Smart FPGA-Based SDR Core for Space Communication

Marwa Albeshr

MSc. Thesis

July 2023

A thesis submitted to Khalifa University of Science and Technology in accordance with the requirements of the degree of Master of Science in Electrical and Computer Engineering in the Department of Electrical Engineering and Computer Science.

Smart FPGA-Based SDR Core for Space Communication

by

Marwa Albeshr

A thesis submitted in partial fulfillment of the
requirements for the degree of

Master of Science in Electrical and Computer Engineering

at

Khalifa University

Thesis Committee

Prof. Hani Saleh (Main Advisor),
Khalifa University

Dr. Lina Bariah (Co-Advisor),
Khalifa University

Prof. Baker Mohammed (Co-Advisor),
Khalifa University

Prof. Sami Muhaidat (RSC Member 1),
Khalifa University

Prof. Arafat Al-Dweik (RSC Member 2),
Khalifa University

July 2023

Abstract

Marwa Albeshr, "**Smart FPGA-Based SDR Core for Space Communication**", M.Sc. Thesis, M.Sc. in Electrical and Computer Engineering, Department of Electrical Engineering and Computer Science, Khalifa University of Science and Technology, United Arab Emirates, July 2023.

Today's communication systems are modular such that each task required by the communication system for effective data transmission and reception is described by a block that optimizes it. Such an approach has proved to be efficient in describing communication systems. However, the individual optimization of these blocks leads to the sub-optimality of the overall system. In contrast, Deep Learning and Machine Learning concepts ensure the end-to-end optimization of systems through training.

Moreover, CubeSats offer attractive research opportunities due to their low cost and small size. However, CubeSats/Nanosatellites are limited due to the small communication window in which they must transmit their data, the power consumption, size, as well as the limited on-board resources. Therefore, CubeSats throughput must be enhanced, which can be achieved using an SDR implemented on an FPGA.

In this paper, an Autoencoder, which is an Artificial Neural Network, is used to build an FPGA-based communication system that targets CubeSats. This implementation can ensure the global optimization of the system, throughput enhancement, and the ability to reprogram. This is done by considering a High-Level model consisting of a communication system based on the Autoencoders concept that includes a Rician Fading Channel to suit space communication applications. The Encoder, Latent Space/Code, and Decoder of the Autoencoder are equivalent to the communication system's Transmitter, Channel, and Receiver, respectively. The High-Level model is then trained repeatedly to obtain the hyperparameters that would result in satisfactory validation and training accuracy. The learnable parameters of the network (i.e., weights and biases) are extracted and converted from Floating-Point to Fixed-Point representation. The system's inference is built using Register Transfer Level (RTL) Modelling to target hardware implementation (e.g., FPGA and ASIC). The simulation results show that the RTL model of the system work as the High-Level model.

Lastly, the RTL Model functionality is verified and synthesized for ASIC and FPGA implementation.

Indexing Terms: Autoencoder, ANN, Communication System, System Verilog, FPGA, ASIC.

Acknowledgement

First, I'd like to express my utmost appreciation and gratitude to my Main Advisor Prof. Hani Saleh for his relentless support throughout the duration of my study. His guidance, numerous words of advice, and unwavering trust has never failed to motivate me when hope was lost. Most importantly, I'd like to thank him for never turning me down whenever I knocked on his office door.

I'd also like to express my deepest gratitude to my Co-Advisor Dr. Lina Bariah for her supportive words and indispensable feedback. Her immense knowledge and insight were invaluable and has helped me tackle many obstacles. Above all, her kind words and advice were essential in motivating me to complete the pending tasks.

I'd like to convey my sincere gratitude to my Research Committee Member Prof. Sami Muhaidat for sharing his imperative knowledge, expertise, and career advice. His thought-provoking questions and insight proved to be monumental towards the completion of this thesis.

I'd also like to extend my appreciation to the remaining Research Committee Members Prof. Arafat Al-Dweik and Prof. Baker Mohammed for being part of the committee and sharing their valuable comments and feedback.

Furthermore, I'd like to thank Dr. Ghadeer Alsuhi, Selina Shrestha, Mohammed Tolba and Abubakar Sani Ali for sharing their knowledge and providing unconditional help. Their aid was truly vital in tackling technical issues and mishaps.

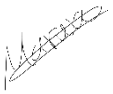
Lastly, I'd like to extend a special thanks to my family and friends for their consistent support. Their encouraging words, unconditional trust, and reassurance when times were rough were much appreciated.

Declaration and Copyright

Declaration

I declare that the work in this thesis was carried out in accordance with the regulations of Khalifa University of Science and Technology. The work is entirely my own except where indicated by special reference in the text. Any views expressed in the thesis are those of the author and in no way represent those of Khalifa University of Science and Technology. No part of the thesis has been presented to any other university for any degree.

Author Name: Marwa Albeshr

Author Signature: 

Date: July 14th, 2023

Copyright ©

Contents

Table of Contents

List of Figures

List of Tables

List of Abbreviations

ADC	Analog to Digital Converter
AE	Autonecoder
ASIC	Application-Specific Integrated Circuit
AWGN	Additive White Gaussian Noise
CPU	Central Processing Unit
DAC	Digital to Analog Converter
DDC	Digital Down Converter
DL	Deep Learning
DSP	Digital Signal Processor
DUC	Digital Up Converter
Eb/No	Energy per bit to the Noise Power Spectral Density
FPGA	Field Programmable Gate Array
GPP	General Purpose Processor
GPU	Graphics Processing Unit
HDL	Hardware Description Language
IBUF	Input Buffer
IOB	Input/Output Box
LDCE	Transparent Data Latch
LEO	Low Earth Orbit
LOS	Line Of Sight
LPF	Low Pass Filter
LSTM	Long Short-Term Memory
LUT	Look Up Table
MAC	Multiply And Accumulate
ML	Machine Learning
MLD	Maximum Likelihood
OBUF	Output Buffer
PE	Processing Element
PSK	Phase Shift Keying
QAM	Quadrature Amplitude Modulation

QPSK	Quadrature Phase Shift Keying
ReLU	Rectified Linear Unit
RF	Radio Frequency
RTL	Register Transfer Level
SCAN	Space Communication and Navigation
SDR	Software Defined Radio
SISO	Single Input Single Output
VHDL	VHSIC Hardware Description Language

Chapter 1. Introduction

This chapter introduces the background information regarding communication systems. It also covers the potential solution and motivation to the issue of the communication systems' non-linearities and sub-optimality. The section ends with the definition of the Problem Statement and Thesis Organization.

1.1 Background

The design of traditional communication systems involves signal processing, information theory, and statistics which makes the design both reliable and efficient [1]. These concepts, which are generally linear, are optimal for describing mathematically tractable channel models and certain components used in the communication system [2]. However, the non-linearities within the communication system can only be approximated [1].

In addition, the different blocks that work together to build a communication system (e.g., Modulator, Encoder, etc.) are designed such that they are optimized individually; each block is designed separately to fulfill a specific function needed by the communication system [3].

This approach has produced communication systems that work well. However, having independently optimized blocks does not necessarily mean that the whole communication system is indeed optimized [1].

Many Machine learning and Deep Learning concepts are actively used in different research areas. For communication, the existence of relevant Deep Learning software libraries and specialized hardware (e.g., GPUs, FPGAs, etc.) enable the application of ML/DL concepts in communication systems and signal processing [4]. Using these concepts in communication systems allows global optimization of the system, including its imperfections, without the need to resort to the commonly used block structure [2].

The possibility of creating a communication system via Neural Networks was proved in [4]; in which the authors replaced the physical layer processing of the conventional communication system by Artificial Neural Networks (i.e., Autoencoder) and introduced two-phase training strategy for the system. Moreover, authors in [5] mention how DL and ML

concepts are used to solve issues related to the physical layer of the communication system, such as modulation and channel estimation.

1.2 Motivation

Regardless of how improved Satellite Communication is, it still suffers from limited number of resources as well as the high speed at which the satellite orbits the earth [6]. The design of Communication Systems in Nanosatellites can prove to be challenging, especially when considering the power consumption and size; in such satellites, power is limited, and the decreased size can also lead to the reduction of the signal strength [7].

According to the Jet Propulsion Laboratory at NASA, CubeSats, which are a type of Nanosatellites, offer attractive new research opportunities due to their low cost and small size. Therefore, they are often used in academia [8]. Additionally, having compact size and lightweight characteristics enables developers to create fully operational satellites in both time- and cost-effective manners [9]. CubeSats are known to operate in low earth orbit (LEO). This creates a disadvantage concerning the duration in which the CubeSat can communicate its data to the ground station, which is relatively short compared to the size of data that must be transmitted back to the ground station [10]. Moreover, the communication system is exposed to inconstant conditions in space, such as atmospheric effects, which can degrade its performance [11], highlighting the need for a reconfigurable communication system.

As will be discussed later, the SDR technology can be reconfigured by reprogrammable hardware such as an FPGA. According to [11], the usage of SDR in small satellites can lead to improved data throughput. Moreover, using an Autoencoder can help in compressing the data before transmission, which can help in saving bandwidth from data transmission as seen in [12]. Autoencoders are a type of an ANN that can reconstruct inputted data. It is mainly used for feature extraction and dimensionality reduction. Moreover, Autoencoders can be used for anomaly detection, denoising, and information retrieval [13].

FPGAs do not run neural networks sequentially which makes the execution much faster. Also, using Fixed-Point representation in the FPGA implementation can reduce resource usage and improve overall performance [14]. The work done in [14] showed that the combination of Autoencoders and FPGAs was able to reduce resources usage and consume less power when compared to other hardware such as CPU and GPU.

Therefore, using Autoencoders on FPGA as an SDR to target CubeSats/Nanosatellite applications can be beneficial in terms obtaining the global optimization of the communication system, the ability of reprogramming the system, reduce resource usage, consume less power, as well as improved data throughput.

1.3 Problem Statement

The SDR-related applications in the literature use ready-made Verilog/VHDL codes obtained from tools. These applications implement a model-based design using MATLAB/Simulink to build the SDR architecture. Then, they use tools to generate the equivalent Verilog/VHDL code and the FPGA configuration file. It was also observed that the applications that develop the design using HDL directly do not implement the entire SDR; they either focus on implementing the receiver or some part of the SDR, such as the modulator.

It is important to note that it is better to design the HDL code than use a ready-made HDL because it is superior in terms of optimization. The sub-optimality of the ready-made HDL applications makes them inappropriate for hardware implementations, such as FPGA and ASIC, which means that they can be inadequate in terms of performance metrics such as speed, area, resources, and power consumption.

Moreover, FPGA-based ML/DL applications found do not implement a communication system in System Verilog. Therefore, the designed communication system is represented using a Deep Learning Neural Network concept (i.e., Autoencoder). This is because the Autoencoder's structure and functionality are similar to that of a communication system. Also, the Autoencoder-based communication system does not require the rigid block structure of the conventional communication system. Most importantly, Autoencoders, like other DL neural networks, are function approximators that enable end-to-end optimization of the system including its imperfections. So, the system's non-linearities will be optimized along with the entire system [2].

This thesis intends to create an entire FPGA-based communication system (i.e., Transmitter and Receiver) using Deep Learning concepts (i.e., Autoencoder) to target satellite applications so that it would represent the satellite's uplink/downlink. The system will be developed in an optimized fashion; by creating the corresponding RTL model from scratch, testing its functionality using FPGA/ASIC, and generating the various performance metrics.

1.4 Thesis Organization

Chapter 2 contains a thorough literature review of the related work as well as the contributions of this thesis. Chapter 3 highlights the functionality of the High-Level Model of the Communication System, the generation of the dataset, the hyperparameters of the system, the training results, and the testing results. Chapter 4 describes the way in which the RTL Model of the Transmitter, Channel, and Receiver were developed, and it demonstrates their simulation results. Chapter 5 touches upon the RTL Verification of the RTL Model through two design flows (i.e., FPGA and ASIC). It also shows the extracted performance metrics.

Chapter 2. Literature Review

This chapter focuses on discussing the previous related work found in the literature. The related work involves: Autoencoders, Autoencoder Applications, SDRs, and SDR Applications. The chapter ends with the contributions done in this thesis paper.

2.1 Autoencoders

An Autoencoder is an unsupervised Neural Network used in Deep Learning. It is used to find the latent space (i.e., a low-dimensional representation of the input), which is used to reconstruct the input at the output layer [1]. As shown in Fig. 1, Autoencoders consist of the Encoder, Latent Space/Code, and the Decoder. The input shrinks in size in the encoder part to generate the latent space, which is then expanded in the Decoder to reconstruct the input [4]. Autoencoders are commonly used for feature extraction as well as dimensionality reduction [15]. However, since both communication systems and Autoencoders have essentially the same functionality (i.e., reconstructing the input at the decoder/receiver), a communication system can be modeled using an Autoencoder; in which the Autoencoder's Encoder, Code, and Decoder represent the Communication System's Transmitter, Channel, and Receiver, respectively.

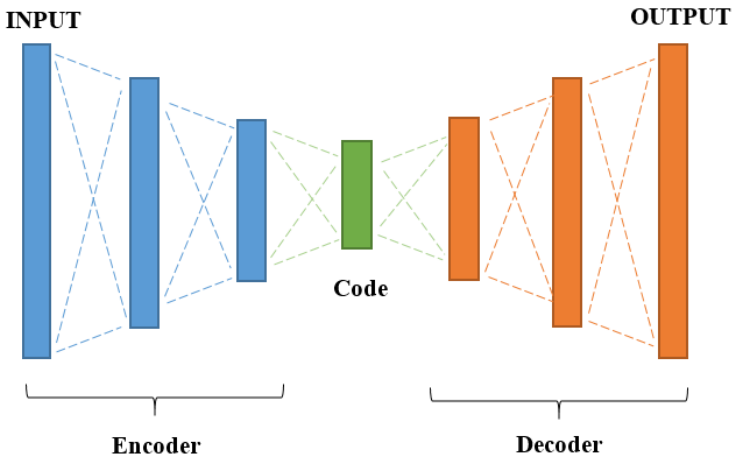


Figure 1: Autoencoder Architecture.

2.2 Autoencoder Applications

A convolutional Autoencoder is built in [14] for image compression. The High-Level model of the network was realized using Keras Library and Python. Periodic layer multiplexing was used to create the network. Using Verilog Description Language and Xilinx's Vivado, the network was tested for FPGA, GPU, and CPU. The results showed that the FPGA is superior in terms of computing time, power consumption, and computing performance.

In [16], an Autoencoder is used along with an LSTM network to detect outliers from a temperature dataset. The Autoencoder is used for feature extraction and dimensionality reduction, whereas the LSTM is used for outlier detection. The Autoencoder and LSTM were trained using Keras Tensorflow. Whereas Xilinx's Vivado was used to design the system's hardware accelerator. The Autoencoder functionality was presented by a Dense Module that performs matrix multiplication. This work also presented the learnable parameters using Fixed-Point representation instead of Floating-Point, which reduced resource utilization, response time, and power consumption.

In [17], an Autoencoder was developed to reconstruct handwritten digits from the MNSIT dataset. This network employed a classifier to evaluate the accuracy or correctness of the reconstruction. The neural network was optimized for hardware implementation by using weight pruning and quantization. Weight pruning sets small weights to zero, and quantization converts the Floating-Point representation of the network parameters to Fixed-Point representation. Three copies of the neural network were built to observe the effect of the optimization techniques; one network contained no optimization techniques, one contained weight pruning only, and one had both weight pruning and quantization. The results showed that the Quantized and Pruned network is the most resource-efficient out of the three networks.

A Stacked Autoencoder is built in [18], which can classify modulation types. The hardware implementation was optimized by using Fixed-Point representation. The modules used to realize the hardware architecture included a PE module and a Control module. The PE module executes the neuron's core functionality, which is the MAC operation. On the other hand, the Control module calls the PE module when it is needed for operation.

