Analysis of Wireline FEC Performance Using an FPGA Platform

by

Richard Barrie

A thesis submitted in conformity with the requirements
for the degree of Master of Applied Science

Graduate Department of Electrical and Computer Engineering
University of Toronto

Analysis of Wireline FEC Performance Using an FPGA Platform

Richard Barrie
Master of Applied Science

Graduate Department of Electrical and Computer Engineering
University of Toronto
2024

Abstract

As wireline communication links transition from 100 Gb/s to 200 Gb/s per lane, new and complex forward error correction (FEC) architectures have been adopted to achieve acceptably low bit error ratios (BERs). Understanding how these architectures impact link performance under various channel conditions is essential. However, software-based time-domain simulations that demonstrate sufficiently low post-FEC BERs are prohibitively long, and statistical BER analysis techniques cannot accurately model the complex architectures proposed for 200 Gb/s wireline systems. This thesis presents a flexible platform for field-programmable gate array (FPGA)-accelerated time-domain simulations. The platform is capable of modeling wireline systems relevant to the upcoming 200 Gb/s Ethernet standard including multi-part links, concatenated FEC codes with soft-decision inner-FEC decoding, and interleaving. This FPGA platform can accurately demonstrate post-FEC BERs at the 10^{-12} level within a day of simulation time, a speed improvement by a factor of 10,000 over software-based simulation platforms.

Acknowledgements

I would first like to thank Professor Tony Chan Carusone. He has given me many golden opportunities, and I am lucky to have been in his group over the past few years. I would also like to thank Ming Yang for being such a supportive mentor, co-author, and friend.

Thanks to Frank Kschischang for serving on my thesis examination committee, for his excellent teaching, and for his support during my studies. Also, thanks to Xilin Liu for serving on my thesis examination committee, and to Stark Draper for stepping in to chair my defense at the last minute.

Thank you to Hossein Shakiba for his valuable perspective and insight on my work.

Finally, I want to thank Tamara, my family, my friends, and my housemates for all the enjoyable times we have shared.

Contents

List of Tables

List of Figures

List of Acronyms

ADC	Analog-to-digital converter
ASIC	Application-specific integrated circuit
AWGN	Additive white Gaussian noise
BER	Bit error ratio
CER	Codeword error ratio
CFEC	Concatenated forward error correction
CI	Convolutional interleaver
DFE	Decision-feedback equalizer
DSP	Digital signal processing
EPF	Error propagation factor
FEC	Forward error correction
FIR	Finite impulse response
FLR	Frame loss ratio
FPGA	Field-programmable gate array
HD	Hard-decision
IEP	Initial error probability
IL	Interleaving
IP	Intellectual property
ISI	Inter-symbol interference
KP4	Reed-Solomon FEC code used in Ethernet standards
LPF	Low-pass filter
LSB	Least-significant bit
LTI	Linear time-invariant

LUT	Look-up table
MLSD	Maximum-likelihood sequence detection
MSB	Most-significant bit
PAM	Pulse amplitude modulation
PDF	Probability density function
PMF	Probability mass function
PRBS	Pseudo-random binary sequence
SD	Soft-decision
SNR	Signal-to-noise ratio
SOVA	Soft-output Viterbi algorithm
UI	Unit interval
URNG	Uniform random number generator

Chapter 1

Introduction

1.1 Motivation

The demand for wireline interconnectivity is rapidly growing, driven by AI applications, video streaming, and cloud services. Forward error correction (FEC) codes are necessary for modern high-speed wireline links to meet stringent bit error ratio (BER) requirements. For wireline links that operate up to 100 Gb/s per lane using four-level pulse amplitude modulation (PAM-4), a single end-to-end FEC code, as shown in Figure 1.1, can achieve a sufficient post-FEC BER with acceptable latency and power. The next generation of PAM-4 wireline links will operate at 200 Gb/s per lane. Doubling the data rate degrades signal integrity, necessitating a stronger FEC architecture. To address this, the IEEE 802.3dj task force has adopted a new concatenated FEC architecture for the 200 Gb/s Ethernet standard. A concatenated FEC has an inner code and an outer code that provide two layers of error correction, as shown in Figure 1.2. A soft-decision decoder may be used to boost the performance of the inner code, while a hard-decision decoder is used for the outer code.

Many factors impact the post-FEC BER of a wireline system, including the choice of codes and decoding algorithms, digital algorithms such as precoding and interleaving, analog and digital equalization, noise, and other channel impairments. Tools that help us understand precisely how these factors affect a system's performance are useful for two purposes: Firstly, they are applicable in developing communication protocol standards that specify the FEC architecture and performance requirements. Secondly, they are helpful for transceiver system architects to know how their design decisions affect the overall performance of a wireline communication system.

Standardized communication protocols such as Ethernet, PCIe, and USB are crucial to ensure the interoperability of products between different vendors. Currently, the 200 Gb/s Ethernet standard is still a work in progress. Industry-leading companies such as Broadcom, Marvell, Intel, Huawei, Ciena, and Alphawave Semi are all heavily invested in the development of 200 Gb/s SerDes products

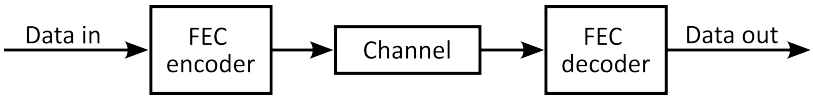

Figure 1.1: End-to-end FEC architecture.

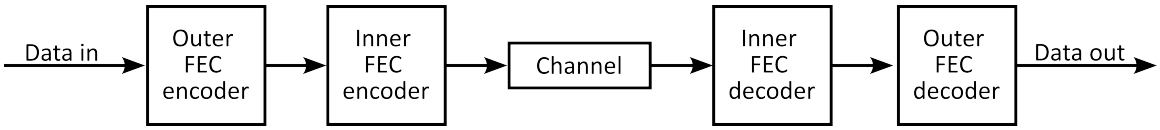

Figure 1.2: Concatenated FEC architecture.

to meet the global demand for wireline connectivity. These companies come together regularly during IEEE 802.dj task force meetings to define the standard with which their products must comply. Many details of the system-level architecture and FEC performance are still up for debate, creating a need for tools that can analyze the post-FEC BER of 200 Gb/s systems [1].

One option is to perform a time-domain software simulation by sending data through a model of the wireline system and counting the number of post-FEC bit errors. However, time-domain software simulations demonstrating the extremely low ($< 10^{-13}$) post-FEC BERs targeted by IEEE standards are prohibitively time-consuming, especially for exploring design alternatives. Another possibility is to use a statistical analysis tool to predict a system's post-FEC BER without having to run a long time-domain simulation. However, there are currently no statistical methods available that are capable of realistically modelling the complex architectures considered for the 200 Gb/s Ethernet standard.

This thesis presents an FPGA platform for analyzing the post-FEC BER of wireline systems down to very low levels. It specifically targets the FEC architectures considered for the upcoming 200 Gb/s Ethernet standard.

1.2 Scope

The work presented in this thesis is distinct from the FPGA emulation of application-specific integrated circuits (ASICs), where a specific digital circuit is synthesized on an FPGA to verify that it operates correctly. Instead, this work aims to evaluate the performance of different FEC architectures under various channel conditions, and so the FPGA platform targets speed and flexibility rather than matching a specific hardware implementation, which may vary between companies. Because of this, the platform cannot be used to evaluate the hardware complexity of a specific FEC implementation. Analysis of the FPGA platform's hardware utilization is included in this thesis, but only to document the FPGA simulation platform rather than a realistic ASIC implementation.

This thesis document aims to give the reader an understanding of how various wireline systems can be modelled on the FPGA platform. It also presents simulation results and analysis. Due to the complexity of 200 Gb/s systems of interest, descriptions of the hardware implementation are presented on an algorithmic level rather than at the register level. For readers and future students who wish to replicate or build on the platform presented in this document, Verilog code for the FPGA platform is included in the online repository at `https://github.com/richard259/FPGA-FEC`.

1.3 Outline

This thesis is organized into nine chapters. Chapter 2 presents background information on the wireline system architecture considered for the upcoming 200 Gb/s Ethernet standard. It also discusses the metrics used to evaluate the performance of communication systems and reviews related works.

Chapter 3 presents the high-level architecture of the proposed FPGA platform. This section focuses on the parts of the platform peripheral to the wireline system model, including the user interface and the software that controls the programmable simulation settings. Chapter 3 also includes additional discussion on BER simulation methodology, such as the platform's speed and the trade-off between simulation length and accuracy.

Chapters 4 to 7 present how different parts of a wireline system are modelled on the FPGA platform. Chapter 4 presents simple wireline systems based on channel models that artificially inject errors into a stream of PAM-4 symbols. Chapter 5 presents a more complex model for the channel and receiver that includes a discrete approximation of the analog signal. In Chapter 6, the inner FEC code is introduced. Chapter 7 presents the Chase algorithm for soft-decision inner FEC decoding and the generation of soft-decision channel outputs. Simulation results are presented in all of these chapters, with a comparison to statistical or software models to verify the accuracy when possible.

Chapter 8 presents a case study where a full multi-part 200 Gb/s wireline system is modeled. The FPGA platform is used to analyze the system's performance under various channel conditions and architecture choices, demonstrating the platform's utility in system-level transceiver design and standard development. Finally, Chapter 9 concludes the thesis.

Chapter 2

Background and Related Works

2.1 FEC Architecture for 200 Gb/s Multi-Part Links

Multi-part links are typically employed for communication between data centers. First, a short electrical link transmits data from a host chip on a server to an optical module. The optical module converts this electrical signal to an optical signal and sends it along a fibre-optic cable to a second optical module at a receiving data center. This optical module converts the signal back to an electrical signal and sends it to a receiving host chip over another electrical link. Both the electrical and optical links use PAM-4 signaling. All three links along the data path can introduce errors in the transmitted data. The electrical links have large amounts of inter-symbol interference (ISI), so equalization techniques such as decision feedback equalization (DFE) or maximum-likelihood sequence detection (MLSD) are used to recover the signal. Both DFE and MLSD suffer from error propagation, which causes the errors from electrical links to be predominantly bursts of correlated errors [2], [3]. In contrast, optical channels are theoretically ISI-free, and so they are often modelled with an additive Gaussian white noise (AWGN) channel, which introduces uncorrelated random errors. However, in reality, PAM-4 optical receivers typically include bandwidth-limited components such as transimpedance amplifiers that introduce ISI, and so MLSD equalization is also common in optical receivers. MLSD error propagation may result in correlated errors from the optical link, although they will typically be less strongly correlated compared to the electrical links [4].

For 200 Gb/s multi-part links, the concatenated FEC architecture proposed in the Ethernet standard [5] is shown in Figure 2.1. The outer code protects from errors in all three links, while the

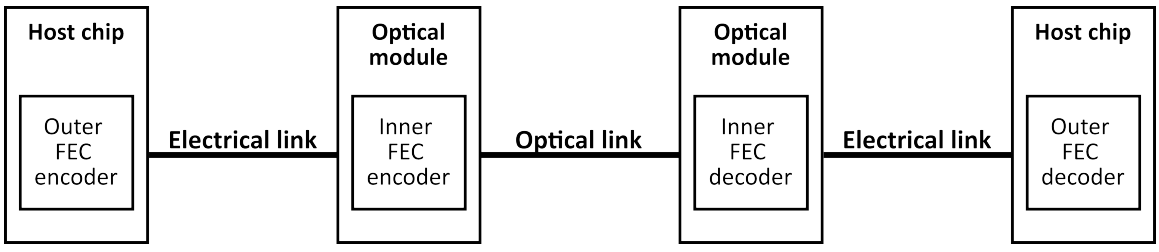

Figure 2.1: 200 Gb/s multi-part link with concatenated FEC.

inner code only protects the middle optical link. The outer code is a Reed-Solomon code with block length $n = 544$ and dimension $k = 514$ in $\mathbb{F}_{2^{10}}$ known as the "KP4" code. The KP4 code is a strong, non-binary linear block code that operates on 10-bit-long FEC symbols. It can correct up to $t = 15$ FEC symbols per 544-FEC-symbol-long codeword, making it is well-suited to correct the long bursts of errors that arise in electrical links due to DFE or MLSD error propagation. The inner code is a binary extended Hamming code that can correct $t = 1$ bit error per 128-bit codeword. Although this code can only correct one bit error per codeword, it is effective against uncorrelated errors that arise in optical links and has low decoding complexity compared to a stronger code. Interleaving and soft-decision decoding can improve the inner FEC's performance so that it is more robust to correlated errors in the optical link. The inner code's role is to reduce the optical link's BER, so that the outer code can work more effectively on correcting the correlated errors produced in the electrical links.

2.2 System Performance Metrics

The post-FEC BER, defined as the ratio of bit errors after FEC decoding to total transmitted bits, is a commonly used measure of communication reliability. However, in the context of the Ethernet standard, this is not the most relevant metric. The Ethernet protocol sends information in 64-byte packets called *frames*, and the Ethernet standard specifies reliability using frame loss ratio (FLR), defined as the ratio of frames received as invalid to the total number of transmitted frames [6]. Note that the outer KP4 FEC decoder can detect when it receives an uncorrectable codeword. When this happens, the decoder flags all the Ethernet frames that may be affected as invalid so that the receiving chip may request a re-transmission of these frames. As such, the Ethernet protocol never experiences actual post-FEC bit errors, just invalid frames that need to be re-transmitted. The current draft of the 200 Gb/s Ethernet standard specifies an FLR requirement of less than 6.2×10^{-11}, which is approximately equivalent to a post-FEC BER below 10^{-13}. This is chosen to be an extremely small number because Ethernet frame re-transmission is very costly to the latency of downstream Ethernet-connected devices.

Although the FLR is the performance metric used in the Ethernet standard, this thesis uses the KP4 codeword error ratio (CER), defined as the ratio of uncorrectable KP4 codewords to total transmitted KP4 codewords. The CER is preferred because it relates directly to the FEC performance and is independent of the Ethernet communication protocol. Although one KP4 codeword contains the same amount of information as eight frames (with some additional padding bits between frames), due to frame alignment and scrambling, each KP4 codeword error may corrupt up to nine Ethernet frames, and so the relationship between CER and FLR is $FLR = \frac{9}{8} \times CER$ [7]. Therefore, the Ethernet standard equivalently requires a CER of less than 5.5×10^{-11}. At this CER level, with 200 Gb/s PAM-4 signaling, one KP4 codeword error happens on average every 8 minutes.

