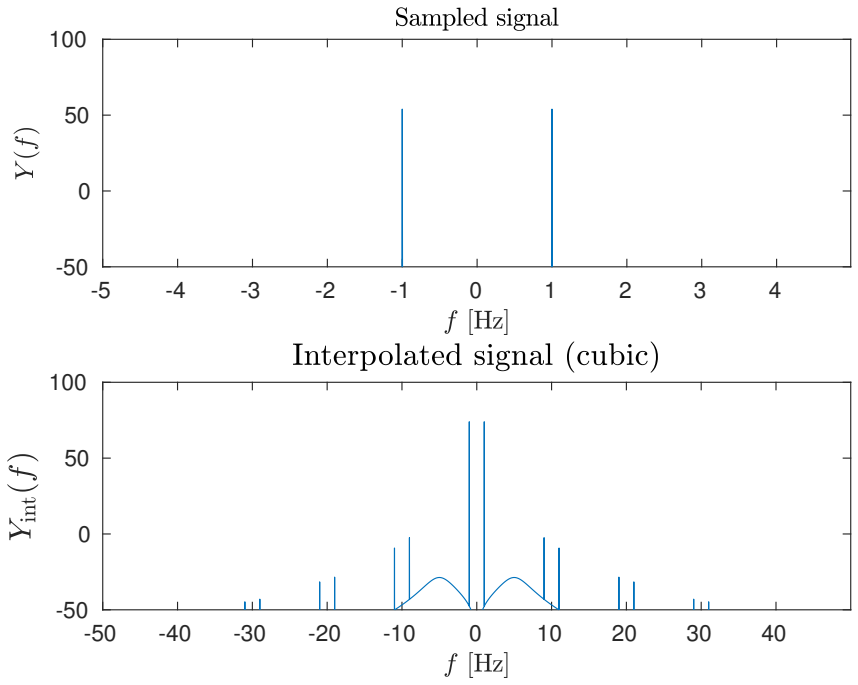


Figure 2.13: Spectrum of cubic interpolated signal $x(t)$.

where

$$T_s^{'} = \frac{R_i}{f_s} \tag{2.18}$$

meaning the frequency of the signal is scaled by $1/R_i$. Therefore, interpolation can be used to produce more samples of a signal, as well as scaling the spectrum of the signal to a desired frequency.

2.5 High Level Synthesis

HLS is a tool used for compiling high-level software code (such as C/C++) into Register Transfer Level (RTL) implementations used to synthesise programmable hardware such as an FPGA. HLS is very useful for hardware design purposes, since it allows developers with basic knowledge in hardware design to easily build custom hardware designs using functional programming. The general steps for HLS design is to create source code functions that performs the desired task. These functions can then be synthesized into RTL implementations, which can be read by the programmable hardware. Alternatively, the HLS tool can translate the functions into Hardware Description Language (HDL) such as VHDL or Verilog, which are languages commonly used for designing FPGAs.

Together with special directives called pragma, the behaviour of the function, such as latency, pipelining and required number of clock cycles, can be enforced in order to fulfill the design requirements. The pragma can also be used to design the flow of the input/output from each function, e.g. to create a FIFO stream on the output of a block. After synthesizing the function, HLS will output a summary of the hardware utilization on the target FPGA. This includes Look-up Tables (LUT), Digital Signal Processing (DSP) slices and memory used, as well as what input/output ports are required. After achieving the desired performance, the synthesized function can be exported as an Intellectual Property (IP) cores to be used in the final block level construction of the design.

3
Methods

This chapter presents the process of implementing the theoretical models in the hardware, as well as how parameters were chosen to achieve the tradeoff between computational complexity and accurate scenario modeling.

3.1 Scenarios and Emulation Scope

During a flight, the airplane will experience several stages which will distinctly alter the behaviour of the wireless channel. This occurs due to the speeds at which the aircraft travels, as well as the distance from the transmitter and angle of scattering. One cycle of a flight is therefore divided into three distinct scenarios by [1], where each channel can be considered unique: en-route, takeoff/landing and taxi/parking. In [5], four scenarios were used, but for this thesis, taxi and parking were merged, as they would experience very similar channel models.

3.1.1 En-Route

When in flight or en-route, the channel can be distinguished by a large Doppler shift as a result a fast moving airplane, along with a large delay spread due to the long distance between the Tx and Rx resulting in the reflections traveling a longer distance relative to the LOS. It can be assumed that LOS is present since there are very few obstructions blocking the main path of the signal, however it is possible that earth curvature or any larger structures are within the Fresnel zone (defined in Sec. 2.3). The reflection could be a single path reflection, such as on the surface of water, or be a combination of many scatters that reaches the en-route plane at approximately the same time, or at least as unresolvable paths. It can therefore be assumed that the channel model is a two-tap model, where the second tap can be modelled with a constant in the single reflection case or a fading channel in the case with multiple scatters.

Fig. 3.1 displays an en-route scenario where the main LOS is present, along with one path that is a reflection from earths surface.

3.1.2 Takeoff/Landing

In the case when the plane is in takeoff or landing, it is usually located in the close quarters with other structures. For example, there may be an airport besides/behind the plane that acts as a reflector. In addition, the plane is close to the ground,

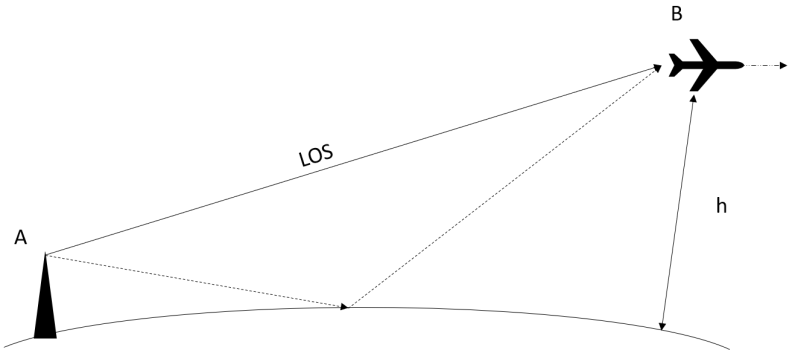


Figure 3.1: Visualization of two-tap model from the en-route scenario. A transmits a signal to the airplane B at altitude h. One LOS path is seen together with one reflection.

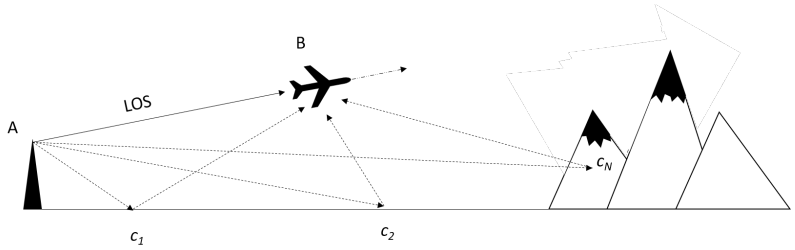


Figure 3.2: Visualization of multi-tap model from the landing scenario. A transmits a signal to the airplane B at altitude h. One LOS path is seen together with multiple reflections $[c_1, c_2, ..., c_N]$.

which may produce reflections. It is however still feasible to assume that the LOS component is still present, as the runway provides a open space where no obstacle blocks the main path between the Tx and Rx antennas.

The LOS can also be used to model the first tap of this scenario. In addition, some reflections with larger delay spread may be visible, meaning the channel experiences wide band fading. Like in the en-route case, each reflective path could be a collection of scatters arriving approximately with the same delay. Fig. 3.2 illustrates how this scenario could look like.