2.3 SDRs

A Software Defined Radio refers to the implementation of a radio in software rather than in hardware, meaning that the components that define a radio (i.e., the communication system) are built using software [19]. This is advantageous in terms of the flexibility given to the system to be updated or reconfigured through reprogramming. On the other hand, the conventional rigid radio would require changing the hardware to alter the system's functionality [20]. An SDR can be realized by using programmable hardware such as Application Specific Integrated Circuit (ASIC), Digital Signal Processor (DSP), General Purpose Processor (GPP), and Field Programmable Gate Array (FPGA) [21]. In the case of DSPs, they are programmed using high-level languages, which run sequentially. This becomes an issue when the program needs to run simultaneously. Therefore, ASICs/FPGAs are used [22]. However, ASICs are inferior to FPGAs in terms of flexibility as they are rigid and cannot be reprogrammed [23].

An SDR transceiver consists of five main components which are: 1) Antenna, 2) RF Front End, 3) Digital Front End, 4) ADCs and DACs, and 5) Signal Processing Block [20]. The signals' frequencies are changed in the RF Front End section [20], which contains components such as RF image Filter and an Oscillator [24]. The RF Front End can also have a power amplifier. The Digital Front End is used to implement Sample Rate Conversion and Channelization. It contains DUCs, DDCs, an Interpolation Filter, and LPFs. The ADCs and DACs are used to convert signals from Analog to Digital and from Digital to Analog, respectively. The Baseband Processing block or Signal Processing block is responsible for implementing various signal processing techniques such as Modulation and Encoding [20].

According to [25], SDR technology is used in different sectors, such as the Military and Space. One of the applications of SDR in space is the SCAN Testbed that was built and previously employed in the International Space Station by NASA. The usage of SDR in the testbed enabled the alteration of its functionality by inserting changes in the corresponding software [26]. Beyond the Testbed, NASA is currently using SDRs and is willing to use SDRs for future missions [27].

2.4 SDR Applications

The literature is brimming with applications containing the usage of SDR, especially communication systems developed using FPGAs. Applications found in [28], [29], and [30] focus on creating the receiver. The application in [28] uses SDR technology to develop a receiver that can be reconfigured in terms of the wireless architecture being implemented. It has shown that it can reconfigure the receiver with other wireless architectures on top of the two predetermined ones. The application in [29] also focuses on creating an SDR receiver. However, it has no hardware implementation of the proposed design. Also, it uses existing Simulink models of HDL-Optimized QPSK Transmitter and Receiver to create the SDR receiver. The Transmitter and Receiver models were altered to meet the required characteristics. Furthermore, the equivalent HDL code was automated using DSP Builder tool. In [30], the author develops an SDR receiver capable of receiving different signals by implementing multiple channels. It showed that the proposed architecture was able to be more resource-efficient when compared the conventional architecture. It is also important to note that this application creates the Verilog model directly.

Other applications, such as [22] and [31], focus on creating a modulator/demodulator using a specific modulation scheme, which is the Quadrature Phase Shift Keying (QPSK). In [22], a model-based approach was adopted to realize a QPSK modulator using SDR technology. This application heavily depends on tools such as Simulink and DSP Builder to simulate the model and automatically generate the HDL code. In [31], a QPSK modulator/demodulator model was designed using SDR technology that was developed using VHDL. The simulation has shown that the proposed model's BER performance matched the theoretical BER performance of QPSK.

The applications in [32], [33], [34], [35], and [36] develop an entire communication system. In [32], an SDR-based communication system that can be used among small satellites to achieve inter-satellite communication was developed. Even though this application develops an entire SDR (including both Transmitter and Receiver) and tests it on an FPGA, it generates the equivalent HDL code using the Workflow Advisor in Simulink. The application in [33] aimed to develop a communication system that utilizes low-cost SDR hardware and software tools. One possible use of this proposed communication system is collecting or receiving signals from several small satellites. In [34], an entire digital communication system is developed based on SDR and is tested on an FPGA. It adopts a model-based design in order

to develop the system. The testing on the chosen FPGA was accomplished by using the Workflow Advisor in Simulink, which produces the HDL code of the model. An implementation of a SISO communication system based on SDR is found in [35]. The author uses a specific modulation/demodulation scheme to modulate/demodulate the signals. The author also investigated and compared two types of symbol timing recovery schemes. In order to test the model in an FPGA, the Xilinx System Generator tool was used to generate a VHDL code. In [36], a communication system is designed using a multicore SDR. Both Transmitter and Receiver were simulated and developed using MATLAB/Simulink. However, only the Transmitter was implemented on the FPGA due to a licensing issue with the chosen decoder. In addition, the required VHDL code was produced by the Xilinx System Generator tool.

2.5 Contributions

In this work, an FPGA-Based framework is developed for satellite networks that is expected to deliver reliable and accurate wireless communication between satellites and ground stations. To realize such a system, an entire communication system is developed (i.e., Transmitter, Channel, and Receiver) using Autoencoders. To the best of the authors' knowledge, such an application is not found in the literature. To establish the said system, the following contributions are executed:

1) A high-level Autoencoder-based communication system is integrated with a Rician Fading Channel such that it is suitable for space applications.

2) The required hyper-parameters of the system are found through trial and error to get an acceptable validation accuracy. The system is trained and tested to obtain the learnable parameters of the network.

3) The learnable parameters are converted to Fixed-Point representation to optimize for hardware implementation.

4) The equivalent RTL model of the system is developed using System Verilog.

5) The RTL model is verified by synthesizing it for ASIC and FPGA to obtain the equivalent gate-netlist and performance metrics.

Chapter 3. High Level Model

This chapter shows the work done to establish the High-Level Model of the Autoencoder-based communication system in MATLAB. It demonstrates how the hyperparameters of the High-Level model were found (i.e., number of layers, neurons, learning rate, etc..), the generation of the dataset needed, the training process, and the testing of the network.

3.1 Communication System Model

The conventional high-level implementation in [37] creates a wireless communication system using an Autoencoder as seen in Fig. 2. The Transmitter/Encoder is given a symbol S, which can be any value between 1 and M; the latter is equal to 2^k, where k is the information bits. The symbol goes through the Transmitter/Encoder to get X, the transmitted signal. The transmitted signal goes through the channel that is modeled by the AWGN Layer, which produces the received signal Y. The received signal goes through the Receiver/Decoder in order to produce an estimation of the input symbol, which is $\hat{S}$.

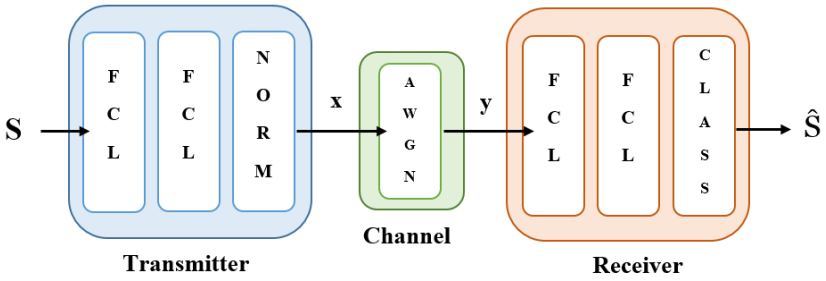


Figure 2: Baseline Communication System Architecture.

The considered CubeSat uplink/downlink channel model is the Rician Fading Channel which has been demonstrated to show accurate representations of the channel between satellites and ground stations. For example, in [38], the Rician Channel Model has proved to have better properties than other systems for a satellite communication system. In addition, the Rician Fading is simulated for several values of the K-Factor, which is a parameter that determines the severity of the fading; evidently, as the K-Factor increases, the probability of having a LOS increases [39]. According to [40], typical values of K-factor lie within the range of 2 to 7.

The considered structure of the Autoencoder can be seen in Fig. 3.

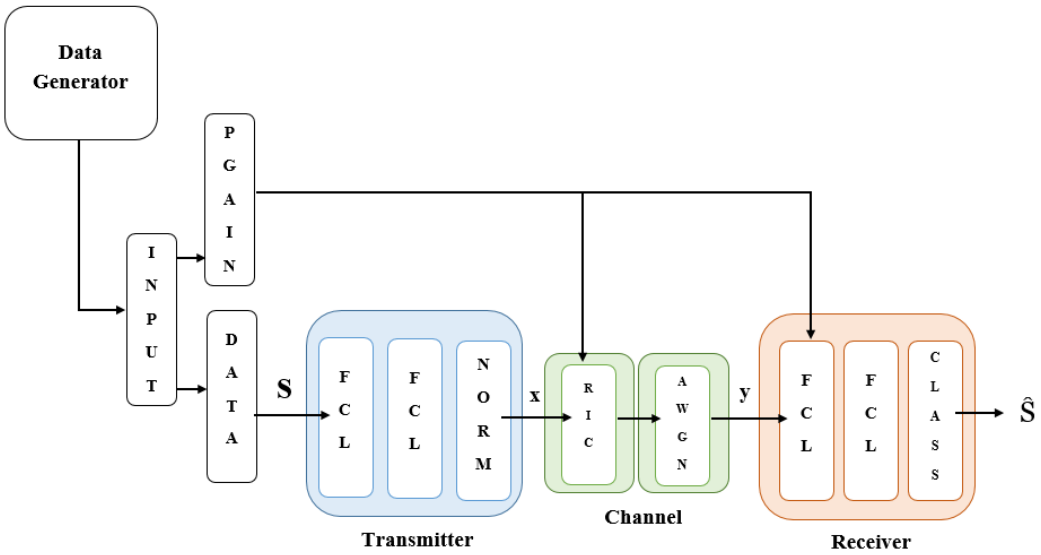


Figure 3: Considered Communication System Architecture.

The channel is represented by Rician followed by AWGN. Rician is fed with the appropriate path gains and the transmitted signal's real and imaginary parts. The Rician channel model then does the complex multiplication of these inputs to generate the effect of the Rician channel. The resultant real and imaginary parts are fed into AWGN to add noise to the signal which generates the received signal that is fed into the receiver.

A Rayleigh Fading Channel System Model was also developed which can be used for terrestrial communication systems. The corresponding High-Level Model, Training results, Testing results, RTL Model, and RTL Verification can be seen in Appendix A.

3.2 Data Generation

As seen in Fig. 3, a Data Generator in MATLAB was implemented to generate synthetic data which included the generation of the symbols S as well as the path gains of the Rician Fading for each symbol. The symbols represent the input to the autoencoder. Whereas the path gains were generated to demonstrate the performance for satellite networks.

The randi() function was used to generate random input data with a max value of M-1. On the other hand, the path gain for each input data was generated using the comm.MIMOChannel() System Object by specifying the required arguments.

The dataset includes 8K samples; where 72% are for training, 18% for validating, and 10% for testing. This division of the dataset was found through trial and error.

3.3 Hyper-Parameters

Numerous trial and error attempts were executed to obtain the number of layers, neurons, learning rate, and R2 regularization that would give a satisfactory training result. Table 1 describes the final architecture of the network. The Transmitter consists of 2 Fully Connected Layers (Layers 0 and 1) followed by a Normalization Layer that encodes the symbol S to create the transmitted signal. The channel, which consists of Rician Fading and AWGN, takes the transmitted signal and the corresponding path gain to actualize the effect of the Rician Fading and AWGN. The received signal from the channel is fed into the Receiver. The Receiver consists of 2 Fully Connected Layers (Layers 2 and 3) that decodes the signal accordingly. Finally, the Classification Layer classifies the decoded signal into one of the existing categories (i.e., 0 to M-1) which represent the reconstructed inputs.

Table 1: Network Layers.

Layer	Number of Neurons
1st Fully Connected Layer (Layer 0)	8
2nd Fully Connected Layer (Layer 1)	2
Rician	2
AWGN	2
3rd Fully Connected Layer (Layer 2)	8
4th Fully Connected Layer (Layer 3)	4

As will be shown later, the input to the Transmitter is a one-hot encoded four-bit number. For that reason, Layer 3 is set to four neurons to represent the four bits of the input that will be reconstructed through the Receiver. Whereas Layer 1 is set to have two neurons because the output of the Transmitter is essentially two numbers: the real and complex values of the transmitted signal. Evidently, the number of neurons in the channel (i.e., Rician and AWGN) will also be two neurons. The number of neurons in Layers 0 and 2 are set through trial and error.

Table 2 displays the communication system parameters and its training options which are set through trial and error prior to the training process. Max epochs represent the number

of times the training goes over the entire training data. The mini batch size represents the segmentation of the training data in each epoch and thus the number of iterations happening in each epoch. Learning rate determines the speed at which the model learns during the training process. An Optimizer is an algorithm that efficiently minimizes the difference between the prediction and the ground truth. Lastly, the R2 Regularization is added to ensure that the weights remain small, and it is one way to overcome the issue of overfitting.

Table 2: Parameters and Training Options.

Parameter	Value
Number of bits k	2
Channel uses n	2
Eb/No	5 dB
Max Epochs	15
Mini Batch Size	50
Initial Learning Rate	0.0155 (drops by 10 every 10 epochs)
Optimizer	Adam
R2 Regularization	0.001

3.4 Training and Validation

The system can learn through the training process. Training is done by feeding the system with the training data and iteratively updating the learnable parameters (i.e., weights and biases) until the prediction error is very small. In MATLAB, the training is achieved by calling the trainNetwork() function and providing the appropriate arguments. The function returns the trained network with all its characteristics (e.g., learnable parameters, no. of layers/neurons, etc....). The following Figure shows the shape of the network produced by MATLAB, which is essentially the same as the architecture depicted in Fig. 3.

Validation is done alongside training, which checks the quality of training by feeding the model with unseen data, that is the validation data.

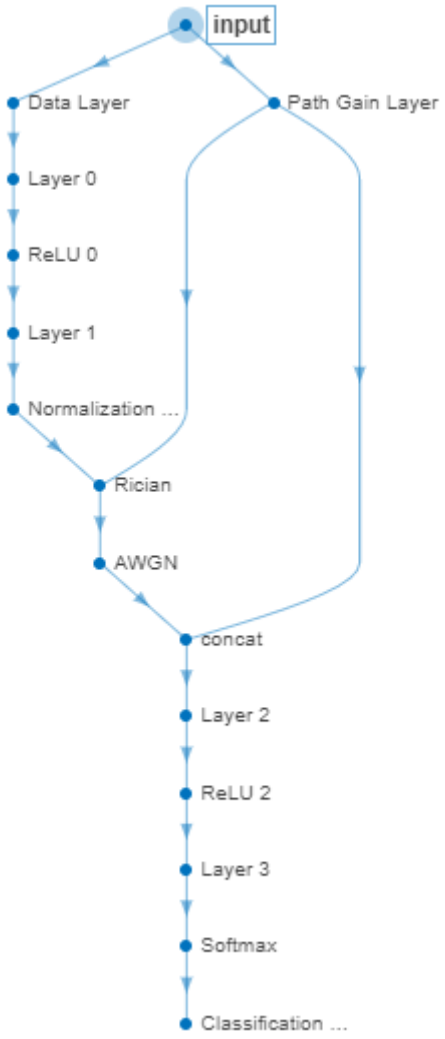


Figure 4: Resultant Network Structure.

To observe the effects of K-Factor on Training and Validation Accuracies. The training process was repeated for seven different K-Factor values, which are: 1, 3, 6, 9, 12, 15, and 150. As can be shown in the following Table, the Validation and Training accuracy for each case is presented. The table shows that as the K-Factor increases, the training quality is improved. Moreover, Figures 5-7 show the training results of the networks having K-Factors of 3,6, and 9.

Table 3: Accuracy for different K-Factor values.