2.3 Related Works

Many approaches for estimating the CER of a wireline system exist, including time-domain software simulations, statistical analysis, and other FPGA platforms. This section presents relevant examples

of these tools and explains why they are not well-suited for demonstrating low CER levels for multi-part links with concatenated FEC codes.

Time-Domain Software Simulation

A standard method for analyzing the performance of error correction codes is using a software-based time-domain simulation where data is sent through a software model of the system, and the CER is estimated by dividing the number of codeword errors by the total number of transmitted codewords. Matlab and Python are attractive platforms for these simulations, as they provide convenient tools for encoding, decoding, and channel modelling. Unfortunately, software simulations are much slower than the data rates of high-speed links, and simulations demonstrating CERs at the 5.5×10^{-11} level are prohibitively long. For example, the Matlab model reported in [8] can simulate an end-to-end KP4 FEC on a wireline link at 1.47×10^5 bits per second when running on 16 parallel CPU cores. At this speed, it would take about 500 years on average to observe one KP4 codeword error at the 5.5×10^{-11} level. Accordingly, the paper only presents software-simulated post-FEC BERs down to the 10^{-9} level, which is equivalent to a KP4 CER of around 10^{-7}. Another example of a software tool that can be used for FEC analysis is a Python library for modelling wireline links that I developed for my undergraduate thesis project [9]. Although this tool is capable of representing a multi-part link with concatenated FEC and soft-decision inner-FEC decoding, this algorithmic complexity results in a simulation speed of only around 2×10^4 bits per second, which is far too slow for demonstrating low CER levels. Because software-based time-domain simulations are too slow, various extrapolation methods are used in industry to estimate lower CERs based on trends at higher CER levels. However, in many cases, this approach is inaccurate because the correlation between signal-to-noise ratio (SNR) and CER may change significantly at low CER [8], [10].

Statistical Analysis

Another approach is to use a statistical model to determine post-FEC BER analytically. This method is attractive because statistical models can determine arbitrarily low CERs in a reasonable amount of time. Ming Yang's PhD Thesis presents a method for statistical post-FEC BER analysis of PAM-4 links subject to DFE error propagation [11]. In 2022-2023, Ming and I extended this model for a concatenated FEC architecture. We presented this work at DesignCon 2023 [12], where it won the best paper award. Unfortunately, our statistical model is subject to several limitations that make it unsuitable for realistic 200 Gb/s wireline systems:

- It only considers a DFE-based receiver and cannot model MLSD-based receivers.
- It assumes a white noise spectrum in the channel and cannot accommodate coloured noise.
- It cannot consider a soft-decision inner FEC decoding or the complex interleaving scheme proposed for the 200 Gb/s Ethernet standard.

During the development of this model, it became clear to us that including all these details in a statistical model would increase the computational complexity to impractical levels.

Hybrid Time-Domain/Statistical Analysis

Another technique for analysis of low CERs currently used in industry is a hybrid time-domain/statistical model [13]. This model uses a time-domain software simulation that is not long enough to observe any KP4 codeword errors, but records statistics including the frequency and correlation between FEC symbol errors. These statistics are input to a statistical model that extrapolates the probability of observing more than t FEC symbol errors in a codeword (the CER). Although this seems to be a reasonable method that is accurate for some link architectures, it is unclear how long a time-domain simulation and how complex of a statistical model is necessary for an accurate CER estimate. Furthermore, it is only possible to validate the model's accuracy at low CER levels with a fast time-domain simulation platform to compare against.

Other FPGA Platforms

Using an FPGA platform for FEC analysis is not a new idea. It was proposed in the 2002 paper "Accelerating BER Analysis Using an FPGA Based Processing Platform" which describes an environment similar to the one presented in this thesis [14]. Since then, FPGAs have often been used to prototype FEC architectures and run BER analysis simulations. For example, Benjamin Smith's 2011 thesis "Error-Correcting Codes for Fibre-Optic Communication Systems" presents FPGA-simulated BERs down to the 10^{-14} level for staircase codes [15]. Another recent publication explores FEC alternatives for 800 Gb/s coherent optical communication, including concatenated Hamming and staircase codes [16]. Using an array of 100 FPGAs, they demonstrate simulations of 16-QAM signalling over an AWGN channel running at 400 Gb/s, allowing for demonstration of BERs down to 10^{-15} levels.

The platform presented in this thesis is similar to these examples but focuses on the end-to-end system for 200 Gb/s multi-part links using the Ethernet standard. A feature that sets this work apart is that other FPGA platforms typically assume an AWGN channel. In contrast, this work can accommodate a more complex system that includes multi-part links, channels with ISI, interleaving, precoding, DFE and MLSD-based receivers, and coloured noise. This work achieves a similar simulation speed per FPGA compared to [16]. However, the results in this thesis only use a single FPGA rather than an array of 100.

Baud Rate vs. Oversampled Signals

Lastly, another FPGA simulation platform for analyzing BER in PAM-4 links is proposed in [17]. However, their platform takes a different approach by using over-sampled time-domain signals, increasing model accuracy but lowering simulation speed. Accordingly, their work only reports FPGA-simulated BERs down to the 10^{-7} level. In contrast, this thesis only uses baud-rate-sampled time-domain signals. While over-sampling can allow for a more faithful representation of an analog signal and impairments on a small timescale, it is common practice to use baud-rate-sampled signals for time-domain FEC simulations for the following reasons:

- A fast simulation speed is the most essential attribute of a FEC simulation platform. A model that is fast enough to demonstrate CERs at the 10^{-11} level in a reasonable amount of time will not realistically be able to model a specific implemented wireline transceiver precisely.

Some model accuracy must be sacrificed to achieve acceptable speed. This is okay because this platform's purpose is not to show perfect correlation to a specific implementation but rather to provide insight on the high-level architecture performance. A simple baud-rate-sampled channel is sufficient to do this at a very high speed.

- It is important to present results that different companies can easily replicate to gain support for standard body contributions. Simple baud-rate channel models, such as an ISI channel with impulse response $1 + \alpha D$ in the presence of AWGN (presented in Section 4.5.3) or the error propagation factor channel (presented in Section 4.5.2), allow for replicable results between companies without the need to disclose complex and proprietary information about their products.

- Modern PAM-4 links typically use an analog-to-digital converter (ADC) to produce baud-rate samples, and digital signal processing (DSP) algorithms to recover the transmitted symbols. Since these DSP algorithms operate on baud-rate samples, their effect on FEC performance can be accurately measured with only a baud-rate simulation platform.

Chapter 3

FPGA Platform Overview

3.1 Architecture

Figure 3.1 shows a system-level diagram of the FPGA simulation platform. The first step in running a simulation is to model the system of interest by defining the "BER simulation IP" block. This block is defined using a library of custom Verilog modules that constitute the main work of this thesis and are described in Chapters 4-7. This library is implemented in a flexible way so that one can mix and match modules to model a wide variety of wireline systems. Many different channel types, FEC architectures, and interleaving schemes are possible. Once the BER simulation IP block is defined, the FPGA is programmed with the system shown in Figure 3.1.

The user configures simulation settings with an interface on a PC. Software running on the on-chip MicroBlaze processor configures the BER simulation IP with these settings and starts the simulation. The BER simulation IP generates binary data and sends it through the wireline system model. As data is sent through the system model, the following statistics are counted:

- Number of transmitted bits
- Number of bit errors before KP4 decoding (pre-FEC bit errors)
- Number of bit errors after KP4 decoding (post-FEC bit errors)

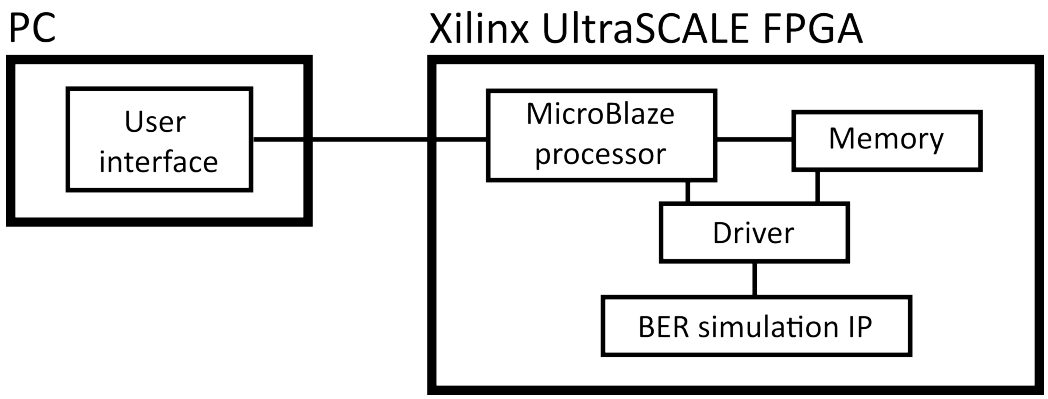

Figure 3.1: FPGA-accelerated simulation platform.

- Number of transmitted KP4 codewords
- Number of transmitted KP4 codeword errors

Once a user-specified number of KP4 codeword errors is reached, the results are automatically written to the memory and can be viewed on the user interface and saved to a .csv file. The software running on the MicroBlaze processor allows the user to run multiple consecutive simulations automatically. This way, the platform can efficiently sweep link SNR over a range of values to view its relationship with CER.

Configurability is one of the platform's main assets, allowing it to explore different FEC architectures and algorithms under various channel conditions. There are three levels at which a user can customize the BER simulation IP:

1. The **system architecture** is defined in a top-level Verilog file. This file specifies all the Verilog modules that make up the wireline system model. For example, this file determines if the system is a single-link or multi-part link, the type of channel models used, if an inner FEC code is used, etc... The system architecture is the least flexible level of customization. To change the system architecture, a user must edit the top-level Verilog file, re-synthesize the hardware and re-program the FPGA.
2. The **Verilog module parameters** are used to customize Verilog modules. These parameters include the number of bits used to represent an analog signal, the value of α in a $1+\alpha D$ channel pulse response, and the number of parallel cores used to speed up the simulation. Changing these parameters is more accessible than changing the system architecture because they can be set in Vivado's "block design" GUI without editing a Verilog file. However, changing these parameters requires re-synthesizing the design and re-programming the FPGA.
3. The **runtime-programmable settings** are the most flexible settings. They can be changed with the user interface after the FPGA is programmed. These settings include the SNR of each link, inner FEC on/off, precoding on/off, and the number of codeword errors to observe before terminating the simulation.

3.2 Simulation Speed

The platform runs at 150 MHz, with one bit per clock cycle simulated, resulting in a simulation speed of 150 million bits per second. However, the platform supports parallel processing, where the BER simulation IP contains multiple parallel cores to increase simulation speed. The number of parallel cores that can be used depends on the hardware complexity of the system of interest. To achieve maximum speed, the number of cores should be set to the highest amount possible subject to the available hardware resources on the FPGA and timing constraints. Simulation results presented in this thesis were run on a single Xilinx Virtex UltraScale FPGA VCU108 evaluation kit. For simulations of the most simple channel model including a single-part link with AWGN, the platform can support 200 parallel cores for a simulation speed of 30 Gb/s. For a full system including multi-part link with soft-decision inner FEC decoding, the hardware complexity of the BER simulation IP is significantly higher. Accordingly only 8 parallel cores can be used, resulting in a slower simulation speed of 1.2 Gb/s.

3.3 Confidence Intervals on CER estimates

When performing a time-domain simulation, an estimate of the CER, CER_{est}, is calculated as the ratio of uncorrectable codewords to total transmitted codewords. The event that each transmitted KP4 codeword is either correctable or uncorrectable can be viewed as a binary random variable. Codewords with 15 or fewer FEC symbol errors are correctable, and those with greater than 15 are not. As the number of transmitted codewords approaches infinity, the ratio of codeword errors to total codewords will approach the true CER of the system, CER_{true}. However, the simulation must be terminated after some finite number of codeword errors. Terminating the simulation after a smaller number of transmitted codewords results in a shorter simulation with lower accuracy, and terminating after a larger number of transmitted codewords results in better accuracy at the cost of longer simulation time. It is helpful to calculate confidence intervals on CER estimates to choose a simulation termination condition with acceptable accuracy in the shortest time. One can approximate the event of correctable or uncorrectable codewords as an independent, identically-distributed (i.i.d) Bernoulli process so that the Clopper-Pearson method [18] can be used to calculate confidence intervals around the CER estimate.

Justification of i.i.d. Bernoulli Assumption

The i.i.d. Bernoulli approximation assumes that the event of each transmitted codeword is an i.i.d. binary random variable with the probability of error $P_e = CER_{true}$, the true system CER, and probability of being correctable $P_c = 1 - CER_{true}$. This approximation is not perfect; DFE or MLSD error propagation, coloured noise, and inner FEC miscorrections result in the PAM-4 symbols at the input to the KP4 decoder to be correlated temporally, resulting correlated KP4 codeword errors. However, this correlation is strongest between neighboring PAM symbols and exponentially decays between symbols that are further apart. Although there may be a significant error correlation between the PAM-4 symbols at the end of one KP4 codeword and the beginning of the next, KP4 codewords are 2720 PAM-4 symbols long, and so the vast majority of the PAM-4 symbols in each codeword are very far from the vast majority of PAM-4 symbols in the neighbouring codeword. This results in an insignificant error correlation between neighbouring codewords, so the i.i.d. Bernoulli approximation is reasonable.

To illustrate this point, consider a simple channel model with error propagation: The probability of a PAM-4 symbol error given that the previous PAM-4 symbol was not an error is 10^{-5}, and the probability of a PAM-4 symbol error given that the previous symbol was an error is 0.75. Using the statistical analysis technique presented in [8], the CER of this channel is evaluated to be 5.5×10^{-11}. The probability of a KP4 codeword error given that the previous codeword was an error is evaluated to be 5.7×10^{-11}, very close to the true CER. Despite the large probability of PAM-4 symbol error propagation, there is a minimal amount of KP4 codeword error correlation, so the process is "almost" i.i.d. Bernoulli. Note that in real wireline systems, this approximation may not hold up. For example, if there is a supply voltage drop for a short period, this may result in a burst of KP4 codeword errors that are correlated in time. However, for the FPGA platform presented in this work, the channel conditions remain constant for the duration of the simulation.