3.1.3 Taxi/Parking

When the airplane is taxiing or parked, it can be assumed that the LOS component is not always present. This of course depends on the airport, the location of the plane, if it is parked in a hangar etc. However considering the worst case scenario, one can assume a Rayleigh fading channel, or a Rician channel where the Rice coefficient $K = 0$. Fig. 3.3 depicts a scenario where LOS is present, and the plane receives scatters from the environment (assumed uniformly distributed around the plane).

This scenario would resemble an urban vehicular communication channel, experiencing narrowband fading and it could be assumed that only one distinct path is resolvable at the receiving end. However, this path will be under the effect of

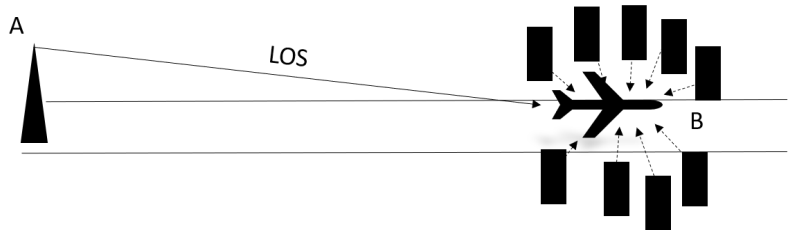


Figure 3.3: Taxi/parking scenario visualization, where the Rx antenna at B receives a LOS and scatters from Tx A.

Rayleigh or Rician fading depending on the gain of the LOS.

3.1.4 Emulation Scope

As was previously seen, during the duration of one flight, a number of different communication channel models will be encountered. To be able to properly model these channels, along with any intermediate channels, the emulator requires some degrees of freedom, where certain parameters are set during run-time to emulate a real flight situation.

The emulator's primary goal is being able to properly emulate the three scenarios derived by Blandino [1]. It will be given the flexibility, however, to achieve a combination of the models if required. Any channel that can not be described as a combination of these are outside of the scope.

A few parameters such as Doppler shift of the LOS component, path loss and shadowing will later be emulated using external hardware. The Doppler shift will be achieved by changing the relative sampling frequency between the receiving and transmitting modem, while the path loss attenuation will be achieved by a serially connected external attenuator. These will therefore not be considered as part of the emulator.

The Additive White Gaussian Noise (AWGN) is generated by hardware in the modem, and SNR will therefore not be modeled in the final emulator. Instead, this will be controlled by the existing noise of the receiver and by varying the external attenuation, alternatively adding an additive noise generating circuit.

3.2 System Requirements

To be able to emulate the three scenarios stated above in real time, some requirements are placed on the system on which the emulator runs.

Firstly, since all three of the scenarios experience some randomness in terms of fading, the emulator needs to be able to produce channel coefficients in order to model this stochastic process. The fading can be modelled as Wide Sense Stationary with Uncorrelated Scattering (WSSUS), which implies that the noise for each tap is uncorrelated. This allows for parallelization of the noise generation process where each parallel noise generation process is independent from the others.

As the velocity experienced during a flight will vary, the emulator requires a method for changing the spectrum of the generated channel coefficients to match

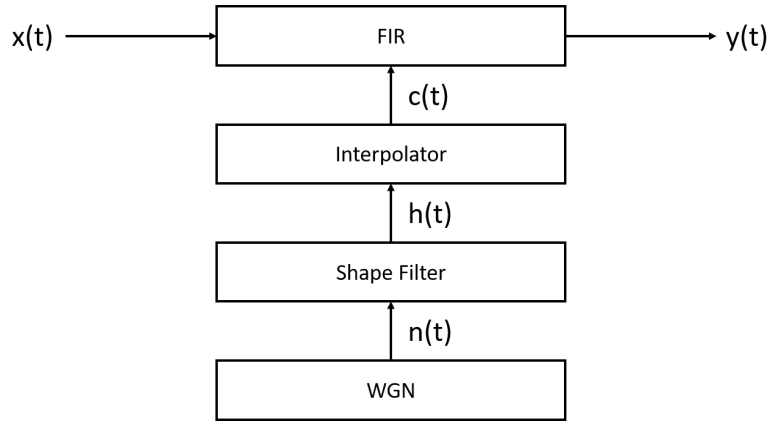


Figure 3.4: Block diagram overview of emulator.

that of the actual channel.

Furthermore, the delay spread for the channel will vary greatly depending on the scenario. For example in the taxi scenario, the scatters may not be resolvable, resulting in a one-tap fading channel. However, en-route, delay spread could reach up to 200 μs [5]. This necessitates a way to change the delay spread of each tap when switching channel.

Lastly, the emulator should not slow down the data rates from Tx modem to Rx, requiring the minimum input/output frequency to match that of the modems. This is required in order to allow real-time emulation to be performed on the communication system.

3.3 Block Design

The task of emulating the response of a wireless channel in aeronautical communication is divided into smaller subtasks. Each subtask is then implemented as an IP block in the final emulator, where the blocks receives some inputs and processes this to compute the output. The block diagram for the divided subtasks can be seen in Fig. 3.4. A noise source called WGN provides complex noise samples $n(t)$. These noise samples are used by the shape filters to produce Rayleigh correlated channel coefficients with normalized Doppler. Since each tap can be modeled as an independent channel, N parallel shape filters provide independent channel coefficients $h_0(t), h_1(t), ..., h_{N-1}(t)$ for each tap at time t. Likewise, N parallel interpolators are used to interpolate the input channel coefficients $h_0, h_1, ..., h_{N-1}$ and output $c_0, c_1, ..., c_{N-1}$ with a rate that matches the actual Doppler of the channel. The interpolated channel coefficients are passed to the FIR filter which filter the coefficients with the input $x(t-\tau_0), x(t-\tau_1), ..., x(t-\tau_{N-1})$ to produce output $y(t)$.

The functions that are implemented in the final emulator are created using Vivado HLS. These could then be exported as IP cores and be used for the final block design.

3.3.1 Shape Filter for Rayleigh Correlated Noise

The shape filter outputs a continuous stream of correlated coefficients for the Rayleigh channel. According to the Jakes' model in Sec. 2.1.4, a complex noise sample is passed to the shape filter corresponding to the auto correlation function of the Rayleigh channel $A_{r_\mathrm{I},r_\mathrm{Q}}$. Using a delay-line structure to store the previous noise samples (discrete convolution), the shape filter can then output one sample of the channel coefficient with the correct statistics according to

$$\hat{h}(t) = \sum_{i=0}^{N_s} A_{r_\mathrm{I},r_\mathrm{Q}}(i) * n(t-i) \tag{3.1}$$

where $n(t-i)$ is the noise sample at time $t-i$ and $A_{r_\mathrm{I},r_\mathrm{Q}}(i)$ is the ith value of the auto correlation function.

As mentioned in Sec. 3.2, the system needs to output N channel coefficients corresponding to the N tap model. This is most easily done by running N shape filters in parallel, where each shape filter i produces $\hat{h}_i(t)$ according to Eq. 3.1.