K-Factor	Validation and Training Accuracy
1	88.19%
3	91.88%
6	94.03%
9	95.28%
12	96.18%
15	97.15%
150	99.17%

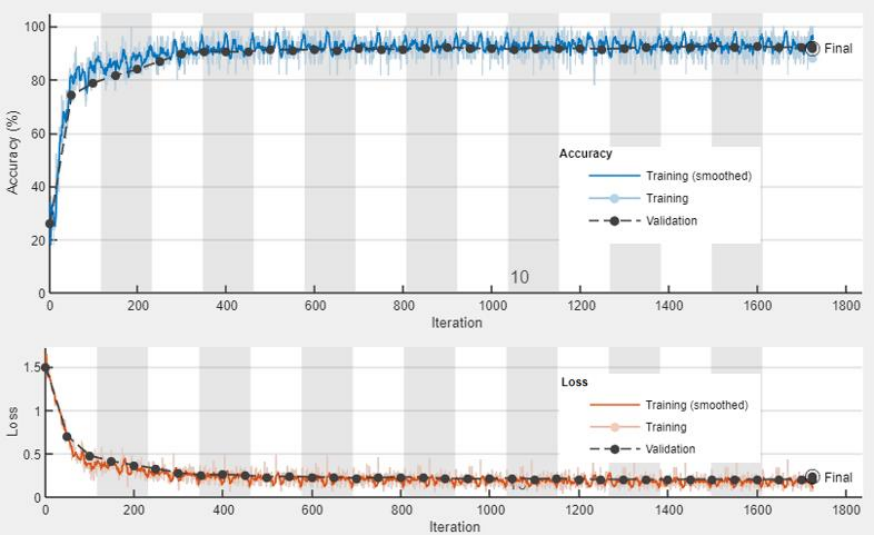


Figure 5: Training and Validating Accuracies for K-Factor = 3.

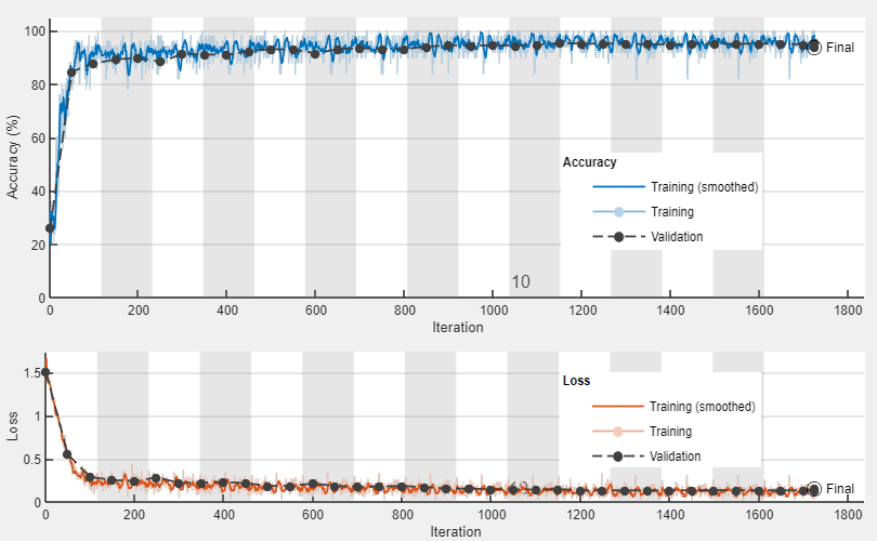


Figure 6: Training and Validation Accuracies for K-Factor = 6.

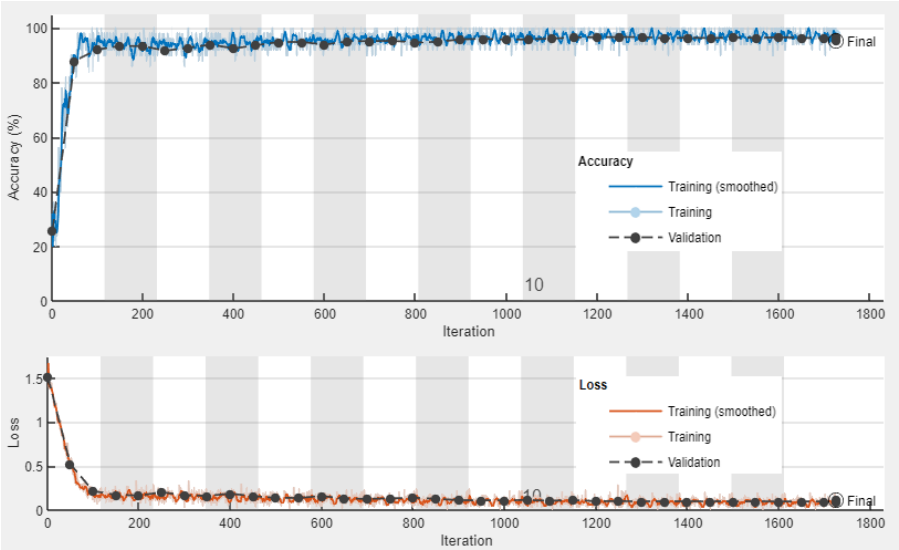


Figure 7: Training and Validation Accuracies for K-Factor = 9.

3.5 Testing

The Autoencoder with K-Factor = 3 was chosen for the subsequent steps which are the Testing, the creation of the RTL model, and the RTL verification. The Testing is done by feeding the testing set through the trained network. The process is done iteratively for varying noise values, starting from high noise values (worst-case scenario) and gradually decreasing the noise until the added noise is minimal. For the worst-case scenario, Testing Accuracy reached 76.125%. Evidently, as the noise decreased, the Testing Accuracy increased. In Table 4, a fragment of the Input Data and the Predicted Data for the worst case is presented.

Table 4: Input and Predicted Data.

Test Data (S + Path Gains)		**Predicted Data ($\hat{S}$)**
Symbol	Path Gains	
$2 + 0i$	$0.758829421861695 - 0.647415873585992i$	2
$3 + 0i$	$0.758829421861695 - 0.647415873585992i$	3
$0 + 0i$	$0.758829421861695 - 0.647415873585992i$	2
$0 + 0i$	$0.758829421861695 - 0.647415873585992i$	0
$0 + 0i$	$0.758829421861695 - 0.647415873585992i$	2
$1 + 0i$	$0.758829421861695 - 0.647415873585992i$	3
$0 + 0i$	$0.758829421861695 - 0.647415873585992i$	0

To quantify the performance of the considered Autoencoder, its performance is compared with conventional communication systems. As seen in the following Figure, the BLER of the considered Autoencoder is evaluated alongside QAM and PSK Models with MLD to benchmark the quality of the communication of the Autoencoder.

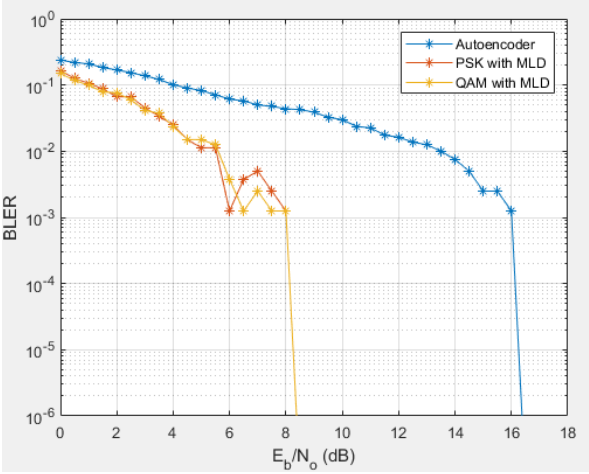


Figure 8: BLER of Autoencoder, QAM with MLD, and PSK with MLD.

As can be seen in the above Figure, the considered Autoencoder is performing worse than the conventional systems. The hyperparameters were fine-tuned to obtain a better BLER performance. However, the fine-tuning did not resolve this issue. Therefore, this result could be due to the way in which the BLER was conducted in MATLAB or other unknown reasons.

Chapter 4. RTL Model

This chapter sheds light on the creation of the communication system's RTL Model through System Verilog in ModelSim. It explains the algorithms applied to ensure a better hardware implementation. It also describes the various hardware modules inside the RTL model that represent the behavior observed in the High-Level model.

4.1 Algorithms for Better Hardware Implementation

4.1.1 Data Quantization

According to [41], Fixed-Point representation is preferred in hardware implementation due to its compatibility with hardware; as there's no support for Floating-Point arithmetic in today's processors. It also helps in expediating the execution and lowering the complexity of software [41]. In addition, using short Fixed-Point representation leads to less memory usage and less resource utilization [42]. Hence, for a better hardware implementation of the Autoencoder, the trained network's learnable parameters extracted from MATLAB were converted to Fixed-Point representation.

The initial attempt of conversion to Fixed-Point included converting the weights and biases of the Transmitter to 32-bits having Q3.28 as seen in Fig. 9; where 3 represents the number of bits for the integer part and 28 represents the number of bits for the fractional part, leaving the most significant bit to represent the sign.

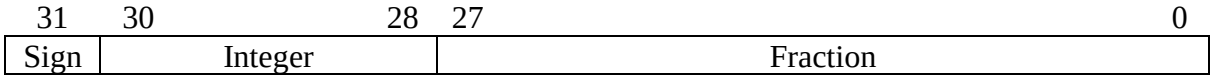


Figure 9: Fixed Point Rep for Weights and Biases of Transmitter (Initial Attempt).

Table 5: 32-bit Floating to Fixed Point conversion example.

Learnable Parameter	Extracted	Q3.28 FP Binary	Q3.28 FP Hexadecimal
Weight0 N0	0.27354577	00000100011000000111000110000111	4607187
Bias0_N0	-0.0081393160	11111111110111101010100101001111	FFDEA94F

Then hand calculations were done to obtain the outputs of the subsequent layers in both Floating-Point and Fixed-Point. This helped in determining the number of bits needed for the

remaining learnable parameters. However, due to the MAC operation occurring in each neuron, the number of bits scaled up leading to the last layer doing the MAC operation on 256-bit numbers and outputting 512-bit numbers. The bits scale up because the multiplication of two Fixed-Point numbers lead to a product that has integer/fraction bits equal to the addition of the integer/fraction bits of both Fixed-Point numbers. This scale up is undesirable as it is very hardware costly.

The second attempt started with reducing the number of bits for the Transmitter weights and biases to 16-bits with Q3.12. Similarly, the weights and other input data needed in the subsequent layers scaled up. Next, all weights were reduced to 16-bits and the outputs of the layers were truncated to 16-bits as well. Therefore, the entire structure had weights, inputs, and outputs that have a Fixed-Point representation of 16-bits with Q3.12. Some biases needed to be 32-bits with Q7.24 to ensure correct MAC operation.

Table 6: Floating to Fixed Point Representation using 16 bits.

Learnable Parameter	Extracted	Q3.12 FP Binary	Q3.12 FP Hexadecimal
Weight0 N0	0.27354577	0000010001100000	0.460
Bias0_N0	-0.0081393160	1111111111011111	FFDF

4.1.2 Pruning

Pruning is the process of setting small weights to zero, which leads to decreasing the number of connections between the layers [42]. According to [43], Pruning can help in reducing the complexity of the Neural Network, and the effects of overfitting. Therefore, the weights and biases were scanned for very small weights and were replaced by zero.

4.2 Transmitter RTL Model

All values of the network were somehow accessible and extractable from MATLAB except for the intermediate values in the Transmitter and Receiver. Since there was no way to extract intermediate outputs from the High-Level model, the first step in actualizing the Transmitter module was by hand-calculating the outputs of Layer 0 for a certain input and then computing the output of Layer 1. The calculation was done in both Fixed- and Floating-Point representations. The computed values were compared to the related extracted values to make sure that the calculations were done correctly. After that, the development of the Transmitter was done using a bottom-up approach starting from the Neuron Module up to the Transmitter

Module. Ultimately, to realize the Transmitter Module, six other modules were needed as seen in Fig. 10. Which are: the Input Neuron Module, Layer 0 Module, Layer 1 Neuron Module, Layer1 Module, ReLU Module, and Normalization Module. Note that the small squares in the Figure represent the Neurons.

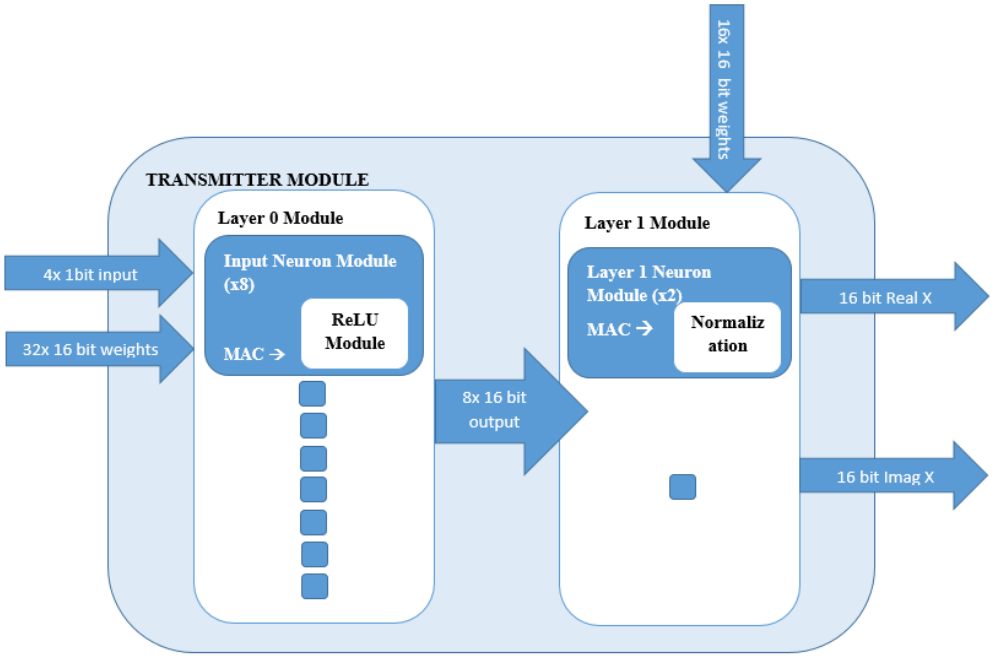


Figure 10: Transmitter RTL Model.

4.2.1 Input Neuron

The Input Neuron Module represents the neuron found in Layer 0. It essentially does the MAC operation on its inputs. The inputs being 1-bit input, 16-bit weight, and 16-bit bias. The signed multiplication of the input and weight and the addition of the bias leads to a 16-bit output. The output is sent to the ReLU function by instantiating it inside the module and specifying its inputs and outputs.

4.2.2 Layer 0

Layer 0 Module represents the very first Layer in the network which consists of eight instantiations of the Input Neuron Module. Therefore, this layer has eight outputs that are fed as inputs to the succeeding layer.

4.2.3 Layer 1 Neuron

Layer 1 Neuron has the same functionality as Input Neuron Module except that it initializes the Normalization Module instead of the ReLU Module.

4.2.4 Layer 1

This layer contains two instantiations of Layer 1 Neuron Module; one would output the real part of the transmitted signal X, while the other would output the complex part of the transmitted signal X.

4.2.5 ReLU

This module depicts the behavior of ReLU Activation Function which maps any given input to an appropriate value. If the input is positive, the output would be equal to the input. However, if the input was negative, the output would be zero.

4.2.6 Normalization

The normalization in the High-Level model does the Energy Normalization on the output of Layer 1. Energy Normalization requires division which is not a synthesizable operation in System Verilog. Therefore, the realization of the normalization step was done through an LUT. In which the input value will be mapped to another value by finding it in the LUT.

4.3 Channel RTL Model

The realization of the Channel Model was done by creating two modules, which are the Rician Fading Module and the AWGN Module as shown in Fig. 11.

