STATISTICAL ANALYSIS OF A CHANNEL EMULATOR FOR NOISY GRADIENT DESCENT LOW DENSITY PARITY CHECK DECODER

by

Rakin Muhammad Shadab

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

of

MASTER OF SCIENCE

in

Electrical Engineering

Approved:

Chris Winstead, Ph.D. Major Professor	Jacob Gunther, Ph.D. Committee Member
Jonathan Phillips, Ph.D. Committee Member	Richard S. Inouye, Ph.D. Vice Provost for Graduate Studies

UTAH STATE UNIVERSITY
Logan, Utah

2019

ABSTRACT

Statistical Analysis of a Channel Emulator for Noisy Gradient Descent Low Density Parity Check Decoder

by

Rakin Muhammad Shadab, Master of Science

Utah State University, 2019

Major Professor: Chris Winstead, Ph.D.
Department: Electrical and Computer Engineering

The Statistical accuracy of a channel emulator is crucial for the proper evaluation of the performance of an error-correcting decoder. Large deviations from the supposed probability distribution of a channel might result in incorrect Bit Error Rate (BER) and Frame Error Rate (FER) estimations. This work investigates the confidence on the statistical analysis of a SystemC based hardware Gaussian channel emulator and its subsequent effect on a Low Density Parity Check (LDPC) decoder based on Noisy Gradient Descent Bit Flipping (NGDBF) algorithm. The examination of channel emulator includes testing the distribution on the extreme tails and checking for correlations with initial seed values for the channel. Both the channel emulator and the decoder are implemented on Xilinx Virtex VCU118 Field Programmable Gate Array (FPGA) hardware platform. The emulator itself is tested separately on Xilinx Virtex-7 VCU707 FPGA device.

(75 pages)

PUBLIC ABSTRACT

Statistical Analysis of a Channel Emulator for Noisy Gradient Descent Low Density Parity Check Decoder

Rakin Muhammad Shadab

The purpose of a channel emulator is to emulate a communication channel in real-life use case scenario. These emulators are often used in the domains of research in digital and wireless communication. One such area is error correction coding, where transmitted data bits over a channel are decoded and corrected to prevent data loss. A channel emulator that does not follow the properties of the channel it is intended to replicate can lead to mistakes while analyzing the performance of an error-correcting decoder. Hence, it is crucial to validate an emulator for a particular communication channel. This work delves into the statistics of a channel emulator and analyzes its effects on a particular decoder.

To my family and all the teachers I've had so far in life

ACKNOWLEDGMENTS

I am indebted to my major advisor, Dr. Chris Winstead for letting me in his research group and familiarizing me with the concepts of error-correcting codes. As a newbie in this domain, I was overwhelmed by the complexity and vastness of the contents. But Dr. Winstead's encouragement and helpful bits of advice gradually made me confident. His mentoring, training and financial supports were invaluable to me. I shall always remain thankful for the time and patience that he invested in me to groom me as a researcher. I learned a great deal about digital system design on FPGA platform while serving as a TA in one of his courses, ECE 3700. His generous company and timely guidance kept me on track for this thesis. This work would never have been possible without him.

I have also been helped and supported by many faculty members in the ECE department. Thanks to Dr. Sanghamitra Roy for the initial funding offer that helped me to come to the USU in the first place. I am grateful to Dr. Todd Moon for his outstanding teaching and all the help in his courses. Also thanks to Dr. Jake Gunther and Dr. Jonathan Phillips for useful suggestions and for serving in my committee. To Dr. Zhang for all his advices and time. My sincere gratitude goes to Dr. Moon and Dr. Phillips for their recommendations that helped me get admission in a good Ph.D. program. Also, thanks to the ECE department for supporting me financially that allowed me to complete my research work. The trio of awesome ladies in the administrative side of the department - Tricia Brandenburg, Diane Buist and Kathy Phippen - deserve special mentions from me for their helpfulness and kindness.

I would like to convey my thankfulness to the fellow lab members at Low Energy Fault-Tolerant Systems Lab - Dr. Tasnuva Tithi and Rejoy Mathews - for their continued support and good friendship. Rejoy, in particular, has been a source of mental strength for me throughout this journey and I can't thank him enough for that. I appreciate all the helps I received from Dr. Mehedi Hasan and Dr. Asaduzzaman Towfiq with critical concepts in various coursework. I would also like to give a shout-out to my friends - Mohammad

Abdullah Al Sarfin, Md Munibun Billah, Md Ferdous Pervej, Abrar Zahin, Nazmus Sakib, Sanat Kumar Saha & Shaju Saha - for their selfless supports and inspirations on countless occasions, for the fun times together, for being family members to me over the course of this degree. The Bangladeshi community in Logan also deserves a special mention for all the parties, potlucks, tours and adventures we had together.

Lastly, I am highly obliged to my parents, my brother, my relatives, friends and family back at home. I wouldn't be here today without their well wishes and constant backing.

Rakin Muhammad Shadab

CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACRONYMS

ASIC	application specific integrated circuit
AWGN	additive white Gaussian noise
AXI	advanced extensible interface
BER	bit error rate
BERT	bit error rate tester
CDF	cumulative distribution function
dB	decibels
DUT	design under test
FER	frame error rate
FPGA	field programmable gate array
HDCE	hardware discrete channel emulator
HDL	hardware description language
IP	intellectual property
LDPC	low density parity check
MSB	most significant bit
NGDBF	noisy gradient descent bit-flip
PDF	probability density function
Q-Q	quantile-quantile
RTL	register transfer logic
UART	universal asynchronous receiver-transmitter

CHAPTER 1

INTRODUCTION

Low Density Parity Check (LDPC) codes are one of the most commonly used coding schemes in Error-Correction Coding. This particular type of code was first theorized by Robert Gallager in 1963 [1]. According to [1], an (n, j, k) LDPC code is characterized by a block length of n and a matrix with fixed j number of 1's along each row and fixed j number of 1's along each column. This matrix is called a parity-check matrix and the number of 0's in a parity check matrix is comparatively larger than the number of 1's [1].

The curve representing the performance of an LDPC code shows a steady decrease of Frame Error Rate (FER) or Bit Error Rate (BER) against the increase in Signal to Noise Ratio (SNR) [2]. However, as mentioned in [2], the curve deviates from a steady decline after a certain point. The area where the curve exhibits this characteristic is known as error-floor region [2]. According to [2], determining the error floors especially at low SNR levels can be very tricky using software simulators because of the significantly long simulation time. Mackay and Postol were the first to find out the error-floor region from a "weakness" in the Margulis construction of regular (3,6) Gallager LDPC codes [3]. The error-floors are the result of the presence of "near-codewords" that are termed as "trapping sets" [4]. The term "absorbing sets" which can be considered as a special subclass of trapping sets, is used to illustrate the failures in decoding for different message-passing algorithms with the convergence to distinct non-codeword states [5]. Very few pieces of literature have explored the confidence in statistical tests while determining the BER with the use of a specific communication channel. Rice et al. compared the outcomes of binomial and negative binomial tests for estimating BER [6]. Mazzeo et al. proposed the notion of a confidence interval estimator using two negative binomial tests [7].

Hardware-based emulations on Field Programmable Gate Array (FPGA) platforms have been used to accurately figure out error floors for LDPC codes with much faster

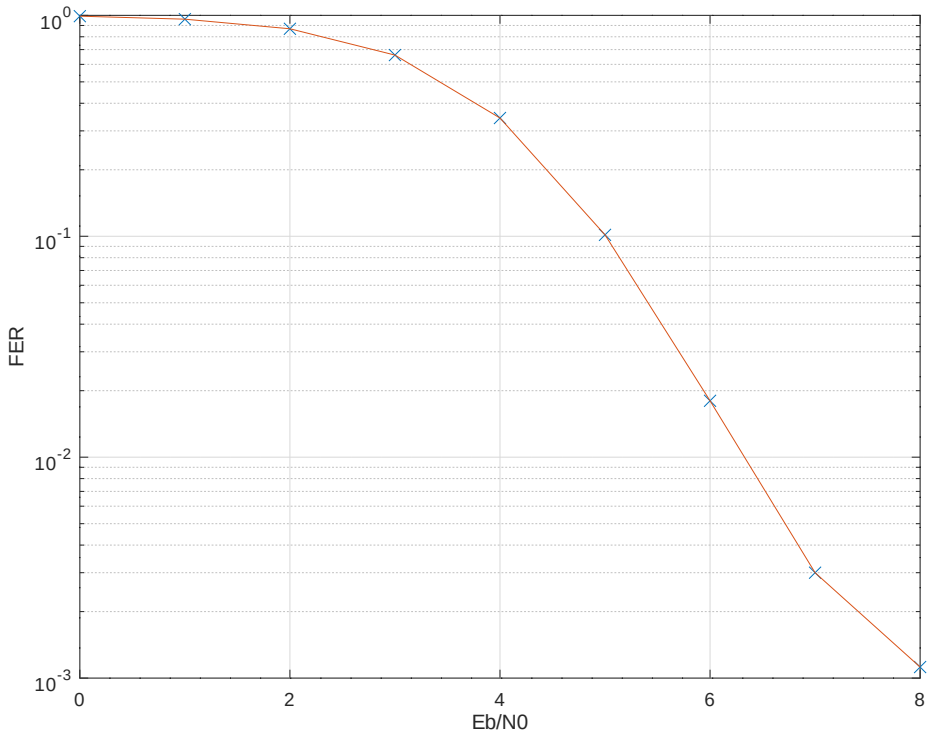

Fig. 1.1: FER results from a sparse LDPC code ((155,64) Tanner Code) using basic NGDBF algorithm with zero noise

execution time compared to software-based approach [8]. Similar emulations have been used for the encoder side, especially to simulate the properties of a noisy communication channel with Gaussian noise at low BER on FPGA platform [9]. Boutillon et al. designed a fast channel emulator named Hardware Discrete Channel Emulator (HDCE) that can be used with LDPC decoders [10].

The focus of this work is on the statistical quality analysis of a hardware-based SystemC derived Gaussian channel emulator and its effect on the performance of an LDPC decoder based on Noisy Gradient Descent Bit-Flip (NGDBF) algorithm in the error floor region. NGDBF is an efficient decoding technique for LDPC codes, proposed by Sundarajan et al. [11]. The SystemC model for the channel emulator was developed by Dr. Bertrand Le

Gal, a professor at Bordeaux INP in France. A series of previous research on the hardware implementation of NGDBF decoder with the aforementioned channel emulator resulted in observations of some randomly anomalous BER outcomes at very high SNR in the error floor region. That discovery suggested the possibility of the anomaly being related to either an inherent feature of the decoder or an artifact in the channel emulator. It also highlighted the importance of independent testing of the emulator in question and thus motivated this research work greatly. For the test purposes related to the investigations of this thesis, Xilinx Virtex-7 VC707 FPGA platform is used to test the channel emulator separately and Xilinx Virtex VCU118 FPGA platform is used to inspect the combination of decoder design along with the emulator. The scope of this work includes but is not limited to:

I Proposing a common sequence of steps that can be used to test similar hardware-based channel emulators.

II Testing the accuracy of statistical distribution for the aforementioned channel emulator.

III Inspecting the quality of Gaussian distribution on extreme tail regions.

IV Evaluating the confidence in the measurement of statistical tests.

V Investigating dependency of the output of Gaussian noise generator on the seed.

VI Analyzing the changes in frame errors for the NGDBF decoder in error-floor region with this channel emulator.

1.1 Chapter Organization

The following parts of this thesis are organized as below:

- Related Work: An overview of the contemporary research on hardware based channel emulators and random noise generators is provided in the second chapter. These works are explored to validate the necessity of the approach for the testing methodology used in this research.

- Verification Methodology: The third chapter includes an in-depth look of the test platforms and the verification steps used to generate and analyze the test results for the systemC based channel emulator. A general order of steps to validate the emulator is proposed in this case.

- Results & Analysis: All the test outputs are observed and scrutinized to complete the research objectives. An analysis of the decoder performance in the error-floor region is also conducted and the anomalies from expected outcomes are inspected.

- Conclusions & Future Work: The chapter summarizes the contributions of this thesis and the outcomes from it. It also highlights the possible premises of any future research endeavor that would extend this work.

CHAPTER 2

RELATED WORK

Different hardware implementations for random Gaussian noise generation were described in past works of literature. Producing stochastic noise on a hardware platform is advantageous while using a hardware-based error-correcting decoder as in that case, a software-based approach is not needed to generate noise and then port it over to the hardware. Boutillon et al. used a combination of a quantized version of the Box-Muller method with the central limit theorem to convert a uniform distribution to Additive White Gaussian noise (AWGN) and implemented it on FPGA platform [12]. Later on, Fang et al. used this as the basis for their Application Specific Integrated Circuit (ASIC) design of a channel emulator [13]. However, in both of these works, the quality of the generated noise sample was not satisfactory and hence Lee et al. proposed a design that uses better function evaluation techniques and produces noise samples with notable improvements in quality [14]. Later on, this design was improved with more efficient hardware, better noise samples and better noise quality in the tails region while getting rid of the central limit theorem [15]. The literature on similar hardware-based noise generators from Lee et al. and Zhang et al. using Wallace and Ziggurat methods respectively are also available [16, 17]. HDCE, on the other hand, was developed using the alias method [10]. Prior works on HDCE were focused on the complexity, performance and accuracy of the platform but there was no analysis on the reliability of error-floor measurements for error-correcting decoders [10]. The noticeable fact about all the aforementioned literatures is that none of them contains the description of hardware design implementation on respective test platforms in great detail. Therefore, making a direct comparison between the test results might be very difficult.

The work in this scope will emphasize on different statistical tests performed on the previously mentioned SystemC based channel emulator, especially in the extreme regions of the tails. Prior research efforts on this emulator focused on its implementation in com-

bination with NGDBF decoder where large deviations in the number of erroneous bits for some specific initial seeds and much fewer variations for other seed values in the error floor analysis were observed. As the reason behind this anomaly was not explored before, this work aspires to look into it to identify whether the deviations are caused by the channel emulator, the design of test platform or the algorithm of the decoder. To the best of my knowledge, this is the first work to investigate the evaluation of the measurements of several statistical tests on this particular channel emulator, the emulator's dependency on different initial seeds and the confidence on test statistics.

CHAPTER 3

VERIFICATION METHODOLOGY

The primary scope of this thesis is to analyze the statistical accuracy of a high-level synthesized channel emulator implemented on a Xilinx Virtex-7 VC707 FPGA up to extreme tail regions, have an approximation of the confidence on these statistics and test the effects of these results on the performance of NGDBF decoder. For this purpose, different statistical tests are going to be conducted on the channel samples. Shapiro-Wilk test and Kolmogorov-Smirnov test are two such tests where the reference distribution from the channel emulator is compared against a null distribution (In this case, the null distribution is standard Gaussian Probability Density Function (PDF)). These tests are carried out on MATLAB & R platform. On R platform, the functions 'shapiro.test' & 'ks.test' are used. On the other hand, built-in function 'kstest2' & a custom function 'swtest' written by Ahmed Ben Saïda (Department of Finance, Accounting & Management, University of Sousse, Tunisia) are used for MATLAB analysis. Quantile-Quantile (Q-Q) plots are also used as they can be a good indicator of how closely the pdf under test follows the null distribution. The outcomes of all these tests, in turn, will be particularly useful to verify if the channel emulator is following a Normal distribution. But since the overall emphasis is to investigate the quality of distribution on extreme tails, there should be additional tests besides the aforementioned ones as those tests are highly sensitive to the tail regions while testing normality for a very large number of samples.

Moreover, further tests are required to determine if the use of different initial seeds has any significant impact on the distribution of channel emulator. The dependency of the statistics on these seed values is needed to be as little as possible for the distribution to be close to that of a standard Gaussian one.