The PDP is then used to scale the output Rayleigh channel coefficient $\hat{h}_i(t)$ of each tap to correctly model the channel, and a constant value K_i is added to this product, which is used to produce a constant power offset for the LOS taps, so that

$$h_i(t) = D_i * \hat{h}_i(t) + K_i \tag{3.2}$$

with $P_i = K_i^2 + 2 * (D_i * \sigma)^2$ (the power of the ith tap). Depending on the values of K_i and D_i, it is now possible to set the number of taps present in the model as well as define if tap i is Rayleigh distributed ($K_i = 0$, $D_i > 0$), Rician distributed ($K_i > 0$, $D_i > 0$) or pure LOS ($K_i > 0$, $D_i = 0$).

3.3.2 Variable Rate Interpolator

In order to generate one output sample $y(t)$ per clock cycle, the FIR filter needs N channel coefficients per clock cycle as well (one per tap). The shape filtering function is computationally expensive. Therefore, it is not feasible to run N instances of this function in parallel with the strict timing requirements of one output per clock cycle. This necessitates an interpolator to produce R_i times the samples from the shape filter, where $1/R_i \in [0, 1]$ is the variable interpolation rate.

The second use of the interpolator is to upsample the filter coefficients generated by the shaping filter function in order to achieve the correct fading of the channel being emulated. The shape filter produces a Rayleigh fading channel with normalized Doppler of $f_{\mathrm{D,norm}}$. In order to reach the desired Doppler frequency at f_D, the output channel has to be upsampled by $R_i = \frac{f_s}{f_\mathrm{D} * f_{\mathrm{D,norm}}}$, f_s being the sampling frequency. This is achieved by interpolating the signal without having to oversample the signal in the shape filter, which would increase the total number of MAC operations, and would be very costly on the hardware.

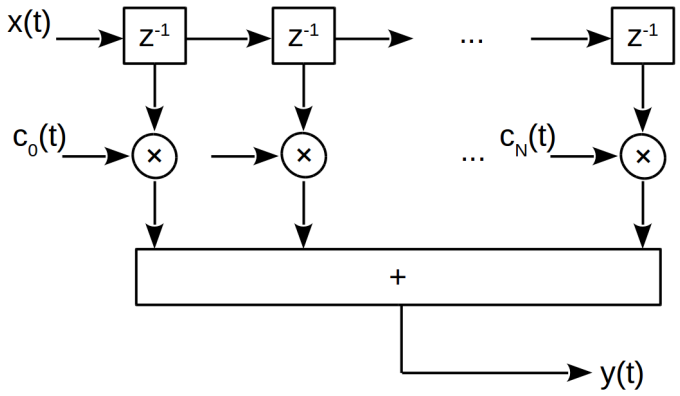


Figure 3.5: FIR tapped delay line design.

3.3.3 Finite Impulse Response Filter

The FIR block is responsible for taking the sampled input $x(t)$ along with the complex channel coefficient $c_i(t)$ of taps $i \in [0, N]$ at time t to calculate the output signal

$$y(t) = \sum_{i=0}^{N} x(t - \tau_i) \cdot c_i(t). \tag{3.3}$$

From Eq. 3.3 it can be seen that the FIR filter at time t produces the output $y(t)$, which depends on the previous N input samples $x(t)$ to $x(t - \tau_{N-1})$. This process can therefore be seen as a tapped delay line where the last N inputs are stored in the delay line, and the output is the sum of the taps, multiplied with the corresponding channel tap coefficients, as can be seen in Fig. 3.5.

The cases where the delay spread is much larger than the sampling time, FIFO queues are used to further delay the samples for each of the taps in the FIR by an integer value N_d. This will instead result in to subsequent taps τ_i and τ_{i+1} being separated by N_d samples which translates to N_d/f_s seconds.

3.3.4 Support Functions

Some support functions were added to support the data flow in the finished emulator, as well as allowing run-time parameters to be set.

3.3.4.1 Axis to Memory

The *axis-to-mem* block is a register block that outputs the K_i, D_i, R_i and N_d parameters to the functional blocks such as interpolator and shape filter. New values can be written to these registers during the emulation. This allows the user to quickly change the channel model during run-time, in order to test a different scenario or model without having to restart the emulator. The input of the block is a normal Advanced eXtensible Interface 4 (AXI4) stream, where an address and a

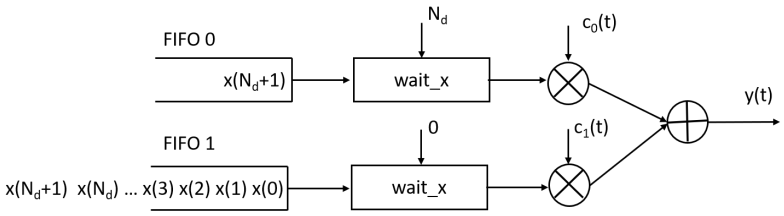


Figure 3.6: FIFO architecture to create a delay of N_d samples on the second tap.

new value is written to the input of the block, which then provides this new value on the output corresponding to that address.

3.3.4.2 Pack Struct

The *pack-struct* support function is used to reduce the clock cycle requirement by the FIR filter. Having N data streams as input requires $2N$ control ports to synchronize this block, prohibiting the FIR block from meeting the hardware timing requirements. This function therefore wraps all the channel parameters from all the taps into one wide data stream and writes this to the FIR filter, allowing it to meet the one output per clock cycle requirement. This is also used in the N_d/f_s delay emulator version to provide the FIR with 10 input samples from the FIFOs, since the delay line was now stored outside the FIR IP core.

3.3.4.3 Noise Stream

As all the N paths need a constant stream of WGN, this block distributes the noise samples to each of the shape filters. Noise samples are passed to the input of this block, which then writes the noise to the shape filters that need new samples (if their input AXI4 are not full).

3.3.4.4 Wait_x

The *wait_x* function block is used to initially delay the data samples in the FIFO for each tap to make sure that the FIR block receives samples with the spacing matching the delay spread of the channel. For example in a two tap scenario, such as in Fig. 3.6, the FIFO of the second tap will fill up with $N_\mathrm{d} + 1$ samples, where the first tap discards of its first N_d samples, leaving it's FIFO with only the latest input sample. This results in the the first output

$$y(t) = c_0(t) * x(t - \tau_0) + c_1(t) * x(t - \tau_1) \tag{3.4}$$

with

$$\tau_1 - \tau_0 = \frac{N_\mathrm{d}}{f_s}, \tag{3.5}$$

so that excess delay was added to the second tap.

The value of N_d is used as the number of samples of delay used between consequent taps.

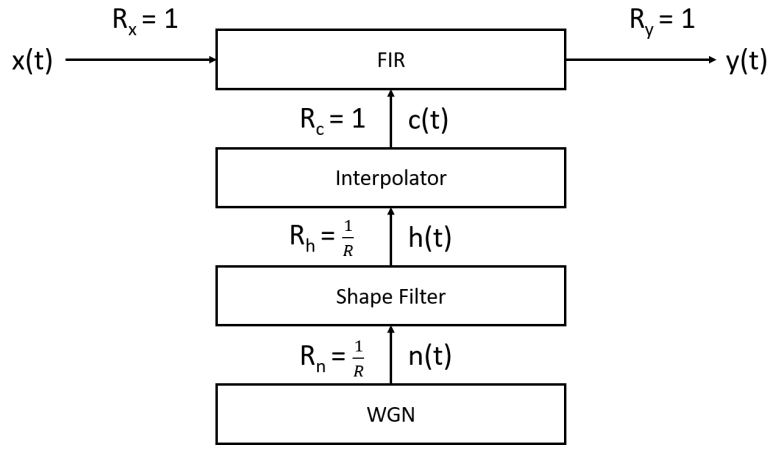


Figure 3.7: Timing criterion at input and output of each functional block in terms of R (samples per clock cycle).