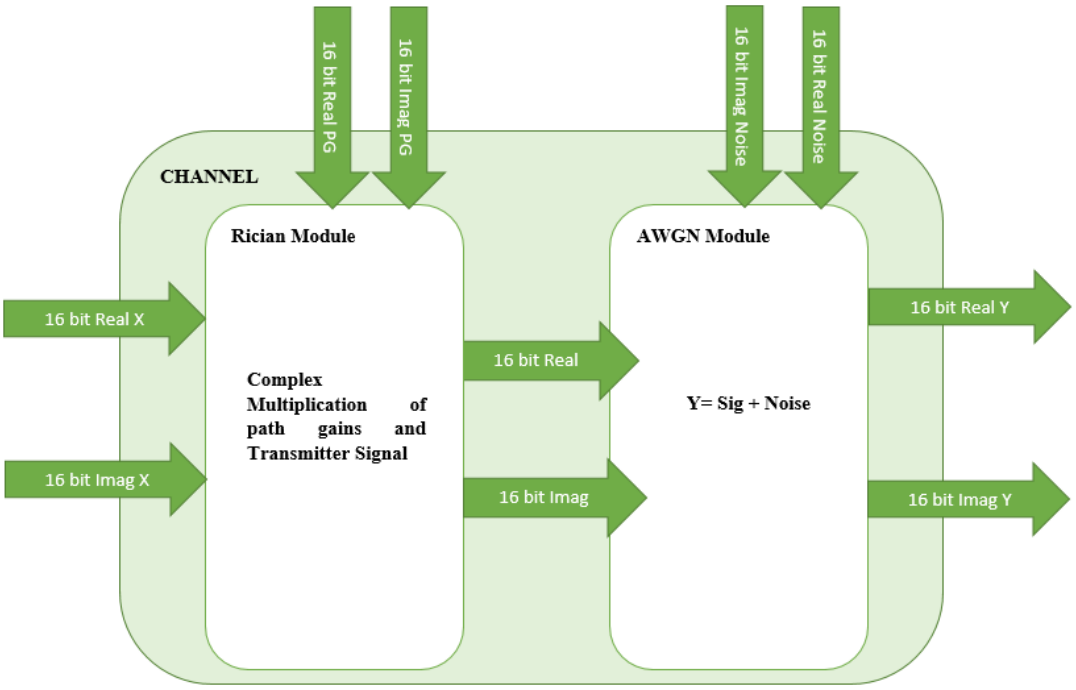


Figure 11: Channel RTL Model.

4.3.1 Rician Fading Channel

The Rician Fading Module accepts four inputs: the transmitted signal's real and complex parts, the corresponding path gains' real and complex parts. The complex multiplication is done inside the module which generates two 32-bit values (i.e., real and imaginary). These values are truncated to 16-bit and sent to the AWGN Module.

4.3.2 AWGN

This module is used to add noise to the two values to generate the received signal Y. To do this, AWGN accepts four values: the real and imaginary outputs coming from Rician and the real and imaginary parts of the noise. The resultant received signal is sent to the Receiver Module.

4.4 Receiver RTL Model

Similarly to the Transmitter Module, the development of the Receiver Module adopted the bottom-up approach. This included creating Layer 2 and 3 Neurons, Layers 2 and 3, and Softmax Modules. The Receiver Module is built by combining these Modules as seen in Fig. 12.

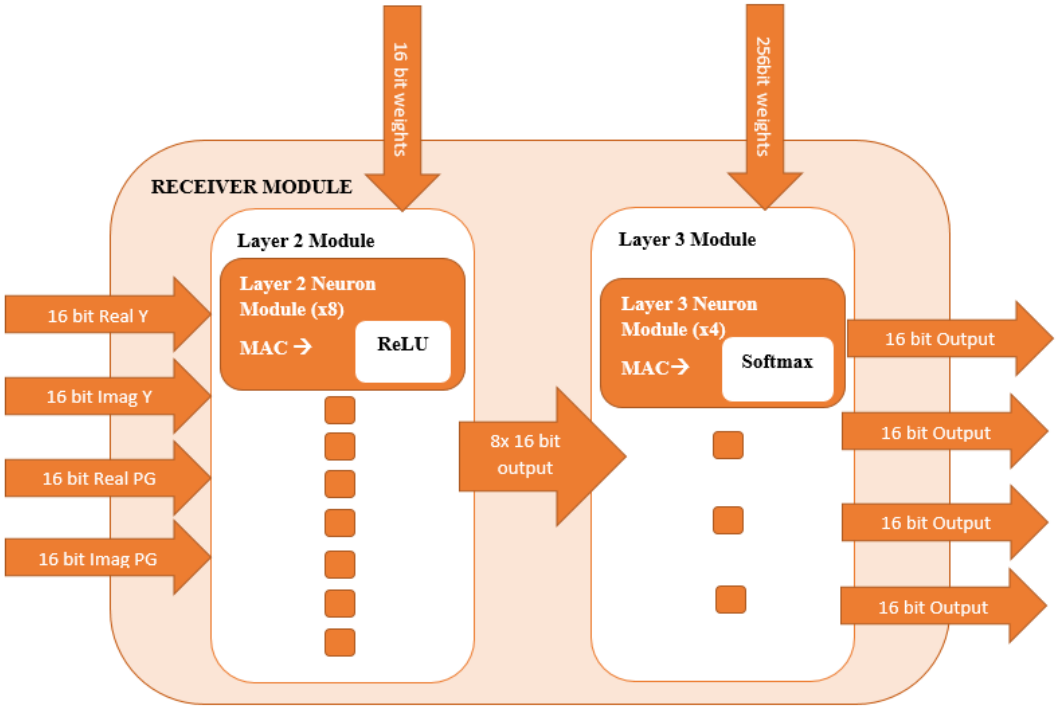


Figure 12 : Receiver RTL Model.

4.4.1 Layer 2 Neuron

The neuron in Layer 2 has similar functionality to the Neuron in Layer 0. However, it takes four 16-bit inputs instead of four 1-bit inputs. The MAC operation produces 32-bit value which is either kept as is or cleared to zero after sending it to the ReLU Module. Lastly, the value is truncated to 16-bit before it's sent to Layer 3.

4.4.2 Layer 2

This module creates Layer 2, which is the first layer in the Receiver, by instantiating eight Layer 2 Neurons. This leads to generating eight outputs for each of the four 16-bit inputs.

4.4.3 Layer 3 Neuron

Layer 3 Neuron Module executes the MAC operation, the resultant value is sent to the Softmax Activation Function by instantiating the Softmax Module inside Layer 3 Neuron.

4.4.4 Layer 3

Development of Layer 3 Module is done by instantiating four Layer 3 Neurons, which would generate four outputs for each of the four 16-bit inputs to the Receiver.

4.4.5 Softmax

Softmax Module represents the behavior of Softmax Activation Function which takes a vector of values and generates the probability for each value.

4.5 Classification Module

The classification Module takes the four outputs of Softmax for each input and does some arithmetic and logic operations on both the output with the highest probability and its index to get the Predicted Data.

4.6 Simulation

Numerous simulations were done to test the functionality of each module by creating appropriate test benches with suitable test cases. The following sub-sections show the simulations of each module. Note that all numbers are set in the decimal Fixed-Point format.

4.6.1 Transmitter Simulation

Once the test bench provides the one-hot encoded input values, the Transmitter uses the learnable parameters of layers 0 and 1 to generate the corresponding real and imaginary parts of the transmitted signal. Fig. 13 shows the waveform obtained when simulating the Transmitter Module. Whereas Table 7 shows the comparison between the MATLAB output and the RTL model output.

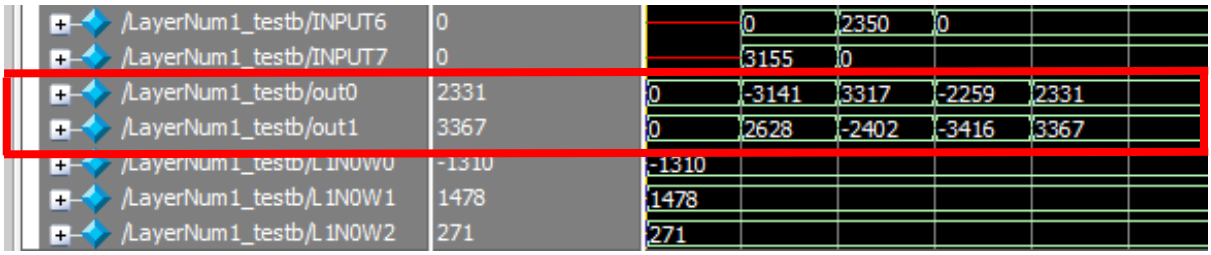


Figure 13: Outputs of Transmitter for each input.

Table 7: High Level vs RTL values.

MATLAB Transmitted Signal in Floating Point	**RTL Transmitted Signal in Fixed Point**	**RTL Transmitted Signal in Floating Point**
$-0.7669173 + 0.6417460i$	$-3141 + 2628i$	$-0.7666015625 + 0.6416015625i$
$0.8099581 - 0.5864878i$	$3317 - 2402i$	$0.8095703125 - 0.58642578125i$
$-0.5516129 - 0.8341003i$	$-2259 - 3416i$	$-0.55126953125 - 0.833984375i$
$0.5691618 + 0.8222256i$	$2331 + 3367i$	$0.56884765625 + 0.82177734375i$

As can be observed in Table 7, the accuracy of the numbers in the RTL are not exactly the same as the High-Level model and that is because of the decrease in the number of bits from 32-bit to 16-bit. Nevertheless, this loss is acceptable because it gives a huge decrease in size and in hardware usage.

4.6.2 Channel Simulations

The channel requires six values from the test bench to correctly simulate it: real and imaginary parts of transmitted signal, real and imaginary path gains, and the real and imaginary noise values. As such the transmitted signal is transformed through the two modules to generate the received signal. Fig. 14 and Fig.15 show the results of simulating the Rician Fading Module and the AWGN Module. Tables 8 and 9 show the comparison of the RTL outputs and the High-Level model outputs.

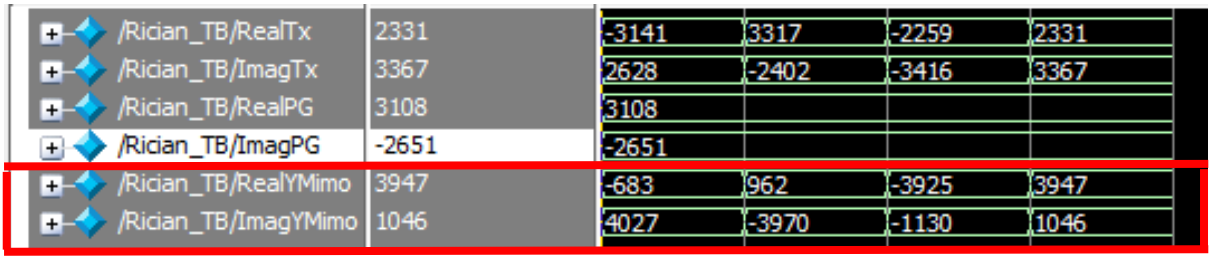


Figure 14: Rician Module IOs.

Table 8: Rician Outputs from MATLAB and ModelSim.

MATLAB Rician Output in Floating Point	RTL Rician Output in Fixed Point	RTL Rician Output in Floating Point
−0.1664829 + 0.9834902i	−683 + 4027i	−0.16650390625 + 0.98291015625i
0.2349185 − 0.9694239i	962 − 3970i	0.23486328125 – 0.96923828125i
−0.9585899 − 0.2758169i	−3925 − 1130i	−0.9580078125 – 0.27587890625i
0.9642186 + 0.2554446i	3947 + 1046i	0.96337890625 + 0.25537109375i

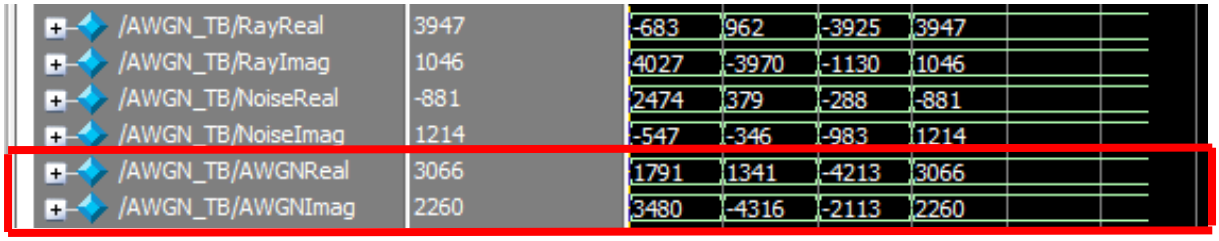


Figure 15: AWGN Module IOs.

Table 9: MATLAB AWGN vs RTL AWGN.

MATLAB AWGN Output in Floating Point	RTL AWGN Output in Fixed Point	RTL AWGN Output in Floating Point
0.4376361 + 0.8498884i	1791 + 3480i	0.43701171875 + 0.849609375i
0.3275313 − 1.054096i	1341 – 4316i	0.327148434375 – 1.0537109375i
−1.028958 − 0.5159034i	−4213 – 2113i	−1.0283203125– 0.515625i
0.7488944 + 0.5520490i	3066 + 2260i	0.74853515625 + 0.5517578125i

Similarly to the Transmitter, the RTL values of both Rician Module and AWGN Module are not exactly the same as the High Level due to the decreased number of bits that represent the values. However, as discussed earlier, this reduction is essential for optimizing the hardware implementation of the communication system.

4.6.3 RTL Model Simulation

To test the functionality of the entire developed system, a Top-Level Module was created, as seen in Fig. 16, and it was embedded with the Transmitter Module, Channel Module, Receiver Module, and Classification Module. A corresponding test bench was created to correctly simulate the Top-Level Module by providing: the 4-bit one-hot encoded input, the 16-bit path gains, 16-bit noise, all required files for printing the output, etc. Fig. 17 shows the waveform resulting from simulating the Top-Level Module. The Figure highlights the one-hot encoded input and the equivalent predicted output.

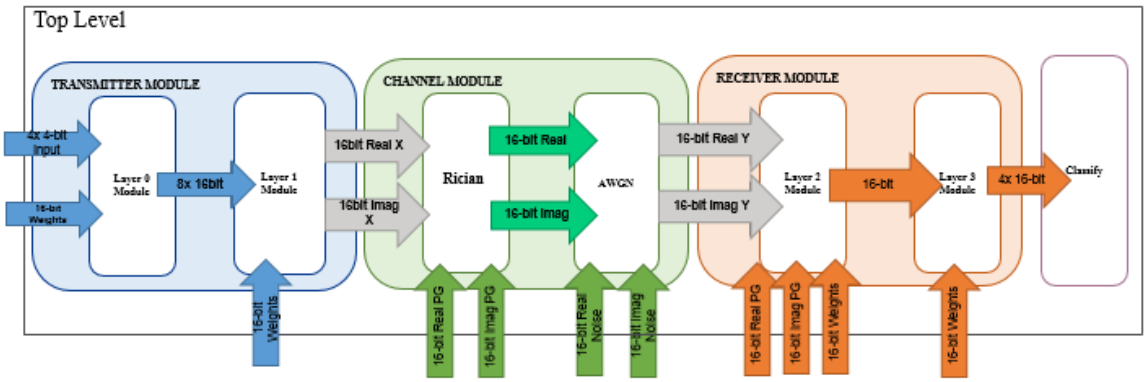


Figure 16: Top Level Module.

Figure 17: Simulation of Top Level.

The following Table shows some of the one-hot encoded inputs, their equivalent value in decimal, and the resultant predicted outputs. It can be observed that the Decoder (i.e., Receiver) was able to reconstruct most of the inputs fed to the Encoder (i.e., Transmitter) even though the transmitted signal X was distorted by the noise added to it in the channel. Similarly to the High-Level model, the RTL model achieved a Testing Accuracy of 76.125% with the added noise. Therefore, the developed RTL model works as expected.

Table 10: Top Level Input and Predicted Data.

One-Hot Encoded Input	Expected Value	Predicted Data ($\hat{S}$)
0010	2	2
0001	3	3
1000	0	2
1000	0	0
1000	0	2
0100	1	3
1000	0	0

The System Verilog RTL of all modules can be found in Appendix B.

Chapter 5. RTL Verification

This chapter verifies the RTL Model by feeding it to two different tools that execute Synthesis on the RTL model. It discusses the hardware implementation flow, the tools in which Synthesis is performed, and the result for each flow (i.e., the performance metrics).