Choosing an Appropriate Simulation Termination Condition

The FPGA platform automatically terminates a simulation after a specified number of codeword errors are observed. Table 3.1 highlights the trade-off between accuracy and simulation time for simulations at the Ethernet standard's target CER of 5.5×10^{-11}. The first column shows options for the number of observed codeword errors to use as the simulation termination condition. The second column shows the expected number of total transmitted codewords for a simulation with $CER_{true} = 5.5 \times 10^{-11}$. The third column shows a 90% confidence interval for the CER estimate expressed as a percentage of the CER_{true}. For example, if the confidence interval is ±10%, one can be 90% confident that CER_{est} is within 10% of CER_{true}. The fourth to seventh columns include the times it would take to complete this simulation at various speeds.

Number of Observed Codeword Errors	Number of Transmitted Codewords	CER (90% Confidence Interval)	Simulation Time (200 Gb/s)	Simulation Time (FPGA, 30 Gb/s)	Simulation Time (FPGA, 1.2 Gb/s)	Simulation Time (Software, 150 Kb/s)
1	1.8×10^{10}	5.5×10^{-11} (-95% to +379%)	8 mins	55 mins	23 hours	21 years
10	1.8×10^{11}	5.5×10^{-11} (-45% to +71%)	1.4 hours	9 hours	9.5 days	213 years
20	3.6×10^{11}	5.5×10^{-11} (-33% to +47%)	2.8 hours	18 hours	19 days	426 years
100	1.8×10^{12}	5.5×10^{-11} (-15% to +19%)	14 hours	3.8 days	95 days	2134 years

Table 3.1: Comparison of simulation time and confidence intervals for simulations demonstrating a CER of 5.5×10^{-11} at different speeds.

For the simulations of low ($< 10^{-8}$) CERs presented in this thesis, the simulation termination condition is chosen to be 20 observed codeword errors, allowing 90% confidence that CER_{est} is within -33% to +47% of CER_{true}. Terminating the simulation at a smaller number of codeword errors, such as 1 or 10, would provide such poor confidence that CER_{est} is less useful. Choosing a larger number of codeword errors, such as 100, provides a relatively small increase in accuracy for a 5× increase in simulation time. However, for the simulations at higher CER ($> 10^{-8}$), even simulations observing 100 codeword errors take a small amount of time, and so the termination condition is set to 100 for increased accuracy. A direction of further research is to investigate if terminating a simulation upon reaching a desired amount of codeword errors introduces a bias into CER_{est}, because it guarantees that the last transmitted codeword in a simulation is an error.

The simulation times in Table 3.1 show that any time-domain simulation of low CERs is prohibitively long with software tools, but the FPGA platform can approach these low CERs in a reasonable time. For simulations terminated after 20 codeword errors, the FPGA platform running at 30 Gb/s can finish a simulation at the targeted CER of 5.5×10^{-11} in 18 hours. Unfortunately,

at this speed, the FPGA platform can only support a simple one-link system with AWGN and end-to-end KP4 FEC. For a more complex system that models a realistic FEC architecture for 200 Gb/s multi-part links, the platform would run at 1.2 Gb/s, so it would take 19 days to observe ten codeword errors at the targeted CER level. Although this is an unreasonable amount of time, it is possible to speed this up by running the platform on multiple FPGAs in parallel, similar to the work presented in [16] that uses 100 parallel FPGAs. With 100× parallelism, the speed would be 120 Gb/s, and a simulation observing 20 bit errors at the targeted CER would be complete in only 4.5 hours. In this work, all FPGA simulation results were run on a single FPGA within one day of simulation time. As a result, many of the more complex systems do not report CERs down to Ethernet's targeted 5.5×10^{-11}.

Figure 3.2 shows an example plot of SNR vs. CER for a PAM-4 link over an AWGN channel protected by a KP4 FEC code. The blue line shows the true CER calculated by the statistical model presented in [8]. The red dots show the FPGA-simulated data points. Error bars showing 90% confidence intervals are included on the FPGA simulated data points. These data points were generated on the FPGA platform with a simulation speed of 30 Gb/s. The lowest CER point was generated with a 24-hour simulation, demonstrating the platform's capability of reaching the Ethernet's targeted CER of 5.5×10^{-11} within one day. The figures presenting FPGA-simulated CERs in the remainder of this thesis use the same simulation termination conditions, but the error bars are omitted for clarity.

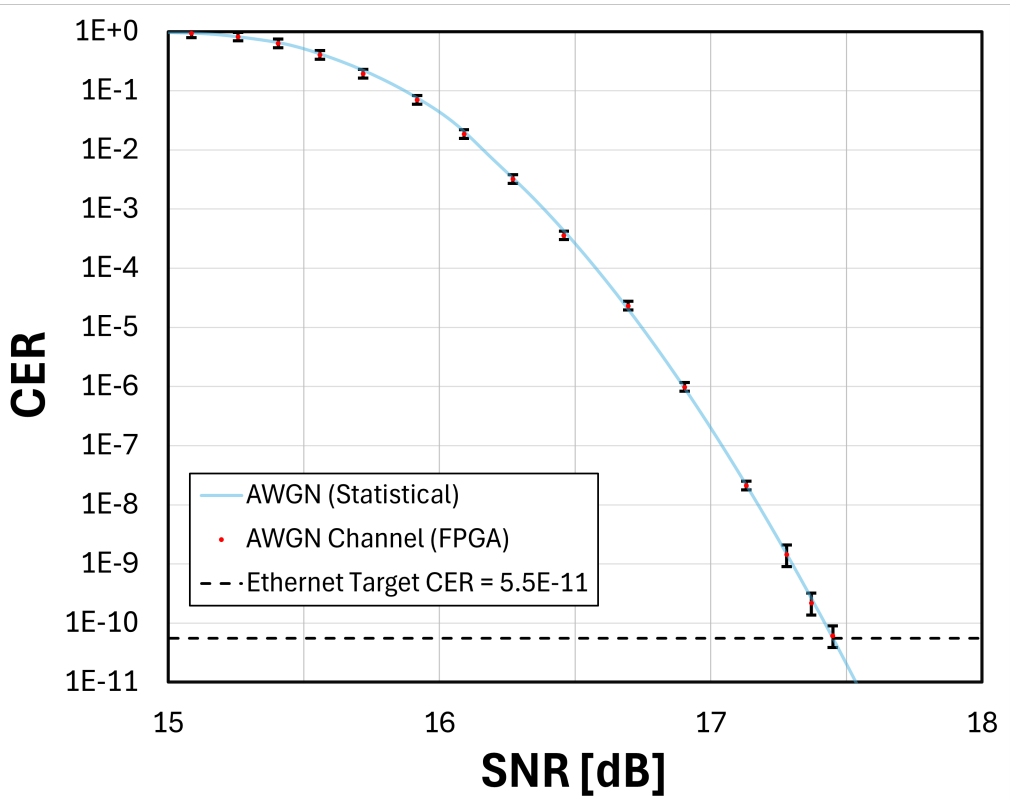

Figure 3.2: SNR vs. CER for PAM-4 AWGN channel: FPGA simulated data with confidence intervals.

Chapter 4

Wireline Systems with Error Injection Channel Models

This chapter describes the FPGA platform's implementation of data generation, PAM-4 modulation, channel models, and the KP4 FEC. This chapter focuses on simple channel models that artificially inject errors into a stream of PAM-4 symbols. These "error injection" channel models have the advantage of a low hardware complexity, and can accurately model the errors introduced by an AWGN channel and channels with error propagation.

4.1 Input PRBS Data

Pseudo-random binary sequences (PRBS) are commonly used as input data for BER testing because they exhibit statistically random behaviour and are easily generated with linear-feedback shift register (LFSR) circuits. The Ethernet standard specifies a PRBS31 pattern for testing BER performance [6]. The PRBS31 pattern is $2^{31} - 1$ bits long and contains all possible binary sequences of length 31 exactly once, except for the all zeros sequence. This pattern is generated using a 31-bit long shift register, with feedback from the 28^{th} stage and 31^{st} stage XORed together driving the input, as shown in Figure 4.1.

The initial value of the shift register (the "seed") determines the starting location in the pattern. After exactly $2^{31} - 1 \approx 2 \times 10^9$ bits, the shift register returns to its initial value and the sequence repeats. A 24-hour-long BER simulation running at 150 MHz will transmit over 10^{13} bits per simulation core so that the PRBS31 sequence will repeat many times during the simulation. For a

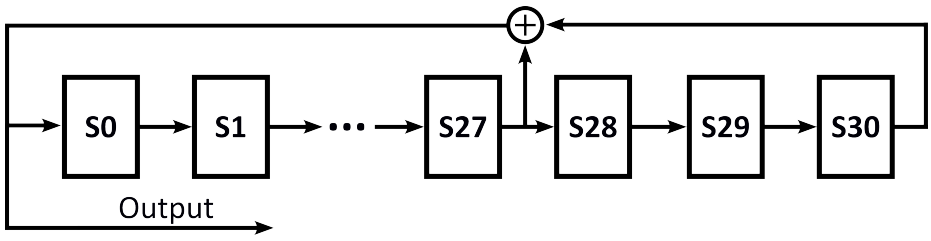

Figure 4.1: PRBS31 pattern generator circuit.

better model of truly random data, a longer PRBS sequence such as the PRBS63 sequence can be used, which is implemented using a 63-bit LFSR with taps after the 62^{th} and 63^{rd} stage [19]. This sequence is of length $2^{63} - 1 \approx 9 \times 10^{18}$, much longer than would ever be repeated.

Both PRBS31 and PRBS63 generators are included in the FPGA platform. They output one new bit each clock cycle. The simulations presented in this thesis use data generated with the PRBS63 sequence to replicate statistical models that assume truly random data. For simulations using parallel processing, each parallel core uses a different random PRBS seed, so they all start in different locations in the sequence, reducing any correlation between the transmitted data in the cores.

4.2 PAM-4 Modulation

Gray-coded PAM-4 modulation is applied to the input binary sequence before transmission through electrical and optical channels, as defined in the Ethernet Standard [6]. Table 4.1 shows how groups of two bits are mapped to PAM-4 symbols. PAM-4 symbols are represented on the FPGA as two-bit unsigned values $\{0, 1, 2, 3\}$. Since the PRBS generation produces one bit per clock cycle, the PAM-4 symbol rate is half the clock frequency.

Bits	PAM-4 Symbol
00	0
01	1
11	2
10	3

Table 4.1: Gray-coded PAM-4 mapping.

It is much more likely for PAM-4 symbol errors to perturb the transmitted symbol to a neighbouring symbol (e.g 0 is sent and 1 is received) rather than across two symbols (eg. 0 is sent and 2 is received) [11]. The Gray coding scheme ensures that neighbouring PAM-4 symbols only differ by one bit, so a PAM-4 symbol error is much more likely to result in one bit error rather than two.

4.3 (1/1+D) Precoding

1/(1+D) precoding is an algorithm used to reduce the impact of bursts of consecutive PAM-4-symbol errors that arise in electrical links due to DFE or MLSD error propagation [20]. Figure 4.2 shows the 1/(1+D) precoding architecture. The blocks labelled "D" delay the signal by one clock cycle, and the $\oplus$ is MOD-4 addition. The negative sign in the red precoding section signifies that the delayed symbol's additive inverse is used at the input to the MOD-4 adder.

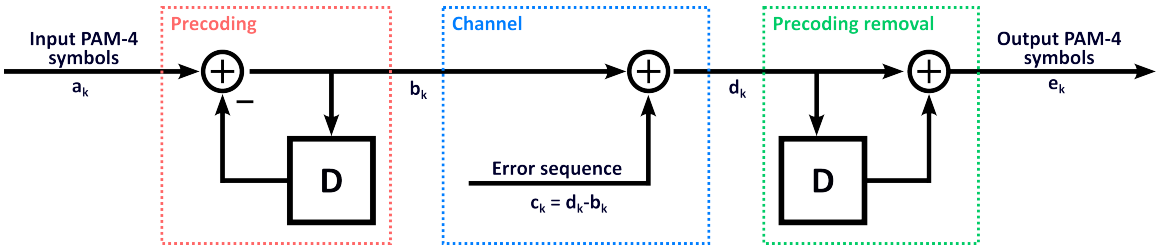

Figure 4.2: (1/1+D) precoding architecture.

Figure 4.2 shows the stream of PAM-4 symbol transmission from input to output, with five points labelled a through e for reference. The channel input at time index k is denoted $b_k \in \{0, 1, 2, 3\}$, and the channel output at time index k is denoted $d_k \in \{0, 1, 2, 3\}$. This notation is used throughout the remainder of this thesis. The channel is represented as an error sequence c_k added to the channel input b_k to produce the channel output d_k. If there is no error at time index k, the error $c_k = 0$, and $b_k = d_k$.

If there is an error at time index k, the error $c_k = d_k - b_k$ is likely to be $\in \{-1, +1\}$, so that PAM-4 symbol errors are only likely to be between neighbouring symbols. Precoding takes advantage of the fact that DFE and MLSD error bursts typically occur with an alternating sign. For example, a length-four burst error sequence could be either $\{+1, -1, +1, -1\}$ or $\{-1, +1, -1, +1\}$. Table 4.2 shows how the signal at points a through e the presence of a length-four burst of errors.

Timestep k	-1	0	1	2	3	4	5	6	7	8
a	-	3	0	0	3	2	2	1	2	3
b	0	3	1	3	0	2	0	1	1	2
c	0	0	0	-1	+1	-1	+1	0	0	0
d	0	3	1	2	1	1	1	1	1	2
e	-	3	0	3	3	2	2	2	2	3

Table 4.2: 1/(1+D) precoding with a burst of consecutive errors.

1/(1+D) precoding eliminates most errors from a burst, leaving only one error at the beginning and one at the end. Table 4.3 shows the same precoding algorithm in the presence of a single isolated error. In this case, precoding introduces an extra error at the output, degrading the link performance. To ensure that precoding improves the link performance, the expected length of a burst of consecutive PAM-4 symbol errors should be greater than two, so that precoding reduces the amount of PAM-4 errors per burst more often than not.

Time	t_{-1}	t_0	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8
a	-	3	0	0	3	2	2	1	2	3
b	0	3	1	3	0	2	0	1	1	2
c	0	0	0	-1	0	0	0	0	0	0
d	0	3	1	2	0	2	0	1	1	2
e	-	3	0	3	2	2	2	1	2	3

Table 4.3: 1/(1+D) precoding with a single error.

The precoding module in the FPGA platform is implemented with a runtime-programmable on/off switch so that the link performance can be measured with precoding both on and off without needing to re-program the FPGA.