Table 3.1: Summary of rate requirements for each function block in terms of rate $0 < R < 1$.

Function	input rate *	output rate *
WGN Generator	N/A	$1/R$
Shape filter	$1/R$	$1/R$
Interpolator	$1/R$	1
FIR	1	1

*unit for rate is samples/clock

3.4 Synthesis and Block Design Build

As previously mentioned in Sec. 3.3, dividing the operations of the emulator into subtasks simplifies the design. Instead, each operation can be performed by one IP block, which is later stitched together in a block design tool with the help of data streams.

3.4.1 Create IP Blocks

Once the functions have been implemented as C++ functions performing the desired operations, Vivado HLS directives or pragma are inserted to ensure that each IP block is able to produce an output within a desired number of clock cycles. Fig. 3.7 shows the final design with the required samples per clock on each input and output. The rates are summarized in Tab. 3.1.

The synthesis summary provided by Vivado HLS gives an estimate of how the function performs when written to hardware. Once the desired values are given, the IP cores are ready for use.

3.4.2 Vivado Block Design and Bit Stream Generation

The custom IP blocks are imported to the Vivado Design Suite, where a block design can be made by connecting the blocks in the correct order using AXI4 streams. Fig. A.1 in Appendix A shows an overview of the final block design.

The SoC core is created using Xillybus IP core generator, while the custom IP blocks are imported and connected using the addIP.tcl script. The software synthesizes the build to a bit stream that is used to program the target device.

3.4.3 Running Emulation

The Kintex 7 board used for this project comes with PCIe ports that allows the board to connect directly to a desktop computer. With correct drivers installed, data streams can directly be written from .bin files to the PCIe ports defined by the Xillybus IP core imported from the vendor. Using the "head" Linux command to write noise to the noise port and input port, a parallel process can read from the output port using the "cat" command and storing it as a output .bin file. With help of Matlab, it is possible to import and plot this data to analyze the output of the emulator in order to validate it or to test a system.

4

Results

This chapter presents the parameters used to synthesize the emulator. It also shows the resulting synthesis summaries from the IP Cores implemented in the final block build, as well as the verification results when testing the final build. Furthermore, some example channels based on the scenarios from Sec. 3.1 are tested.

4.1 Emulator Parameters and Synthesis

After implementing the blocks from Sec. 3.3 in C++, the functions are ready to be synthesized into IP cores for the final block design. A number of fixed global parameters were set in the header files of the source code. A summary of the system usage is provided by Vivado HLS.

4.1.1 Fixed Emulator Parameters

Certain parameters were set before the complete project was synthesized. This includes: the max number of taps N, the shape filer $A_{r_\mathrm{I},r_\mathrm{Q}}$, the number of samples in the shape filter N_s, the sampling frequency f_s, the interpolation method, and the delay between taps N_d (in samples). A summary of these parameters can be seen in Tab. 4.1. These values were used for the verification of the emulator. In Tab. 4.1,

Table 4.1: Parameters fixed during synthesis.

f_s	100 MHz
$A_{r_\mathrm{I},r_\mathrm{Q}}$	$J_0(0.2\pi\tau)$
N_s	256
N	10
N_d	100

the auto correlation function $A_{r_\mathrm{I},r_\mathrm{Q}}$ represents a Doppler shift fixed at 0.1 Hz.

The function for $A_{r_\mathrm{I},r_\mathrm{Q}}$ is theoretical, while the fixed N and N_s provided a good trade off of channel model and computational cost. f_s is chosen to match that of the modem, and N_d together with the set sampling frequency allowed delay taps to span the region suggested by [5].

Table 4.2: HLS summary of synthesized functions. Values include how many of each block was used in the emulator, the clock cycles required per output as well as % of total available hardware used by each block.

Function	Number of blocks	Clock cycles	DSP Slices	Flip-flops	LUT
Shape filter	10	256	1%	3%	2%
Interpolator	10	1	0%	0%	1%
FIR	1	1	4%	1%	0%
Pack_struct	1	1	0%	0%	0%
Noise_stream	1	1	0%	0%	0%
Total	N/A	N/A	14%	31%	30%

4.1.2 Synthesis Summary

After synthesising the C++ functions, the Vivado HLS tool provided a summary of the hardware requirements for each function. The most important of these parameters for each function can be seen in Tab. 4.2. Note that the percentages in Tab. 4.2 are not exact values but just rough estimates to get an idea of system utilization in terms of DSP slices, flip-flops and LUT, and are based on the total number of resources available on the FPGA on the Kintex 7 development board (xc7k325tffg900-2). The *Number of blocks* value states how many time each block is used and *Clock cycles* defines how many clocks are required by the hardware to calculate one output.

4.2 Performance Validation

To ensure that the synthesized system performs according to the requirements mentioned in Sec. 3.2, tests are performed on the input and output rates of the emulator. It is also necessary to make sure that the output from the emulator matches with that of the simulations to verify that each block is working as expected.

4.2.1 Rates

The first performance test is on the rate of the input and output of the system, that is, to verify that the emulator can produce one output sample per input received. Recall from Fig. 3.1 that the interpolators receives samples at rate $0 < R_\mathrm{h} < 1$. This means that, in order to supply 1 of each channel tap coefficients per input, the minimum interpolation rate R_i has to be $> 1/R_\mathrm{h} = 1/256$, resulting in a output rate of $R_\mathrm{c} = 1$ (sample per clock cycle).

With a interpolation rate of $R_i > 1/R_\mathrm{h}$, the linux "dd" command is used to write data on the input while measuring write rates. This is also done on the output simultaneously. The result shows that the rate on both the input and output was 396 MB/s. Since the clock was set to 100 MHz and the input/output streams consists of $2 * 16$ bits (real and imaginary part of signals $x(t)$ and $y(t)$), the expected

output data rate is

$$\frac{2 * 16 \text{ [bits]}}{8 \text{ [bits/byte]}} \text{ [bits]} * 100 \text{ MHz} = 400 \text{ MB/s} \tag{4.1}$$

which is close to the measured value. If the "dd" command is run for a longer time, the value converges to 400 MB/s.

The rate on the input of the noise stream is not important, since it can be seen from the resulting values from the "dd" command above that the noise rates will not slow down the overall data rates of the input and output of the FIR, as long as the interpolation rate is large enough.

4.2.2 Validation on Generic Input Signal

The emulator is run with a generic input signal and compared to the simulation results in order to ensure that it performs as expected. Tab. 4.3 shows the run-time parameter configuration used to validate the emulator.

Table 4.3: Run-time parameters for validation of emulator.

R_i	1/256
D_0	0
D_i	$\sqrt{e^{-\tau/\tau_{\text{slope}}}..},\ i = [1, ..., N]$
K_0	1
K_i	$0,\ i = [1, ..., N]$

The run-time parameters correspond to a 10 tap model where the power delay profile is set to exponentially decay with factor $\tau_{\text{slope}} = 1$ μs. The first tap will have a LOS component of 1. The interpolation rate is set to 1/256, which corresponds to a Doppler spectrum of

$$f_{\text{D}} = R_{\text{i}} * f_{\text{s}} * f_{\text{D,norm}} \approx 39 \text{ [kHz]}. \tag{4.2}$$

R_{i} is the interpolation rate, f_{s} is the sampling frequency and $f_{\text{D,norm}}$ corresponds to the normalized Doppler frequency fixed by the filter $A_{r_{\text{I}},r_{\text{Q}}}$ in the shaping filter.