5.1 Synthesis

Synthesis is the process of mapping the RTL model into the equivalent gate netlist. The chosen tools, Synopsys Design Compiler and Xilinx Vivado, perform Synthesis for the ASIC and FPGA implementation flow, respectively. The tools also provide the performance metrics after Synthesis is done executing. Synthesizing the 32-bit scaled-up RTL model showed that the model used a large number of resources. This is what led to the decrease in the number of bits to 16-bits and the use of truncation. Also, an internal memory mechanism was built to reduce the number of IOs.

5.2 ASIC Synthesis

ASIC Synthesis is done through the Design Compiler tool. An appropriate synthesis script was written to provide the tool with the necessary timing constraints (e.g., clock period, clock latency, etc....), the required files (i.e., System Verilog files), the cell library, and other auxiliary settings. To run the synthesis script, the commands were written into the DC Shell Terminal.

The tool provides essential performance metrics post-Synthesis, such as: timing report, power report, and an area report which are summarized in Table 11.

Table 11: ASIC Performance Metrics.

Metric	Value
Internal Power (mW)	0.3446
Switching Power (mW)	0.2591
Leakage Power (uW)	1.0300
Total Power (mW)	0.6047
Combinational Area	224.640001
Buffer/Inverter Area	43.130880
Non-Combinational Area	63.897598
Total Cell Area	288.537599

5.3 FPGA Synthesis

FPGA Synthesis is done through the Xilinx Vivado tool. The synthesis process is applied on Artix-7 XC7A100T-1CSG324C FPGA. Vivado was provided with the necessary files (i.e., System Verilog files).

The tool provided the following performance metrics post-synthesis: Slice Logic Utilization, DSP Utilization, and IO and Specific Utilization. These are listed in Table 12. The tool also provides the Primitives Utilization as seen in Table 13.

Table 12: Slice Logic, DSP, IO, and Specific Utilization.

Site Type	Used	Fixed	Available	Utilization %
Slice LUTs	439	0	63400	0.69
LUT as Logic	439	0	63400	0.69
LUT as Memory	0	0	19000	0.00
Slice Registers	64	0	126800	0.05
Register as Flip Flop	0	0	126800	0.00
Register as Latch	64	0	126800	0.05
F7 Muxes	0	0	31700	0.00
F8 Muxes	0	0	15850	0.00
DSPs	48	0	240	20.00
DSP48E1	48	-	-	-
Bonded IOB	258	0	210	122.85
Bonded IPADS	0	0	2	0.00
PHY_CONTROL	0	0	6	0.00
PHASER_REF	0	0	6	0.00
OUT_FIFO	0	0	24	0.00
IN_FIFO	0	0	24	0.00
IDELAYCTRL	0	0	6	0.00
IBUFGDS	0	0	202	0.00
PHASER_OUT/PHASER_OUT_PHY	0	0	24	0.00
PHASER_IN/PHASER_IN_PHY	0	0	24	0.00
IDELAY2/IDELAY2_FINEDELAY	0	0	300	0.00
IBUFDS_GIE2	0	0	4	0.00
ILOGIC	0	0	210	0.00
OLOGIC	0	0	210	0,00

Table 13: Primitive Utilization.

Ref Name	Used	Functional Category
IBUF	256	IO
LUT6	201	LUT
LUT5	92	LUT
LUT3	87	LUT
LUT4	70	LUT
LDCE	64	Flop & Latch
CARRY4	62	Carry Logic
LUT2	56	LUT
DSP48E1	48	Block Arithmetic
OBUF	2	IO
LUT1	1	LUT

The post-Synthesis findings are summarized in Table 14. Even though the system decreased in size in the second attempt, the designed system still consumed more IOs than what's available on Artix-7 FPGA. Therefore, the Synthesis should be re-done on a bigger FPGA that has more resources than the currently selected FPGA.

Table 14: Utilization Summary.

Resource	Estimation	Available	Utilization %
Flip Flops	64	126800	0.05
LUT	439	63400	0.69
I/O	258	210	122.86
DSP48	28	240	20.00

From the FPGA and ASIC hardware implementation results, it can be observed that the implementation uses a low number of resources. This can be beneficial if used in future satellite communication systems as it is known that these networks have a limited number of resources. However, since there is no similar application in the literature, the hardware implementation of the Autoencoder-based communication system could not be compared and benchmarked against other applications. As such, it would be favorable to further study the hardware implementation, improve it if possible, and perform comparisons with the baseline; that is the application done in this thesis. Nevertheless, the hardware implementation can be advantageous if used in future networks as it utilizes a low number of resources and has low power consumption, which are two major deficiencies found in satellite communications.

Chapter 6. Conclusion and Future Work

In this thesis, an Autoencoder-based Communication System High-Level model was considered to satisfy CubeSat/Nanosatellite requirement. The system was trained and tested accordingly to extract the required learnable parameters to build its equivalent RTL model. All learnable parameters were converted to Fixed-Point representation for improved hardware implementation. The simulation of the RTL model revealed that the model worked exactly as the High-Level model. The Synthesis of the built model was done for both ASIC and FPGA design flow and the vital performance metrics were also extracted. Future work would include: 1) Increasing the number of Antennas and applying Spatial Modulation, 2) Expanding the range of numbers that the system can re-construct correctly and re-train; as the current one can send and receive four numbers only, 3) Fine-tuning the hyperparameters to ensure that the Autoencoder training quality does not drop due to the expansion, 4) Re-writing the Normalization, AWGN, and Softmax Modules by using a divider primitive instead of an LUT, 5) Completing the FPGA design flow by going through the Implementation and Generation of the corresponding bitstream, and 6) Testing the programmed FPGA.

Bibliography

[1] T. O'Shea and J. Hoydis, "An Introduction to Deep Learning for the Physical Layer," *IEEE Transactions on Cognitive Communications and Networking*, vol. 3, no. 4, pp. 563–575, 12 2017.

[2] J. Schmitz, C. von Lengerke, N. Airee, A. Behboodi and R. Mathar, "A Deep Learning Wireless Transceiver with Fully Learned Modulation and Synchronization," *2019 IEEE International Conference on Communications Workshops (ICC Workshops)*, pp. 1-6, 2019.

[3] M. E. Morocho-Cayamcela, J. N. Njoku, J. Park and W. Lim, "Learning to Communicate with Autoencoders: Rethinking Wireless Systems with Deep Learning," *2020 International Conference on Artificial Intelligence in Information and Communication (ICAIIC)*, pp. 308-311, 2020.

[4] S. Dorner, S. Cammerer, J. Hoydis, and S. T. Brink, "Deep learning based communication over the air," IEEE Journalof Selected Topics in Signal Processing, vol. 12, no. 1, pp. 132–143, 2018.

[5] M. Soltani *et al*., "Autoencoder-Based Optical Wireless Communications Systems," *2018 IEEE Globecom Workshops (GC Workshops)*, pp. 1-6, 2018.

[6] F. Fourati and M. Alouini, "Artificial intelligence for satellite communication: A review," in *Intelligent and Converged Networks*, vol. 2, no. 3, pp. 213-243, Sept. 2021.

[7] D. Homan and Q. Young, "The Challenges of Developing an Operational Nanosatellite," in *Small Satellite Conference*, 2008.

[8] "CubeSats and SmallSats", *NASA Jet Propulsion Laboratory (JPL)*, 2022. [Online]. Available: https://www.jpl.nasa.gov/topics/cubesats.

[9] S. Liu et al., "A Survey on CubeSat Missions and Their Antenna Designs," Electronics, vol. 11, no. 13, p. 2021, Jun. 2022.

[10] P. I. Theoharis, R. Raad, F. Tubbal, M. U. Ali Khan, and S. Liu, "Software-Defined Radios for CubeSat Applications: A Brief Review and Methodology," *IEEE Journal on Miniaturization for Air and Space Systems*, vol. 2, no. 1, pp. 10-16, March 2021.

[11] N. A. Salam Bauomy and E. A. Elbeh, "Design of SDR Simulation for Wireless Communication between Ground Station and CubeSat Implemented by LabVIEW," *2020 8th International Japan-Africa Conference on Electronics, Communications, and Computations (JAC-ECC)*, pp. 60-63, 2020.

[12] G. Guerrisi, F. Del Frate and G. Schiavon, "Convolutional Autoencoder Algorithm for On-Board Image Compression," *IGARSS 2022 - 2022 IEEE International Geoscience and Remote Sensing Symposium*, pp. 151-154, 2022.

[13] J. Jordan, "Introduction to Autoencoders," Jeremy Jordan, 2018. [Online]. Available: https://www.jeremyjordan.me/autoencoders/

[14] W. Zhao, Z. Jia, X. Wei, and H. Wang, "An FPGA Implementation of a Convolutional Auto-Encoder," *Applied Sciences*, vol. 8, no. 4, p. 504, Mar. 2018.

[15] J. Zhai, S. Zhang, J. Chen, and Q. He, "Autoencoder and Its Various Variants," *2018 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pp. 415-419, 2018.

[16] N. A. Mohamed and J. R. Cavallaro, "Real-time FPGA-Based Outlier Detection using Autoencoder and LSTM," *2021 55th Asilomar Conference on Signals, Systems, and Computers*, pp. 1195-1199, 2021.

[17] L. Valente, "Autoencoder for FPGA" GitHub, 2022. [Online]. Available: https://github.com/LorenzoValente3/Autoencoder-for-FPGA

[18] Z. Li *et al*., "FPGA Realization of Stacked Auto-encoder with Three Fully Connected Layers," *2021 IEEE International Conference on Advances in Electrical Engineering and Computer Applications (AEECA)*, pp. 997-1001, 2021.

[19] S L. Chaparro and A. Akan, *Signals and systems using MATLAB*, 3rd ed. Academic Press, 2019. [Online]. doi: 10.1016/C2017-0-00826-1

[20] R. Akeela and B. Dezfouli, "Software-defined Radios: Architecture, state-of-the-art, and challenges", *Computer Communications*, vol. 128, pp. 106-125, 2018.

[21] M. S. Safadi and D. L. Ndzi, "Digital Hardware Choices For Software Radio (SDR) Baseband Implementation," *2006 2nd International Conference on Information & Communication Technologies*, pp. 2623-2628, 2006.

[22] T. Kazaz, M. Kulin and M. Hadzialic, "Design and implementation of SDR based QPSK modulator on FPGA," *2013 36th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, pp. 513-518, 2013.

[23] S. Gandhare and B. Karthikeyan, "Survey on FPGA Architecture and Recent Applications," *2019 International Conference on Vision Towards Emerging Trends in Communication and Networking (ViTECoN)*, pp. 1-4, 2019.

[24] D. Dardari, E. Falletti, and M. Luise, *Satellite and Terrestrial Radio Positioning Techniques*. Academic Press, 2012. [Online]. doi: 10.1016/C2009-0-61856-0

[25] H. Nasser, A. Badawieh, and A. Assalem, "A Survey Of Software Radios: Reconfigurable Platforms, Development Tools And Future Directions," *Radioelectronics. Nanosystems. Information Technologies*, pp. 207-218, 2020.

[26] I. Tzinis, "Space Communications and Navigation (SCAN) Testbed", *NASA*, 2012. [Online]. Available: https://www.nasa.gov/directorates/heo/scan/engineering/technology/txt_scantestbed.html.

[27] I. Tzinis, "Software Defined Radios", *NASA*, 2012. [Online]. Available: https://www.nasa.gov/directorates/heo/scan/engineering/technology/txt_sdr.html.

[28] A. Di Stefano, G. Fiscelli and C. G. Giaconia, "An FPGA-Based Software Defined Radio Platform for the 2.4GHz ISM Band," *2006 Ph.D. Research in Microelectronics and Electronics*, pp. 73-76, 2006.

[29] A. Skärpe, "Implementation of an SDR in Verilog." M.S. thesis, Dept. of Electrical Engineering, Linköping University, Linköping, 2016.

[30] V. Allui, "Multiple Channel Coherent Amplitude Modulated (AM) Time Division Multiplexing (TDM) Software Defined Radio (SDR) Receiver," M.S. thesis, University of Kentucky, 2008.

[31] S. B. Junior, V. C. de Oliveira, and G. B. Junior, "Software defined radio implementation of a QPSK modulator/demodulator in an extensive hardware platform based on FPGAs Xilinx ZYNQ," *Journal of Computer Science*, vol. 11, no. 4, pp. 598–611, 2015.

[32] X. Cai, M. Zhou, T. Xia, W. H. Fong, W. Lee, and X. Huang, "Low-Power SDR Design on an FPGA for Intersatellite Communications," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 26, no. 11, pp. 2419-2430, Nov. 2018.

[33] M. Maheshwarappa, “Software defined radio (SDR) architecture for concurrent multi-satellite communications,” Ph.D dissertation, University of Surrey, 2017.

[34] X. Cai, M. Zhou, and X. Huang, "Model-Based Design for Software Defined Radio on an FPGA," *IEEE Access*, vol. 5, pp. 8276-8283, 2017.

[35] P. Dong, “Design and FPGA implementation of a SISO and a MIMO Wireless System for Software Defined Radio,” M.S. thesis, Dept. of Electrical and Computer Engineering, Concordia University, 2009.

[36] S. Joshi, "Integrating FPGA with Multicore SDR Development Platform to Design Wireless Communication System," M.S. thesis, Dept. of Electrical and Computer Engineering, California State University, Northridge, 2012.

[37] “Autoencoder for Wireless Communications”, *MathWorks*, [Online]. Available:https://www.mathworks.com/help/comm/ug/autoencoders-for-wireless-communications.html?searchHighlight=autoencoder&s_tid=srchtitle

[38] L. Lu et al., “Analysis of Channel Model for GEO Satellite Mobile Communication System,” *National Conference on Information Technology and Computer Science*, pp. 563-567, 2012.

[39] M. Delibasic and M. Pejanovic-Djurisic, "Impact of random K factor on Ricean fading wireless system performance," *16th IEEE Mediterranean Electrotechnical Conference, Yasmine Hammamet, Tunisia*, pp. 233-236, 2012.

[40] S. Zhu et al., “Probability Distribution of Rician K -Factor in Urban, Suburban and Rural Areas Using Real-World Captured Data,” *IEEE Transactions on Antennas and Propagation*, vol. 62, no. 7, pp. 3835-3839, July 2014.

[41] E. Oberstar, “Fixed-Point Representation & Fractional Math,” *RSGC ACES*, 2007.

[42] J. Qiu et al., “Going Deeper with Embedded FPGA Platform for Conventional Neural Network,” *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2016.

[43] S. Han, H. Mao, and W. Dally, “Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding,” *International Conference on Learning Representations*, 2016.

Appendices

Appendix A

Communication System with Rayleigh Fading Channel

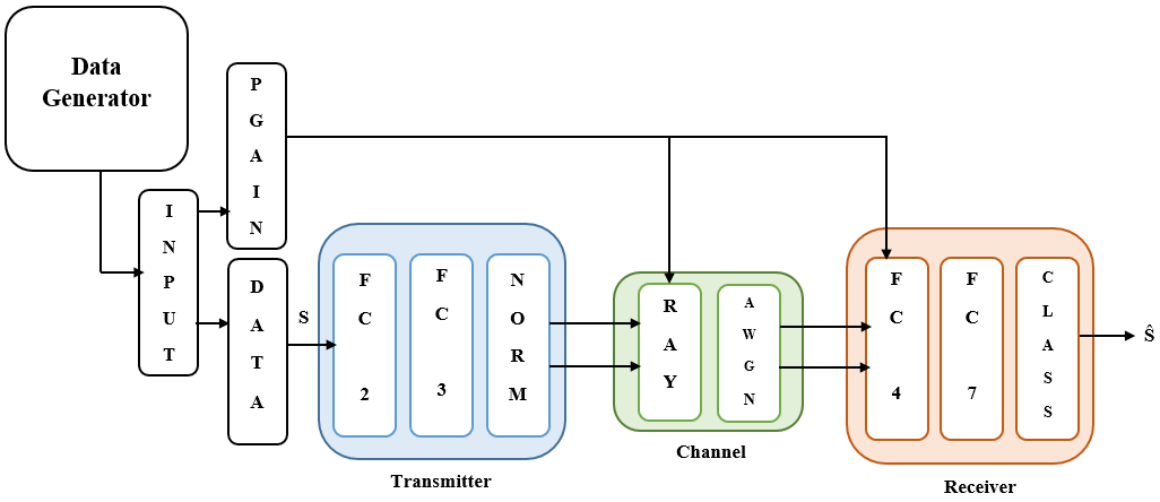


Figure 18: High Level Communication System with Rayleigh Fading Channel.