4.4 Outer Code Model

Using a deterministic PRBS as input data provides a convenient way of checking the BER of a system: The same PRBS can be generated and compared the data at the system's output. Pre-FEC bit errors are identified if the output data does not match the PRBS. This concept can be extended to check the CER of a system without the need for a complex and resource-intensive KP4 encoder and decoder on the FPGA. To do this, a "FEC checker" module checks each group of 10 consecutive bits at the system's output against the reference PRBS. A FEC-symbol error is identified if there are one or more bit errors in a 10-bit-long sequence. The FEC checker tracks the number of bit errors and FEC-symbol errors for each sequence of FEC symbols that make up a codeword. If there are more than 15 FEC-symbol errors at the end of a 544 FEC-symbol-long KP4 codeword, the codeword is not correctable, so a codeword error count is incremented. All the bit errors in the uncorrectable codeword are added to a count of post-FEC bit errors. Otherwise, the codeword is correctable, and no codeword error or post-FEC bit errors are counted.

This FEC checker has the drawback of being unable to consider miscorrections (decoding errors), where a received codeword with more than t errors is mistakenly corrected to the wrong codeword, adding more errors. However, it has been shown that for a t-error-correcting code, the probability of miscorrecting a codeword with greater than t errors is less than $1/t!$. For the KP4 code with $t = 15$, this probability is less than 7×10^{-13}, so neglecting miscorrections results in an insignificant loss of accuracy [21]. The hardware implementation of this FEC checker module has Verilog parameters to set the FEC code parameters n, k and t, as well as the number of bits per FEC symbol so that it is able to model any linear block code. 64-bit unsigned values are used for all counters so they will not overflow during long simulations.

4.4.1 Block Interleaving

Block interleaving mitigates the impact of consecutive errors on FEC performance [22]. With N-way block interleaving, N different codewords are transmitted through the channel simultaneously, with

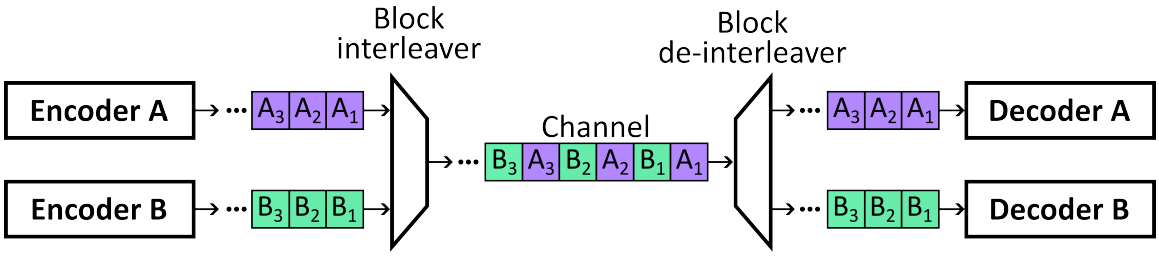

Figure 4.3: 2-way block interleaving.

their FEC symbols interleaved in a round-robin fashion. An example of 2-way block interleaving is shown in Figure 4.3.

This architecture distributes consecutive FEC-symbol errors over multiple codewords, giving them a greater chance of being corrected. For example, with the KP4 code that can correct up to 15 FEC-symbol errors per codeword, a codeword with a burst of 16 consecutive FEC-symbol errors cannot be corrected. However, with 2-way interleaving, two codewords would receive 8 FEC symbol errors each and could each be corrected given that the rest of the codewords are error-free. Both 2-way and 4-way block interleaving schemes are considered for 200 Gb/s applications [23]. Our FPGA platform supports N-way interleaving schemes by making a simple modification to the KP4 checker module described in Section 4.4: the module keeps track of which of the N codewords the current FEC-symbol belongs to and maintains bit errors and FEC-symbol error counts for each of the N codewords individually. The value of N for the KP4 FEC checker module interleaving is implemented as a runtime-programmable setting.

4.5 Error Injection Channel Models

This section describes the implementation of three different error injection channel models included in the FPGA platform. The purpose of these modules is to take in the channel input PAM-4 symbols $b_k \in \{0, 1, 2, 3\}$ and produce channel output symbols $d_k \in \{0, 1, 2, 3\}$ to model the behaviour of a wireline link.

4.5.1 AWGN Channel

The AWGN channel is commonly used to evaluate FEC performance. Figure 4.4 (a) shows an analog representation of the AWGN channel with PAM-4 signaling. First, the transmitted PAM-4 symbols $b_k \in \{0, 1, 2, 3\}$ are mapped to analog levels $t_k \in \{-3, -1, +1, +3\}$ by a driver. These analog levels may represent a differential voltage in the case of an electrical channel or a light intensity in the case of an optical link. Then, uncorrelated Gaussian noise samples denoted n_k with zero mean and variance σ^2 are added to each transmitted symbol, producing r_k, the received noisy samples. Finally, an ideal hard-decision slicer outputs the PAM-4 symbol $d_k \in \{0, 1, 2, 3\}$ corresponding to the analog voltage level $\{-3, -1, +1, +3\}$ that is closest to r_k. Because the noise is Gaussian and has zero mean, this slicer output corresponds to the most likely transmitted symbol given the received noisy sample, i.e., d_k is the b_k that maximizes $P(b_k|r_k)$.

Figure 4.4 (b) shows the FPGA's AWGN channel module, simply taking PAM-4 symbols b_k as

input and producing d_k as a channel output with the same conditional probability $P(d_k|b_k)$ as the true analog channel.

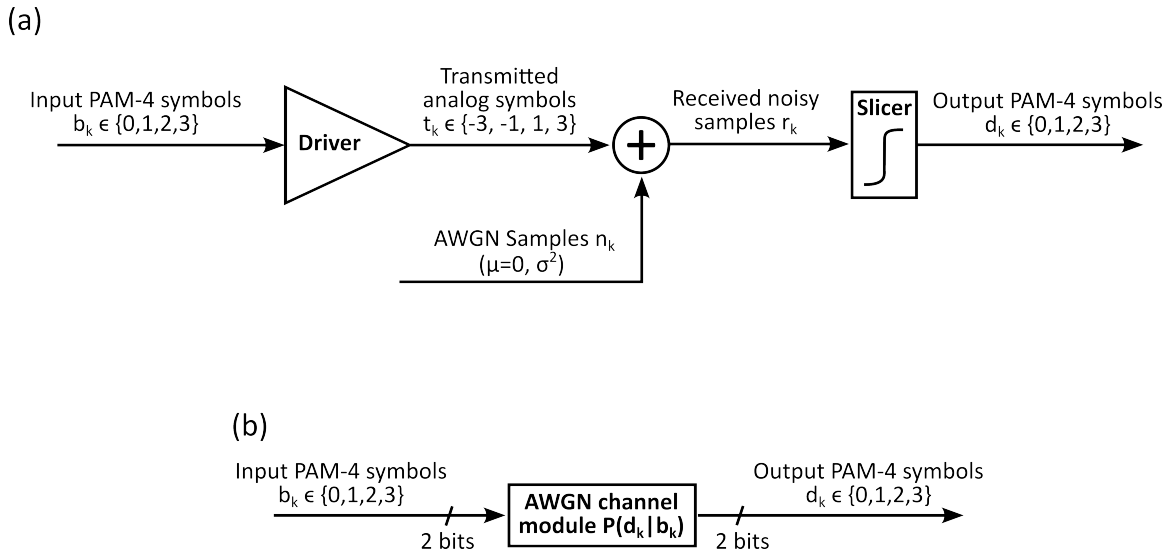

Figure 4.4: (a) PAM-4 AWGN Channel. (b) Error injection model of AWGN channel on FPGA.

Signal-to-Noise Ratio (SNR)

Assuming each symbol is equally likely to be transmitted, the average transmitted signal power per symbol for the symbol set $t_k \in \{-3, -1, +1, +3\}$ is

$$P_{signal} = \frac{(-3)^2 + (-1)^2 + 1^2 + 3^2}{4} = 5$$

For the AWGN channel, the average noise power per symbol is the variance of the noise samples,

$$P_{noise} = \sigma^2$$

In this thesis, SNR is the main metric for signal integrity and is defined as the ratio of signal power to noise power,

$$\text{SNR} = P_{sig}/P_{noise}$$

This is expressed in decibels as,

$$\text{SNR}_{dB} = 10 \times \log_{10}(P_{sig}/P_{noise})$$

The signal and noise power values in this SNR calculation do not necessarily depict a realistic system, which may not use the signal levels $\{-3, -1, +1, +3\}$ at the transmitter and may also be attenuated by the channel. However, because the SNR expresses a ratio and is independent of the actual amplitude of signal and noise, the SNR used in this thesis still applies to real systems. Also, note that this definition of SNR is consistent with what is used in Ethernet standard body

contributions, so one can directly compare them with the results presented in this thesis. It is not the same as the E_b/N_0 commonly used in information theory.

Calculating $P(d_k|b_k)$

For the PAM-4 channel in the presence of AWGN with zero mean and variance $= \sigma^2$, the probability of receiving the symbol d_k given that b_k was sent, $P(d_k|b_k)$, can be calculated using the area under the tail of the Gaussian probability density function (PDF). For example, Figure 4.5 shows the probability of receiving symbols $d_k \in \{0, 1, 2, 3\}$, given that $b_k = 2$ is sent. The probability density function for r_k is shown with the Gaussian curve (not to scale) and is divided into sections that are mapped to $d_k \in 0, 1, 2, 3$ by the slicer. The tail distribution function of the standard normal distribution,

$$Q(x) = \frac{1}{\sqrt{2\pi}} \int_x^\infty e^{\frac{-u^2}{2}} du$$

can be used to calculate the probability of receiving the symbol d_k given that $b_k = 2$ was sent, as shown in the right column in Figure 4.5. For these calculations, the SNR is 17 dB, corresponding to a noise variance of $\sigma^2 = 0.01$, a value that results in a pre-FEC BER of around 6×10^{-4}.

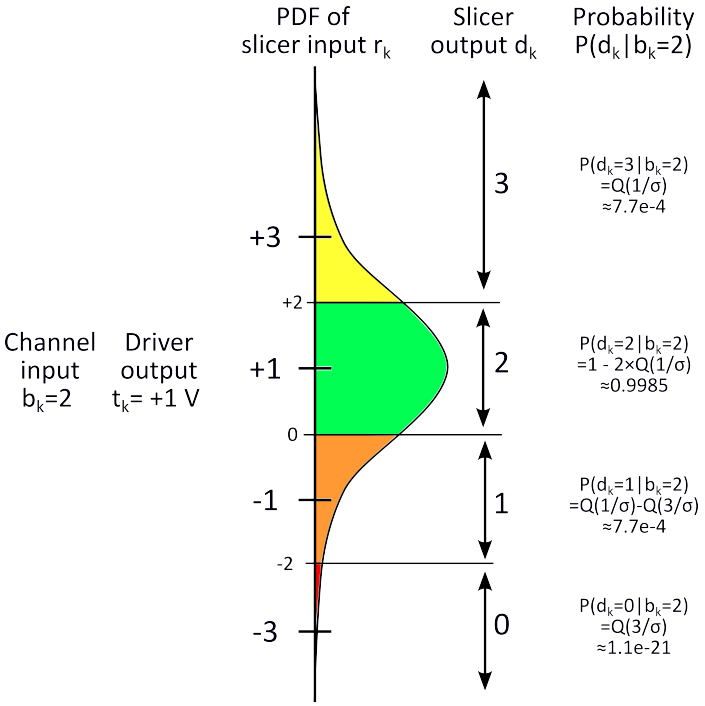

Figure 4.5: PDF of r_k for AWGN channel given $b_k = 2$ is transmitted.

Figure 4.5 shows that for an AWGN channel with 17 dB SNR, the most likely outcome is no PAM-4 error. Errors between neighbouring PAM-4 symbols are possible and constitute almost all error events. Due to the exponential decay of the Gaussian PDF, errors across two PAM-4 symbols (i.e. $b_k = 2$ is sent and $d_k = 0$) is received are with extremely low probability ($\approx 10^{-21}$). In the context of the FPGA simulations that will only transmit on the order of 10^{15} PAM-4 symbols maximum, this event is unlikely ever to occur.

Hardware Implementation

Uniform random number generators (URNGs) are the FPGA platform's primary tool for realizing random processes. Like PRBS patterns, URNGs can be implemented with shift register circuits to give pseudo-random uncorrelated uniform numbers. In this work, the Tausworthe algorithm was chosen for it's hardware simplicity and suitable pseudo-random properties [24]. A 64-bit Tausworthe URNG is used to produce a new random 64-bit sample each clock cycle. It has a repetition period of approximately 2^{176}, far longer than any simulation will be. This Tausworthe URNG requires three 64-bit numbers as a random seed. Similar to the PRBS generation, when parallel cores are used, each Tausworthe URNG is initialized with different random seed values, so the URNG outputs are not correlated.

The AWGN channel probabilities $P(d_k|b_k)$ for $d_k, b_k \in \{0, 1, 2, 3\}$ are calculated for a given SNR in a Matlab program using the method illustrated in Figure 4.5 for a range of SNRs of interest. Each conditional probability value between 0 and 1 is scaled to a 64-bit unsigned integer representation, where 0 represents a zero-probability event, and 2^{64} represents an event with probability 1. This 64-bit representation introduces a minimal $< 2^{-64} \approx 5 \times 10^{-20}$ amount of rounding error in the probability values. However, the dominant source of error is due to the precision of floating point numbers in the Matlab program used to calculate them. The float precision introduces an error of $\pm 10^{-15}$ in the PAM-4 symbol error rate. While this may seem significant given that the platform's goal is to measure very low CERs, this amount of error is insignificant to the PAM-4 symbol error rate, which is typically on the order of 10^{-3} to 10^{-5}. Statistical analysis confirms that this does not significantly affect the CER.

The 64-bit probability values are runtime-programmable. At the beginning of a simulation, the probability values for a given SNR are loaded into registers in the AWGN channel module for use during the simulation. Algorithm 3 describes how the AWGN channel model on the FPGA platform produces output PAM-4 symbols d_k.

Algorithm 1 Hardware implementation of PAM-4 AWGN error injection channel model.