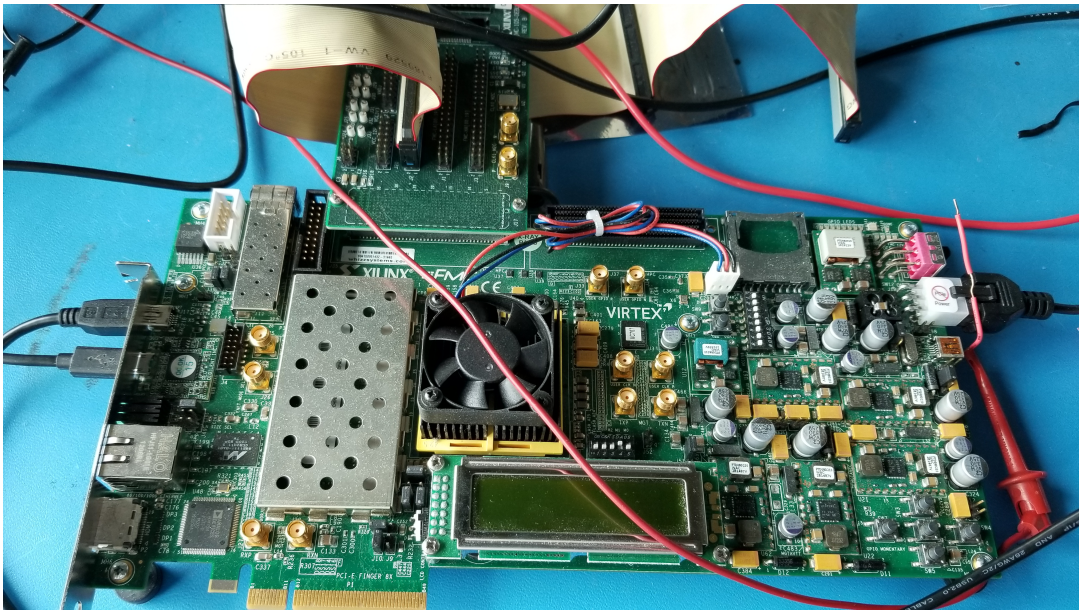

Fig. 3.1: Xilinx Virtex-7 VC707 FPGA platform

A convenient way to measure the quality of a huge number of random quantized channel samples coming out of the emulator is to represent the occurrence of different threshold levels by a histogram. In this case, the histogram data is taken out of a testbench simulation while also being reported from a hardware-level implementation. Both the simulation and the hardware implementation work in identical ways to achieve these results. The results are then compared against the outputs from two reference models in MATLAB.

The Hardware setup for the emulator includes the use of Microbalze soft-core processor-based design. An Advanced Extensible Interface (AXI) for Universal Asynchronous Receiver-Transmitter (UART) peripheral module is used to test the output of the channel emulator with different 8-bit seed values. The outputs for 256 distinct seed values are analyzed and the correlation between a particular initial seed and its corresponding channel samples is determined. If any of the seed values exhibit some specific influence on the distribution of the channel, that value will be ignored for further analysis.

Past works of research contributed to a design that combines the relevant channel emulator with NGDBF decoder and it is implemented on Xilinx Virtex VCU118 FPGA

platform to analyze decoder output with respect to the emulator. This work will explore the relationship of channel emulator with the performance of decoder in the error floor region. In regards to the anomalies in the result observed in previous research, the focus would be to inspect and test the Register Transfer Logic (RTL) design to try to replicate the anomalies and figure out their nature.

The first two sections of this chapter focus on the background of Shapiro-Wilk & Kolmogorov-Smirnov tests. The third section elaborates on proposing a common pattern of steps that can be used to validate similar hardware-based channel emulators. After that, the following sections describe the test methodology in detail. The last section suggests a new state machine design to modify the existing RTL design for Bit Error Rate Tester (BERT) module.

Fig. 3.2: Xilinx Virtex VCU118 FPGA platform

3.1 Shapiro-Wilk Test & Its Extensions

In 1965, Shapiro and Wilk [18] proposed a normality test where the test statistic W can be represented as :

$$W = (\sum_{i=1}^{n} a_i y_i)^2 / \sum_{i=1}^{n} (y_i - \bar{y}^2) \tag{3.1}$$

Here y_i is random observation $(i = 1, 2, ..., n)$ and the coefficients $(a_1, a_2, ..., a_n)$ are determined by:

$$(a_1, ..., a_n) = (m' V^{-1}) / (m' V^{-1} V^{-1} m)^{1/2} \tag{3.2}$$

where $m' = (m_1, m_2, ...m_n)$ are the reference values for standard normal distribution and $V = (\nu_{ij})$ is an nXn covariance matrix that corresponds to those values [18]. However, the approximations were developed to account for sample size of up to 50 samples in this case and later on, Shapiro & Francia presented a modification of this test for larger sample size [19]. In 1997, Rahman & Govindarajulu modified the test statistic to extend the sample size up to 5000 [20].

3.2 Kolmogorov-Smirnov Test

Kolmogorov-Smirnov test can be used to compare any test samples against a standard distribution (In this case, Normal distribution). In 1933, Andrey Kolmogorov theorized the details of Kolmogorov-Smirnov test statistic [21]. A table for the related empirical distribution of the test was provided by Smirnov in 1948 [22]. According to Kolmogorov & Smirnov [21, 22], If $(X_1, X_2, ...X_n)$ are independent variables with Cumulative Distribution Function (CDF) $F(x)$ then the empirical distribution can be defined as

$$F_n^* = N(z)/n \tag{3.3}$$

Where $N(z)$ is the number of observations less than z. From [22], the maximum difference between $F_n^*(z)$ and $F(z)$ is

$$D_n = max|F_n^*(z) - F(z)| \tag{3.4}$$

The CDF that limits the random variable D_n [22] is

$$L(z) = n^{1/2} D_n \tag{3.5}$$

Unlike Shapiro-Wilk test, the Kolmogorov-Smirnov test does not appear to have a real theoretical limit on its test sample size.

3.3 A Common Pattern for Test Methodology

Since the hardware implementation of different channel emulators can vary widely, establishing a common test and verification pattern would be useful. A sequence of steps to validate a hardware-based channel emulator is proposed in Fig. 3.3.

The design under test is the input in this scenario. The verification process starts with the statistical tests mentioned in the previous sections. As a follow-up, a high-level golden reference model is developed to simulate ideal behavior for an AWGN channel emulator. The results from the testbench simulation and hardware platform are then compared against the results of the reference model. If there is a mismatch, the design for test setup needs to be reviewed. In case of consistent results, the next logical step would be to check if the outputs are particularly dependent on any parameter (for example, dependence on the initial random seed) and whether the channel emulator is experiencing a proper reset after every operation. The test platform will be required to go through another review process if the results are anomalous. Finally, if all the tests return favorable outcomes, the channel emulator is validated. Otherwise the inference would be the assumption that there are major issues with the design of the emulator.

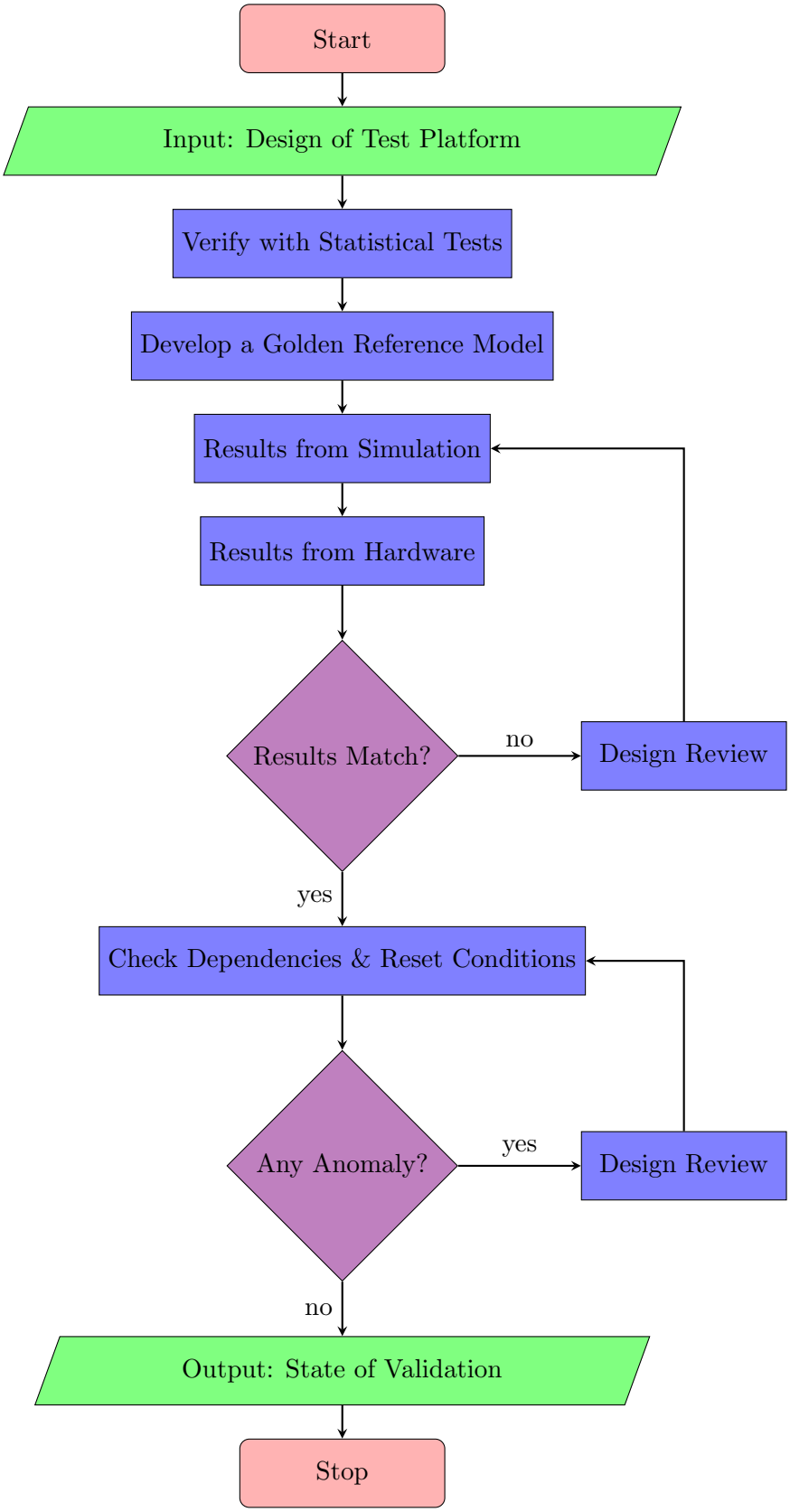

Fig. 3.3: Flow chart of verification steps to test channel emulator

3.4 Test Procedure - MATLAB Models

Keeping in touch with the steps proposed in the last section, two MATLAB reference models are created to validate the testbench and hardware results. The first model takes in non-quantized fractional-integer data from testbench simulation and then performs 5-bit quantization on the output. It takes in 21-bit samples and extracts the fractional part and 2's complement part from each of them. It then calculates the magnitude for every sample from this information and scales it by a scaling factor. The scaled value of magnitude is 42 bits wide and it is truncated to a 32-bit value called *intsample*. The value of *intsample* is then shifted by *delta* which is a value based on the flipping threshold of NGDBF algorithm. The channel samples are quantized to 5 bits depending on the value of shifted *intsample* where the Most Significant Bit (MSB) is sign bit and the other 4 bits represent the magnitude. These quantized samples depict 32 separate threshold levels. In the end, the first model generates histogram results with these threshold values. The second model imports non-quantized random sample values ranging in between 0 to 1 from RTL simulation and generates a data set with normal distribution using the mean and variance of testbench data with normrnd function. It then quantizes that set of data into 5-bit samples with similar quantization method used in the first model. It also outputs the results in a histogram format. If the outcomes from testbench and FPGA line up with the results produced by these reference models, those outputs can be assumed to be valid.

3.5 Test Procedure - Testbench Simulation

Simulation of four different Hardware Description Language (HDL) testbenches is used for the test purposes. The first testbench for the channel emulator generates non-quantized sample values both in general and in fractional-integer format. The common format is a set of floating-point random numbers that has a span in between 0 to 1. In fractional-integer format, the samples are 21 bits in size where the first bit or the MSB is a sign bit, followed by 5 bits in 2's complement format to represent integer part. The remaining 16 bits are always positive and constitute the fractional part of the sample. These samples are generated with a mean value of +1. The non-quantized results in the output are suitable for

the previously mentioned normality tests. However, the outcome of those tests still won't have conclusive results on the probability distribution in the tail region for a large number of samples. Hence, the aforementioned testbench is used to also produce 5-bit quantized sign-magnitude values using a quantization technique identical to the ones used in the MATLAB models. A histogram format then portrays the distribution of these channel samples and the repetition of particular quantization levels. A second testbench is used to log the bin values for the histogram. Another testbench simulation is employed to test if the emulator is reset properly after a change in the initial seed. Finally, another Verilog testbench instantiates the entire design of the test platform with the quantization and histogram generation being done in the design level.

3.6 Test Procedure - Hardware Implementation of Channel Emulator

Even though the simulation results would represent an accurate picture of the emulator characteristics, these would not be very reliable for the bin values in the tail regions because the simulation would only be able to generate thousands of samples within a considerable amount of time.

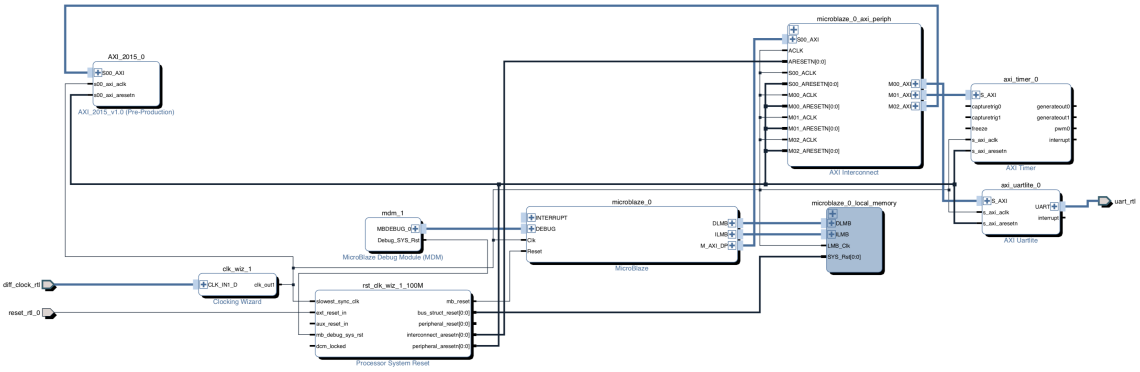

Fig. 3.4: Block Diagram for Microblaze Embedded System to Test Channel Emulator

The hardware implementation of the channel emulator goes through similar 5-bit quantization compared to the simulation. However, the FPGA platform is capable of generating

billions of samples within a very short duration whereas the testbench simulation would be limited to a much smaller data set. To obtain these values from the hardware, the required modifications are done at the top level of the design. Packaging the channel emulator design into an Intellectual Property (IP) and connecting it with a Microblaze processor-based system using the AXI UART interface helps the serial communication with FPGA device. Microblaze is a soft-core processor system exclusive to Xilinx FPGA platforms. As shown in Fig. 3.4, the Microblaze processor system comprises of a clocking wizard, a reset module, a local memory block, a debug module, an AXI peripheral block and the processor itself. An AXI timer is part of the design to control the time delay in the communication interface. This setup is also very reliable for the analysis of the impact of different initial seed values on the channel samples for a particular SNR.

A representation of the design hierarchy of the test platform is seen in Fig. 3.5. As evident in Fig. 3.5, the custom AXI peripheral IP, `AXI_2015_0` is where the RTL code base for the channel emulator lies in. `AXI_2015_v1_0.v` is the top module of that IP and it instantiates `AXI_2015_v1_0_S00_AXI.v` which contains the design and handshaking operations for Microblaze slave registers. `AXI_2015_v1_0_S00_AXI.v` makes an instantiation of custom IP `channel_top_0` that contains the Design Under Test (DUT). The top module of `channel_top_0` is a SystemVerilog design, `channel_top.sv` that performs 5-bit quantization on the data coming from DUT module, `sc_awgn.vhd` and generates histogram for the quantized data.

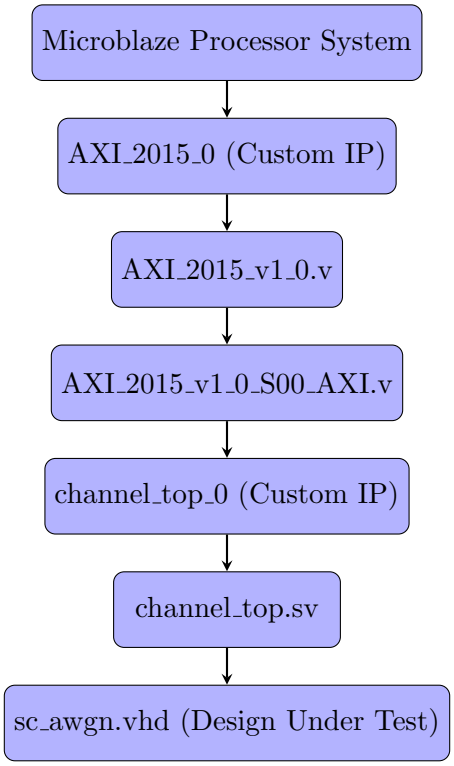

Fig. 3.5: Hierarchy of the test platform design to validate channel emulator

3.7 Bit Error Rate Tester (BERT) Design

Previous research merged the NGDBF decoder design with the channel emulator and it was implemented on a separate test setup. The hardware implementation of NGDBF decoder contains six Bit Error Rate Tester (BERT) modules. Each of the BERT modules contains a channel emulator (In this case, the high-level synthesized channel emulator described in previous sections) that simulates transmission of bits in an Additive White Gaussian Noise (AWGN) channel. Every BERT block also contains a noise generator which produces a pseudo-random sequence of threshold perturbations used in the NGDBF decoding algorithm. The noise emulator and channel emulator each produce one pseudorandom sample per clock cycle. On the channel emulator side, samples are loaded serially into a shift register until a complete frame comprising of 2048 samples is acquired. Once the decoder is initialized, the full frame of channel samples is loaded in parallel into a different

set of registers. The NGDBF decoder operates on this fixed frame of channel samples until decoding is done. The BERT then samples the decoder output, counts error events, and re-initializes the decoder with a fresh frame of channel samples.