Table 15: Communication System Neurons and Layers.

Layer	Number of Neurons
1st Fully Connected Layer (Layer 0)	8
2nd Fully Connected Layer (Layer 1)	2
Rayleigh	2
AWGN	2
3rd Fully Connected Layer (Layer 2)	8
4th Fully Connected Layer (Layer 3)	4

Table 16: Hyperparamaters of the system.

Parameter	Value
Number of bits k	2
Channel uses n	2
Eb/No	5 dB
Max Epochs	30
Mini Batch Size	50
Initial Learning Rate	0.008 (drops by 10 every 10 epochs)
Optimizer	Adam
R2 Regularization	0.001

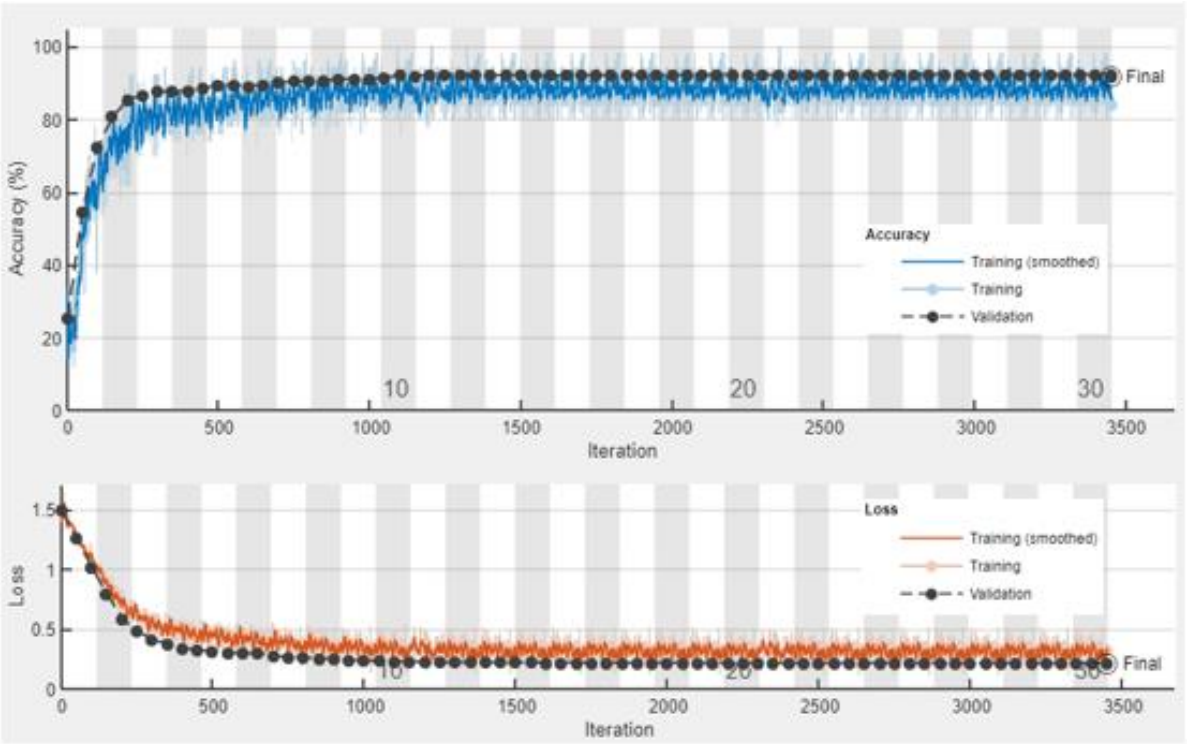


Figure 19: Training Results.

Table 17:High Level Testing.

Input	Predicted
0	0
1	1
0	3
0	0
1	1
3	2
2	3
0	0

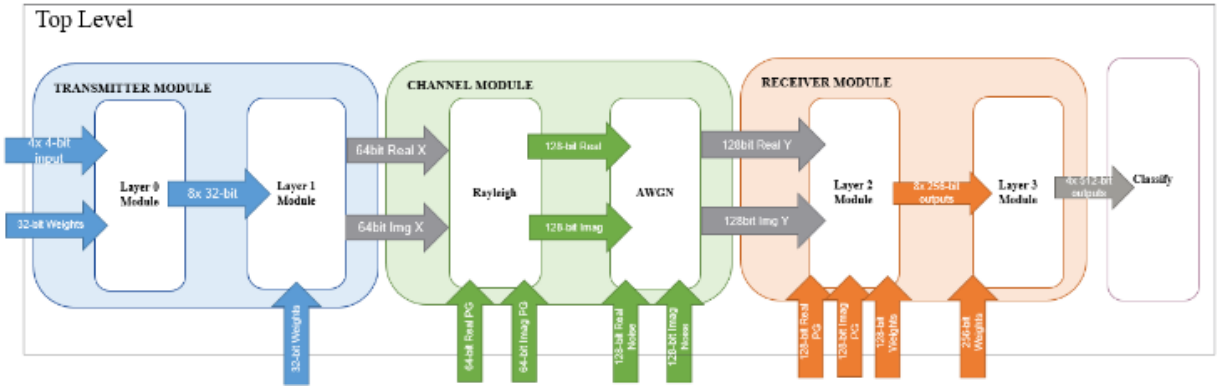


Figure 20: Top Level of RTL Model

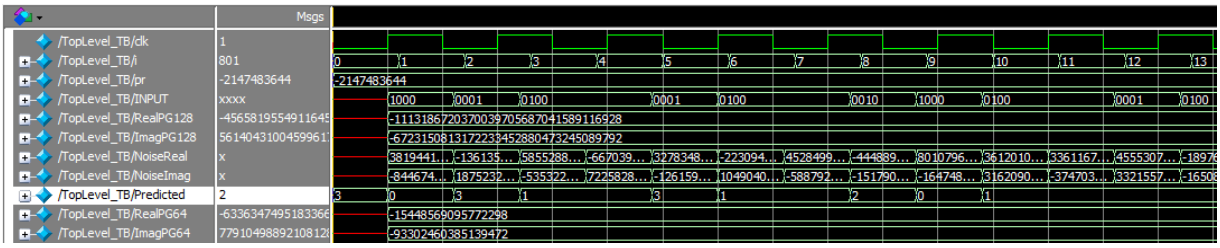


Figure 21: Simulation Results of Top Level.

Table 18:RTL Input and Predicted Data.

One-Hot Encoded Input	Expected Value	Predicted Data (Ŝ)
0100	1	1
0100	1	1
0010	2	2
1000	0	0
0100	1	1
0100	1	1
0001	3	1
0100	1	1

Table 19: ASIC Synthesis results.

Metric	Value
Internal Power (mW)	778.9294
Switching Power (mW)	1.6045e+03
Leakage Power (uW)	5.9029e+033
Total Power (mW)	2.3911e+03
Combinational Area	2908177.932765
Buffer/Inverter Area	64499.635376
Non-Combinational Area	2044.723145
Total Cell Area	2910222.655913

Table 20: FPGA Primitive Utilization.

Ref Name	Used	Functional Category
LUT6	1437529	LUT
LUT2	1368136	LUT
LUT4	763215	LUT
LUT3	689767	LUT
CARRY4	584508	Carry Logic
LUT5	153441	LUT
LUT1	151539	LUT
IBUF	22916	IO
LDCE	2048	Flop & Latch
DSP48E1	240	Block Arithmetic
BUFG	4	Clock
OBUF	2	IO

Table 21: FPGA Slice Logic, DSP, IO, Specific, and Clocking Utilization.

Site Type	Used	Fixed	Available	Utilization %
Slice LUTs	3388674	0	63400	5344.91
LUT as Logic	3388674	0	63400	5344.91
LUT as Memory	0	0	19000	0.00
Slice Registers	2048	0	126800	1.61
Register as Flip Flop	0	0	126800	0.00
Register as Latch	2048	0	126800	1.61
F7 Muxes	0	0	31700	0.00
F8 Muxes	0	0	15850	0.00
DSPs	240	0	240	100.00
DSP48E1	240	-	-	-
Bonded IOB	22918	0	210	10913.33
Bonded IPADS	0	0	2	0.00
PHY_CONTROL	0	0	6	0.00
PHASER_REF	0	0	6	0.00
OUT_FIFO	0	0	24	0.00
IN_FIFO	0	0	24	0.00
IDELAYCTRL	0	0	6	0.00
IBUFGDS	0	0	202	0.00
PHASER_OUT/PHASER_OUT_PHY	0	0	24	0.00
PHASER_IN/PHASER_IN_PHY	0	0	24	0.00
IDELAY2/IDELAY2_FINEDELAY	0	0	300	0.00
IBUFDS_GIE2	0	0	4	0.00
ILOGIC	0	0	210	0.00
OLOGIC	0	0	210	0.00
BUFGCTRL	4	0	32	12.50

Table 22: FPGA Synthesis Summary.

Resource	Estimation	Available	Utilization %
Flip Flops	2048	126800	1.62
LUT	338867	63400	5344.91
I/O	22918	210	10913.33
DSP48	240	240	100.00
BUFG	4	32	12.50