Initial Information:

$P(d_k|b_k)$: Channel probabilities in 64-bit representation

Algorithm:

for Each incoming transmitted symbol b_k **do**
  Generate a 64-bit random sample R
  if $R < P(0|b_k)$ **then**
    Output $d_k = 0$
  else if $R < P(0|b_k) + P(1|b_k)$ **then**
    Output $d_k = 1$
  else if $R < P(0|b_k) + P(1|b_k) + P(2|b_k)$ **then**
    Output $d_k = 2$
  else
    Output $d_k = 3$
  end if
end for

Simulation Results

Figure 4.6 shows a plot of SNR vs. CER for the AWGN channel described in this section. The accuracy is confirmed with the analytically-determined CER from the statistical model presented in [8], shown with the blue line. The system-level diagram of the BER Simulation IP programmed on the FPGA is at the top of Figure 4.6. A similar system-level diagram is included for all figures that present FPGA-simulated CERs in the remainder of the thesis.

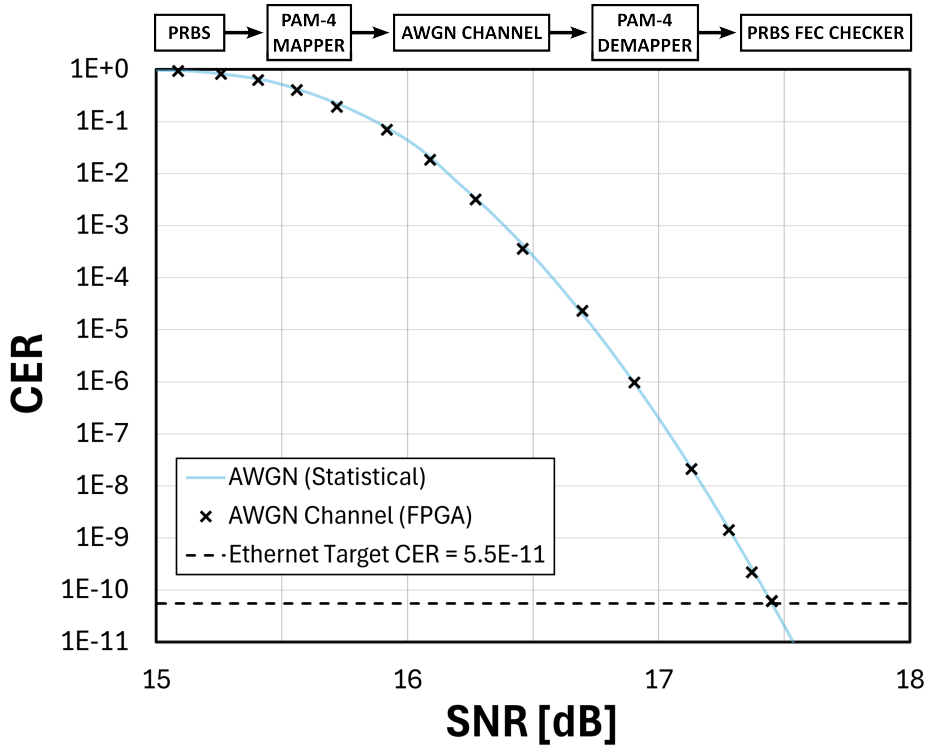

Figure 4.6: SNR vs. CER for AWGN channel.

Figure 4.6 shows that an SNR greater than 17.4 dB is needed to achieve the Ethernet standard's target CER for the PAM-4 AWGN channel. This simulation was run with 200 parallel cores, each running at 150 MHz, for a simulation speed of 30 Gb/s. The hardware utilization is summarized in Table 4.4. Only the look-up table and flip-flop utilization are presented, which are the critical resources for the FPGA platform.

Module Name	**# Look-up Tables**	**# Flip Flops**
PRBS	32	65
PAM-4 Mapper	3	5
AWGN Channel	251	0
PAM-4 Demapper	3	5
FEC Checker	720	831
Total	1009	906
× **200 Cores**	202,000	181,000
Processor, Driver & Memory	5,000	5,000
Grand Total	207,000 (38%)	186,000 (17%)

Table 4.4: FPGA utilization for simulation of the AWGN error injection channel using 200 parallel cores at 150 MHz for a simulation speed of 30 Gb/s.

The first five rows from Table 4.4 show a breakdown of the hardware utilization for one simulation core. The numbers in these rows represent an average because the actual number of look-up table and flip-flop resources will typically vary between parallel cores. This system requires relatively low hardware resources due to the simple one-link architecture and error-injection channel model. The FEC checker is the module with the highest hardware utilization due to its 64-bit counters and logic to interpret FEC statistics and interleaving. Because this system is small, the speed is limited by timing constraints rather than hardware resources, which is why only 38% of LUT resources are used.

4.5.2 Error Propagation Factor Channel

The error propagation factor (EPF) channel model is a two-state Markov model for modelling DFE and MLSD error propagation in Ethernet standard contributions [3]. Figure 4.7 shows a state diagram of the Markov model.

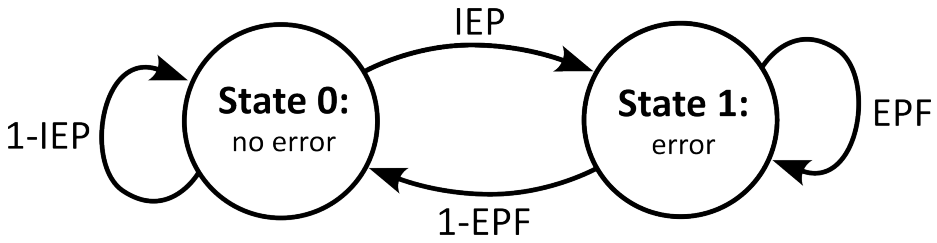

Figure 4.7: EPF Markov model.

Visiting the non-error state at time k signifies no PAM-4 symbol error at that time index, and visiting the error state signifies that there is. Given that the current state is the non-error state,

the probability of transitioning to the error state is the initial error probability (IEP). Given that the current state is the error state, the probability of a subsequent PAM-4 symbol error is called the error propagation factor (EPF). For consecutive PAM-4 symbol errors, the error sequence alternates between +1 and -1 to mimic the behaviour of DFE and MLSD error propagation, so precoding is effective against the artificial errors injected by the EPF channel model. These two parameters, IEP and EPF, entirely define the behaviour of the EPF channel model.

Hardware Implementation

The IEP and EPF parameters are converted to a 64-bit representation over a range of interest, as described in Section 4.5.1. The EPF module contains a register to represent the Markov state. A 64-bit URNG realizes the random transitions between the error and non-error states. The output d_k is assigned to be the same as the input b_k in the non-error state and is perturbed by ± 1 in the error state, as shown in Algorithm 2.

Algorithm 2 Implementation of EPF Channel

Initial Information:
IEP, EPF: State transition probabilities in 64-bit representation
Initialize state variable: state = 0
Initialize error sign variable: sign = 1

Algorithm:
for Each incoming transmitted symbol b_k **do**
  Generate a 64-bit random sample R
  if state == 0 **then** ▷ No error state
    Output $d_k = b_k$
    if $R <$ IEP **then**
      state $\leftarrow 1$ ▷ Transition to error state
    end if
  else if state == 1 **then** ▷ Error state
    Output $d_k = (b_k +$ sign) mod 4
    sign $\leftarrow -$sign
    if $R < 1 -$ EPF **then**
      state $\leftarrow 0$ ▷ Transition to no error state
    end if
  end if
end for

Simulation Results

Figure 4.8 shows pre-FEC BER vs. IEP. Two channels are considered, having EPF = 0 and EPF = 0.75.

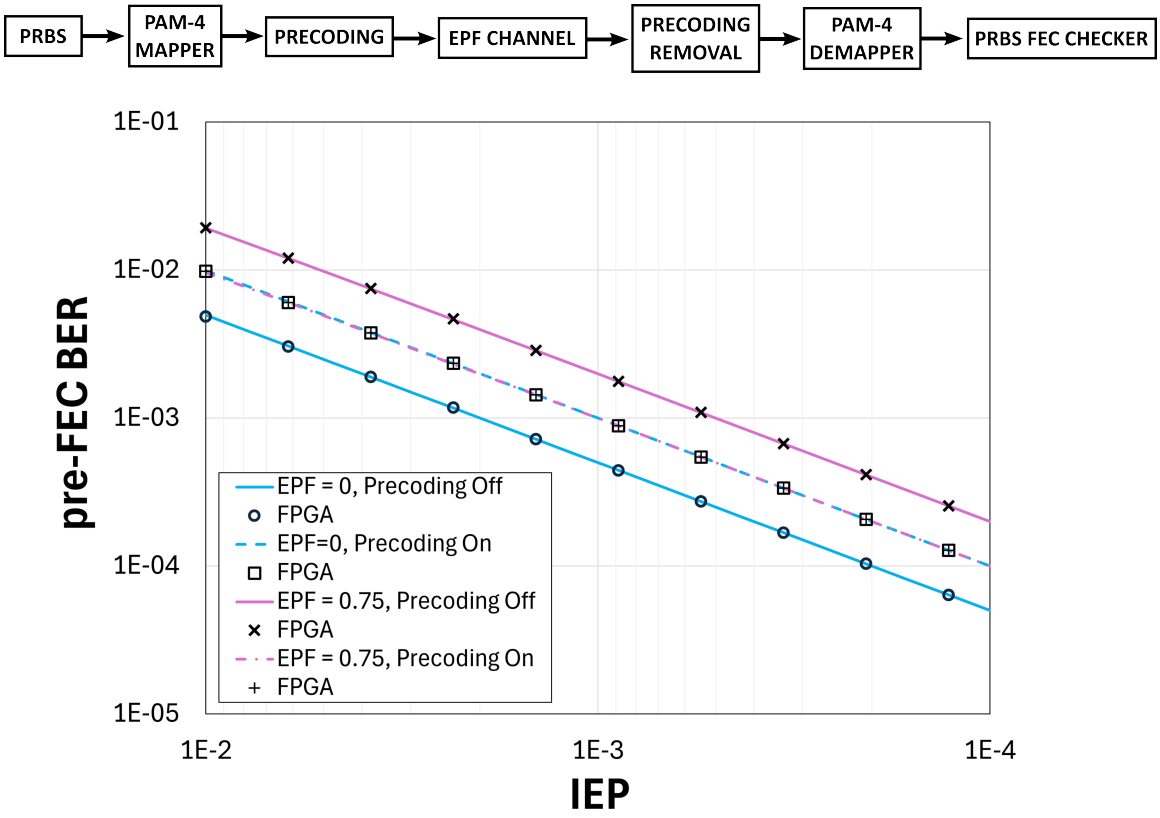

Figure 4.8: Pre-FEC BER vs. IEP for EPF channels with and without precoding.

For the channel with EPF = 0 and precoding off, the pre-FEC BER is about half the initial PAM-4 symbol error probability because each PAM-4 symbol error only produces one-bit error. In contrast, with precoding, the pre-FEC BER is the same as the IEP because the precoding turns each single PAM-symbol error into two PAM symbol errors, each containing one bit error. The pre-FEC BER is much higher for the link with EPF = 0.75 and precoding off because each initial error may result in a burst. However, with precoding on, it eliminates most errors in a burst, leaving only two PAM-symbol errors 4.2. Note that for both EPF levels with precoding on, each initial error results in two pre-FEC bit errors; therefore, they have the same pre-FEC BER for the same IEP.

Figure 4.9 shows the CER vs. IEP for the same channel conditions.

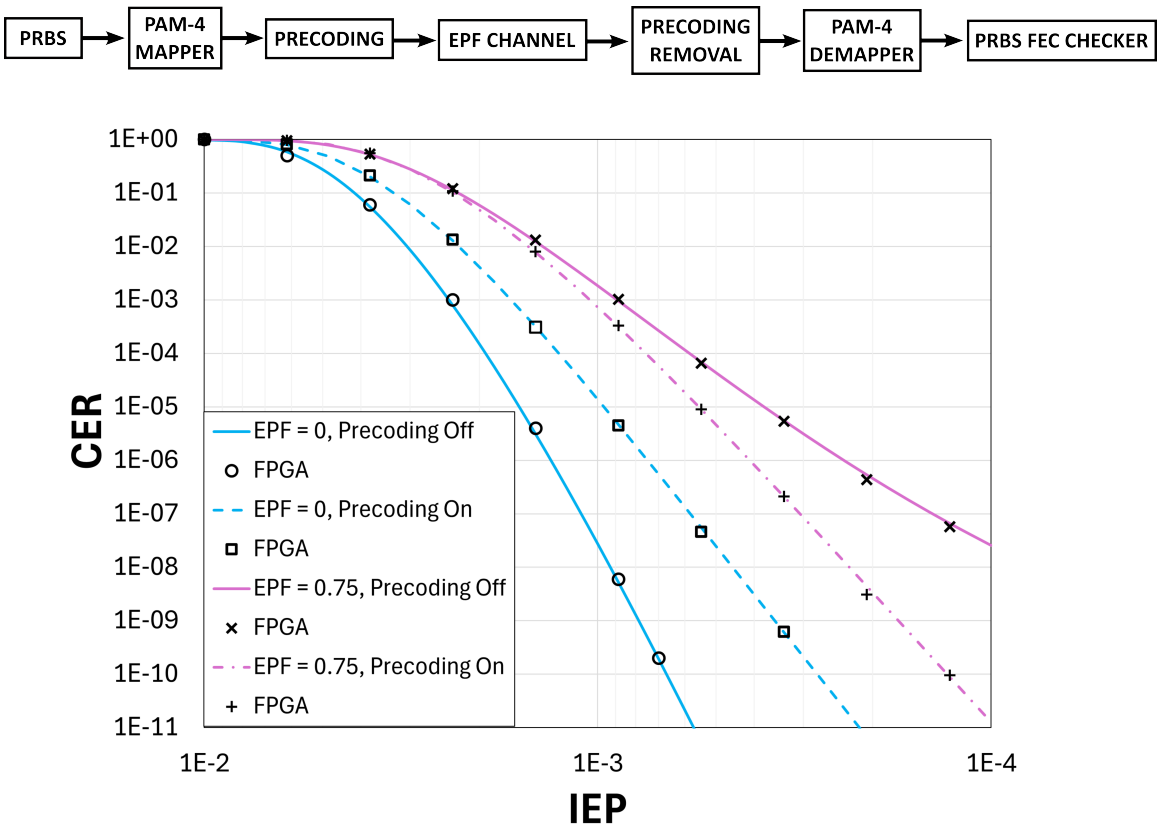

Figure 4.9: CER vs. IEP for EPF channels with and without precoding.