The emulator is validated with the sinusoidal input signal

$$x(t) = e^{-j2\pi f_{\text{c}}} \tag{4.3}$$

at a carrier frequency of $f_{\text{c}} = 5$ GHz. The output $y(t)$ of the simulator as well as the emulator in time-domain are plotted in Fig. 4.1.

The frequency response of the simulator and emulator outputs can further be seen in Fig. 4.2. Fig. 4.3 shows the frequency response of the emulator output zoomed in at the 5 MHz region.

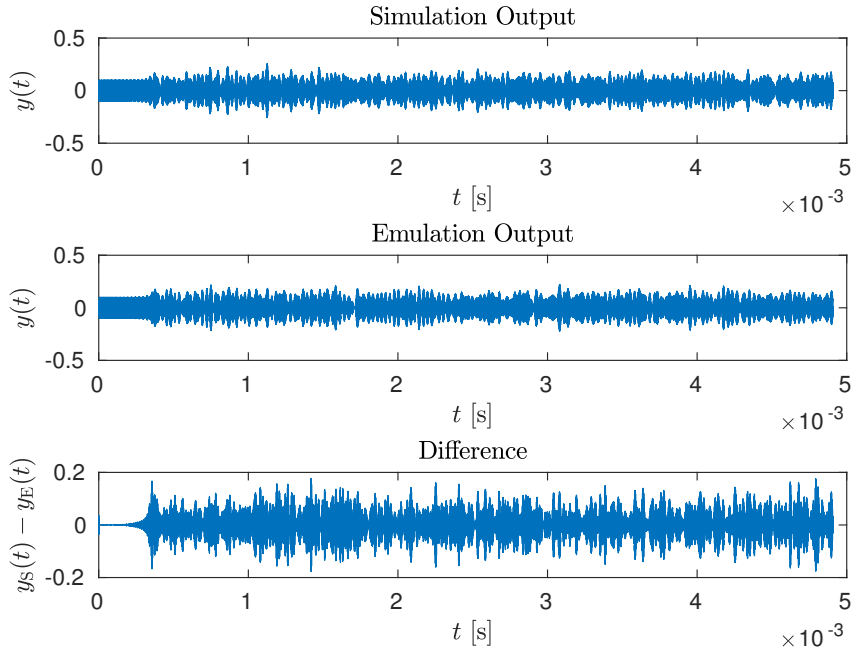


Figure 4.1: Output of simulation and emulation with sinusoidal input signal, along with the difference between the simulation and emulation results $y_{\mathrm{S}}(t) - y_{\mathrm{E}}(t)$.

4.3 Scenario Parameters and Results

In order to ensure that the emulator is able to successfully recreate all the scenarios mentioned in Sec. 3.1.4, run time parameters corresponding to each scenario are set, and the output of the emulator is analyzed. The values used to generate the channel models for the scenarios are based on values used by [1].

4.3.1 En-route Emulation

As mentioned in Sec. 3.1.1, the en-route scenario is characterized as a two-tap channel consisting of LOS and one reflection (or cluster of scatters). The relative delay of the second tap is inversely proportional to the altitude of the plane. This channel also experiences the highest Doppler shift, since the airplane is at cruising speed. Tab. 4.4 summarizes the parameters used to emulate this channel.

Table 4.4: Run-time parameters for testing the en-route scenario.

Velocity [m/s]	300
f_{c} [GHz]	5
Height [m]	1800
N	2

The square root of the PDP that is fixed for this emulation scenario to create the 2 tap model can be seen from Fig. 4.4. The resulting channel impulse response

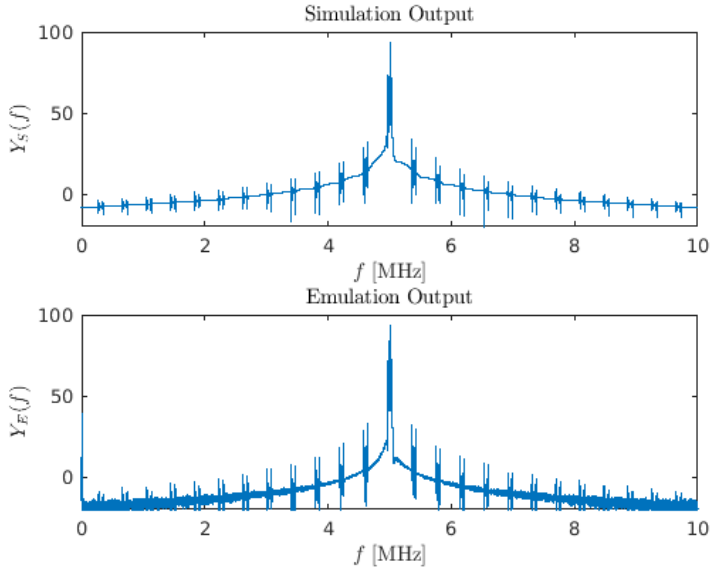


Figure 4.2: Frequency response of simulation and emulation with sinusoidal input signal.

$|c(t,\tau)|$ is shown in Fig. 4.5.

4.3.2 Takeoff/Landing Emulation

The takeoff/landing scenario will, as stated in Sec. 3.1.2, resemble a multi-tap channel where the power of each tap has an exponential decay with slope $\tau_{slope} = 1\mu$s, just like in Sec. 4.2.2. The parameters of the test scenario can be seen in Tab. 4.5.

Table 4.5: Run-time parameters for testing takeoff/landing scenario

Velocity [m/s]	50
Height [m]	500
f_c [GHz]	5
N	10

The square root of the PDP that is set for this scenario can be seen in Fig. 4.6. Fig. 4.7 shows the impulse response of the channel $|c(t,\tau)|$ with the parameters set as given.

4.3.3 Taxi/Parking Emulation

The taxi/parking scenario with a one-tap Rician model can be described by one LOS factor (K_0) and one Rayleigh factor (D_0). The ratio of this can be seen in Fig. 4.8. The run-time parameters are summarized in Tab. 4.6. The resulting impulse response $|c(t,0)|$ is plotted in Fig. 4.9. $|c(t,\tau)| = 0$ for $\tau \neq 0$, so the these taps are ignored.

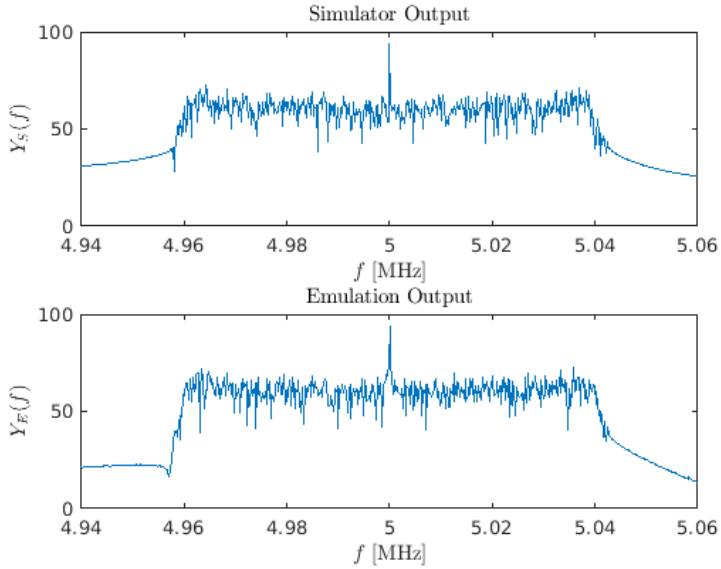


Figure 4.3: Frequency response of simulator and emulation with sinusoidal input, zoomed in at 5 MHz.