In case of the noise generator, the noise samples are loaded serially into a shift register. After each clock cycle, the noise samples are provided directly to the decoder in parallel and are shifted serially in each clock cycle. Hence, each bit-flipping processor is provided with a fresh noise sample at each clock cycle.

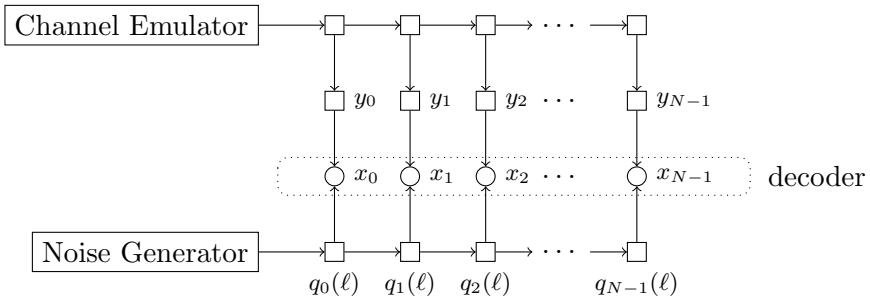

Fig. 3.6: Simplified BERT architecture

Inside the NGDBF decoder, there is one bit-flipping processor for every bit in the frame, for a total of 2048 processors. In addition, there is a set of parity-check modules that detect parity violations in the code by performing XOR operations. Every bit in a frame is associated with six parity checks, and the parity-check outputs are termed s_j, with the condition $0 \leq j < 6$. At iteration ℓ, the bit-flipping processors compute a version of the NGDBF decoding rule:

$$\text{Flip bit } i \text{ if } x_i y_i + \sum_{j=0}^{5} s_j < \theta + q_i(\ell),$$

where $x_i \in \{+1, -1\}$ is the bipolar bit decision at position i, y_i is the corresponding channel sample, θ is a global flipping threshold parameter, and $q_i(\ell)$ is a noise perturbation for processor i at iteration ℓ [11]. The purpose of $q_i(\ell)$ is to perturb the flipping threshold, which empirically improves decoding performance. At present, there is no explicit theory

for constructing good perturbation sequences, so pseudorandom noise samples are used for the $q_i(\ell)$ sequence. The $q_i(\ell)$ values are called "noise samples," but randomness is not necessarily a mandatory feature of these samples. A simplified architecture of the BERT design is shown in Fig. 3.7.

The actual hardware design is highlighted in Fig. 3.7. It uses 802.3 an LDPC code [23].This design, just like the other one, includes a Microblaze processor system. The decoder test configuration, as mentioned before, has six identical Bit Error Rate Tester (BERT) modules in addition to the processor system. The BERT blocks operate in parallel for determination of the total number of bits and frames with errors to calculate BER & FER for different test parameters. In the context of this work, the design in question will be explored to investigate the influence of channel emulator on decoder outcomes in the error floor region. In the case of the channel emulator being validated, the source of the anomaly is likely to be the design of BERT modules or even the algorithm of NGDBF decoder. The state machine incorporated in the existing BERT design is represented in Fig. 3.8.

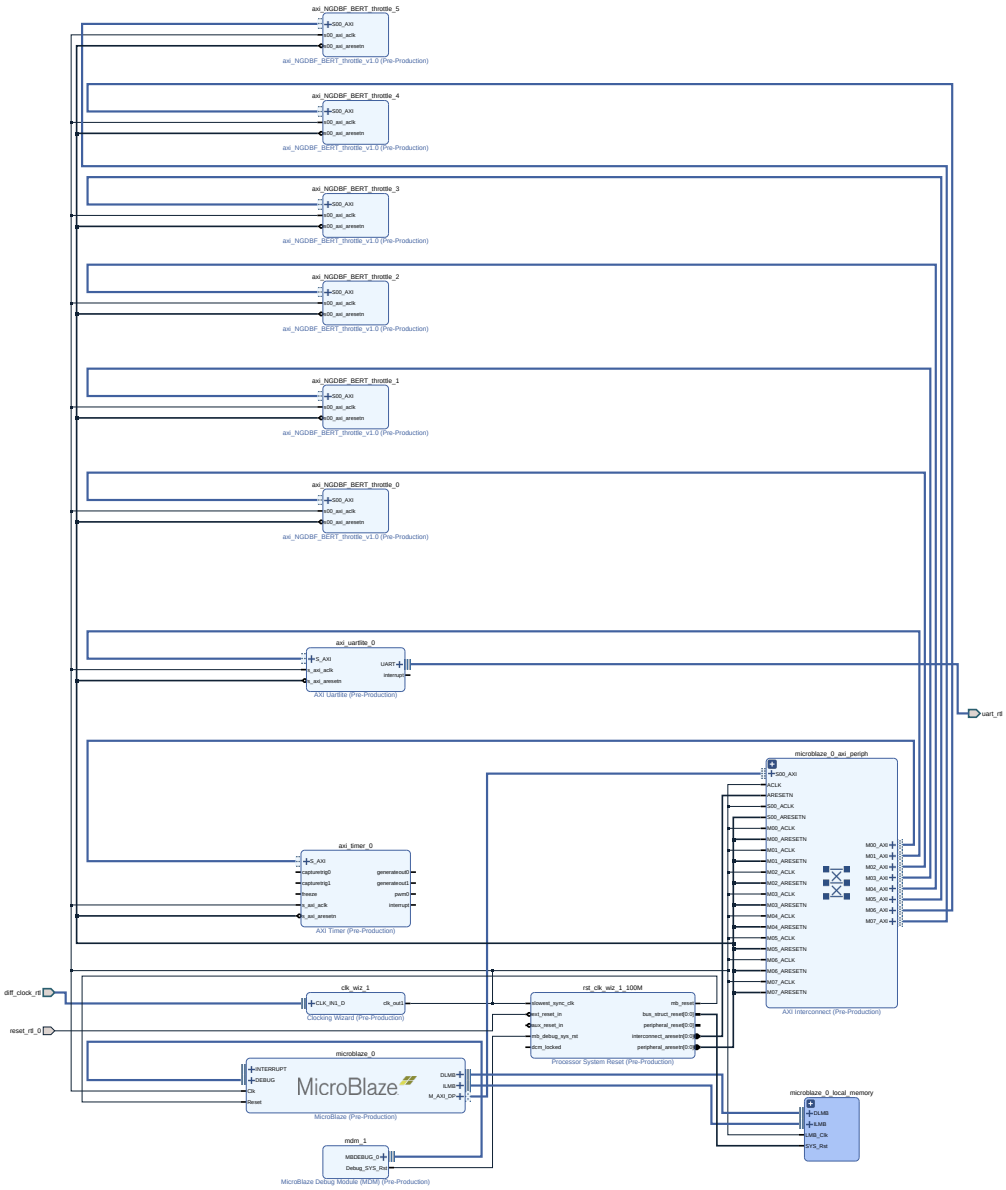

Fig. 3.7: Block Diagram for BERT design with Microblaze Embedded System

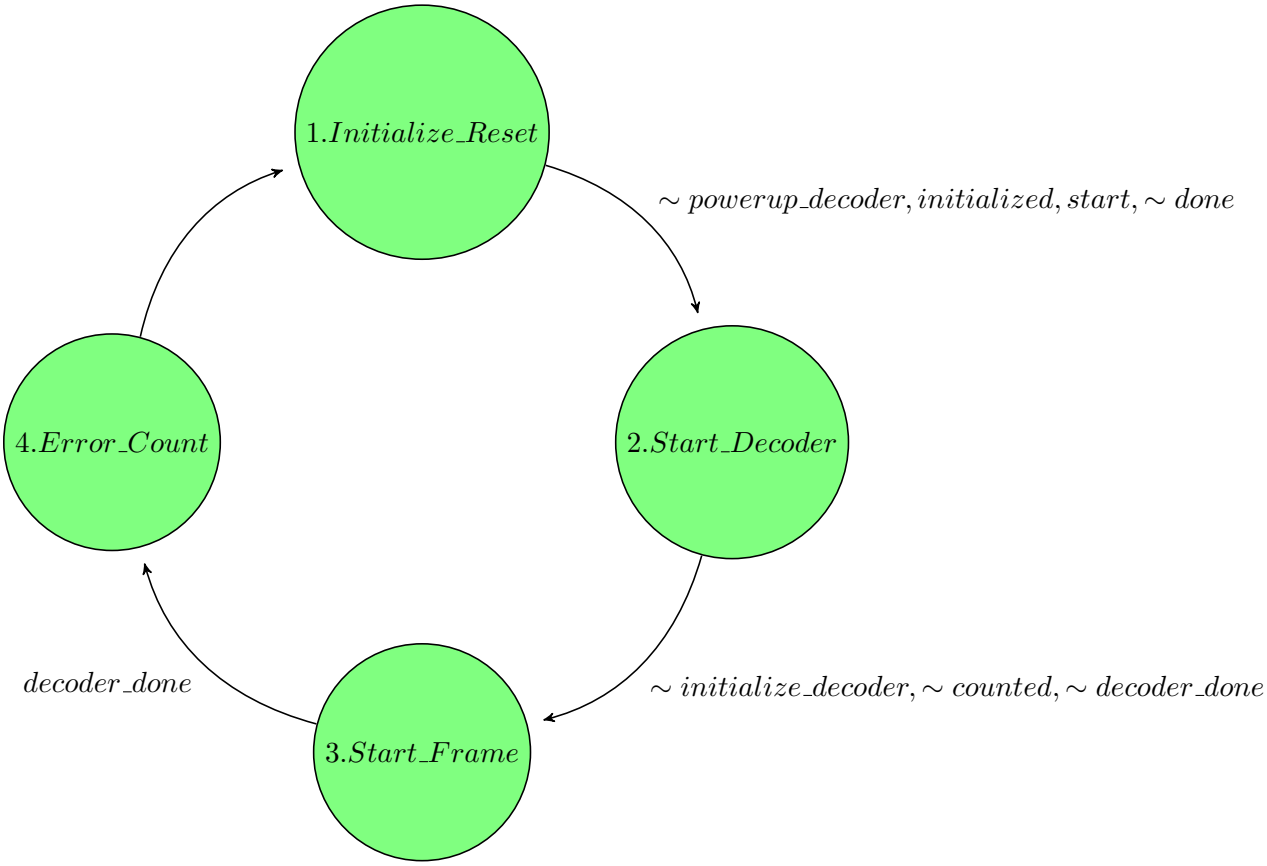

Fig. 3.8: State machine of the BERT module

The first state performs reset and then initializes all the parameters before the start of decoding. When the handshaking signals `powerup_decoder` & `initialized` are asserted low and high respectively- indicating the initialization is complete, depending on the high value of `start` and a low value of `done`, the system transitions to the next state. Then the NGDBF decoder gets started in the second state. A zero value of `initialize_decoder` indicates that the decoder is ready to start decoding and a low value the flag signal `counted` (indicating that next frame is yet to be decoded) & `decoder_done` (signaling that the state of decode operation has not changed) at the same time takes the system to the following state. The third state depicts the beginning of the decode operation on an incoming frame of samples. Once a frame is decoded, `decoder_done` is set high and a transition to the fourth state takes place. The following state counts the total number of erroneous bits and

frames and reports the error count. Then the system goes back to the initial state. After that, the state transitions keep repeating to decode next incoming frames until the total number of frames is reached.

CHAPTER 4

RESULTS & ANALYSIS

This chapter starts with the outcomes obtained from Shapiro-Wilk & Kolmogorov-Smirnov tests. In the second section. the acquired sets of data from the simulation are used to draw some Q-Q plots. Then the results of the reference MATLAB models are explained in detail in the following section. After that, the focus shifts towards the pictorial representation of the histogram data from testbench simulation and hardware implementation. Finally, there are some insights on the significance of these results on the performance of NGDBF decoder in the error floor region and an investigation of the error floor results both with the original and modified BERT design.

4.1 Results from Statistical Tests

The extended Shapiro-Wilk Test results are valid only up to 5000 test samples. However, even with this relatively small sample size, this should be a good starting point for the comparison against a standard Gaussian distribution. The following results are obtained at a significance level of 5%.

Output Parameters			
MATLAB		R	
Test Statistic, W	P-value	Test Statistic, W	P-value
0.9996	0.3075	0.99955	0.3075

Table 4.1: Sapiro-Wilk Test results on MATLAB & R platform

As shown in Tab. 4.1, Sapiro-Wilk test statistic, W is very close to 1 and p-value is greater than 0.05 which suggests that the pdf of channel emulator is extremely similar to that of a Gaussian Normal. It strongly indicates that the distribution from which the

samples are drawn is indeed a Normal distribution. But the limited number of samples implies that these results are far from conclusive.

Kolmogorov-Smirnov test is usually valid for a representation of the distribution with much larger sample size. The test statistic, D_n is a good representative of how much the test data deviates from the null hypothesis. In this case, the hypothesis under test is compared against a randomly generated Gaussian data set with the mean and variance of original data.

Number of Samples	Output Parameters			
	MATLAB		R	
	Test Statistic, D_n	Accept/Reject Null Hypothesis	Test Statistic, D_n	Accept/Reject Null Hypothesis
5k	0.0140	Accept	0.0104	Accept
10k	0.0195	Accept	0.0116	Accept
20k	0.0064	Accept	0.0092	Accept
50k	0.0042	Accept	0.00468	Accept
100k	0.0031	Accept	0.00352	Accept

Table 4.2: Kolmogorov-Smirnov Test results on MATLAB & R platform

In Tab. 4.2, As the number of samples goes up, the value of D_n approaches zero. It implies more and more confidence in the acceptance of the null hypothesis with larger test vectors. However, these tests alone are not enough to infer if the samples come from a normal distribution. Some visual representation like Q-Q plot might help in this case. The following section is based on empirical Q-Q plot analysis on the samples used in the aforementioned tests.

4.2 Q-Q Plots for Statistical Analysis

A Q-Q plot compares a pdf with another distribution by plotting quantiles where the coordinates of the points on the plot correspond to the same quantiles from each distribution [24]. In this case, the quantiles from the test samples are compared against a set of normally distributed data generated using the mean and covariance of those samples. In theory, this type of plot should be linear apart from some stochastic fluctuations of the data in reference distribution [25]. Any significant deviation from the linear pattern would imply that the data being tested do not follow the Normal distribution [25]. If the samples being tested and the null hypothesis both have the same distribution, then the points of Q-Q plot should approximately be on a straight line that goes towards the origin.

Fig. 4.1 depicts a Q-Q plot for 100k test samples. As shown in Fig. 4.1, the outcome is almost linear with some small outliers towards both ends of the range, suggesting minor deviations in the extreme parts of the tail. This phenomenon might be a case of not having enough samples in the tail regions. The deviations are less pronounced with larger sample counts. These results provide sound evidence in support of the argument that the distributions are similar in nature. However, as one of the main objectives of this work is to focus on the quality of distribution on extreme tails, the test set obtained from the simulation is not going to be adequate for full analysis. There needs to be additional testing especially with hardware generated, extremely large test vectors to put emphasis on the tail region. The next section focuses specifically on the MATLAB analysis of quantized data samples.