The link with EPF = 0 and precoding off performs best because it has the lowest pre-FEC BER. The link with EPF = 0.75 and precoding off has the highest CER due to its highest pre-FEC BER. Also, the burst errors degrade performance, which can be seen in the slope of the IEP vs. CER relationship lowering at low IEP in an effect known as the "error floor". Despite the two EPF levels having the same pre-FEC BER with precoding on, the link with EPF = 0 has a lower CER due to the spacing between PAM-symbol errors. For a single PAM-4 symbol error, the precoding will insert an additional PAM-4 symbol error directly after the first error, as shown in Table 4.3, making it likely that the two PAM-4 symbol errors only corrupt one 5-symbol-long FEC symbol. In contrast, the precoding removes errors from the middle of the burst, leaving two errors that may be further spaced apart, making it more likely for a single initial error event to end up corrupting multiple FEC symbols.

Figure 4.10 shows a link with EPF = 0.75 and precoding off, with no interleaving, 2-way, and 4-way block interleaving. Although all three links have the same pre-FEC BER, the interleaving significantly improves the CER for links with more interleaving by splitting up bursts of consecutive FEC symbol errors among different codewords. Note that block interleaving would not affect the CER of a channel with EPF = 0.

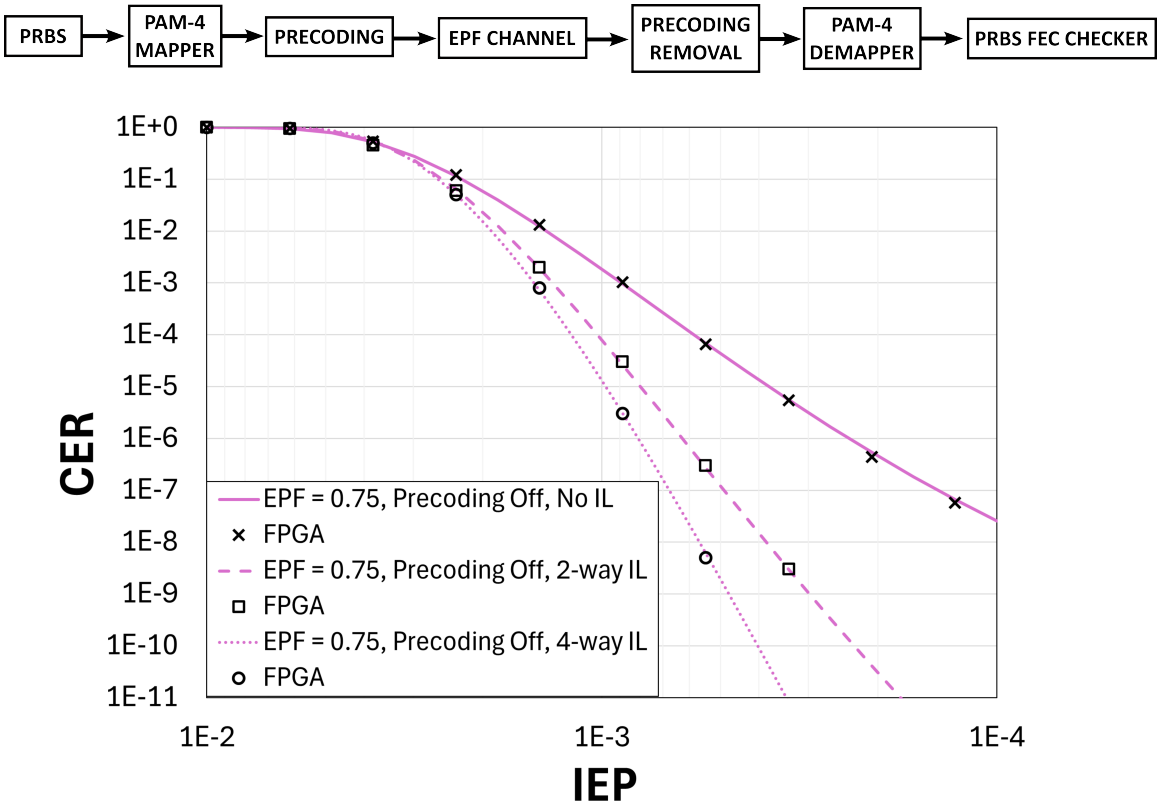

Figure 4.10: CER vs. IEP for EPF channels with block interleaving.

The simulation results presented in Figures 4.8 to 4.10 were also run with 200 parallel cores, each running at 150 MHz, for a simulation speed of 30 Gb/s. The hardware utilization is summarized in Table 4.5.

Module Name	**# Look-up Tables**	**# Flip Flops**
PRBS	32	65
PAM-4 Mapper	3	5
Precoder	4	5
EPF Channel	135	0
Precoding Removal	3	5
PAM-4 Demapper	3	5
FEC Checker	720	831
Total	900	916
× 200 Parallel Cores	180,000	183,000
Processor, Driver & Memory	5,000	5,000
Grand Total	185,000 (34%)	185,000 (18%)

Table 4.5: FPGA utilization for EPF simulation using 200 parallel cores at 150 MHz for a simulation speed of 30 Gb/s.

4.5.3 $1 + \alpha D$ Channel with DFE

Although the EPF model can model DFE error propagation, it is not perfectly accurate. While the EPF is only a two-state Markov model, it has been shown that a 4^{2n}-state Markov model is needed to precisely model the errors introduced by an n-tap DFE-based electrical receiver with PAM-4 signalling in the presence of AWGN [8]. Because a one-tap DFE is commonly used for 200 Gb/s electrical links, a more complex Markov model that precisely models the error propagation introduced by one is included in this FPGA platform.

Figure 4.11 (a) shows a block diagram of a $1 + \alpha D$ channel, under the presence of AWGN, equalized with a zero-forcing DFE that completely cancels the ISI. In this model, the channel response and the response of any transmitter pre-emphasis, receiver analog equalization, and feed-forward equalization are lumped into one system with a finite impulse response $1+\alpha D$. This $1+\alpha D$ response means that the channel's output before adding noise is calculated as $1t_k + \alpha t_{k-1}$. This channel has "channel memory" $\mu = 2$ because each output depends on the channel input from the two most recent timesteps. AWGN noise samples are added to the signal after $1 + \alpha D$ the response and before the DFE input. Figure 4.11 (b) depicts the hardware implementation of an equivalent Markov model that injects errors with the same statistical properties as in Figure 4.11 (a).

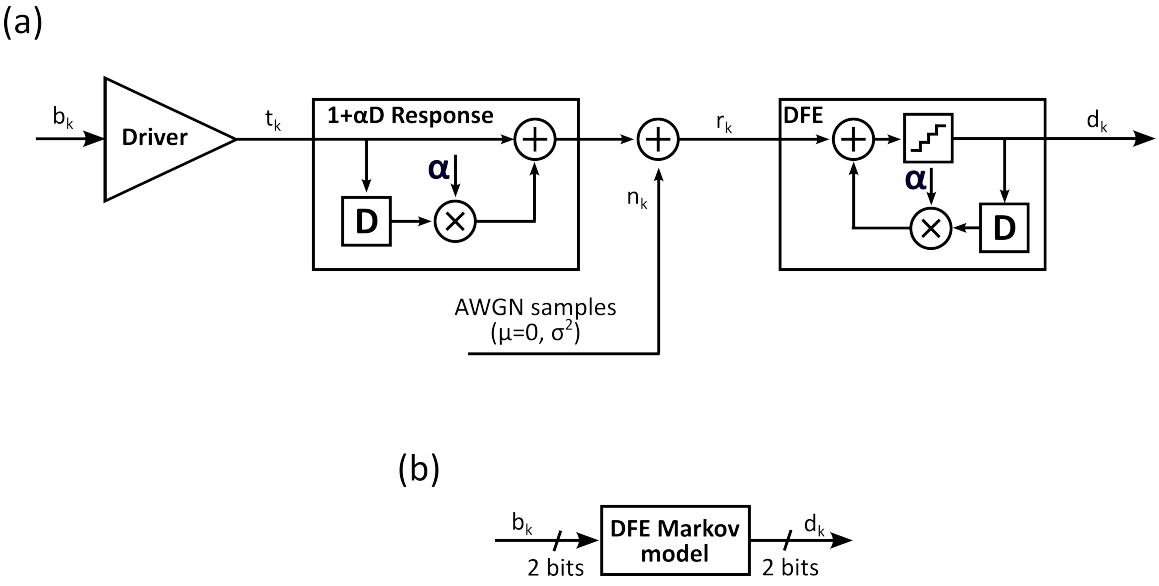

Figure 4.11: (a) $1+\alpha D$ ISI channel in the presence of AWGN with zero-forcing DFE. (b) Hardware implementation of equivalent Markov model.

Hardware Implementation

The derivation of the $2^4 = 16$ states needed to model errors introduced by this 1-tap zero-forcing DFE is omitted from this thesis, as this is work done by Ming Yang and presented in Section 2.2 of his thesis [11]. The transition probabilities are determined from the α value and link SNR and are converted to a 64-bit representation. A similar algorithm to Algorithm 2 is used to realize the state transitions and produce channel outputs d_k based on the channel inputs b_k and the current Markov state.

Simulation Results

The simulation results in Figure 4.12 use a $1 + \alpha D$ with $\alpha = 0.8$ to model a long-reach electrical channel with significant ISI. The error propagation from this channel illustrates the benefits of precoding and interleaving. Figure 4.12 shows this channel's SNR vs. CER relationship with and without precoding and interleaving. Note that for the SNR reported on the x-axis, the signal power P_{sig} is still 5, corresponding to the transmitted symbol intensity values $\{-3, -1, 1, 3\}$, even though the $1 + \alpha D$ ends up amplifying the signal, increasing the power per symbol due to the added ISI. Like the EPF channel model, the error propagation caused by the $1+0.8D$ channel with DFE causes significant degradation to the CER, but precoding and interleaving mitigate this. The statistical model presented in [8] verifies the simulation results' accuracy down to low CER levels. The FPGA simulation closely matches the statistical model's results, which are shown with solid lines.

The simulation results presented in Figures 4.12 were run with 200 parallel cores, each running at 150 MHz for a simulation speed of 30 Gb/s. Table 4.6 summarizes the hardware utilization for this simulation. The DFE channel model is significantly more complex than the others presented in this chapter and requires more hardware resources.

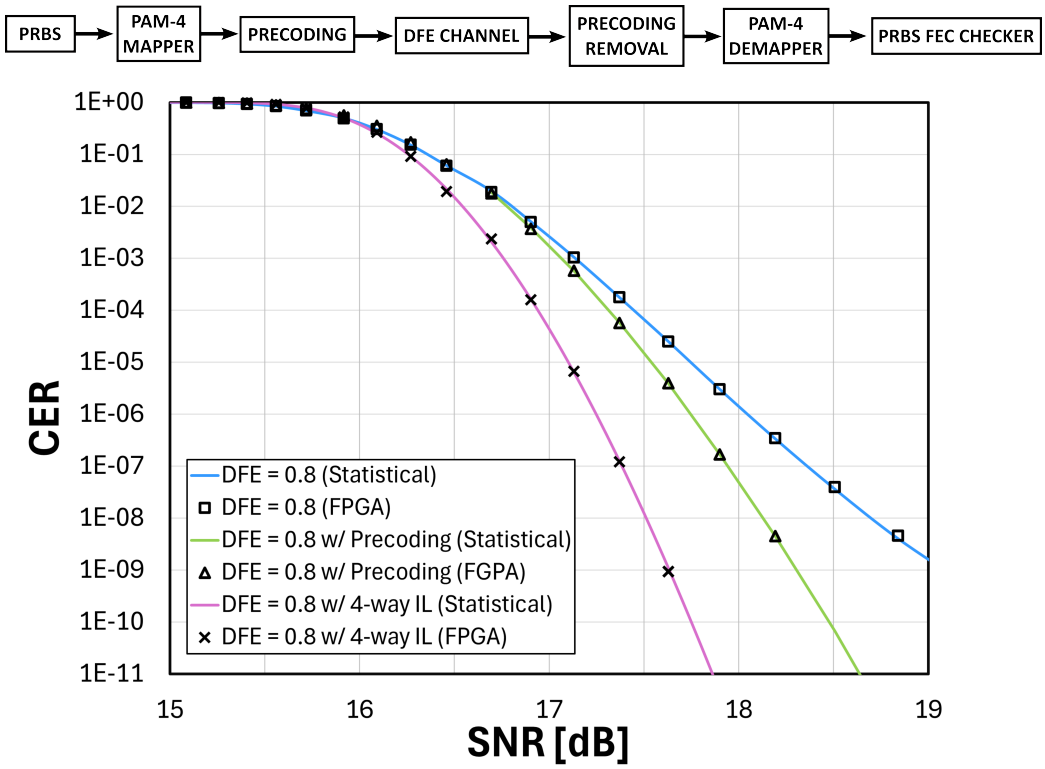

Figure 4.12: CER vs. SNR for $1 + \alpha D$ channel with AWGN and zero-forcing DFE.

Module Name	**# Look-up Tables**	**# Flip Flops**
PRBS	32	65
PAM-4 Mapper	3	5
Precoder	4	5
DFE Channel	955	217
Precoding Removal	3	5
PAM-4 Demapper	3	5
FEC Checker	720	831
Total	1,720	1,133
× **200 Parallel Cores**	344,000	227,000
Processor, Driver & Memory	5,000	5,000
Grand Total	349,000 (65%)	232,000 (22%)

Table 4.6: FPGA utilization for DFE simulation using 200 parallel cores at 150 MHz for a simulation speed of 30 Gb/s.

Chapter 5

Analog Channel and Receiver Models

Chapter 4 presented channel models that artificially inject errors into a stream of PAM-4 symbols. Although these models have a relatively low hardware complexity, they have some shortcomings. Firstly, they cannot precisely model an MLSD-based receiver, which is a popular choice for 200 Gb/s applications. Secondly, they are not well-suited to produce the soft reliability information required for soft-decision inner FEC decoding because they do not include any representation of an analog signal level or time-domain noise signal.