Table 4.6: Run-time parameters for testing taxi/parking scenario

Velocity [m/s]	5
Height [m]	0
f_c [GHz]	5
N	1

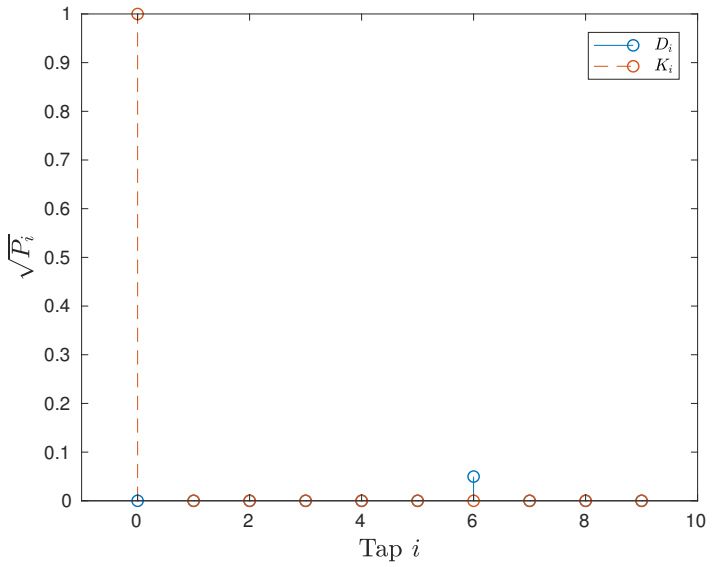


Figure 4.4: Square root PDP of en-route scenario. The plot shows the LOS constant factor K_i and average noise scalar D_i for each tap i.

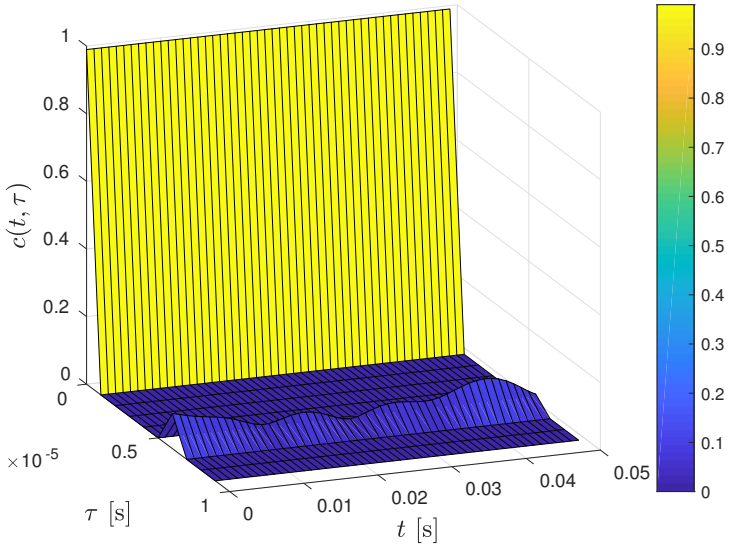


Figure 4.5: Impulse response $|c(t, \tau)|$ of en-route channel model

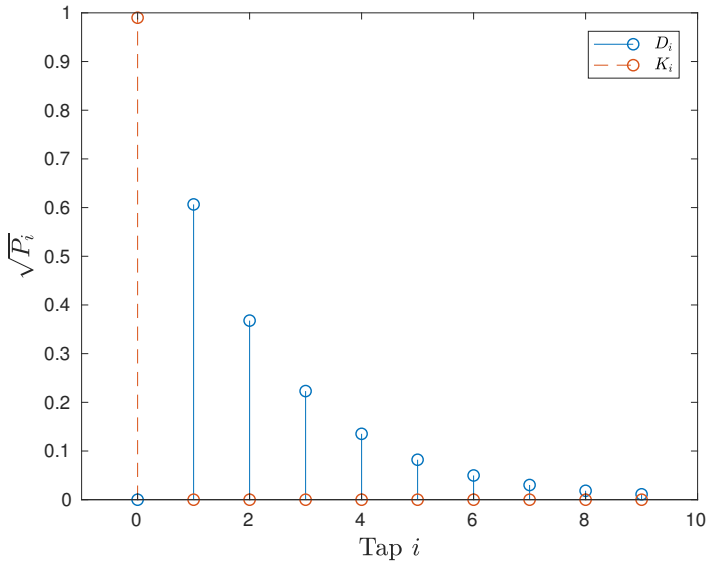


Figure 4.6: Square root PDP of takeoff/landing scenario. The plot shows the LOS constant factor K_i and average noise scalar D_i for each tap i.

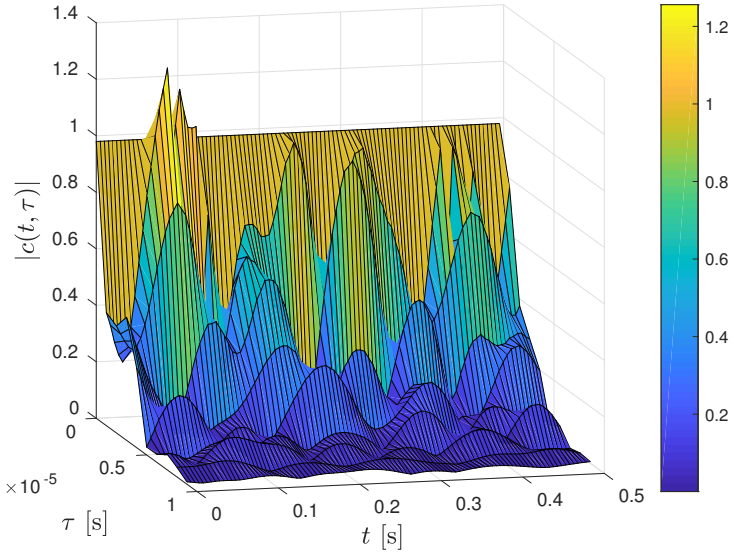


Figure 4.7: Impulse response $|c(t, \tau)|$ of takeoff/landing channel model.

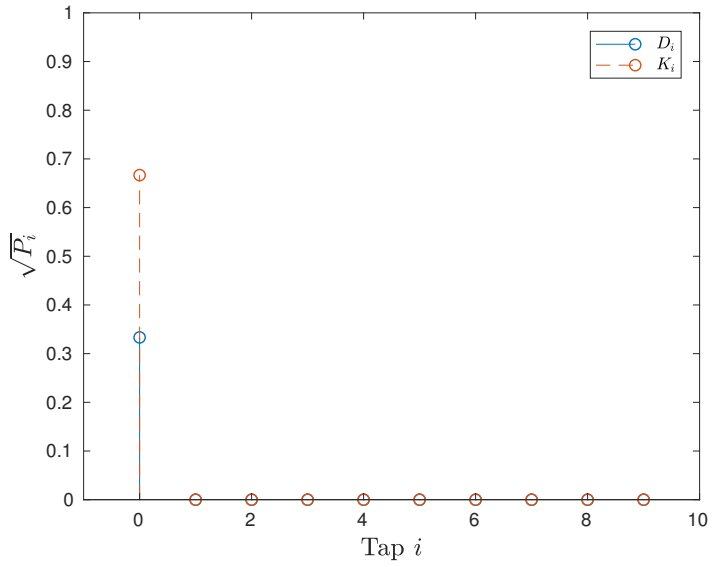


Figure 4.8: Square root PDP of taxi/parking scenario. The plot shows the LOS constant factor K_i and average noise scalar D_i for each tap i.