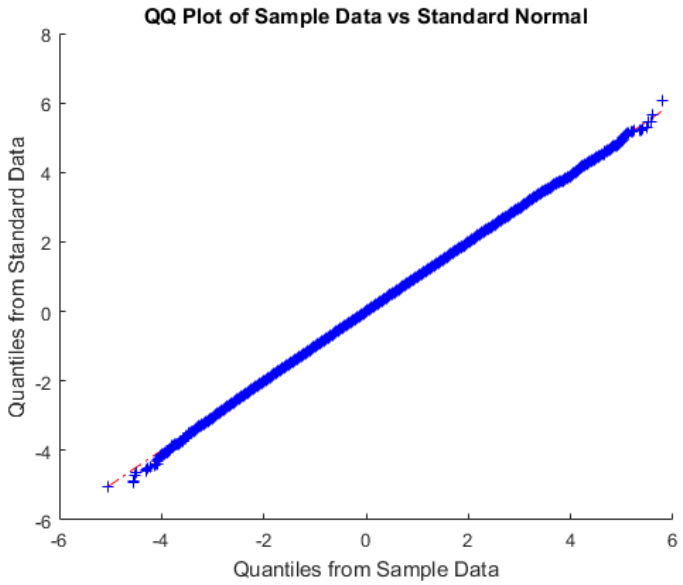

Fig. 4.1: Q-Q plot for 100k samples

4.3 Results from MATLAB Models

The first model takes in non-quantized samples generated by the channel emulator from the testbench simulation and performs 5-bit quantization to produce results in sign-magnitude format. Then it generates a histogram based on the quantized output depicting the number of samples in each of the 32 threshold levels. The second model imports the test samples in floating-point format ranging in between 0 to 1 from the RTL simulation and generates a set of random Gaussian distributed samples using the statistical information of the test vector. This data set then goes through a similar 5-bit quantization process to generate another histogram. In both cases, the testbench simulation is done with an SNR value of 4 decibels (dB) and an initial seed of 123. The SNR is calculated as the ratio of energy per bit (E_b) and noise (N_o), E_b/N_o.

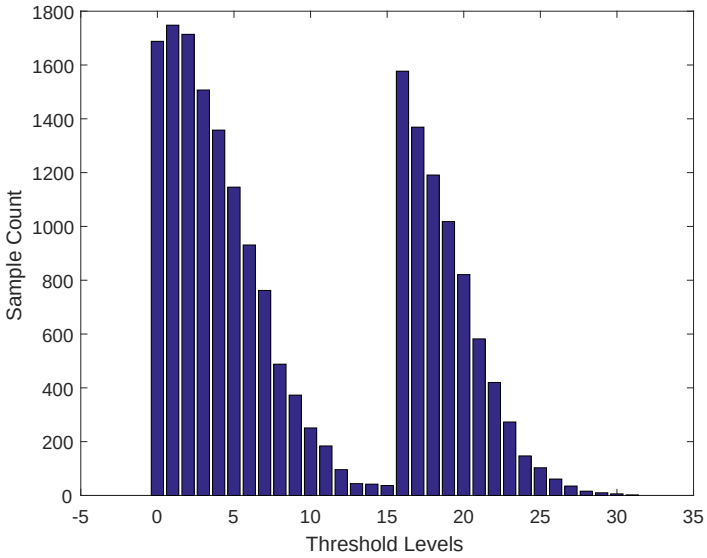

Fig. 4.2: Histogram plot generated by MATLAB model 1

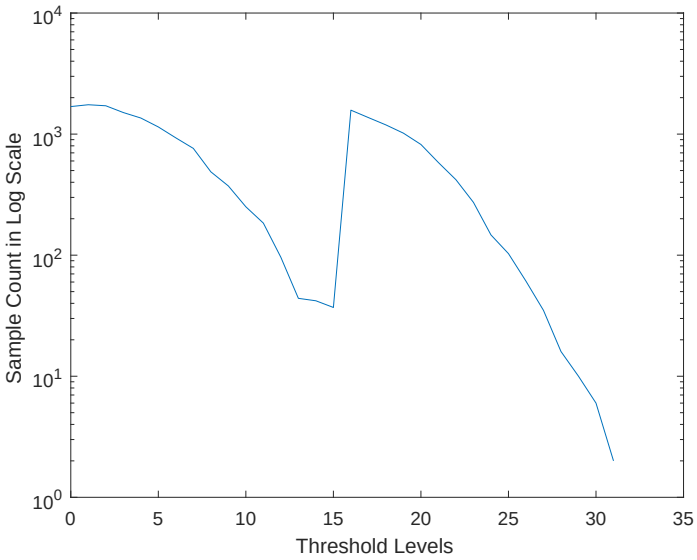

Fig. 4.3: Semilog plot generated by MATLAB model 1

The comparative study of Fig. 4.2 & 4.4 reveals that both the models generate very similar outputs. Fig. 4.4 shows a standard Gaussian distribution in a sign-magnitude format where the first 16 threshold levels display positive half of the distribution starting from the peak to the positive tail whereas the threshold levels 17-32 represent the negative side of the distribution. The histogram from model 1 displays the same format. This implies that the quantized channel emulator samples from the first model fall in line with the output from the second model. In other words, the channel emulator samples follow Gaussian distribution closely. In both the outputs, there is a very small number of samples on the tails due to the limited size of the data set. However, the pattern of these results provides a base for an expected outcome from the FPGA implementation. The pattern is also visible in semilog plots in figures 4.3 & 4.5.

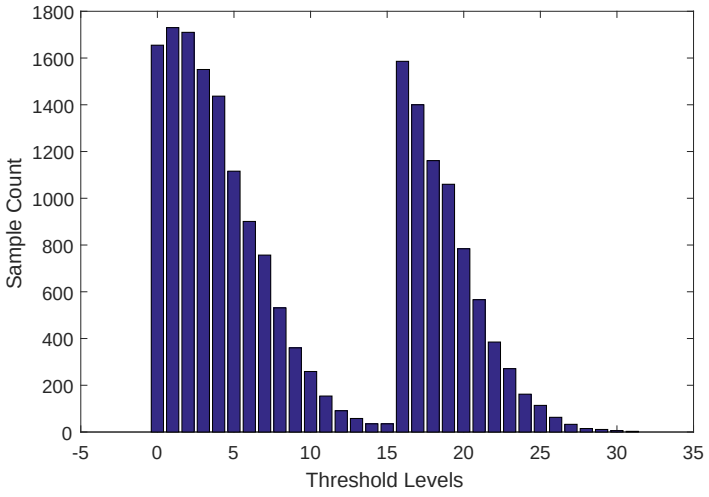

Fig. 4.4: Histogram plot generated by MATLAB model 2

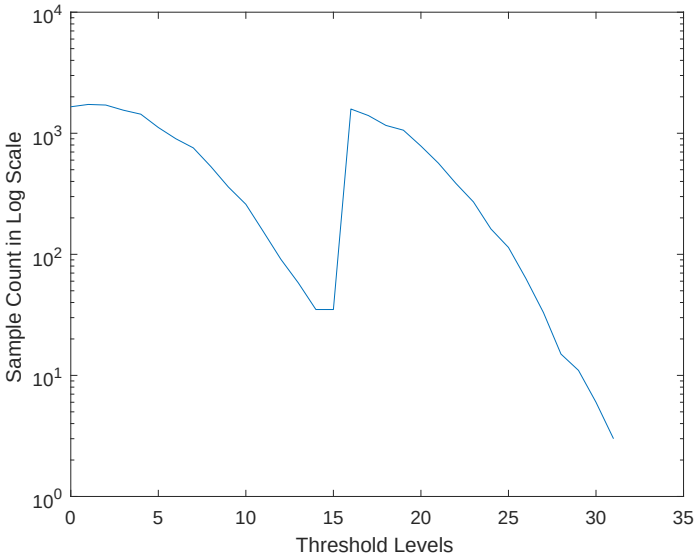

Fig. 4.5: Semilog plot generated by MATLAB model 2

4.4 Results from Testbench Simulation & Hardware Implementation

Two Verilog testbenches for the channel emulator generate histogram data for quantized samples. One of them instantiates top RTL design module, thereby simulating the entire design whereas the other one simulates a lower level design (the AWGN generator design module that generates non-quantized sample values using random number generation) and does the quantization separately. As similar techniques are used, both of the simulations should produce identical results. Comparison of these outputs with the results from reference MATLAB models should be an effective way to check the validity of the quantization process.

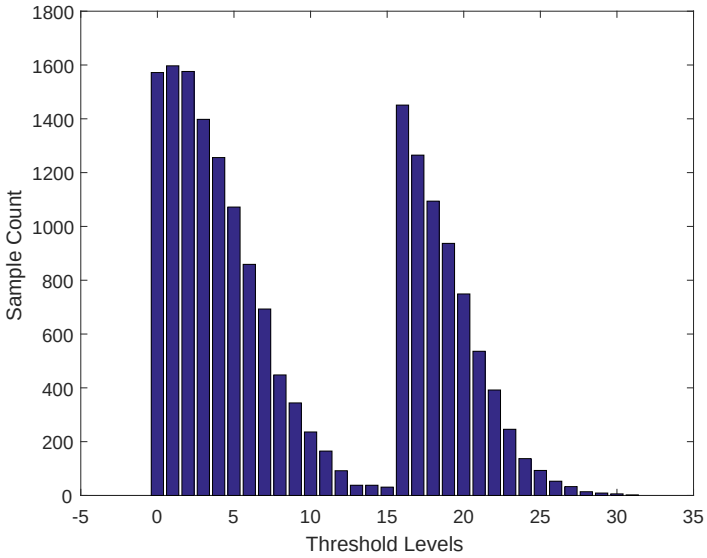

Fig. 4.6: Histogram plot generated by AWGN design simulation

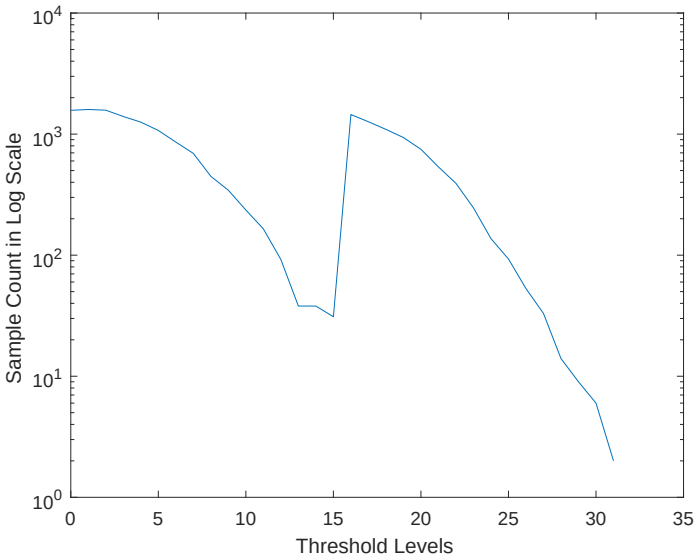

Fig. 4.7: Semilog plot generated by AWGN design simulation

Scrutiny on Fig. 4.6 & 4.8 reveals that both of them are very similar apart from the difference in the number of samples. The evidence from semilog plots of Fig. 4.7 & 4.9 also points to identical outputs. A plot pitting both of these testbench simulations against the MATLAB reference models are shown in Fig. 4.10. Fig. 4.10 confirms that all these results are also coherent with the outcomes of reference MATLAB models. Therefore, the simulation results are validated.

The results obtained from VC707 FPGA platform in figures 4.11 & 4.12 also fall in line with the same pattern as previous outputs. In this case, the number of samples in both the tail regions is very high and it increases the confidence on the outcome of these tests, even though the sample count in the peak is significantly larger for the samples to properly show up in the tail region of the histogram plot. The semilog plot in 4.12 reveals that there are more than thousands of samples even in the extreme tails. Similar to the comparisons with RTL simulation, Fig. 4.13 illustrates how the hardware results stack up against the MATLAB reference models. As apparent from Fig. 4.13, the FPGA outcomes are very similar to the reference model results. The takeaway from all these investigations is that the distribution of the channel emulator properly follows that of standard Gaussian even in the extreme parts of both the tails.