This chapter presents "analog" channel models that are included in the FPGA platform to address these shortcomings. The critical difference is that instead of simply injecting artificial errors into a stream of PAM-4 symbols, each PAM-4 symbol is represented as an analog level in a fixed-point signed decimal representation. This baud-rate signal may include a $1 + \alpha D$ response that adds ISI. A time-domain noise signal in fixed-point decimal representation is generated on the FPGA and added to the signal. The output symbols d_k may be recovered by a simple ideal slicer or an MLSD equalizer if ISI is present in the channel. Next, Chapter 7 builds on this analog channel model by introducing models of soft-output receivers that generate the reliability information required for soft-decision decoding.

5.1 Analog AWGN Channel Model

This section presents a new analog model of the AWGN channel on the FPGA platform. Figure 5.1 (a) shows a block diagram of an ideal PAM-4 AWGN channel, and (b) shows the hardware implementation on the FPGA. The PAM-4 symbols b_k are mapped to the transmitted signal levels t_k^{bin} in an m-bit signed decimal representation. The superscript "bin" denotes that t_k^{bin} is the FPGA platform's binary representation of the analog signal level t_k. The Gaussian noise samples n_k^{bin} are then added to t_k^{bin}. The signal is clipped to maintain the m-bit resolution for the received noisy samples r_k^{bin}. An ideal slicer outputs the most likely output signal d_k.

A Verilog parameter m controls the number of bits representing the analog signal. Increasing m produces a finer resolution that may match better to a statistical or software model that assumes a floating point representation of analog signals. However, increasing m comes at the cost of additional

hardware complexity. A difference between this model and a real 200 Gb/s receiver is that the FPGA platform has no analog-to-digital converter (ADC). Instead, the resolution m is assumed to be the same as the ADC resolution of interest. This way, the same signal resolution can be used throughout the model, reducing hardware complexity. An eight-bit signal resolution is chosen for a realistic ADC resolution in a 200 Gb/s receiver, and so $m = 8$ is used for all simulation results shown in this thesis.

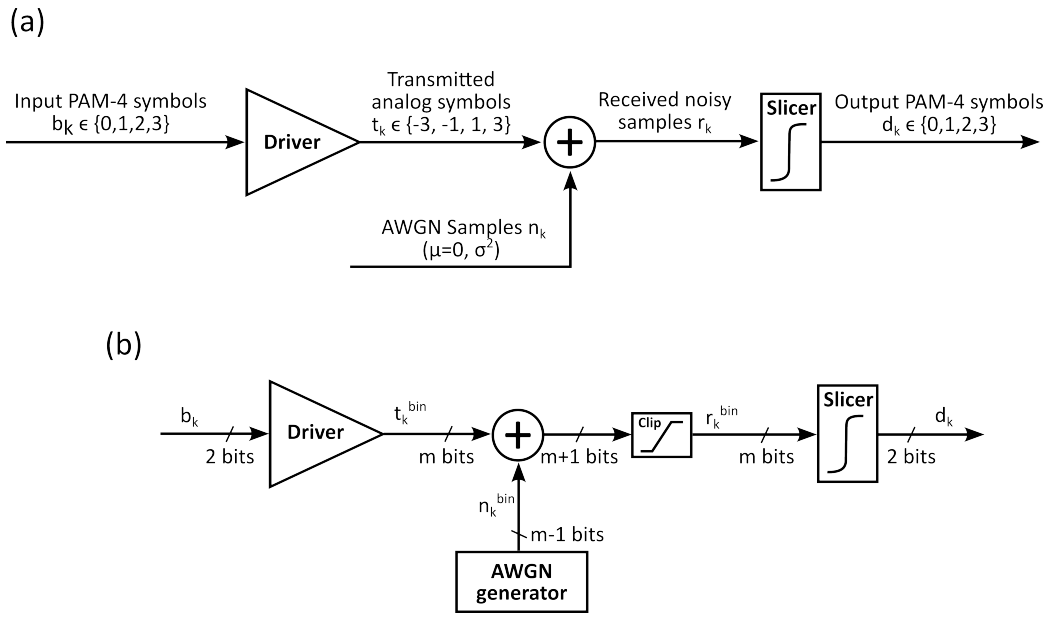

Figure 5.1: (a) Analog PAM-4 channel with AWGN. (b) FPGA implementation.

5.1.1 PAM-4 Mapping

The PAM-4 mapper takes in PAM-4 symbols $b_k \in \{0, 1, 2, 3\}$, and outputs a corresponding signed decimal representation of the voltages levels $t_k \in \{-3, -1, +1, +3\}$. A scaling factor s determines the signed decimal numbers corresponding to each PAM-4 voltage level. The eight-bit signal resolution can represent integers from -128 to 127. The scaling factor should be chosen so that the signal at the channel output can fit in the range of signed decimal numbers with some margin for added noise. The simulations in this thesis use $s = 24$, and the mapping is shown in Table 5.1.

PAM-4 Symbol	Signal Level	Decimal Representation
0	-3	-72
1	-1	-24
2	+1	24
3	+3	72

Table 5.1: PAM-4 symbol mapping with scaling factor $s = 24$.

For the AWGN channel, t_k^{bin} will be at maximum 72 and minimum -72, leaving a margin of 56

for additional noise to be added to the signal before overflowing the eight-bit range. With a $1+0.5D$ channel response, t_k^{bin} is amplified to ± 108, still leaving a margin of 20.

5.1.2 AWGN Generator

There are multiple approaches for generating AWGN samples on an FPGA; for example, the Box-Muller method is a commonly used [25]. However, this method is not well-suited for in-runtime programming of the noise variance, which is essential for the platform to efficiently sweep over a range of SNR levels. Instead, the AWGN generator on this FPGA platform uses a pre-computed look-up table that contains the probability for each noise sample, and a URNG is used to randomly choose noise samples with the correct frequency. This approach is similar to the error injection AWGN model presented in Section 4.5.1.

To find the probability for each possible noise sample, the PDF of a Gaussian distribution for the SNR of interest is first calculated. Since the signed decimal representation of the signal at the channel output is scaled by s, the noise standard deviation is scaled by the same factor. Figure 5.2 shows Gaussian PDFs for 15 and 18 dB SNR, scaled to match the signed integer representation with $s = 24$. This 15-18 dB level is a realistic range for 200 Gb/s wireline channels.

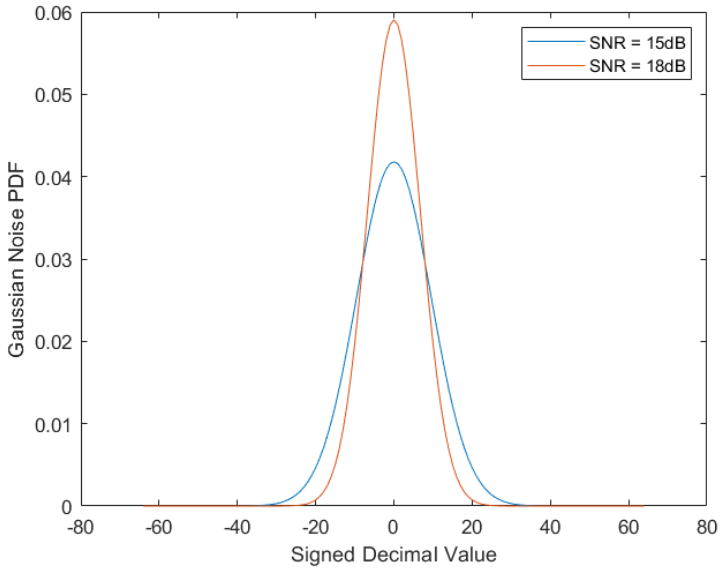

Figure 5.2: Gaussian noise PDF scaled by $s = 24$ for 15 and 18 dB SNR.

The continuous PDF is discretized into bins of length one to find the probability of each possible noise sample $n_k^{bin} = \{\dots, -2, -1, 0, +1, +2, \dots\}$. The probability of an integer noise sample n_k^{bin} being generated from the scaled Gaussian noise PDF $f_{noise}(x)$ can be calculated as

$$P(n_k^{bin} = i) = \int_{i-0.5}^{i+0.5} f_{noise}(x)dx$$

.

For SNRs greater than 15 dB, the noise PDF decays to a very small probability before reaching ± 64. For example, with 15 dB SNR, $P(n_k^{bin} = 63) = 5.8 \times 10^{-11}$. Because of this, the FPGA implementation of the noise generator used for the simulations in this thesis represents the noise

samples n_k^{bin} with a seven-bit number from -63 to 63 without much loss of accuracy. However, the number of bits used to represent noise samples is a customizable Verilog parameter that may be increased for better accuracy. To calculate the probabilities of the noise samples at ± 63, the bounds of integration are set to infinity to ensure that the total quantized PMF sums to 1:

$$P(n_k^{bin} = 63) = \int_{63-0.5}^{\infty} f_{noise}(x)dx$$

.

By confining the noise samples to a seven-bit resolution, no noise samples greater than 63 or less than -63 are generated. However, for reasonable SNR levels, statistical analysis confirms that this has insignificant affect on the pre-FEC BER or CER of the system.

The discretized noise PMF is scaled to 64-bit values and loaded into the AWGN module as a runtime-programmable setting. Only 64 probabilities are stored in the noise generator module, corresponding to $P(n_k = i), i \in \{0, 1, \ldots, 63\}$, rather than the full range of $i \in \{-63, -62, \ldots, 62, 63\}$ since the Gaussian is symmetrical ($P(n_k = i) = P(n_k = -i)$). A 64-bit URNG is used to pick the magnitude of each noise sample, and an uncorrelated binary random number generator determines the sign, as described in Algorithm 3. Although Algorithm 3 uses a long chain of if/else statements, the hardware-synthesized design executes the mapping from URNG sample to Gaussian sample in only one clock cycle. For a more hardware-efficient implementation of this mapping, the alias method can also be used [26].

Algorithm 3 Implementation of AWGN sample generator.

Initial Information:

$P(n), n \in \{0, \pm 1, \pm 2, \ldots, \pm 63\}$: Noise PMF in 64-bit representation

Algorithm:

for each clock cycle **do**
  Generate a 64-bit random sample R
  Generate a 1-bit random sample R_b
  if $R < P(0)$ **then**
    Output $n = 0$ ▷ Zero noise
  else if $R < P(n = 0) + P(n = \pm 1)$ **then** ▷ Noise magnitude = 1
    if $R_b = 0$ **then**
      Output $n = +1$ ▷ Noise sign is positive
    else if $R_b = 1$ **then**
      Output $n = -1$ ▷ Noise sign is negative
    end if
    ⋮ ▷ Repeat else if blocks for noise magnitudes 2 to 62

  else if $r < P(n = 0) + P(n = \pm 1) + P(n = \pm) + \ldots P(n = \pm 63)$ **then**
    if $R_b = 0$ **then**
      Output $n = +63$
    else if $R_b = 1$ **then**
      Output $n = -63$
    end if
  end if
end for

Despite setting s so that there is some range to add noise sample n_k^{bin} to the signal without overflowing the 8-bit resolution, it is still possible that the noise and signal will sum to be greater than 127 or less than -127. Therefore, this sum is clipped to stay within this range. Simulation results show that this clipping has little effect on the BER compared to a software simulation that represents analog signal levels with a floating point number.

5.1.3 Slicer

A hard-decision PAM-4 slicer recovers the PAM-4 symbols from the received noisy samples. It calculates the absolute value of the distance of each sample to each of the transmitted symbol levels and outputs the symbol with the lowest distance. A channel output may be equidistant from two of the transmitted symbol levels. In this case, the slicer's tie-breaking rule is always to output the smaller PAM-4 symbol (e.g. if the channel output is equidistant from $b_k = 0$ and $b_k = 1$, the slicer outputs $d_k = 0$). Because all four PAM-4 symbols are equally likely to be transmitted, this results in an error half the time when the channel output is equidistant from two PAM-4 symbols.

Due to the quantization of the Gaussian PDF, this tie-breaking rule is the source of some discrepancy with the statistical model. Consider an ideal, continuous probability distribution of channel outputs, shown in Figure 5.3. Each of the four Gaussian curves has area $\frac{1}{4}$ because each

symbol is equally likely to be transmitted. The probability shown by the red-shaded region denoted P_{tail} is the probability of a symbol error between neighbouring PAM symbols. This red error probability is comprised of six Gaussian "tails". Therefore, the PAM-4 symbol error probability is equal to $\frac{6}{4}\ P_{tail}$.

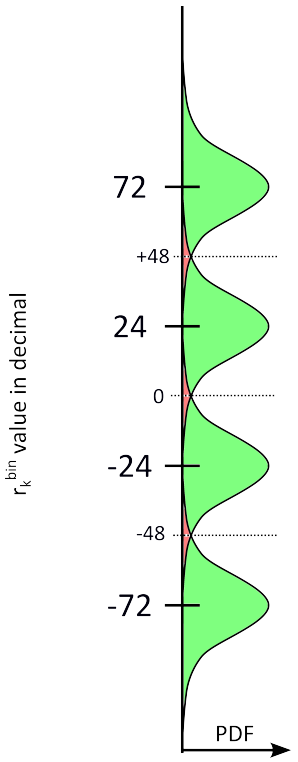

Figure 5.3: PDF of noisy samples r_k^{bin} at AWGN channel output.

The statistical model assumes one of these red tails is exactly $\int_{24}^{\infty} \mathcal{N}(\mu = 0, \sigma)$, as shown in Figure 5.4.

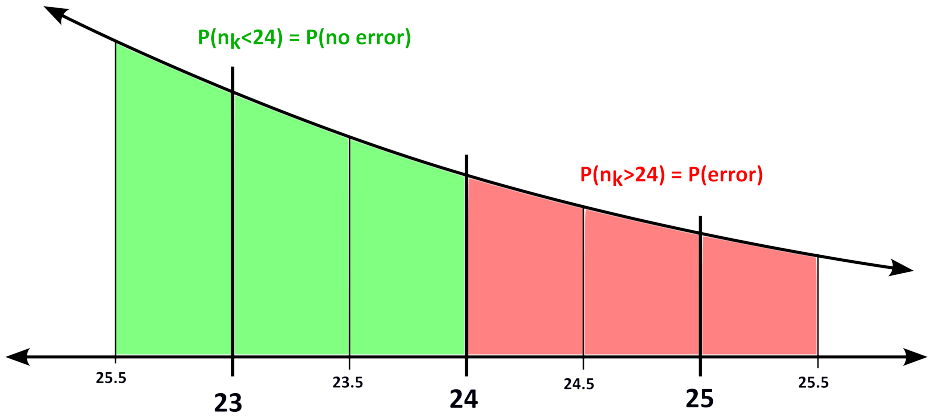

Figure 5.4: Tail of Gaussian distribution responsible for PAM-4 symbol errors.