Appendix B

```
/*
  Input --> [Real(H * x) Imag(H * x)]
  Output --> [Real(y) Imag(y)]
  where y = H*x + n (n is white gaussian noise)
*/
module AWGN
  (
  input logic signed [15:0] RayReal, RayImag, NoiseReal, NoiseImag,
  output logic signed [15:0] AWGNReal, AWGNImag
  );

  always_comb
  begin
    AWGNReal <= RayReal + NoiseReal;
    AWGNImag <= RayImag + NoiseImag;
  end
endmodule
```

```
/*
        Takes 4 inputs (probabilities) and assigns an index for each value.
        The maximum value is determined and its index is set as max index.
        dHat is set equal to max index.
        Predicted value is found by subtracting one from dHat
*/
module Classify
  (
  input logic signed [15:0] In0, In1, In2, In3,
  output logic [1:0] Predicted
  );

  logic signed [15:0] max1, max2, max;
  logic [2:0] dHat, maxIndex, max1Index, max2Index;
  logic [2:0] IndexIn0, IndexIn1, IndexIn2, IndexIn3;
```

```
assign IndexIn0= 3'd1;
assign IndexIn1= 3'd2;
assign IndexIn2= 3'd3;
assign IndexIn3= 3'd4;

always_comb
begin
  if (In0 >In1)
    begin
    max1= In0;
    max1Index=IndexIn0;
  end
  else
    begin
    max1=In1;
    max1Index=IndexIn1;
   end
  if(In2 > In3)
    begin
    max2=In2;
    max2Index=IndexIn2;
  end
  else
    begin
    max2=In3;
    max2Index=IndexIn3;
end
  if(max1 > max2)
    begin
    max= max1;
    maxIndex=max1Index;
  end
  else
    begin
```

```
      max = max2;
      maxIndex=max2Index;
    end

    dHat= maxIndex;

  end


  always_comb
  begin
    Predicted = dHat- 1'd1;
  end

endmodule
```

```
/*
        Represents the first layer of the network. Learnable parameters are first
        read and assigned to the corresponding variables.
        The module has 8 instantiations of input neuron module which correspond
        to the 8 neurons in the first layer.
*/
module Layer0 #(parameter NN=8, numWeight=4, dataWidth=1, weightWidth=16)
(
input logic clk,
input logic  signed in0,in1,in2,in3,
output logic signed [dataWidth*weightWidth-1:0] out0,out1,out2,out3, out4,out5,out6,out7
);
//Layer 0 has 8 Neurons 0-->7
//has 4 inputs
logic [15:0] WeightMem0 [0:31];
logic [31:0] Bias0 [0:7];
logic  signed [weightWidth-1:0] N0W0, N0W1, N0W2, N0W3, N0Bias;
logic  signed [weightWidth-1:0] N1W0, N1W1, N1W2, N1W3, N1Bias;
logic  signed [weightWidth-1:0] N2W0, N2W1, N2W2, N2W3, N2Bias;
```

```
logic  signed [weightWidth-1:0] N3W0, N3W1, N3W2, N3W3, N3Bias;
logic  signed [weightWidth-1:0] N4W0, N4W1, N4W2, N4W3, N4Bias;
logic  signed [weightWidth-1:0] N5W0, N5W1, N5W2, N5W3, N5Bias;
logic  signed [weightWidth-1:0] N6W0, N6W1, N6W2, N6W3, N6Bias;
logic  signed [weightWidth-1:0] N7W0, N7W1, N7W2, N7W3, N7Bias;
//Read weight and bias values for layer0
initial
begin
  $readmemb("Layer0WeightsBinary", WeightMem0);
  $readmemb("Layer0BiasBinary", Bias0);

  //W+B for Layer 0 Neuron 0
  N0W0 = WeightMem0[0];
  N0W1 = WeightMem0[1];
  N0W2 = WeightMem0[2];
  N0W3 = WeightMem0[3];
  N0Bias = Bias0[0];
  //W+B for Layer 0 Neuron 1
  N1W0 = WeightMem0[4];
  N1W1 = WeightMem0[5];
  N1W2 = WeightMem0[6];
  N1W3 = WeightMem0[7];
  N1Bias = Bias0[1];
  //W+B for Layer 0 Neuron 2
  N2W0 = WeightMem0[8];
  N2W1 = WeightMem0[9];
  N2W2 = WeightMem0[10];
  N2W3 = WeightMem0[11];
  N2Bias = Bias0[2];
  //W+B for Layer 0 Neuron 3
  N3W0 = WeightMem0[12];
  N3W1 = WeightMem0[13];
  N3W2 = WeightMem0[14];
  N3W3 = WeightMem0[15];
```

```
    N3Bias = Bias0[3];
    //W+B for Layer 0 Neuron 4
    N4W0 = WeightMem0[16];
    N4W1 = WeightMem0[17];
    N4W2 = WeightMem0[18];
    N4W3 = WeightMem0[19];
    N4Bias = Bias0[4];
    //W+B for Layer 0 Neuron 5
    N5W0 = WeightMem0[20];
    N5W1 = WeightMem0[21];
    N5W2 = WeightMem0[22];
    N5W3 = WeightMem0[23];
    N5Bias = Bias0[5];
    //W+B for Layer 0 Neuron 6
    N6W0 = WeightMem0[24];
    N6W1 = WeightMem0[25];
    N6W2 = WeightMem0[26];
    N6W3 = WeightMem0[27];
    N6Bias = Bias0[6];
    //W+B for Layer 0 Neuron 7
    N7W0 = WeightMem0[28];
    N7W1 = WeightMem0[29];
    N7W2 = WeightMem0[30];
    N7W3 = WeightMem0[31];
    N7Bias = Bias0[7];
end

//Neuron 0
input_neuron #(.numWeight(4),.dataWidth(1),
.weightWidth(weightWidth)) Lay0Neu0
(
.clk(clk),
.in0(in0),.in1(in1),.in2(in2),.in3(in3),
.W0(N0W0),.W1(N0W1),.W2(N0W2),.W3(N0W3),.Bias(N0Bias),
```

```
.out(out0)
);

//Neuron 1
input_neuron #(.numWeight(4), .dataWidth(1),
.weightWidth(weightWidth)) Lay0Neu1
(
.clk(clk),
.in0(in0),.in1(in1),.in2(in2),.in3(in3),
.W0(N1W0),.W1(N1W1),.W2(N1W2),.W3(N1W3),.Bias(N1Bias),
.out(out1)
);

//Neuron 2
input_neuron #(.numWeight(4), .dataWidth(1),
.weightWidth(weightWidth)) Lay0Neu2
(
.clk(clk),
.in0(in0),.in1(in1),.in2(in2),.in3(in3),
.W0(N2W0),.W1(N2W1),.W2(N2W2),.W3(N2W3),.Bias(N2Bias),
.out(out2)
);

//Neuron 3
input_neuron #(.numWeight(4), .dataWidth(1),
.weightWidth(weightWidth)) Lay0Neu3
(
.clk(clk),
.in0(in0),.in1(in1),.in2(in2),.in3(in3),
.W0(N3W0),.W1(N3W1),.W2(N3W2),.W3(N3W3),.Bias(N3Bias),
.out(out3)
);

//Neuron 4
```

```
input_neuron #(.numWeight(4), .dataWidth(1),
.weightWidth(weightWidth)) Lay0Neu4
(
.clk(clk),
.in0(in0),.in1(in1),.in2(in2),.in3(in3),
.W0(N4W0),.W1(N4W1),.W2(N4W2),.W3(N4W3),.Bias(N4Bias),
.out(out4)
);

//Neuron 5
input_neuron #(.numWeight(4), .dataWidth(1),
.weightWidth(weightWidth)) Lay0Neu5
(
.clk(clk),
.in0(in0),.in1(in1),.in2(in2),.in3(in3),
.W0(N5W0),.W1(N5W1),.W2(N5W2),.W3(N5W3),.Bias(N5Bias),
.out(out5)
);

//Neuron 6
input_neuron #(.numWeight(4), .dataWidth(1),
.weightWidth(weightWidth)) Lay0Neu6
(
.clk(clk),
.in0(in0),.in1(in1),.in2(in2),.in3(in3),
.W0(N6W0),.W1(N6W1),.W2(N6W2),.W3(N6W3),.Bias(N6Bias),
.out(out6)
);

//Neuron 7
input_neuron #(.numWeight(4), .dataWidth(1),
.weightWidth(weightWidth)) Lay0Neu7
(
.clk(clk),
```

```
.in0(in0),.in1(in1),.in2(in2),.in3(in3),
.W0(N7W0),.W1(N7W1),.W2(N7W2),.W3(N7W3),.Bias(N7Bias),
.out(out7)
);
endmodule
```

```
/*
        Represents the second layer of the network. Learnable parameters are first
        read and assigned to the corresponding variables.
        The module has 2 instantiations of layer 1 neuron module which correspond
        to the 2 neurons in the second layer.
*/
module Layer1 #(parameter NN=2, numWeight=4, dataWidth=16, weightWidth=16)
(
input logic clk,
input logic  signed[dataWidth-1:0]in0,in1,in2,in3,in4, in5, in6, in7,
output logic signed [weightWidth-1:0] out0,out1//,out2,out3
);
//Layer 1 has 2 Neurons and 8 inputs
logic  signed [weightWidth-1:0] N0W0,N0W1,N0W2,N0W3,N0W4,N0W5,N0W6,N0W7;
logic  signed [weightWidth-1:0] N1W0,N1W1,N1W2,N1W3,N1W4,N1W5,N1W6,N1W7;
logic signed [2*weightWidth-1:0] N0Bias, N1Bias;
logic [15:0] WeightMem1 [0:15];
logic [31:0] Bias1 [0:1];

initial
begin
  $readmemb("Layer1WeightsBinary", WeightMem1);
  $readmemb("Layer1BiasBinary", Bias1);
   //LAYER 1//
   //W+B for Layer 1 Neuron 0
   N0W0 = WeightMem1[0];
   N0W1 = WeightMem1[1];
   N0W2 = WeightMem1[2];
   N0W3 = WeightMem1[3];
```

```
  N0W4 = WeightMem1[4];
  N0W5 = WeightMem1[5];
  N0W6 = WeightMem1[6];
  N0W7 = WeightMem1[7];
  N0Bias = Bias1[0];
  //W+B for Layer 1 Neuron 1
  N1W0 = WeightMem1[8];
  N1W1 = WeightMem1[9];
  N1W2 = WeightMem1[10];
  N1W3 = WeightMem1[11];
  N1W4 = WeightMem1[12];
  N1W5 = WeightMem1[13];
  N1W6 = WeightMem1[14];
  N1W7 = WeightMem1[15];
  N1Bias = Bias1[1];
end


//Neuron 0
Layer1_neuron #(.numWeight(8),.dataWidth(16),
.weightWidth(16)) Lay1Neu0
(
.clk(clk),
.in0(in0),.in1(in1),.in2(in2),.in3(in3),.in4(in4),.in5(in5),.in6(in6),.in7(in7),
.W0(N0W0),.W1(N0W1),.W2(N0W2),.W3(N0W3), .W4(N0W4), .W5(N0W5), .W6(N0W6), .W7(N0W7),
.Bias(N0Bias),.out(out0)
);

//Neuron 1
Layer1_neuron #(.numWeight(8),.dataWidth(16),
.weightWidth(16)) Lay1Neu1
(
.clk(clk),
.in0(in0),.in1(in1),.in2(in2),.in3(in3),.in4(in4),.in5(in5),.in6(in6),.in7(in7),
```

```
.W0(N1W0),.W1(N1W1),.W2(N1W2),.W3(N1W3), .W4(N1W4), .W5(N1W5), .W6(N1W6), .W7(N1W7),
.Bias(N1Bias),.out(out1)
);
endmodule
```

```
/*
        Represents the neuron in the second layer. It performs the MAC operation
        followed by energy normalization through the instantiation of the normalization
        module.
*/
module Layer1_neuron #(parameter numWeight=8, dataWidth=16,weightWidth=16)
(
input logic  clk, //read_en,
input logic  signed [dataWidth-1:0]in0,in1,in2,in3, in4, in5, in6, in7,
input logic  signed [weightWidth-1:0] W0,W1,W2,W3,W4,W5,W6,W7,//Bias,
input logic signed [2*weightWidth-1:0] Bias,
output logic [weightWidth-1:0] out
);

logic signed [2*weightWidth-1:0] mult0,mult1,mult2,mult3, mult4,mult5,mult6,mult7;
logic signed [2*weightWidth-1:0] sum;

always_comb begin
  mult0= in0*W0;
  mult1= in1*W1;
  mult2= in2*W2;
  mult3= in3*W3;

  mult4= in4*W4;
  mult5= in5*W5;
  mult6= in6*W6;
  mult7= in7*W7;
  sum = mult0 + mult1 + mult2 + mult3+ mult4+ mult5+ mult6+ mult7+ Bias;
end
```

```
//Normalize
Normalization #(.dataWidth(dataWidth),.weightWidth(weightWidth))Norm (
.sum(sum),
.sum_norm(out)
);
endmodule
```

```
/*
        Represents the third layer of the network. Learnable parameters are first
        read and assigned to the corresponding variables.
        The module has 8 instantiations of layer 2 neuron module which correspond
        to the 8 neurons in the third layer.
*/
module Layer2
 (
 input logic clk,
 input logic signed [15:0] AWGNReal, AWGNImag, //RealPG, ImagPG,
 input logic signed [15:0] RealPG, ImagPG,
 output logic signed [15:0] out0, out1, out2, out3, out4, out5, out6, out7
 );

 //Layer 2 has 8 neurons: neuron 0 to neuron 7
 logic  signed [15:0] N0W0, N0W1, N0W2, N0W3;
 logic  signed [15:0] N1W0, N1W1, N1W2, N1W3;
 logic  signed [15:0] N2W0, N2W1, N2W2, N2W3;
 logic  signed [15:0] N3W0, N3W1, N3W2, N3W3;
 logic  signed [15:0] N4W0, N4W1, N4W2, N4W3;
 logic  signed [15:0] N5W0, N5W1, N5W2, N5W3;
 logic  signed [15:0] N6W0, N6W1, N6W2, N6W3;
 logic  signed [15:0] N7W0, N7W1, N7W2, N7W3;
 logic signed [31:0] N0Bias, N1Bias,N2Bias,N3Bias,N4Bias,N5Bias,N6Bias,N7Bias;
 logic [15:0] WeightMem2 [0:31];
 logic [31:0] Bias2 [0:7];

 initial
```

```
begin
        $readmemb("Layer2WeightBinary", WeightMem2);
 $readmemb("Layer2BiasBinary", Bias2);
                    //LAYER 2//
 //W+B for Layer 2 Neuron 0
 N0W0 = WeightMem2[0];
 N0W1 = WeightMem2[1];
 N0W2 = WeightMem2[2];
 N0W3 = WeightMem2[3];
 N0Bias = Bias2[0];
 //W+B for Layer 2 Neuron 1
 N1W0 = WeightMem2[4];
 N1W1 = WeightMem2[5];
 N1W2 = WeightMem2[6];
 N1W3 = WeightMem2[7];
 N1Bias = Bias2[1];
 //W+B for Layer 2 Neuron 2
 N2W0 = WeightMem2[8];
 N2W1 = WeightMem2[9];
 N2W2 = WeightMem2[10];
 N2W3 = WeightMem2[11];
 N2Bias = Bias2[2];
 //W+B for Layer 2 Neuron 3
 N3W0 = WeightMem2[12];
 N3W1 = WeightMem2[13];
 N3W2 = WeightMem2[14];
 N3W3 = WeightMem2[15];
 N3Bias = Bias2[3];
 //W+B for Layer 2 Neuron 4
 N4W0 = WeightMem2[16];
 N4W1 = WeightMem2[17];
 N4W2 = WeightMem2[18];
 N4W3 = WeightMem2[19];
 N4Bias = Bias2[4];
```

```
  //W+B for Layer 2 Neuron 5
  N5W0 = WeightMem2[20];
  N5W1 = WeightMem2[21];
  N5W2 = WeightMem2[22];
  N5W3 = WeightMem2[23];
  N5Bias = Bias2[5];
  //W+B for Layer 2 Neuron 6
  N6W0 = WeightMem2[24];
  N6W1 = WeightMem2[25];
  N6W2 = WeightMem2[26];
  N6W3 = WeightMem2[27];
  N6Bias = Bias2[6];
  //W+B for Layer 2 Neuron 7
  N7W0 = WeightMem2[28];
  N7W1 = WeightMem2[29];
  N7W2 = WeightMem2[30];
  N7W3 = WeightMem2[31];
  N7Bias = Bias2[7];
end

//Neuron 0
Layer2_neuron Lay2Neu0
(
.clk(clk),
.AWGNReal(AWGNReal), .AWGNImag(AWGNImag), .RealPG(RealPG), .ImagPG(ImagPG),
.W0(N0W0), .W1(N0W1), .W2(N0W2), .W3(N0W3), .Bias(N0Bias),
.out(out0)
);

//Neuron 1
Layer2_neuron Lay2Neu1
(
.clk(clk),
.AWGNReal(AWGNReal), .AWGNImag(AWGNImag), .RealPG(RealPG), .ImagPG(ImagPG),
```

```
.W0(N1W0), .W1(N1W1), .W2(N1W2), .W3(N1W3), .Bias(N1Bias),
.out(out1)
);

//Neuron 2
Layer2_neuron Lay2Neu2
(
.clk(clk),
.AWGNReal(AWGNReal), .AWGNImag(AWGNImag), .RealPG(RealPG), .ImagPG(ImagPG),
.W0(N2W0), .W1(N2W1), .W2(N2W2), .W3(N2W3), .Bias(N2Bias),
.out(out2)
);

//Neuron 3
Layer2_neuron Lay2Neu3
(
.clk(clk),
.AWGNReal(AWGNReal), .AWGNImag(AWGNImag), .RealPG(RealPG), .ImagPG(ImagPG),
.W0(N3W0), .W1(N3W1), .W2(N3W2), .W3(N3W3), .Bias(N3Bias),
.out(out3)
);

//Neuron 4
Layer2_neuron Lay2Neu4
(
.clk(clk),
.AWGNReal(AWGNReal), .AWGNImag(AWGNImag), .RealPG(RealPG), .ImagPG(ImagPG),
.W0(N4W0), .W1(N4W1), .W2(N4W2), .W3(N4W3), .Bias(N4Bias),
.out(out4)
);

//Neuron 5
Layer2_neuron Lay2Neu5
(
```

```
  .clk(clk),
  .AWGNReal(AWGNReal), .AWGNImag(AWGNImag), .RealPG(RealPG), .ImagPG(ImagPG),
  .W0(N5W0), .W1(N5W1), .W2(N5W2), .W3(N5W3), .Bias(N5Bias),
  .out(out5)
  );

  //Neuron 6
  Layer2_neuron Lay2Neu6
  (
  .clk(clk),
  .AWGNReal(AWGNReal), .AWGNImag(AWGNImag), .RealPG(RealPG), .ImagPG(ImagPG),
  .W0(N6W0), .W1(N6W1), .W2(N6W2), .W3(N6W3), .Bias(N6Bias),
  .out(out6)
  );

  //Neuron 7
  Layer2_neuron Lay2Neu7
  (
  .clk(clk),
  .AWGNReal(AWGNReal), .AWGNImag(AWGNImag), .RealPG(RealPG), .ImagPG(ImagPG),
  .W0(N7W0), .W1(N7W1), .W2(N7W2), .W3(N7W3), .Bias(N7Bias),
  .out(out7)
  );
endmodule
```

```
/*
        Represents the neuron in the third layer. It performs the MAC operation
        followed by a ReLU activation function through the instantiation of the ReLU
        module.
*/
module Layer2_neuron
  (
  input logic clk,
  input logic signed [15:0] AWGNReal, AWGNImag,