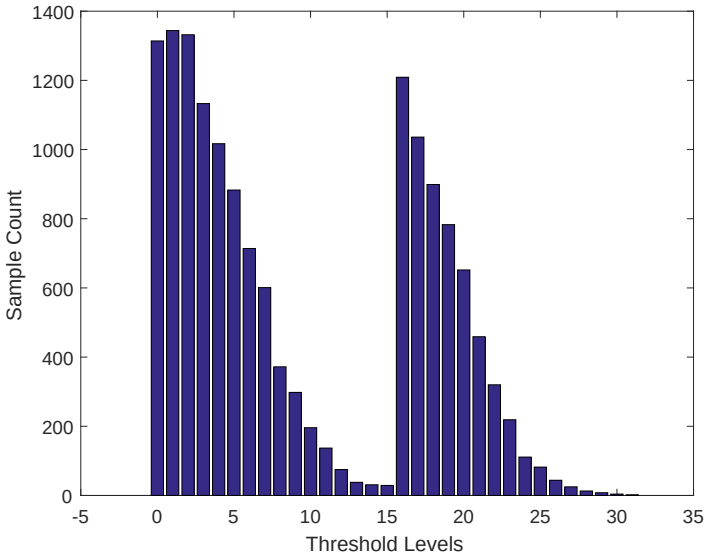

Fig. 4.8: Histogram plot generated by synthesizable RTL design simulation

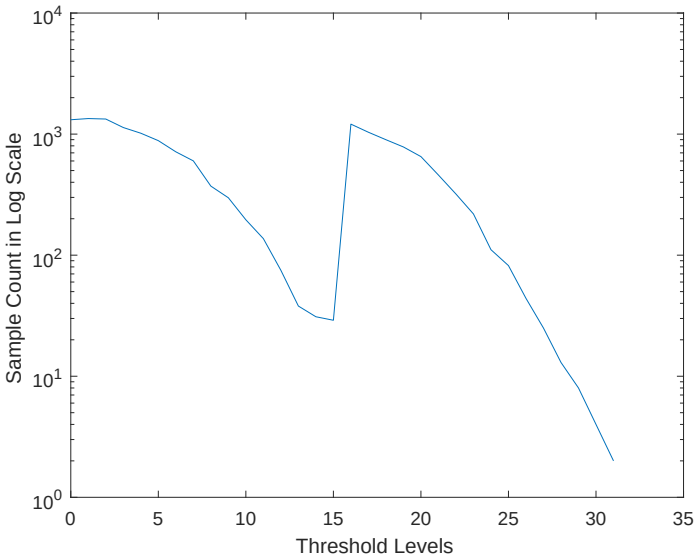

Fig. 4.9: Semilog plot generated by synthesizable RTL design simulation

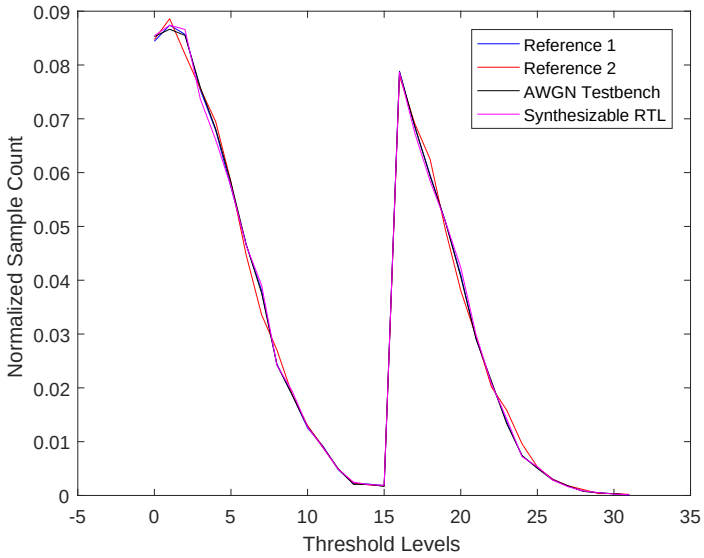

Fig. 4.10: Comparison of RTL simulation against MATLAB reference models

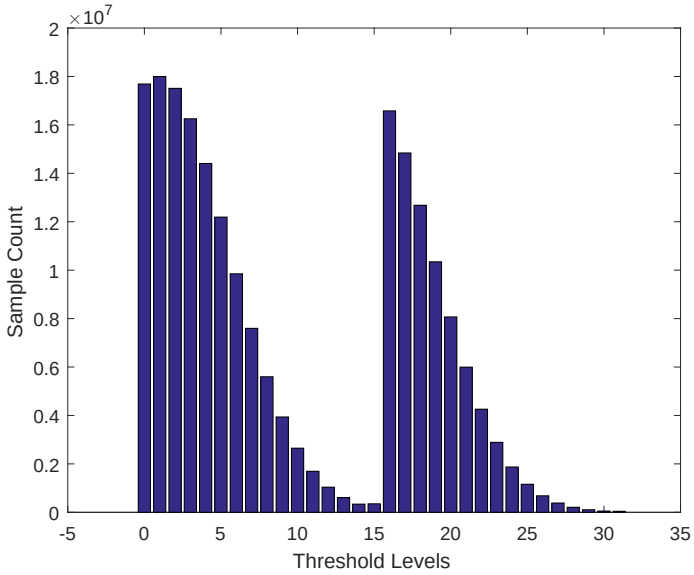

Fig. 4.11: Histogram plot generated from VC707 hardware platform

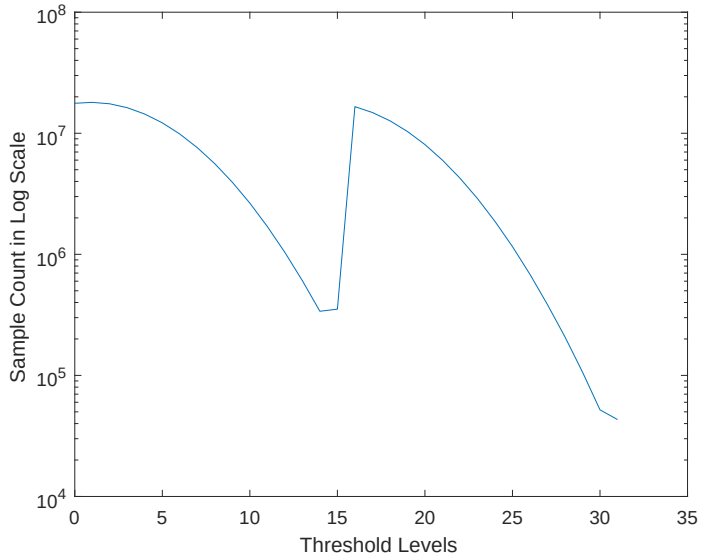

Fig. 4.12: Semilog plot generated from VC707 hardware platform

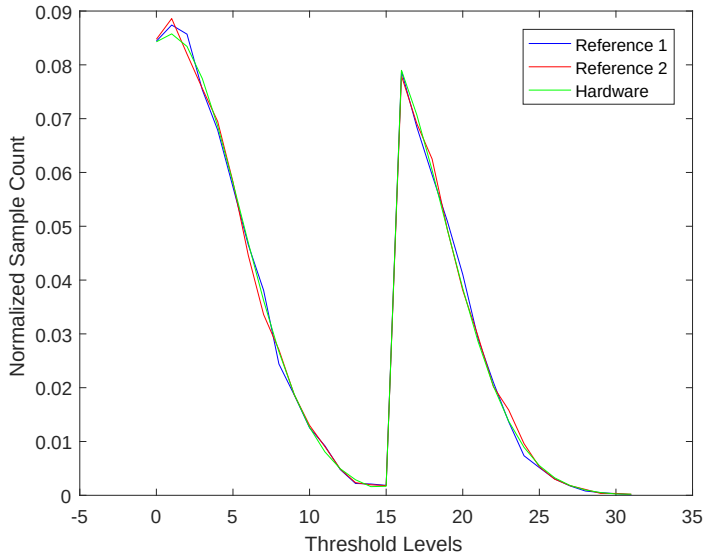

Fig. 4.13: Comparison of hardware results against MATLAB reference models

4.5 Checking the Dependency on Initial Seed Value

According to the steps of verifications proposed in chapter 3, the focus should now be shifted to find the dependencies of the emulator results on some internal parameters that are generally not supposed to have any influence on the outcome. Since in the past works, the NGDBF decoder operating on the error-floor region produced anomalous results with the change in seed values while using the channel emulator in concern, the next logical step would be to investigate if the initial seed values have any impact on these outcomes. In this case, the individual tests with all 256 different seeds generated identical results. This finding is helpful to conclude that the initial seed values don't have any influence on the distribution of the noise samples from channel emulator. The Four test cases presented in Fig. 4.14, 4.15, 4.16, 4.17, 4.18, 4.19, 4.20 & 4.21 provide evidence to this deduction. Two plots in Fig. 4.22 & 4.23 compare the histogram outputs obtained using all four seed values. As shown in Fig. 4.22, all four outcomes are so identical that there is virtually no difference. It is further confirmed with a zoomed-in view in Fig. 4.23.

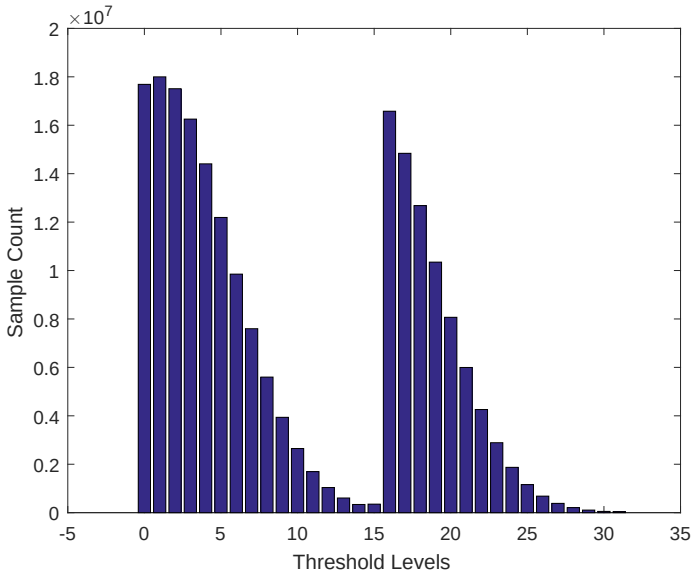

Fig. 4.14: Histogram plot for seed value 0

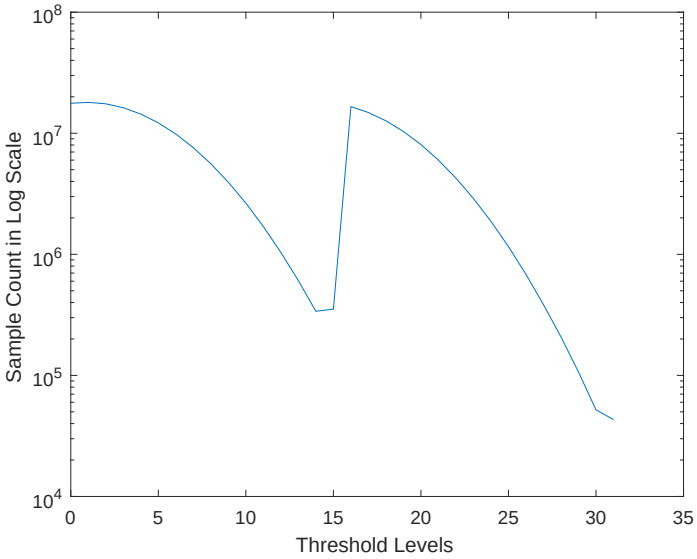

Fig. 4.15: Semilog plot for seed value 0

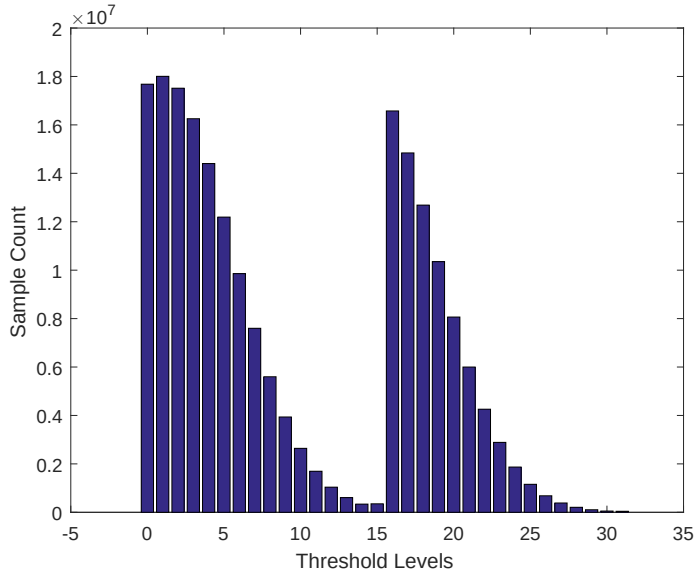

Fig. 4.16: Histogram plot for seed value 50

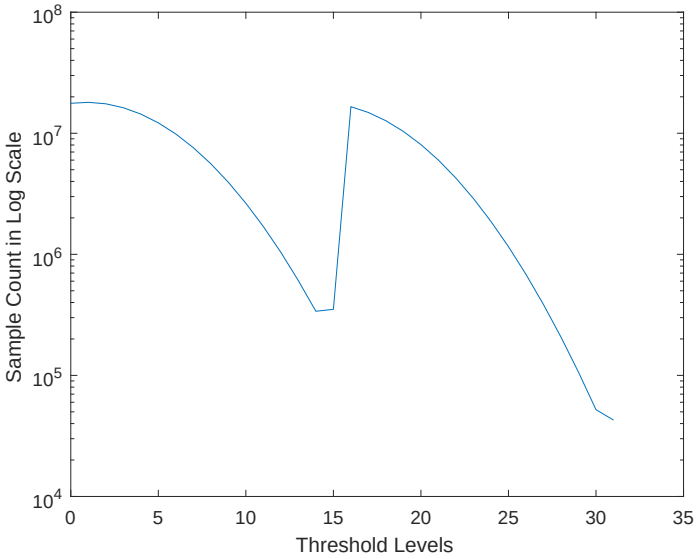

Fig. 4.17: Semilog plot for seed value 50

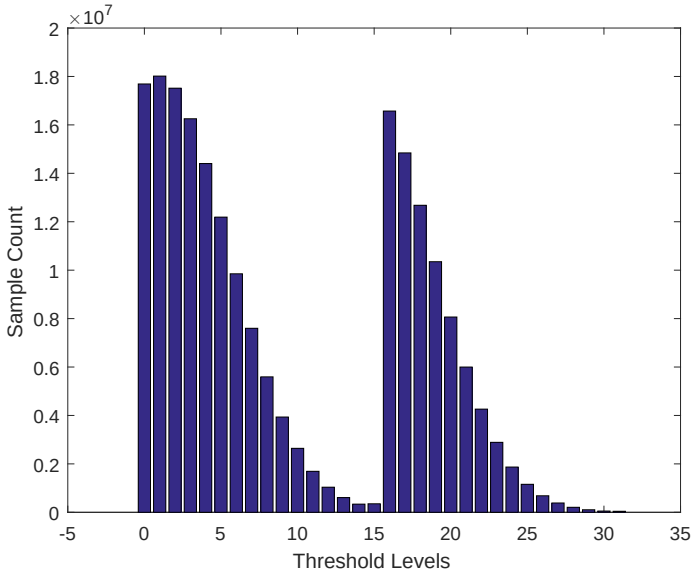

Fig. 4.18: Histogram plot for seed value 100

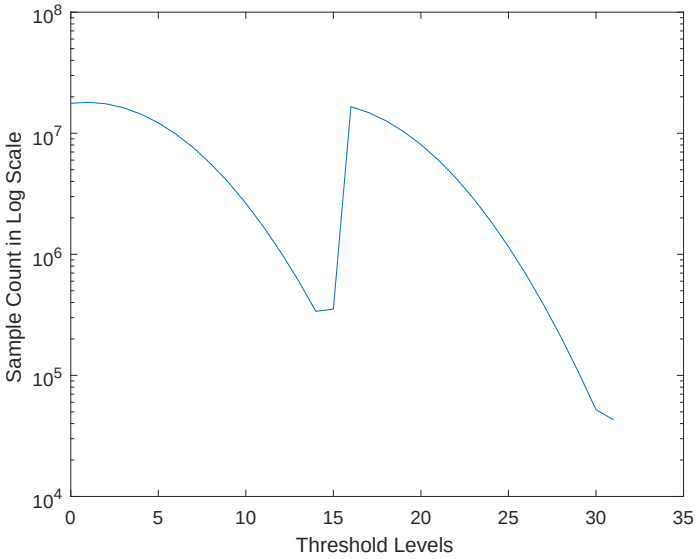

Fig. 4.19: Semilog plot for seed value 100

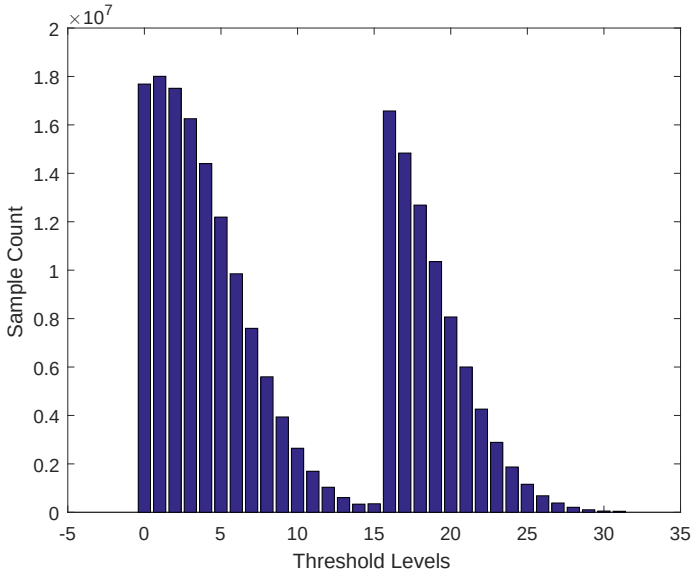

Fig. 4.20: Histogram plot for seed value 200

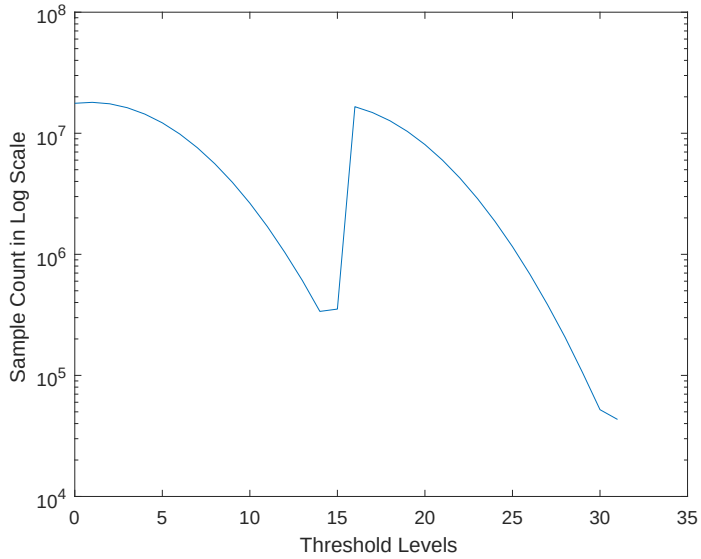

Fig. 4.21: Semilog plot for seed value 200

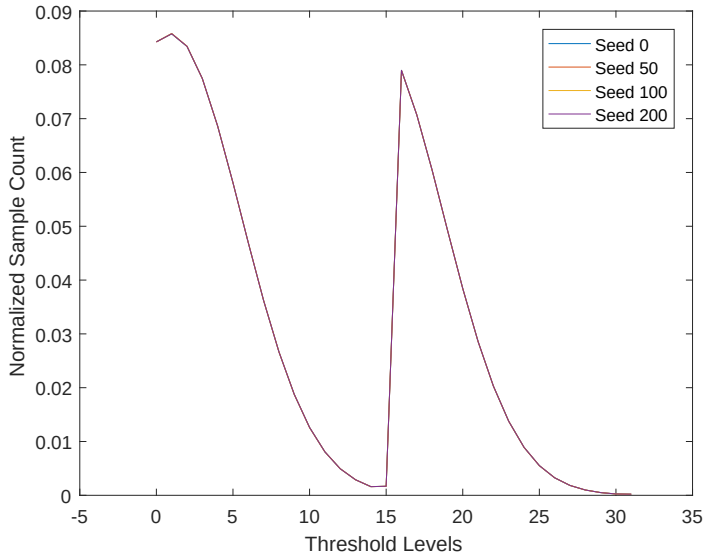

Fig. 4.22: Comparison of seed outputs

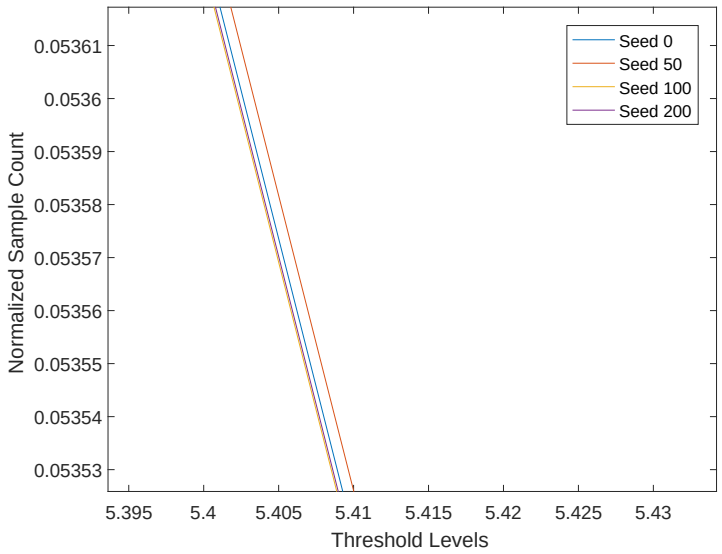

Fig. 4.23: A zoomed-in view for the seed comparisons

4.6 Verifying Reset Functionality

The next step to validate the channel emulator is to check if resetting the parameters to their original values and then changing them results in expected outputs. In other words, this section tests the effectiveness of the reset operation. This step is done with a change of initial seed in mind. For this purpose, the emulator is started with a random seed value. After some time, it is reset and restarted with another random seed value. The emulator then gets reset again after some further time delay and it now begins its operation again with the original seed. The seed values used for this test are 123 and 221 respectively, but it can be performed with any random seed. The idea here is to inspect whether the changes in the seed are getting recognized after every reset. A Q-Q plot for every operation until the next reset is used to infer if the reset is working in a proper manner. The Q-Q plots for this operation are produced by plotting the quantiles of respective output samples against the quantiles of a set of random samples generated using the statistical information of the data set generated by the use of the first random seed.