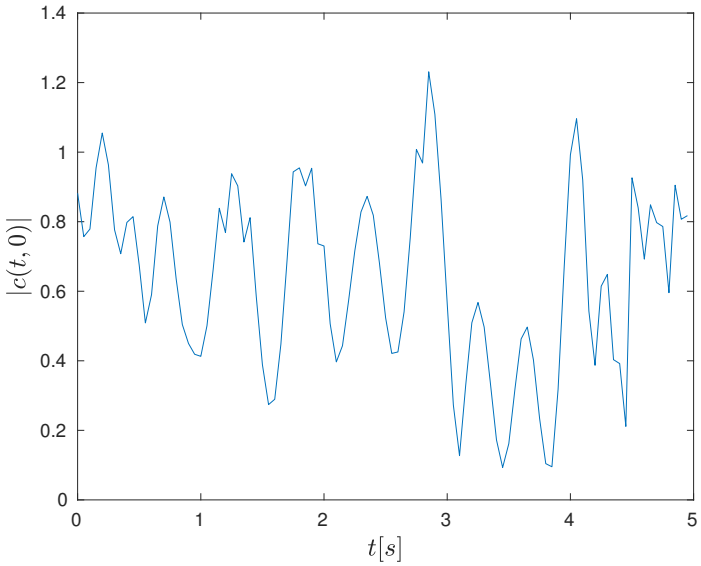


Figure 4.9: Impulse response $|c(t,\tau)|,\ \tau = 0$ of taxi/parking channel model.

5
Discussion

In this chapter, the results are discussed and analyzed. Based on these results, some changes to the emulator are suggested in order to further improve performance, channel accuracy, as well as flexibility.

5.1 Performance

The primary performance measure of the emulator is how fast data can be written to and read from the two input/output ports. The purpose of this test is to make sure that the rates match that of the requirements.

As could be seen from Sec. 4.2.1, the data rates on both the input and output converge to the expected 400 MB/s. The reason that shorter runs of the "dd" command resulted in lower rates is due to the setup time of the system. This includes first the delay architecture that was introduced to postpone some samples in order to create a wider delay spread. The second source of reducing the rate comes from the fact that the shaping filter requires a larger number of clock cycles, due to the amount of operations required per output. The read/write rate of the desktop computer may also factor into the reduced rate.

5.2 Emulation with Sinusoidal Input

The first step of validating the channel model produced by the emulator is to compare the results to a simulation with the same parameters. In Fig. 4.1, the time-domain output of both the simulator and emulator are plotted.

The time domain response of the emulator varies slightly from that of the simulation. This is because, unlike the software implementation of the data streams, the hardware AXI4 has finite memory. This will result in the *noise stream* block distributing the noise samples differently than in the simulation, as a noise sample is assigned if the AXI4 is not full, which in turn produces a time-response different from the result observed in the simulation.

In Figs. 4.2 and 4.3 however, it becomes quite clear that the emulator is able to reproduce the model from the simulation. The frequency response from the emulator output matches both the theoretical Doppler spread, as well as the simulation results. Although the output is not the exact same as that of the simulation, the channel parameters produced still follow the expected Rayleigh fading channel model in combination with a LOS, which is seen by the peak at 5 GHz in Fig. 4.2 and 4.3.

5.3 Scenario Emulation

The results of the three scenarios mentioned in the Sec. 3.1 are presented in Sec. 4.3. The plots show how the channel produced by the emulator change over time for each delay tap.

In the en-route scenario, it can be seen in Fig. 4.5 that the amplitude of the LOS tap remains constant at around 1 throughout the whole emulation as expected. In reality this may increase or decrease as the distance between the airplane and transmitter changes, or if something else interferes with the LOS component of the signal. The second tap, which is the result of a reflection or cluster of reflections will vary over time as expected, and due to the fast moving airplane, the amplitude of this tap will change very quickly as well (in the order of 1 μs).

In the resulting channel for the takeoff/landing scenario, it can be seen that the impulse response now consists of 10 different paths with varying average amplitude. The LOS component still remains constant through the emulation. The amplitude of the remaining 9 taps will however vary according to the Rayleigh distribution over time. These variations will, due to the lower average speed of the airplane during this stage, vary much slower that the en-route scenario (order of 10 μs).

The last scenario, the taxi/parking scenario, will only consist of one resolvable path, and therefore will be modeled using the 1 tap Rayleigh or Rician fading model. The result of this relatively slow changing channel seen in Fig. 4.9 shows a situation where LOS is still present, resulting in a higher average received power, but due to movement, the amplitude will vary over time. The movement of the plane in this scenario is however very slow, resulting in slow variations over time.

5.4 Improvements to the Emulator

From the synthesis summaries in Sec. 4.1.2, it can be seen that the hardware is not being 100% utilized by the current emulator architecture. The spare resources that are not being used by the system could be utilized to improve the performance of the channel model generation.

5.4.1 Variable Delay

In the case of the en-route scenario, the second tap was emulated with a delay of 7 μs. However, this is only one example of the delays that may be experienced for this scenario. If the delay observed is $> 9\mu$s, the emulator at its current state would not be able to reproduce this channel model. In addition, the current setup requires that the delay is rounded to the nearest μs since each tap is separated by a fixed number of samples.

To solve this issue, a variable delay mechanism is needed that is able to increase or decrease the delay between each tap. This could be achieved by setting the wait_x delay input to a run-time variable. By changing this value, each separation of each tap could be changed in order to stretch over a larger span of delay taps, as well as to be able to emulate delays at higher resolution.

5.4.2 Improved Interpolator

Although the sidelobes caused by the linear interpolator are relatively low compared to the main Doppler spectrum (as seen in Fig. 4.2), a spline interpolator could be used to improve the channel model. The spline interpolator will, as previously stated, be more costly for the hardware. However, since the system is not fully utilized, a spline interpolator could be considered for testing more sensitive applications. Another alternative is to implement a cascade of interpolators, such as in [4]. As mentioned, the lowest interpolation rate possible to achieve real-time performance is 1/256. Therefore, the hardware requirement of the complete system can be effectively reduced with a fixed interpolator at this rate, followed by a variable rate interpolator, which enables increased resource sharing.

5.4.3 Improved Shape Filter

Increasing the number of samples in the shape filter is another option for improving the channel model for the channel generation, and would be achieved by choosing a wider window function. This would result in a cleaner Rayleigh spectrum for the interpolator. The benefits of doing this, however would be marginal, since most of the deficiencies in the spectrum are produced by the interpolator. It would also add a large number of MAC operations for a minor improvement.

5.4.4 Increased Number of Taps

Additional taps could be added to the system to be able to model more reflections. However, this is not beneficial for the en-route scenario, nor for the taxi/parking scenario, since they only require a fixed number of taps < 3 to emulate. However, if the takeoff/landing scenario of the measured channel contains larger number of taps, this is an option to produce a more realistic impulse response.