  input logic signed [15:0] RealPG, ImagPG,
```

```
 input logic signed [15:0] W0, W1, W2, W3, //Q3.12
 input logic signed [31:0] Bias, //Q3.12*Q7.24=Q11.36
 output logic signed [15:0] out //Q11.36
 );

 //wires
 logic signed [31:0] mult0,mult1,mult2,mult3;
 logic signed [31:0] sum;
 logic signed [31:0] initOut;

 //multiply and add
 always_comb
 begin
  mult0= AWGNReal*W0;
  mult1= AWGNImag*W1;
  mult2= RealPG*W2;
  mult3= ImagPG*W3;

  sum = mult0 + mult1 + mult2 + mult3+ Bias;
  end

  //ReLU
  ReLU #(.dataWidth(1),.weightWidth(32)) ReluAct (
        .clk(clk),
        .sum(sum), //sum of MAC and bias fed to activation func
        .out(initOut)
      );

  assign out = {initOut[31], initOut[26:12]};
endmodule
```

```
/*
        Represents the fourth layer of the network. Learnable parameters are first
        read and assigned to the corresponding variables.
        The module has 4 instantiations of layer 3 neuron module which correspond
```

```
        to the 4 neurons in the third layer.
*/
module Layer3 #(parameter InputNum=0)
 (
 input logic clk,
 input logic signed [31:0] Pre0, Pre1, Pre2, Pre3,
 input logic signed [15:0] Post0, Post1, Post2, Post3,
 input logic signed [15:0] in0,in1, in2, in3, in4, in5, in6, in7,
 output logic signed [15:0] out0, out1, out2, out3
 );

 //Layer 3 has 4 neurons 0-->3
 logic signed [15:0] N0W0, N0W1, N0W2, N0W3, N0W4, N0W5, N0W6, N0W7;
 logic signed [15:0] N1W0, N1W1, N1W2, N1W3, N1W4, N1W5, N1W6, N1W7;
 logic signed [15:0] N2W0, N2W1, N2W2, N2W3, N2W4, N2W5, N2W6, N2W7;
 logic signed [15:0] N3W0, N3W1, N3W2, N3W3, N3W4, N3W5, N3W6, N3W7;
 logic signed [31:0] N0Bias, N1Bias, N2Bias, N3Bias;
 logic [15:0] WeightMem3 [0:31];
 logic [31:0] Bias3 [0:3];

 initial
 begin
        $readmemb("Layer3WeightsBinary",WeightMem3);
  $readmemb("Layer3BiasBinary", Bias3);
        //LAYER 3//
  //W+B for Neuron 0
  N0W0 = WeightMem3[0];
  N0W1 = WeightMem3[1];
  N0W2 = WeightMem3[2];
  N0W3 = WeightMem3[3];
  N0W4 = WeightMem3[4];
  N0W5 = WeightMem3[5];
  N0W6 = WeightMem3[6];
  N0W7 = WeightMem3[7];
```

```
  N0Bias = Bias3[0];
  //W+B for Neuron 1
  N1W0 = WeightMem3[8];
  N1W1 = WeightMem3[9];
  N1W2 = WeightMem3[10];
  N1W3 = WeightMem3[11];
  N1W4 = WeightMem3[12];
  N1W5 = WeightMem3[13];
  N1W6 = WeightMem3[14];
  N1W7 = WeightMem3[15];
  N1Bias = Bias3[1];
  //W+B for Neuron 2
  N2W0 = WeightMem3[16];
  N2W1 = WeightMem3[17];
  N2W2 = WeightMem3[18];
  N2W3 = WeightMem3[19];
  N2W4 = WeightMem3[20];
  N2W5 = WeightMem3[21];
  N2W6 = WeightMem3[22];
  N2W7 = WeightMem3[23];
  N2Bias = Bias3[2];
  //W+B for Neuron 3
  N3W0 = WeightMem3[24];
  N3W1 = WeightMem3[25];
  N3W2 = WeightMem3[26];
  N3W3 = WeightMem3[27];
  N3W4 = WeightMem3[28];
  N3W5 = WeightMem3[29];
  N3W6 = WeightMem3[30];
  N3W7 = WeightMem3[31];
  N3Bias = Bias3[3];
end
//Neuron 0
Layer3_neuron #(.Neuron(0)) Lay3Neu0
```

```
(
.clk(clk), //.inputNumber(inputNumber), //.PreSoft(PreSoft), .PostSoft(PostSoft),
.Pre0(Pre0), .Pre1(Pre1),.Pre2(Pre2), .Pre3(Pre3),
.Post0(Post0), .Post1(Post1),.Post2(Post2), .Post3(Post3),
.in0(in0), .in1(in1), .in2(in2), .in3(in3), .in4(in4), .in5(in5), .in6(in6), .in7(in7),
.W0(N0W0), .W1(N0W1), .W2(N0W2), .W3(N0W3), .W4(N0W4), .W5(N0W5), .W6(N0W6), .W7(N0W7),
.Bias(N0Bias), .out(out0)
);
//Neuron 1
Layer3_neuron #(.Neuron(1)) Lay3Neu1
(
.clk(clk), //.inputNumber(inputNumber), //.PreSoft(PreSoft), .PostSoft(PostSoft),
.Pre0(Pre0), .Pre1(Pre1),.Pre2(Pre2), .Pre3(Pre3),
.Post0(Post0), .Post1(Post1),.Post2(Post2), .Post3(Post3),
.in0(in0), .in1(in1), .in2(in2), .in3(in3), .in4(in4), .in5(in5), .in6(in6), .in7(in7),
.W0(N1W0), .W1(N1W1), .W2(N1W2), .W3(N1W3), .W4(N1W4), .W5(N1W5), .W6(N1W6), .W7(N1W7),
.Bias(N1Bias), .out(out1)
);
//Neuron 2
Layer3_neuron #(.Neuron(2)) Lay3Neu2
(
.clk(clk), //.inputNumber(inputNumber), //.PreSoft(PreSoft), .PostSoft(PostSoft),
.Pre0(Pre0), .Pre1(Pre1),.Pre2(Pre2), .Pre3(Pre3),
.Post0(Post0), .Post1(Post1),.Post2(Post2), .Post3(Post3),
.in0(in0), .in1(in1), .in2(in2), .in3(in3), .in4(in4), .in5(in5), .in6(in6), .in7(in7),
.W0(N2W0), .W1(N2W1), .W2(N2W2), .W3(N2W3), .W4(N2W4), .W5(N2W5), .W6(N2W6), .W7(N2W7),
.Bias(N2Bias), .out(out2)
);
//Neuron 3
Layer3_neuron #(.Neuron(3)) Lay3Neu3
(
.clk(clk), //.inputNumber(inputNumber), //.PreSoft(PreSoft), .PostSoft(PostSoft),
.Pre0(Pre0), .Pre1(Pre1),.Pre2(Pre2), .Pre3(Pre3),
.Post0(Post0), .Post1(Post1),.Post2(Post2), .Post3(Post3),
```

```
  .in0(in0), .in1(in1), .in2(in2), .in3(in3), .in4(in4), .in5(in5), .in6(in6), .in7(in7),
  .W0(N3W0), .W1(N3W1), .W2(N3W2), .W3(N3W3), .W4(N3W4), .W5(N3W5), .W6(N3W6), .W7(N3W7),
  .Bias(N3Bias), .out(out3)
  );
endmodule
```

```
/*
        Represents the neuron in the fourth layer. It performs the MAC operation
        followed by a Softmax activation function through the instantiation of the Softmax
        module.
*/
module Layer3_neuron #(parameter Neuron=6)
  (
  input logic clk,
  input logic signed [31:0] Pre0, Pre1, Pre2, Pre3,
  input logic signed [15:0] Post0, Post1, Post2, Post3,
  input logic signed [15:0] in0, in1, in2, in3, in4, in5, in6, in7,
  input logic signed [15:0] W0, W1, W2, W3, W4, W5, W6, W7,
  input logic signed [31:0] Bias,
  output logic signed [15:0] out
  );

  logic signed [31:0] mult0,mult1,mult2,mult3, mult4,mult5,mult6,mult7, presoft;

  always_comb begin
   mult0= in0*W0;
   mult1= in1*W1;
   mult2= in2*W2;
   mult3= in3*W3;

   mult4= in4*W4;
   mult5= in5*W5;
   mult6= in6*W6;
   mult7= in7*W7;
```

```
  presoft = mult0 + mult1 + mult2 + mult3+ mult4+ mult5+ mult6+ mult7+ Bias;
 end

 //Instantiate Softmax
 Softmax #(.Neuron(Neuron))SM
 (
 .Pre0(Pre0), .Pre1(Pre1),.Pre2(Pre2), .Pre3(Pre3),
 .Post0(Post0), .Post1(Post1),.Post2(Post2), .Post3(Post3),
 .inputNumber(inputNumber),
 .SoftmaxInput(presoft), .SoftmaxOutput(out)
 );
endmodule
```

```
/*
        Energy Normalization is done through an LUT.
*/
module Normalization #(parameter dataWidth=2, weightWidth=16)
(
input logic [2*weightWidth-1:0]sum,
output logic [weightWidth-1:0] sum_norm
);



always_comb
begin
        case(sum)
                //Real
                -32'd11156953  : sum_norm<= -16'd3141 ; //if -0.665 get -0.7669
                32'd13004771    : sum_norm<= 16'd3317    ; //if 0.7755 get 0.8099
          -32'd11284568    : sum_norm<= -16'd2259   ; //if -0.6727 get -0.55161
                32'd4380842   : sum_norm<= 16'd2331    ; //if 0.2611 get 0.56916
                //Imaginary
                32'd9331493    : sum_norm<= 16'd2628   ; //if 0.5565 get 0.6417i
                -32'd9416784  : sum_norm<= -16'd2402    ; //if -0.5615 get -0.5864i
                -32'd17060644 : sum_norm<= -16'd3416    ; //if -1.017 get -0.8341i
```

```
                32'd6328295  : sum_norm<= 16'd3367   ; //if 0.3772228 get 0.8222i
                default: sum_norm<=0;
        endcase
end
endmodule
```

```
/*
        Represents communication system's decoder. It conatains instantiation of
        Layer 2 and Layer 3.
*/
module Rec_Decode
 (
 input logic clk,
 input logic signed [31:0] Pre0, Pre1, Pre2, Pre3,
 input logic signed [15:0] Post0, Post1, Post2, Post3, RealPG, ImagPG,
 input logic signed [15:0] AWGNReal, AWGNImag,
 output logic signed [15:0] out0, out1, out2, out3
 );
 logic signed [15:0] L2Out0,L2Out1,L2Out2,L2Out3,L2Out4,L2Out5,L2Out6,L2Out7;

 //Receiver or Decoder includes Layer 2 and Layer 3 (fc_4 and fc_7)

 Layer2 fc_4
 (
  .clk(clk),.AWGNReal(AWGNReal), .AWGNImag(AWGNImag), .RealPG(RealPG), .ImagPG(ImagPG),
 .out0(L2Out0), .out1(L2Out1),.out2(L2Out2),.out3(L2Out3),.out4(L2Out4),.out5(L2Out5),
 .out6(L2Out6),.out7(L2Out7)
 );

 Layer3 fc_7
 (
 .clk(clk),
 .Pre0(Pre0), .Pre1(Pre1),.Pre2(Pre2), .Pre3(Pre3),
 .Post0(Post0), .Post1(Post1),.Post2(Post2), .Post3(Post3),
```

```
  .in0(L2Out0), .in1(L2Out1), .in2(L2Out2), .in3(L2Out3), .in4(L2Out4), .in5(L2Out5), .in6(L2Out6),
.in7(L2Out7),

  .out0(out0), .out1(out1), .out2(out2), .out3(out3)

  );

endmodule
```

```
/*

        Represents ReLU activation function.

        If input is negative --> output =0

        If input is positive --> output =input

*/

module ReLU #(parameter dataWidth=1, weightWidth=32)

(input logic clk,

input logic signed [dataWidth*weightWidth-1:0] sum,

output logic signed [dataWidth*weightWidth-1:0] out

);


always_comb begin

        if (sum < 0)

                out=32'b0;

        else

                out=sum;

end

endmodule
```

```
/*

        Represents the Rician Fading Channel.

        Performs the complex multiplication between the transmitted signal

        and the path gains.

*/

module Rician

  (

  input logic signed [15:0] RealTx, ImagTx, RealPG, ImagPG,

  output logic signed [15:0] RealYMimo, ImagYMimo

  );
```

```
 logic signed [31:0] RealYMimo32,ImagYMimo32;
 always_comb
 begin
   RealYMimo32<= (RealTx*RealPG - ImagTx*ImagPG);
   ImagYMimo32<= (RealTx*ImagPG + ImagTx*RealPG);

 end
  assign RealYMimo= {RealYMimo32[31], RealYMimo32[26:12]};
  assign  ImagYMimo= {ImagYMimo32[31], ImagYMimo32[26:12]};
endmodule
```

```
/*
        Perfoms Softmax activation function through an LUT.
*/
module Softmax #(parameter Neuron=6)
 (
 input logic signed [31:0] Pre0, Pre1, Pre2, Pre3,
 input logic signed [15:0] Post0, Post1, Post2, Post3,
 input logic signed [31:0] SoftmaxInput, //input to softmax
 input logic inputNumber,
 output logic signed [15:0] SoftmaxOutput //output of softmax
 );

 always_comb
 begin
   if(SoftmaxInput==Pre0)
     SoftmaxOutput=Post0;
   else if (SoftmaxInput==Pre1)
     SoftmaxOutput=Post1;
   else if (SoftmaxInput==Pre2)
     SoftmaxOutput=Post2;
   else if (SoftmaxInput==Pre3)
     SoftmaxOutput=Post3;
 end
endmodule
```

```
/*
        Represents the entire communication system by linking the Transmitter,
        Channel, Reciever, and Classification.
*/
module TopLevel
 (
 input logic clk,
 input logic signed [31:0] Pre0, Pre1, Pre2, Pre3,
 input logic signed [15:0] Post0, Post1, Post2, Post3,
 input logic signed [4-1:0] in,
 input logic signed [15:0] RealPG16, ImagPG16,
 input logic signed [15:0] NoiseReal, NoiseImag,
 output logic signed [1:0] Predicted
 );
 logic signed [15:0] Out0L1, Out1L1;
 logic signed [15:0] RayReal, RayImag;
 logic signed [15:0] AWGNReal, AWGNImag;
 logic signed [15:0] Out0L3, Out1L3, Out2L3, Out3L3;

 //Instantiate Transmitter
 Trans_Encode Tx
 (
 .clk(clk),.in(in),
 .Out0L1(Out0L1), .Out1L1(Out1L1)
 );

 //Instantiatiate Rayliegh
 Rician Ray
 (
 .RealTx(Out0L1), .ImagTx(Out1L1), .RealPG(RealPG16), .ImagPG(ImagPG16),
 .RealYMimo(RayReal), .ImagYMimo(RayImag)
 );

 //Instantiale AWGN
```

```
 AWGN WhiteGN
 (
 .RayReal(RayReal), .RayImag(RayImag), .NoiseReal(NoiseReal), .NoiseImag(NoiseImag),
 .AWGNReal(AWGNReal), .AWGNImag(AWGNImag)
 );

 //Instantiate Receiver
 Rec_Decode Rx
 (
 .clk(clk),
 .Pre0(Pre0), .Pre1(Pre1),.Pre2(Pre2), .Pre3(Pre3),
 .Post0(Post0), .Post1(Post1),.Post2(Post2), .Post3(Post3),
 .AWGNReal(AWGNReal), .AWGNImag(AWGNImag), .RealPG(RealPG16), .ImagPG(ImagPG16),
 .out0(Out0L3), .out1(Out1L3), .out2(Out2L3), .out3(Out3L3)
 );

 //Instantiate Classification Layer
 Classify CL
 (
 .In0(Out0L3), .In1(Out1L3), .In2(Out2L3), .In3(Out3L3),
 .Predicted(Predicted)
 );
endmodule
```

```
/*
        Represents communication system's transmitter. It conatains instantiation of
        Layer 0 and Layer 1.
*/
module Trans_Encode
(
input logic clk,
input logic signed [4-1:0] in,
output logic signed [16-1:0] Out0L1, Out1L1
);
```

```
logic signed [15:0] Out0L0,Out1L0, Out2L0, Out3L0, Out4L0, Out5L0, Out6L0, Out7L0;


Layer0 #(.NN(8), .numWeight(4), .dataWidth(1), .weightWidth(16)) fc_2
(
.clk(clk),.in0(in[3]), .in1(in[2]), .in2(in[1]), .in3(in[0]),
.out0(Out0L0), .out1(Out1L0), .out2(Out2L0), .out3(Out3L0), .out4(Out4L0), .out5(Out5L0), .out6(Out6L0), .out7(Out7L0)
);


Layer1 #(.NN(2), .numWeight(8), .dataWidth(16), .weightWidth(16)) fc_3
(
.clk(clk), .in0(Out0L0), .in1(Out1L0), .in2(Out2L0), .in3(Out3L0), .in4(Out4L0), .in5(Out5L0), .in6(Out6L0), .in7(Out7L0),
.out0(Out0L1), .out1(Out1L1)
);
endmodule
```

```
/*
        Represents the neuron in the first layer. It performs the MAC operation
        followed by a ReLU activation function through the instantiation of the Softmax
        module.
        It takes each bit of the input as input and multiplies it with the
        corresponding weight. The output is fed to the ReLU.
*/
module input_neuron #(parameter numWeight=4, dataWidth=1,weightWidth=16)
(
input logic  clk, //read_en,
input logic  [dataWidth-1:0]in0,in1,in2,in3, //input taken is 4 bit, each neuron takes 1
input logic  signed [weightWidth-1:0] W0,W1,W2,W3,Bias,
output logic  [dataWidth*weightWidth-1:0] out
);

logic signed [dataWidth*weightWidth-1:0] mult0,mult1,mult2,mult3; //////1 bit x 32 bit, output width=33
logic signed [dataWidth*weightWidth-1:0] sum;

always_comb begin
```

```
  mult0= in0*W0;
  mult1= in1*W1;
  mult2= in2*W2;
  mult3= in3*W3;

  sum = mult0 + mult1 + mult2 + mult3+ Bias;
end
//instantiate RELU
ReLU #(.dataWidth(dataWidth),.weightWidth(weightWidth)) ReluAct (
        .clk(clk),
        .sum(sum), //sum of MAC and bias fed to activation func
        .out(out)
      );
endmodule
```