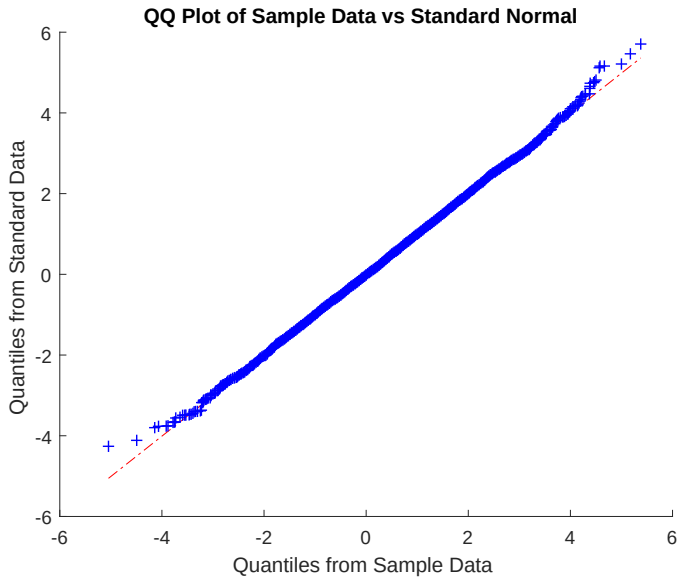

Fig. 4.24: Q-Q plot for seed 123

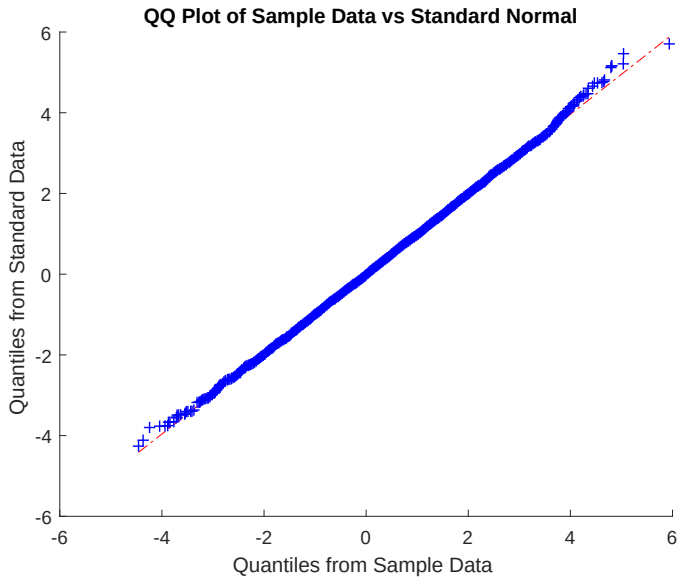

Fig. 4.25: Q-Q plot for seed 221 after first reset

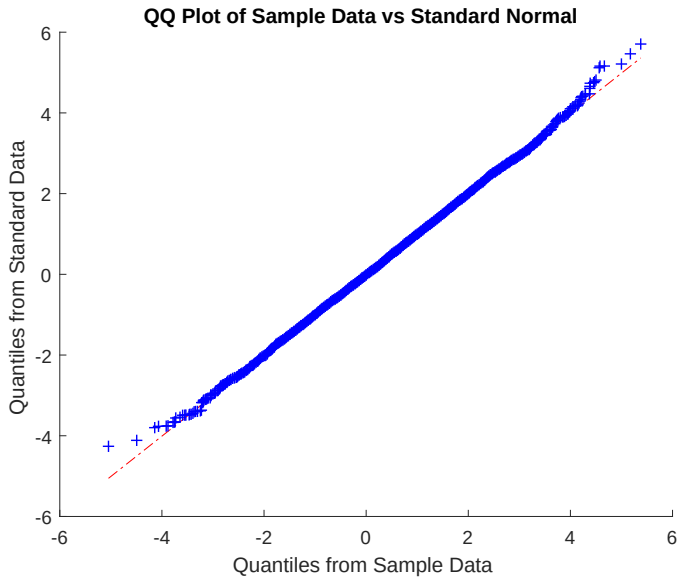

Fig. 4.26: Q-Q plot for seed 123 after second reset

As seen in Fig. 4.24 & 4.25, change in the overall position of the data points is evident after the first reset. This result strongly suggests that the reset operation is erasing all the previous statistics. The identical figures, Fig. 4.24 & 4.26 present further proof in favor of this deduction. These outcomes lead to the interpretation that the reset function in the RTL design of channel emulator is happening as intended. Therefore, these inferences lead to the conclusion that the AWGN channel emulator in concern is validated.

4.7 Analyzing the results of Decoder Implementation

As the channel emulator is validated, it is fair to deduce that the preceding erratic results observed with the NGDBF decoder derived from some faults in either the design of test platform or the algorithm of the decoder itself. To investigate the source of this issue, an attempt is made to reproduce the anomalous behavior in the results using BERT design at high SNR values approaching the error-floor region.

The terminology for the parameters used here are described below:

- `index_channel`: This represents the SNR value for channel emulator. The index value is 4 times the SNR. For example, an SNR value of 4 would result in `index_channel` value of $4*4 = 16$.
- `index_noise`: This parameter indicates a similar expression of SNR for random noise generator used in decoder design.
- `bitErrors`: It is indicative of the total number of bits with an error.
- `frameErrors`: It depicts the total number of failed frames with one or more erroneous bits.
- `numFrames`: This highlights the total number of frames to be decoded.
- `channelSeed`: This is the initial seed value for channel emulator.
- `noiseSeed`: This is the initial seed value for the noise generator used in NGDBF decoder. In this implementation, the same noise generation techniques from channel emulator are used to produce perturbation noise for decoder operation. Therefore, it's imperative to use different initial seeds for channel emulator and the additional perturbed noise generation for the decoder to reduce the correlation between them.

Tab. 4.3 highlights differences in `bitErrors` & `frameErrors` with the changes in `index_channel` & `index_noise` for the original BERT design. The test is operated with `index_channel` & `index_noise` values ranging from 17 to 20 & 18 to 21 respectively. These values typically correspond to high SNRs and they are selected specifically because previous works observed exceptions from standard outputs at those SNR levels. As the SNR values increase, the number of errors start to go down noticeably. The changes in `bitErrors` & `frameErrors` are most drastic at `index_channel` value of 17 with varying `index_noise` values. The random initial seeds are set to different values for channel and noise generator respectively. It is worth mentioning that all the results from onwards are taken from one particular run but similar outputs have been observed with multiple runs for each scenario. All of these test runs are conducted with one million frames unless stated otherwise.

index_channel	*index_noise*	*bitErrors*	*frameErrors*	*numFrames*	*channelSeed*	*noiseSeed*
17	18	568453	5875	1000000	180	120
17	19	31447	387	1000000	180	120
17	20	2740	54	1000000	180	120
17	21	892	28	1000000	180	120
18	18	10957	116	1000000	180	120
18	19	97	2	1000000	180	120
18	20	44	4	1000000	180	120
18	21	24	6	1000000	180	120
19	18	126	2	1000000	180	120
19	19	0	0	1000000	180	120
19	20	4	1	1000000	180	120
19	21	2	1	1000000	180	120
20	18	6	1	1000000	180	120
20	19	3	1	1000000	180	120
20	20	9	1	1000000	180	120
20	21	2	1	1000000	180	120

Table 4.3: Results for original BERT design at different SNRs & a channel seed value of 180 & a noise seed value of 120

Tab. 4.4 portrays an interesting picture. This test run is performed with a fixed `index_channel` value of 18 while varying the `index_noise` & seed values. The idea here is to observe how the error count changes with respect to slightly different seeds. Theoretically, the number of erroneous bits and frames should hover around some constant low values at error-floor. But most of the outcomes of this run don't conform to this hypothesis. As before, the initial seeds for channel and noise generator start at 180 & 120 respectively. Each BERT module increases the seed values by 1, therefore the sixth BERT block operates with seed values 185 & 125. The results for `index_channel` 19 & 20 are usually consistent, but `bitErrors` & `frameErrors` produce significantly distinct results with different values of `channelSeed` & `noiseSeed` at `index_channel` value of 17. This trend is also visible at `index_channel` 18.

As the variations in errors seem to be related to specific seed values, it is important to observe which of the two seeds - `channelSeed` & `noiseSeed`- have the most impact on the outputs. Tab. 4.5 & 4.6 show a reproduction of the test depicted in table 4.4, but with different initial `noiseSeed` & `channelSeed` values respectively in the first and second case. The inference from these runs is obvious. The error results from the aforementioned tables show that the deviations are largely caused by changes in `noiseSeed` while the change in `channelSeed` value has minimal effect on the output. As shown in Fig. 4.27, 4.29, 4.31 & 4.33, there is a very wide range of deviations in error counts while varying the `noiseSeed` with a fixed `channelSeed` value but this variation is negligibly small when sweeping through all possible values of `channelSeed` with a fixed `noiseSeed`. These phenomena are more pronounced in the semilog plots of Fig. 4.28, 4.30, 4.32 & 4.34. As the same noise generator from channel emulator is used to produce random perturbation noise for decoder and the channel emulator is already validated, this outcome suggests that there needs to be further inspection on `noiseSeed` values.

A run of rigorous tests with different `noiseSeed` values reveals that some of the seeds reduce `bitErrors` & `frameErrors` drastically. These seeds generate significantly lower number of bit & frame errors compared to the other one. Out of all such values, 47, 123 and 241 in particular are chosen for the following analyses as they are found to be producing very low error counts, but it is worth mentioning that there are other seed values that generate results close to these seeds. The effect of these seeds with the change in index values for channel and noise is shown in Fig. 4.35, 4.36, 4.37, 4.38, 4.39, 4.40, 4.41, 4.42, 4.43, 4.44, 4.45 & 4.46. Since the outliers were mainly evident on `index_channel` value of 17 & 18, the aforementioned analyses are done at those index levels. The `bitErrors` counts are from test runs of 1 million frames whereas the `frameErrors` counts are from test runs of 10 million frames.

index_channel	*index_noise*	*bitErrors*	*frameErrors*	*numFrames*	*channelSeed*	*noiseSeed*
17	18	568453	5875	1000000	180	120
17	18	144158	1680	1000000	181	121
17	18	236282	2608	1000000	182	122
17	18	1750	42	1000000	183	123
17	18	21792	272	1000000	184	124
17	18	306670	3464	1000000	185	125
18	18	10957	116	1000000	180	120
18	18	2322	34	1000000	181	121
18	18	4825	56	1000000	182	122
18	18	0	0	1000000	183	123
18	18	53	1	1000000	184	124
18	18	5337	65	1000000	185	125
19	18	126	2	1000000	180	120
19	18	4	2	1000000	181	121
19	18	8	2	1000000	182	122
19	18	0	0	1000000	183	123
19	18	0	0	1000000	184	124
19	18	0	0	1000000	185	125
20	18	6	1	1000000	180	120
20	18	2	1	1000000	181	121
20	18	0	0	1000000	182	122
20	18	2	1	1000000	183	123
20	18	0	0	1000000	184	124
20	18	0	0	1000000	185	125

Table 4.4: Results for original BERT design at constant noise SNR & different channel SNR with varying seed values

$index\_channel$	$index\_noise$	$bitErrors$	$frameErrors$	$numFrames$	$channelSeed$	$noiseSeed$
17	18	48812	565	1000000	180	70
17	18	568566	5922	1000000	181	71
17	18	70885	761	1000000	182	72
17	18	18121	261	1000000	183	73
17	18	1201430	12648	1000000	184	74
17	18	684904	7052	1000000	185	75
18	18	444	5	1000000	180	70
18	18	11820	125	1000000	181	71
18	18	918	11	1000000	182	72
18	18	0	0	1000000	183	73
18	18	34699	392	1000000	184	74
18	18	15409	370	1000000	185	75
19	18	0	0	1000000	180	70
19	18	106	1	1000000	181	71
19	18	0	0	1000000	182	72
19	18	0	0	1000000	183	73
19	18	336	5	1000000	184	74
19	18	111	2	1000000	185	75
20	18	2	1	1000000	180	70
20	18	0	0	1000000	181	71
20	18	0	0	1000000	182	72
20	18	2	1	1000000	183	73
20	18	3	1	1000000	184	74
20	18	3	1	1000000	185	75

Table 4.5: Results for original BERT design at constant noise SNR & same initial channel seed with different channel SNR and noise seed values

index_channel	*index_noise*	*bitErrors*	*frameErrors*	*numFrames*	*channelSeed*	*noiseSeed*
17	18	558005	5735	1000000	70	120
17	18	148408	1754	1000000	71	121
17	18	232697	2547	1000000	72	122
17	18	1800	40	1000000	73	123
17	18	20346	256	1000000	74	124
17	18	303001	3440	1000000	75	125
18	18	13843	144	1000000	70	120
18	18	2223	26	1000000	71	121
18	18	4952	55	1000000	72	122
18	18	118	2	1000000	73	123
18	18	0	0	1000000	74	124
18	18	4596	59	1000000	75	125
19	18	232	3	1000000	70	120
19	18	5	2	1000000	71	121
19	18	0	0	1000000	72	122
19	18	0	0	1000000	73	123
19	18	0	0	1000000	74	124
19	18	0	0	1000000	75	125
20	18	0	0	1000000	70	120
20	18	3	1	1000000	71	121
20	18	0	0	1000000	72	122
20	18	0	0	1000000	73	123
20	18	0	0	1000000	74	124
20	18	0	0	1000000	75	125

Table 4.6: Results for original BERT design at constant noise SNR & same initial noise seed with different channel SNR and channel seed values

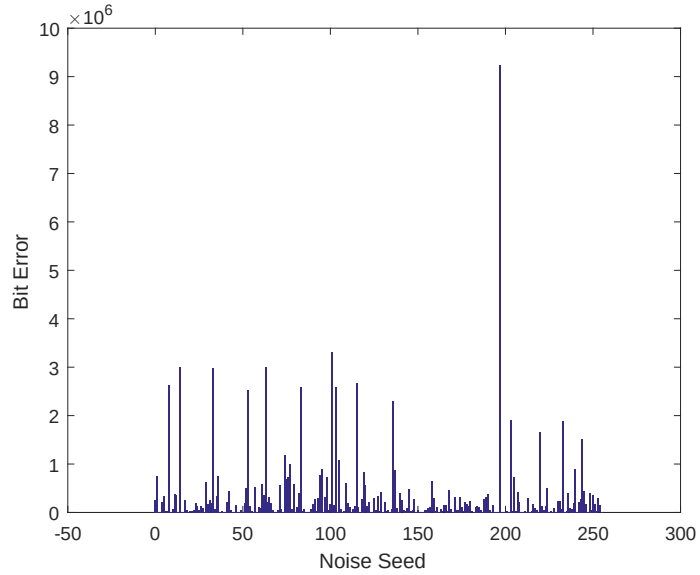

Fig. 4.27: Bit errors with all noise seeds at index value of 17 for channel and 18 for noise with a fixed channel seed value of 180

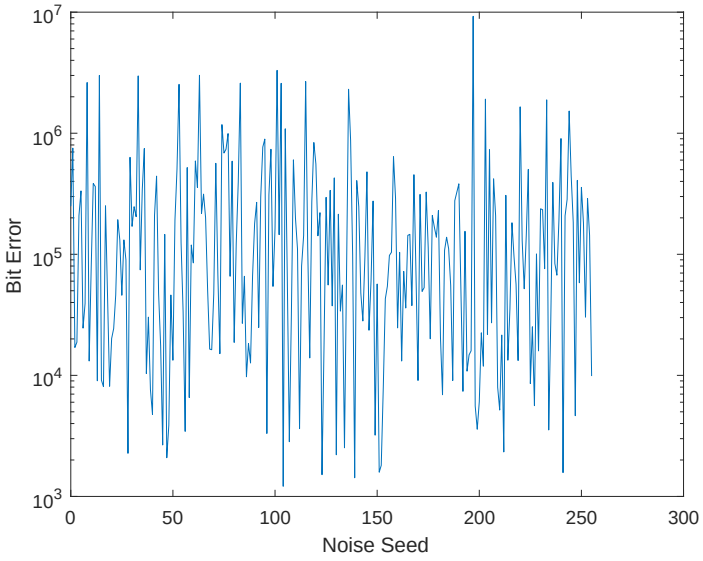

Fig. 4.28: Semilog plot for bit errors with all noise seeds at index value of 17 for channel and 18 for noise with a fixed channel seed value of 180