6
Conclusion

6.1 General Discussion

The resulting IP core is a complete channel emulator that can be used for testing aeronautical communication systems. The emulator is a flexible implementation where the scattering environment, the strength of the LOS component as well as the velocity and delay of each reflection can be decided by externally writing to the on-board registers.

In order to run the emulation, Gaussian noise is written to one of the input, while the actual input signal is written to the other. Using the registers and the collection of IP blocks, the emulator will then swiftly calculate the output signal, based on how the channel model has been set by the run-time variables.

The emulator was implemented around the theory introduced by [1] as well as [5], and is fully able to reproduce the channel models which were presented, including the en-route flight, along with landing/takeoff and taxi/parking scenarios. The current implementation of the emulator does not have an easy method for emulating shadowing nor path loss, as this is easier to generate through the external hardware. However, by choosing the channel parameters to values that reflect an increased attenuation, it is possible to mimic the effects of these additional power losses.

6.2 Emulator Testing Capacity

The emulator is only as good as the underlying model. Therefore, work has to go into finding the exact statistical model for specific aeronautical communication channels in order to accurately emulate the live environment. The implemented model, however does serve as a good start for stress testing the modem used for communicating. With the flexibility of being able to easily set channel parameters during run-time, and to handle the three main scenarios that will be experienced in a flight, all that is missing is the statistics that one might find in a real aeronautical channel.

This emulator is not limited to aeronautical communication. Many other usage areas such as vehicular communication would have similar channels as the one implemented. Therefore, many of the building blocks used for this implementation could even be reused for the testing of other channels that experience similar kinds of fading, Doppler and scattering environments.

6.3 Further Work

As the main intention with the emulator is to test the existing communication system, the next step would therefore be to write the complete IP to a dedicated chip that can be used to test the modem. This chip would require first a AD converter to sample the output of the transmitting modem. On the other end, the output needs to be converted to an analog signal that represents the transmitted signal after the channel. In order to make a fully free-standing emulator that only requires a input signal, a circuit design would also need to be created in order to generate the noise. There exist numbers of noise generating circuits for purchase, and some chips with on-board FPGA also offer a built in noise source solution.

Bibliography

[1] S. Blandino, "Wireless channel-simulation for mobile receivers in airborne environment," 2014.

[2] J. Sparks, "Aircraft communication," *Aviation Pros*, 2007.

[3] M. Garcia, "Inflight connectivity to hit the skies for two-thirds of airlines by 2020." https://www.phocuswire.com/Inflight-connectivity-to-hit-the-skies-for-two-thirds-of-airlines-by-2020, 2017. [Online; accessed 4-July-2019].

[4] D. M. E. Briggs, B. Nutter, "A real-time multi-path fading channel emulatordeveloped for lte testing," *SDR 11 Technical Conference*, 2011.

[5] E. Haas, "Aeronautical channel modeling," *IEEE Transactions on Vehicular Technology*, vol. 51, pp. 254–264, March 2002.

[6] A. Goldsmith, *Wireless Communication*. NY, USA: Cambridge University Press, 2005.

[7] W. Jakes, *Microwave Mobile Communication*. NY, USA: John Wiley Sons Inc., 1975.

[8] P. Bello, "Correlation functions in a tapped delay line model of the orbital dipole channel," *IEEE Transactions on Information Theory*, vol. 9, pp. 2–11, January 1963.

[9] G. L. Turin, *Communication Through Noisy, Random-Multipath channels*. PhD thesis, Massachusetts Institute of Technology, 1956.

[10] ITU, "Propagation by diffraction," *Recommendation ITU-R P.526-14*, 2018.

[11] J. O. Smith, *Physical Audio Signal Processing*. W3K Publishing, 2010.

A

Appendix 1

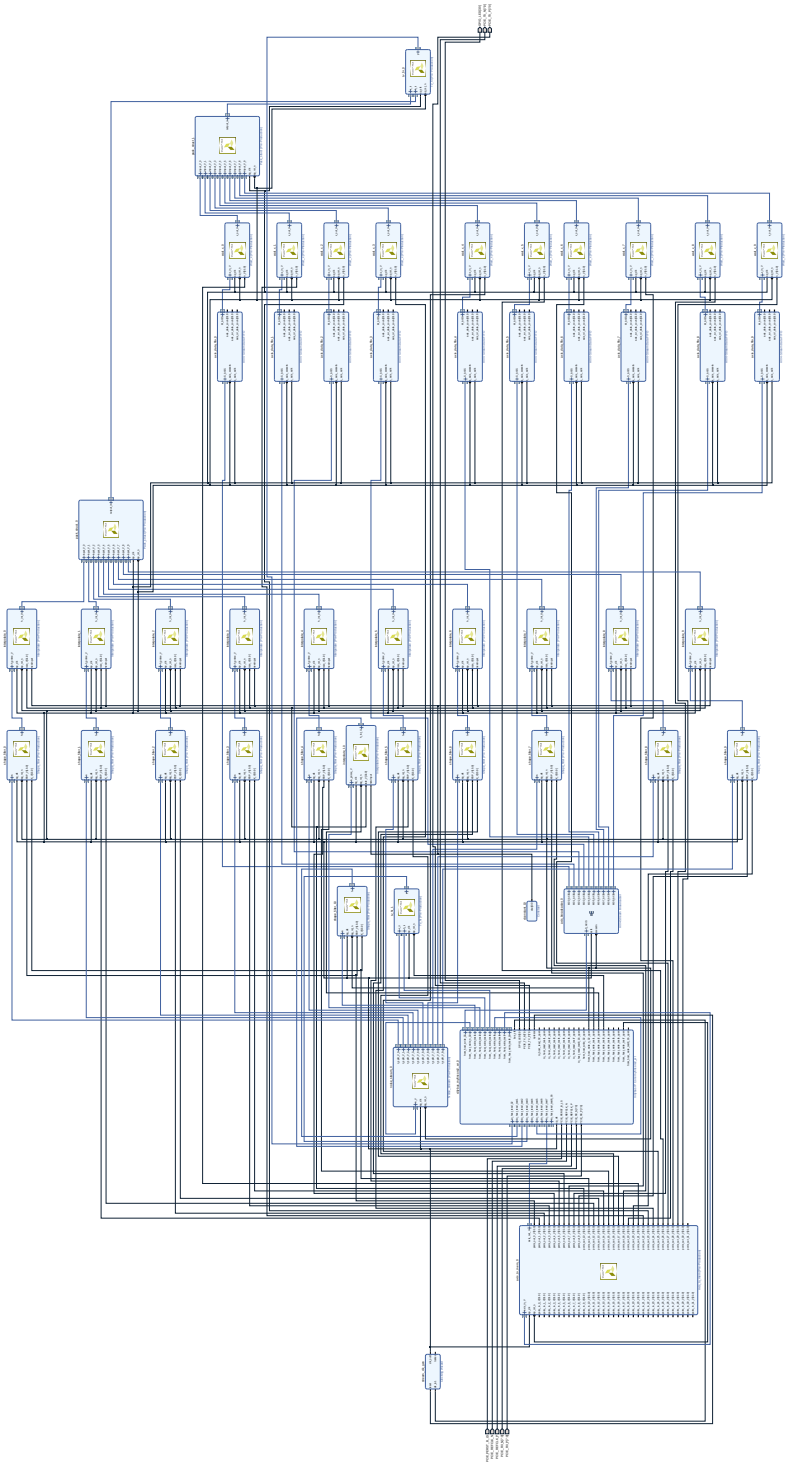


Figure A.1: Final block design in Vivado.