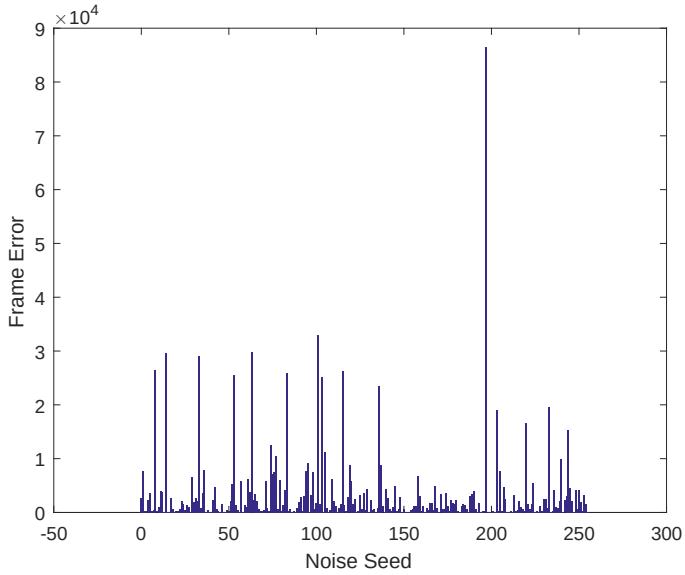

Fig. 4.29: Frame errors with all noise seeds at index value of 17 for channel and 18 for noise with a fixed channel seed value of 180

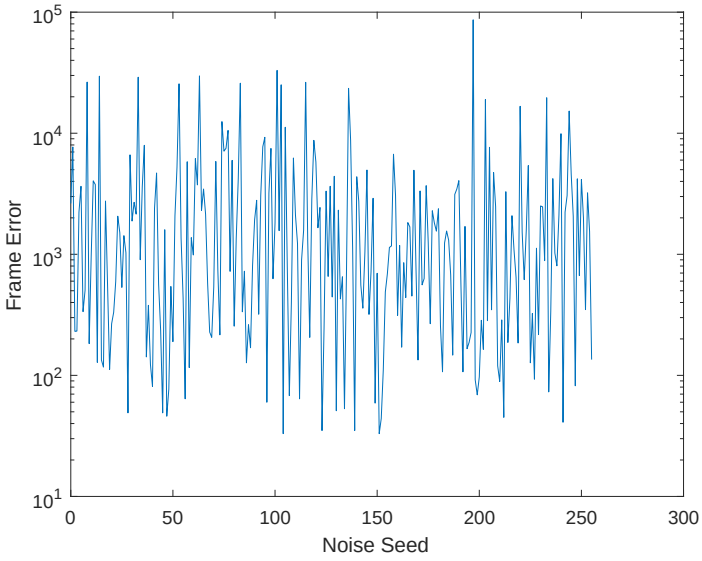

Fig. 4.30: Semilog plot for frame errors with all noise seeds at index value of 17 for channel and 18 for noise with a fixed channel seed value of 180

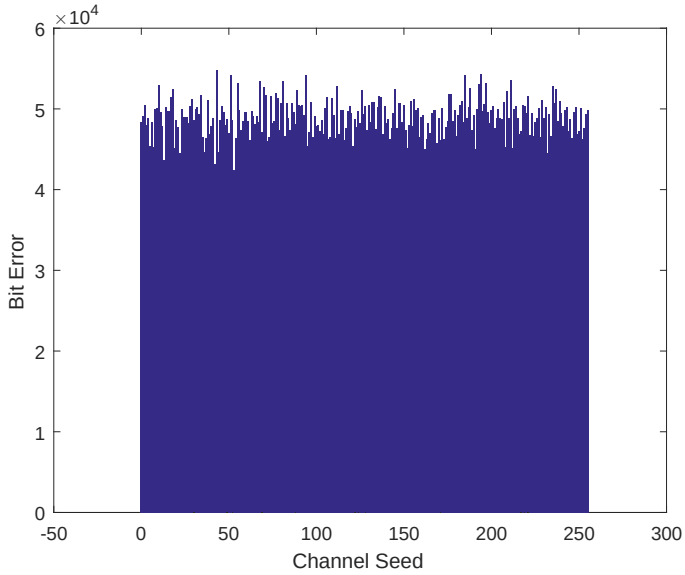

Fig. 4.31: Bit errors with all channel seeds at index value of 17 for channel and 18 for noise with a fixed noise seed value of 70

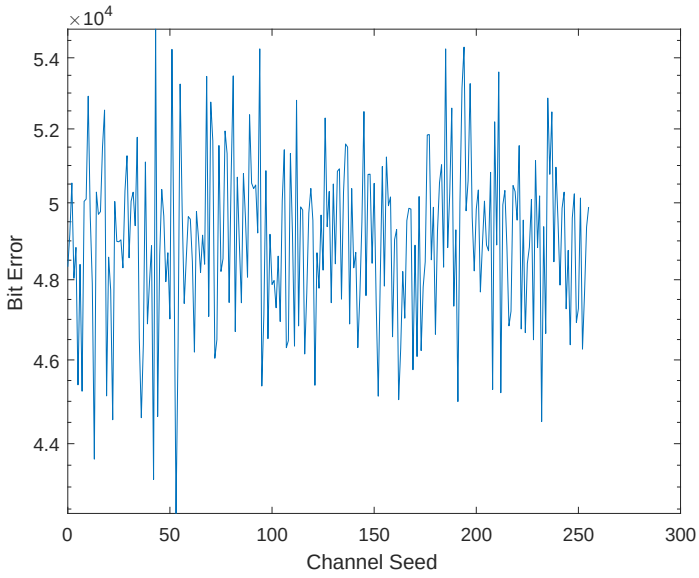

Fig. 4.32: Semilog plot for bit errors with all channel seeds at index value of 17 for channel and 18 for noise with a fixed noise seed value of 70

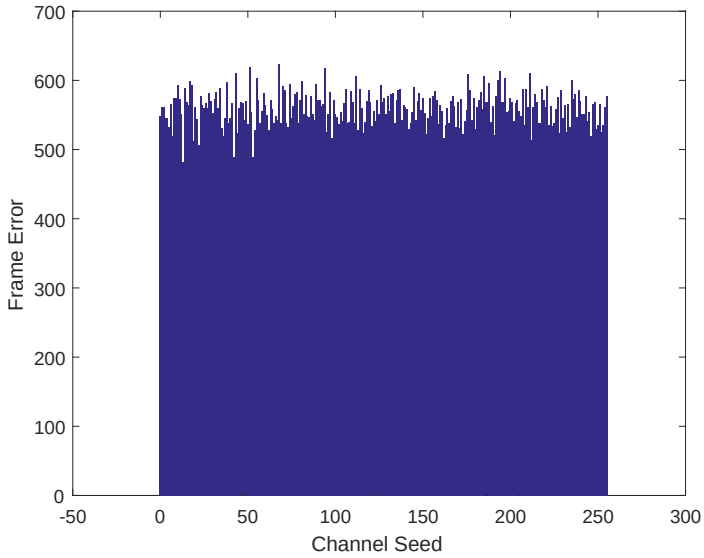

Fig. 4.33: Frame errors with all channel seeds at index value of 17 for channel and 18 for noise with a fixed noise seed value of 70

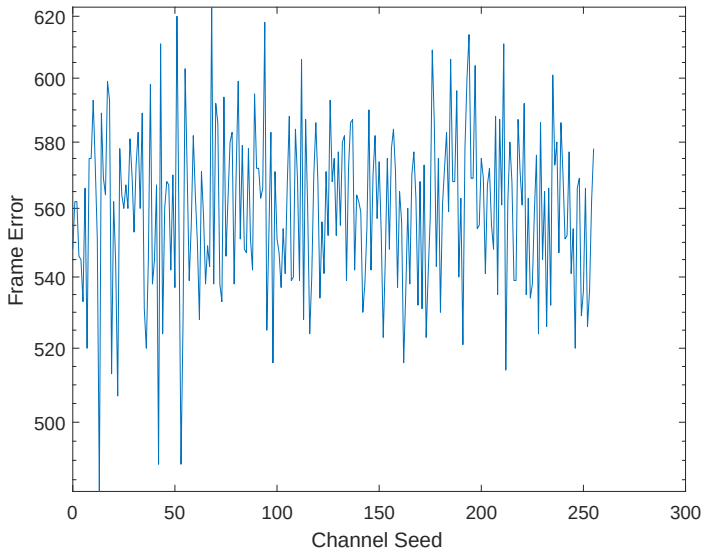

Fig. 4.34: Semilog plot for frame errors with all channel seeds at index value of 17 for channel and 18 for noise with a fixed noise seed value of 70

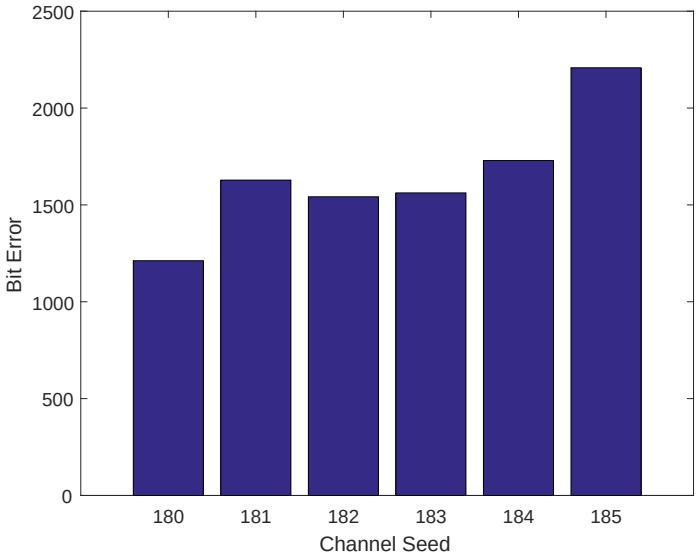

Fig. 4.35: Bit errors for varying channel seeds at index value of 17 for channel and 18 for noise with a fixed noise seed value of 47

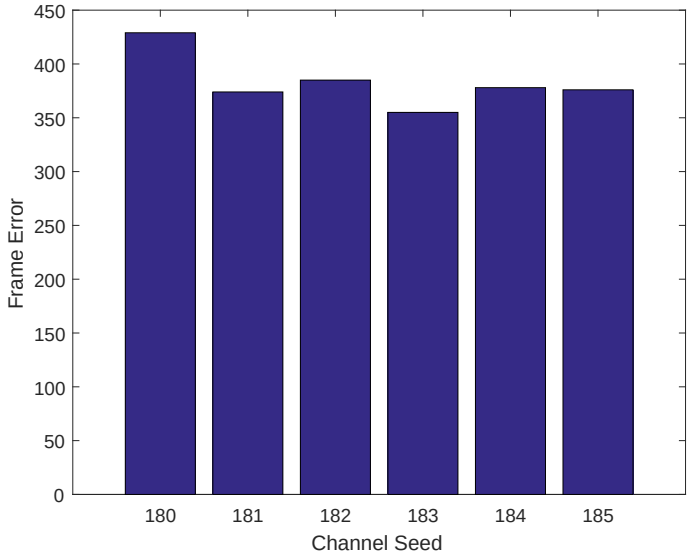

Fig. 4.36: Frame errors for varying channel seeds at index value of 17 for channel and 18 for noise with a fixed noise value seed of 47

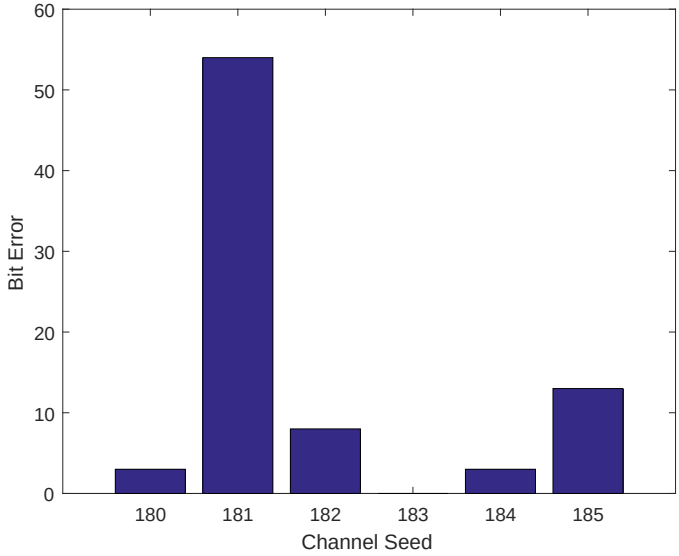

Fig. 4.37: Bit errors for varying channel seeds at index value of 18 for channel and 18 for noise with a fixed noise seed of 47

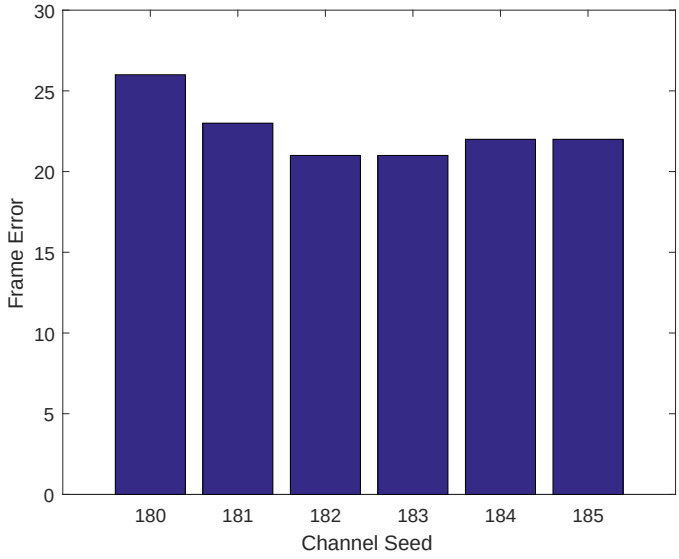

Fig. 4.38: Frame errors for varying channel seeds at index value of 18 for channel and 18 for noise with a fixed noise seed of 47

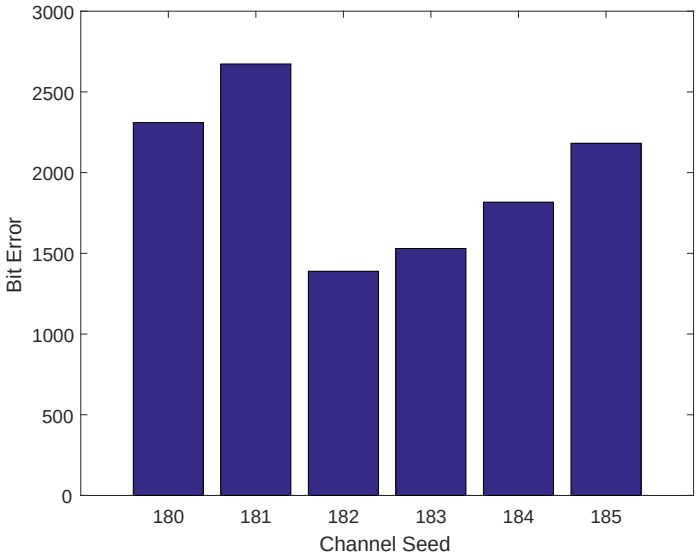

Fig. 4.39: Bit errors for varying channel seeds at index value of 17 for channel and 18 for noise with a fixed noise seed of 241

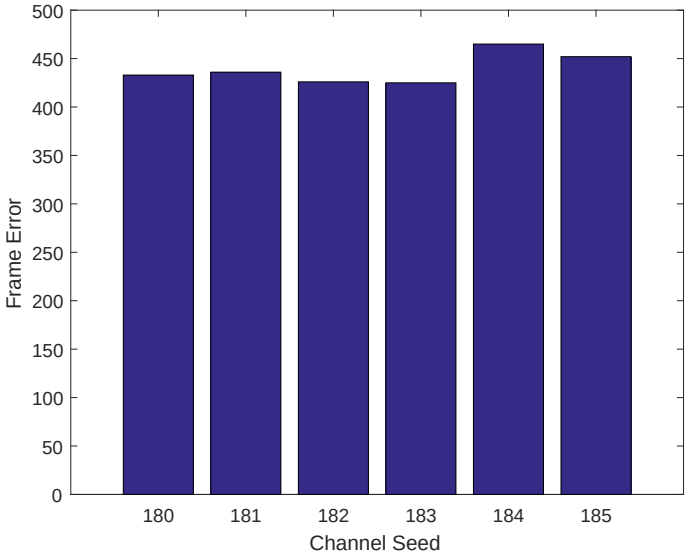

Fig. 4.40: Frame errors for varying channel seeds at index value of 17 for channel and 18 for noise with a fixed noise seed of 241

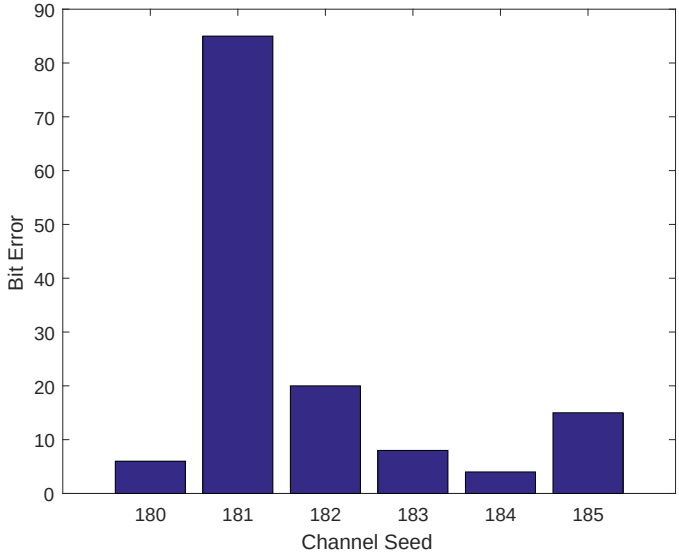

Fig. 4.41: Bit errors for varying channel seeds at index value of 18 for channel and 18 for noise with a fixed noise seed of 241

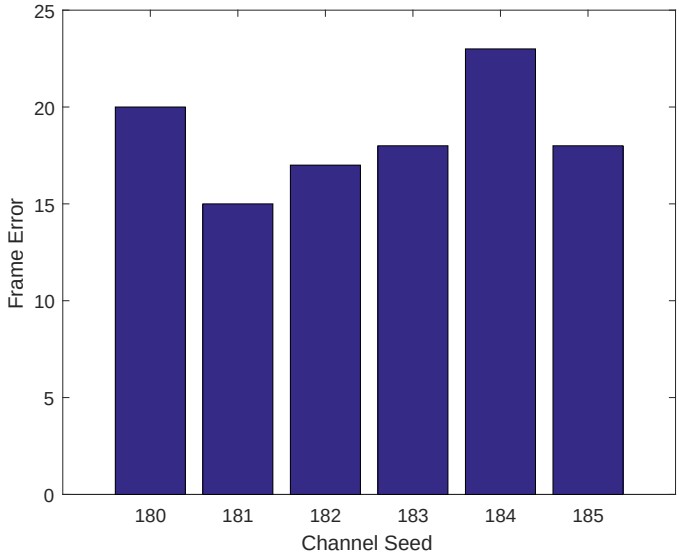

Fig. 4.42: Frame errors for varying channel seeds at index value of 18 for channel and 18 for noise with a fixed noise seed of 241

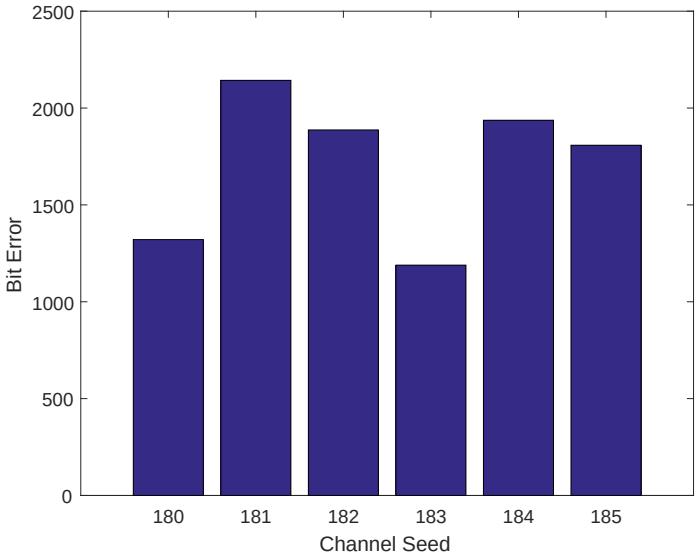

Fig. 4.43: Bit errors for varying channel seeds at index value of 17 for channel and 18 for noise with a fixed noise seed of 123

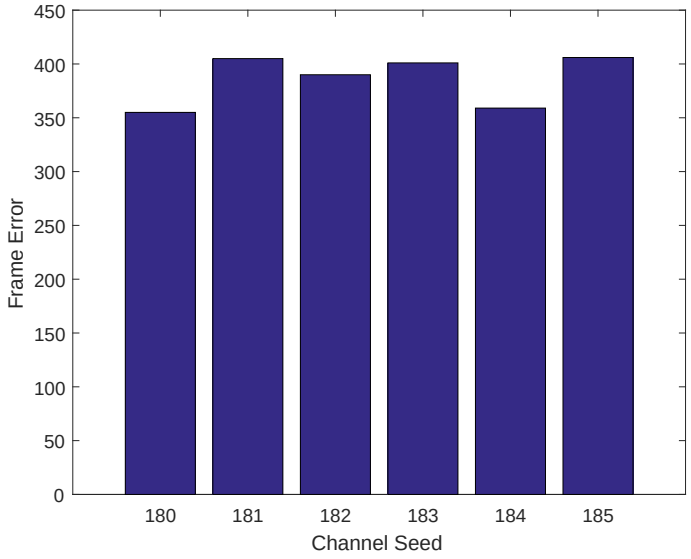

Fig. 4.44: Frame errors for varying channel seeds at index value of 17 for channel and 18 for noise with a fixed noise seed of 123

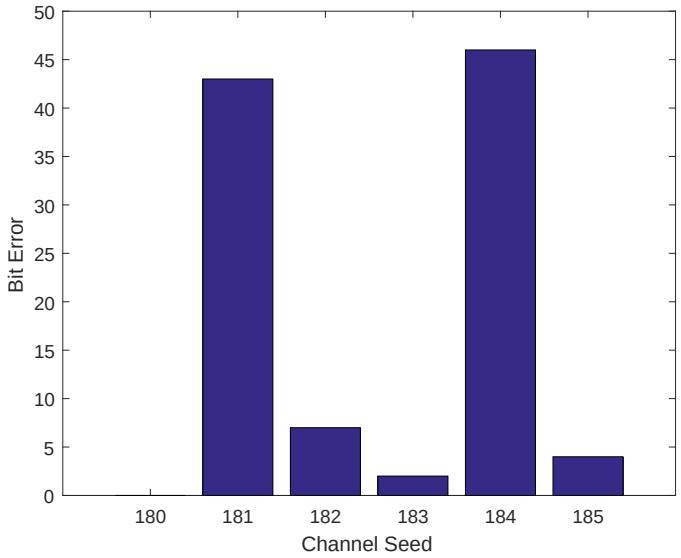

Fig. 4.45: Bit errors for varying channel seeds at index value of 18 for channel and 18 for noise with a fixed noise seed of 123

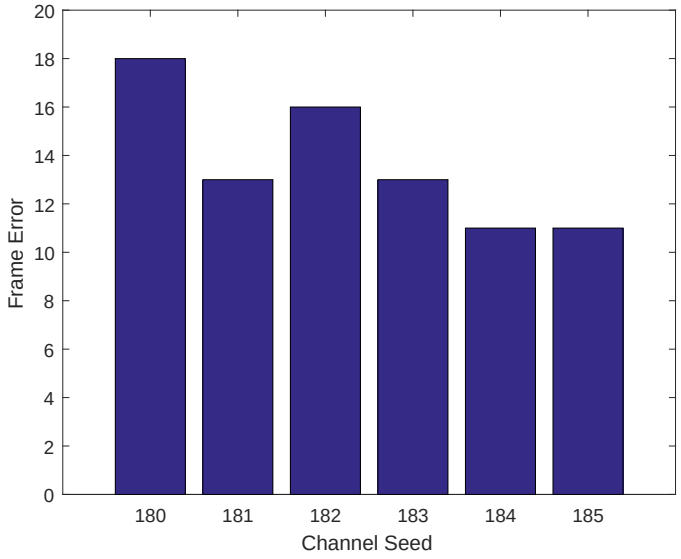

Fig. 4.46: Frame errors for varying channel seeds at index value of 18 for channel and 18 for noise with a fixed noise seed of 123

The results illustrate low values the `bitErrors` and `frameErrors` count with `index_channel` values of 17 & 18 while using `noiseSeed` values of 47, 241 & 123. As evident from these figures, the variations between the number of errors with different `channelSeed` values are usually small and hence they can be overlooked. Since there is no correlation between the initial seed and the performance of channel emulator and the `channelSeed` value does not have noticeable effects on the error counts of the decoder and the `noiseSeed` is strongly dictating the pattern of results, the anomalies might not be related to the design of the test platform. These outcomes strongly point that the outliers are caused by the sequence of the generated perturbation noise samples. The two `noiseSeed` values that are empirically found out to be the best suit for producing low error counts might just have a very good random sequence. Hence, integrating any of the aforementioned two `noiseSeed` values in the source design should get rid of the aberrations in error-floor analysis. Further testing and verifications might be required in the future to find out how the random noise sequences are actually influencing the decoder performance.

CHAPTER 5

CONCLUSIONS & FUTURE WORK

The work in concern emphasizes different statistical aspects of a hardware-based AWGN channel emulator to validate its functionality. This is crucial for the operation of an error-correcting decoder as a slight deviation of the emulator from the desired probability distribution might cause significant errors in the performance analysis. The channel emulator in question is validated following a pattern of general verification steps. First, the distribution of the emulator is verified by the use of a couple of statistical tests and Q-Q plots. Then two MATLAB reference models are designed and compared against testbench and hardware results to account for the samples in extreme tails. The reset operation and dependency on initial seeds are tested within the validation process. The final deduction from all these tests is that the emulator is verified to be functioning as expected.

This thesis mainly focuses on some anomalous behavior of the LDPC decoder observed previously during the analysis of error-floors with said channel emulator. Those outliers have prompted a thorough review of the emulator in the first place. Investigating the BERT implementation gives an obvious indication that the anomalies are primarily influenced by seed values of the perturbation noise generator for the decoder. This phenomenon might be linked to the particular pattern of the noise generated by that specific seed. Some of these seeds might have a better sequence of producing noise samples than the other ones, hence yielding lower error counts. Adding the perturbation noise is a design feature of the NGDBF decoder and changing the initial noise seed only changes the sequence of the noise samples generated by noise generator. Since the perturbed noise samples are not a part of the statistical tests to validate the hardware design of the test platform, choosing a constant noise seed value should not affect the statistical validity of the results. Therefore, the initial noise seeds that are empirically found to be causing better results might be integrated into the hardware design to improve the performance. Additional test runs are operated to

figure out the seed values that cause the lowest deviations from usual bit errors and frame errors in the error-floor region. These values might be useful for any future modification of the BERT module or even some new designs for the hardware-based NGDBF decoder.

There are certain scopes to further improve upon the consistency of these outputs from NGDBF decoder. An attempt to figure out the relationships between the sequence of noise seeds and thus formulating an algorithm to find the best possible pattern of noise generation to help with low BER region would be a potential path to extend this work. Also employing a better state machine, changing how the handshaking and control sequences take place or even moving to a completely different design with the knowledge of best noise generation pattern might be explored in the future works.

REFERENCES

[1] R. Gallager, "Low-density parity-check codes," *IRE Trans. on information theory*, vol. 8, no. 1, pp. 21–28, 1962.

[2] W. Ryan and S. Lin, *Channel codes: classical and modern*. Cambridge University Press, 2009.

[3] D. J. MacKay and M. S. Postol, "Weaknesses of Margulis and Ramanujan-Margulis low-density parity-check codes," *Electronic Notes in Theoretical Computer Science*, vol. 74, pp. 97–104, 2003.

[4] T. Richardson, "Error floors of LDPC codes," in *Proc. of the annual Allerton conference on communication control and computing*, vol. 41, no. 3. The University; 1998, 2003, pp. 1426–1435.

[5] L. Dolecek, Z. Zhang, V. Anantharam, M. J. Wainwright, and B. Nikolic, "Analysis of absorbing sets and fully absorbing sets of array-based LDPC codes," *IEEE Transactions on Information Theory*, vol. 56, no. 1, pp. 181–201, 2010.

[6] M. Rice and B. Mazzeo, "On the superiority of the negative binomial test over the binomial test for estimating the bit error rate," *IEEE Transactions on Communications*, vol. 60, no. 10, pp. 2971–2981, 2012.

[7] B. A. Mazzeo and M. Rice, "Bit error rate comparison statistics and hypothesis tests for inverse sampling (negative binomial) experiments," *IEEE Transactions on Communications*, vol. 64, no. 5, pp. 2192–2203, 2016.

[8] Z. Zhang, L. Dolecek, B. Nikolic, V. Anantharam, and M. Wainwright, "Gen03-6: Investigation of error floors of structured low-density parity-check codes by hardware emulation," in *IEEE Globecom 2006*. IEEE, 2006, pp. 1–6.

[9] J.-L. Danger, A. Ghazel, E. Boutillon, and H. Laamari, "Efficient FPGA implementation of Gaussian noise generator for communication channel emulation," in *Electronics, Circuits and Systems, 2000. ICECS 2000. The 7th IEEE International Conf. on*, vol. 1. IEEE, 2000, pp. 366–369.

[10] E. Boutillon, T. Yangyang, C. Marchand, and P. Bomel, "Hardware discrete channel emulator," in *High Performance Computing and Simulation (HPCS), 2010 International Conf. on*, 2010, pp. 452–458.

[11] G. Sundararajan, C. Winstead, and E. Boutillon, "Noisy gradient descent bit-flip decoding for LDPC codes," *arXiv preprint arXiv:1402.2773*, 2014.

[12] E. Boutillon, J.-L. Danger, and A. Ghazel, "Design of high speed AWGN communication channel emulator," *Analog Integrated Circuits and Signal Processing*, vol. 34, no. 2, pp. 133–142, 2003.

[13] E. Fung, K. Leung, N. Parimi, M. Purnaprajna, and V. C. Gaudet, "ASIC implementation of a high speed WGNG for communication channel emulation [white Gaussian noise generator]," in *Signal Processing Systems, 2004. SIPS 2004. IEEE Workshop on.* IEEE, 2004, pp. 304–309.

[14] D.-U. Lee, W. Luk, J. D. Villasenor, and P. Y. Cheung, "A Gaussian noise generator for hardware-based simulations," *IEEE Trans. on Computers*, no. 12, pp. 1523–1534, 2004.

[15] D.-U. Lee, J. D. Villasenor, W. Luk, and P. H. W. Leong, "A hardware Gaussian noise generator using the Box-Muller method and its error analysis," *IEEE Trans. on Computers*, vol. 55, no. 6, pp. 659–671, 2006.

[16] D.-U. Lee, W. Luk, J. D. Villasenor, G. Zhang, and P. H. W. Leong, "A hardware Gaussian noise generator using the Wallace method," *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, vol. 13, no. 8, pp. 911–920, 2005.

[17] G. Zhang, P. H. W. Leong, D.-U. Lee, J. D. Villasenor, R. C. Cheung, and W. Luk, "Ziggurat-based hardware Gaussian random number generator," in *International Conf. on Field Programmable Logic and Applications, 2005.* IEEE, 2005, pp. 275–280.

[18] S. S. Shapiro and M. B. Wilk, "An analysis of variance test for normality (complete samples)," *Biometrika*, vol. 52, no. 3/4, pp. 591–611, 1965.

[19] S. S. Shapiro and R. Francia, "An approximate analysis of variance test for normality," *Journal of the American Statistical Association*, vol. 67, no. 337, pp. 215–216, 1972.

[20] M. M. Rahman and Z. Govindarajulu, "A modification of the test of shapiro and wilk for normality," *Journal of Applied Statistics*, vol. 24, no. 2, pp. 219–236, 1997.

[21] A. Kolmogorov, "Sulla determinazione empirica di una lgge di distribuzione," *Inst. Ital. Attuari, Giorn.*, vol. 4, pp. 83–91, 1933.

[22] N. Smirnov *et al.*, "Table for estimating the goodness of fit of empirical distributions," *Annals of Mathematical Statistics*, vol. 19, no. 2, pp. 279–281, 1948.

[23] T. T. Tithi, "Error-floors of the 802.3 an ldpc code for noise assisted decoding," 2019.

[24] M. Biometrika19685511Wilk and R. Gnanadesikan, "Probability plotting methods for the analysis of data," *Biometrika*, vol. 55, pp. 1–17, 1968.

[25] H. C. Thode, *Testing for normality.* CRC press, 2002.