The quantized hardware model has a slightly different symbol error probability for the same SNR level. Any noise samples less than or equal to 23 falls outside the tail, and any noise samples greater

than or equal to 25 are included in the tail. However, for the noise sample 24, the probability is divided inside and outside the red tail due to the slicer's tie-breaking rule, as shown in Figure 5.5. Because the Gaussian probability distribution has a slope, this quantized approximation results in a small amount of additional error probability in the decimal hardware model. As a result, there is a slight increase in the overall symbol error rate. Although this discrepancy is very small (about 2%) for the pre-FEC BER, it can result in a significant margin of difference between the statistical model and the quantized FPGA model for the CER, as shown in the following simulation results.

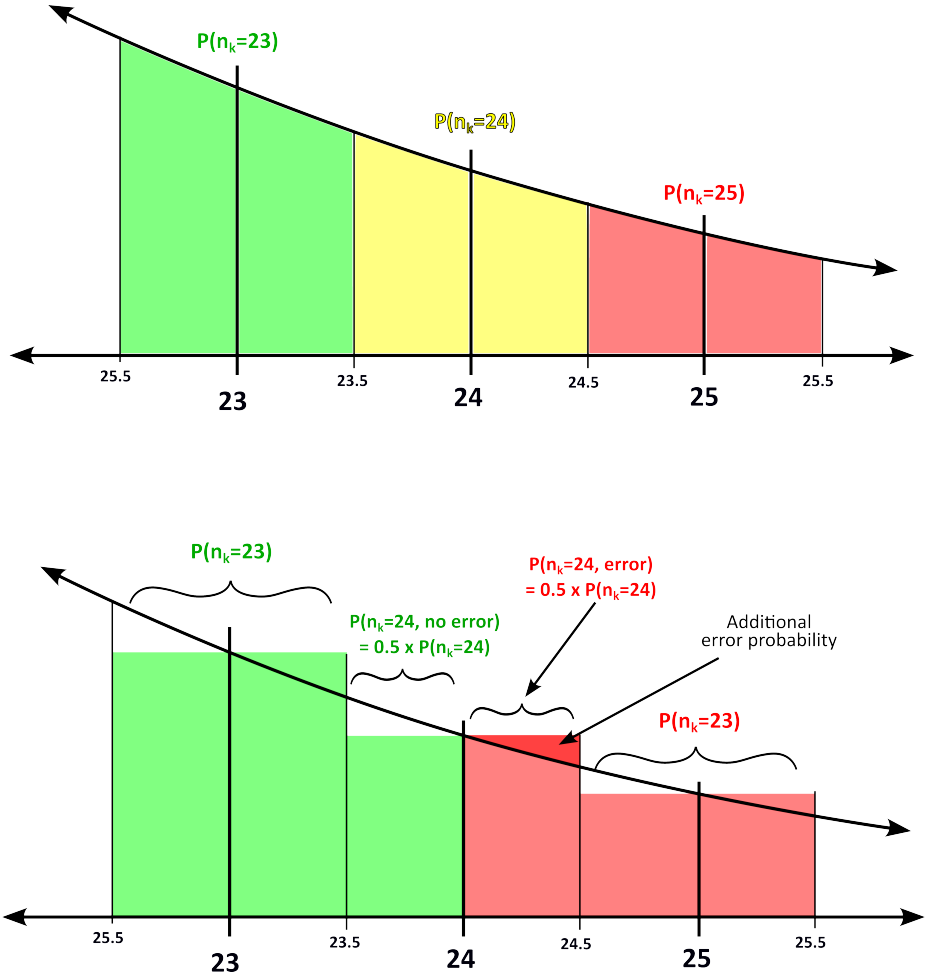

Figure 5.5: Added PAM-4 error probability in FPGA model due to quantization of signal and slicer tie-breaking.

Simulation Results

Figure 5.6 shows the SNR vs. CER for the analog AWGN channel model. The FPGA simulation results report a slightly higher CER than the statistical model due to the quantization of the noise PDF and the slicer's tie-breaking rule.

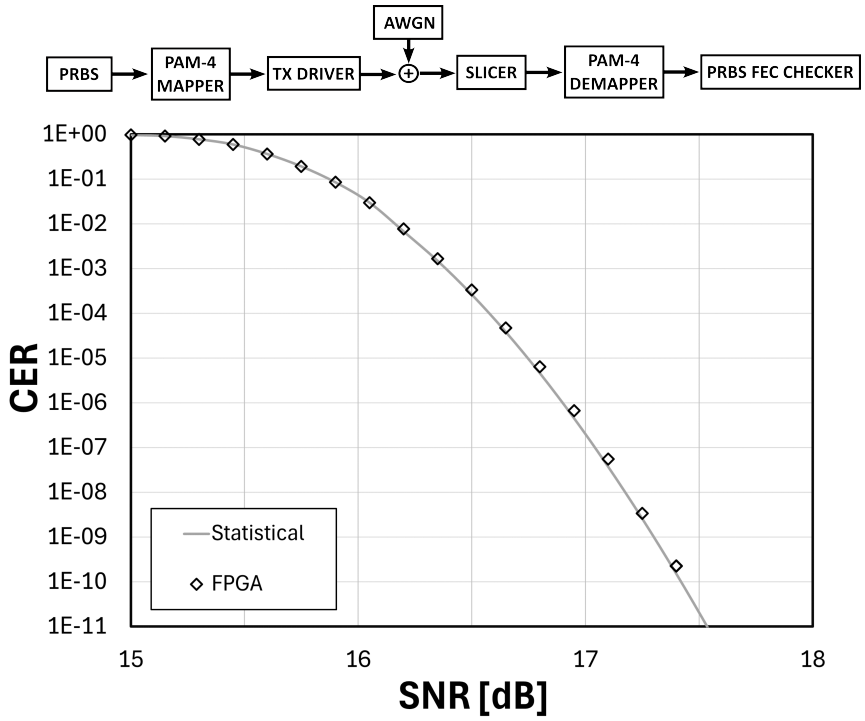

Figure 5.6: SNR vs. CER for analog AWGN channel model.

Despite this discrepancy between the SNR and pre-FEC BER, the errors introduced by the analog AWGN model are still random and uncorrelated, so the relationship between pre-FEC BER and CER matches the statistical model, as shown in Figure 5.7.

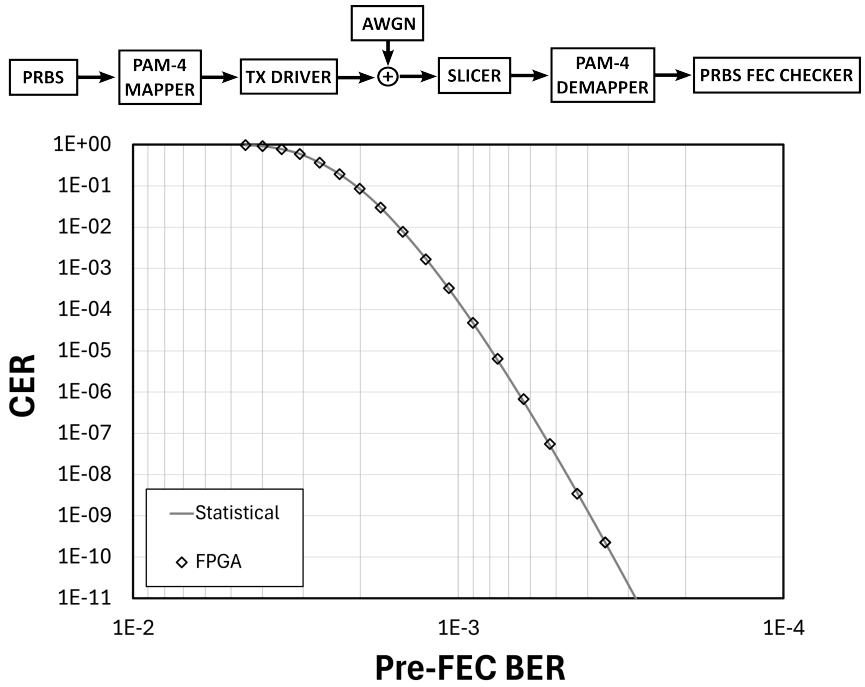

Figure 5.7: Pre-FEC BER vs. CER for analog AWGN channel model.

Table 5.2 shows the hardware utilization for the analog AWGN channel. This analog channel model takes significantly more hardware resources than the error injection model. This is because the AWGN noise generator holds sixty-four 64-bit re-programmable values to determine the sample probabilities. The simulations in this section use 60 cores at 150 MHz for a speed of 9 Gb/s.

Module Name	**# Look-up Tables**	**# Flip Flops**
PRBS	32	65
PAM-4 Mapper	3	5
Driver	3	4
AWGN Generator	2183	4280
Slicer	174	3
PAM-4 Demapper	3	5
FEC Checker	720	831
Total	3118	5193
× 60 Parallel Cores	187,000	312,000
Processor, Driver & Memory	5,000	5,000
Grand Total	192,000 (36%)	317,000 (29%)

Table 5.2: FPGA utilization for analog AWGN simulation using 60 parallel cores at 150 MHz for a simulation speed of 9 Gb/s.

5.2 Coloured Noise

Although a white noise spectrum is commonly used for FEC performance evaluation, real 200 Gb/s links typically experience coloured noise due to the bandwidth limitations of the electronic components. The impact of correlated noise samples on FEC performance is not commonly studied, and it is not easy to analyze this with statistical models. Intuitively, one would expect coloured noise to create correlated PAM-symbol errors and degrade FEC performance similarly to DFE error propagation. However, the results presented in this section show a more complex story.

The method for generating coloured noise samples on the FPGA platform is to pass the AWGN through an FIR filter. The number of filter coefficients and their values are implemented as customizable Verilog parameters. In this section, the filter coefficients are chosen to implement the low pass filter (LPF) with magnitude response shown with the blue line in Figure 5.8. This filter has two poles at 16 GHz, resulting in a -20 dB loss at the Nyquist frequency of 50 GHz for 200 Gb/s PAM-4 signalling. A white noise spectrum with a cut-off at the Nyquist frequency is also shown with the black line.

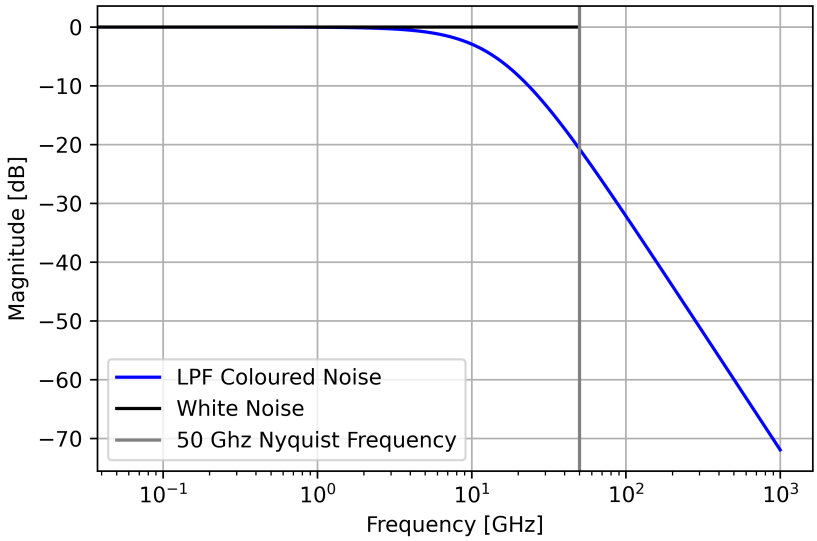

Figure 5.8: Magnitude response of LPF.

The pulse responses for the low-pass filter and white spectrum are calculated using the inverse Fourier transform. Figure 5.9 shows the pulse responses normalized for a maximum value of 1. 10 fs unit intervals (UI) for 200 Gb/s PAM-4 are shown with the gray lines. The intersection of the impulse response with the unit interval lines determines the FIR filter coefficients. The LPF filter impulse response has the first five coefficients $h = [1, 0.73, 0.41, 0.20, 0.09]$. The remaining coefficients are less than 0.05 and are disregarded for the hardware simplicity of the FIR filter. Also, these small filter coefficients are washed out by the eight-bit signal resolution. The filter coefficients are normalized so that the sum of their squares is equal to one to keep the noise power approximately the same as the unfiltered AWGN signal. After normalization, the final digital filter coefficients for the low-pass filter are $h = [0.75, 0.56, 0.31, 0.15, 0.07]$.

The effect of this noise colouring is that one large noise sample increases the likelihood of more large samples in the following five UI. Therefore, PAM-4 symbol errors are more likely to occur within five UI of each other, with the strongest correlation between neighbouring PAM-4 symbols.

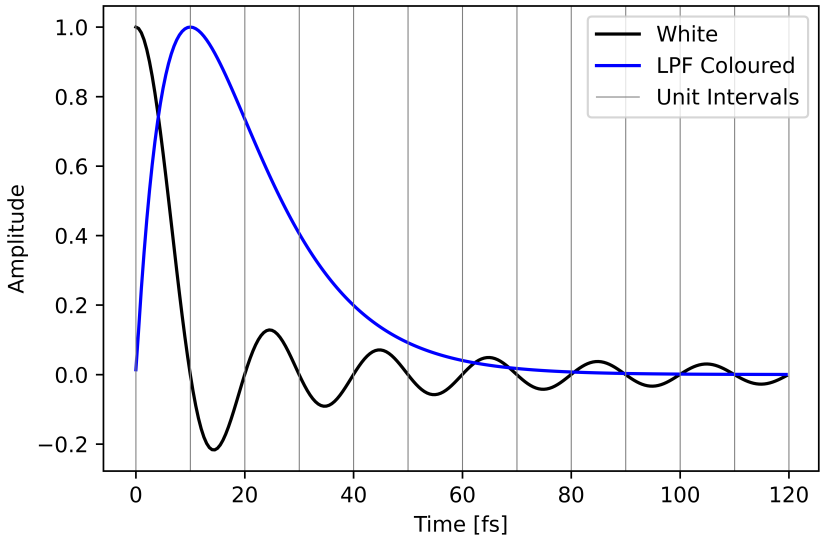

Figure 5.9: Pulse response of low-pass filter.

Figure 5.10 shows pre-FEC BER vs. CER simulation results for a PAM-4 channel with low-pass filtered noise.

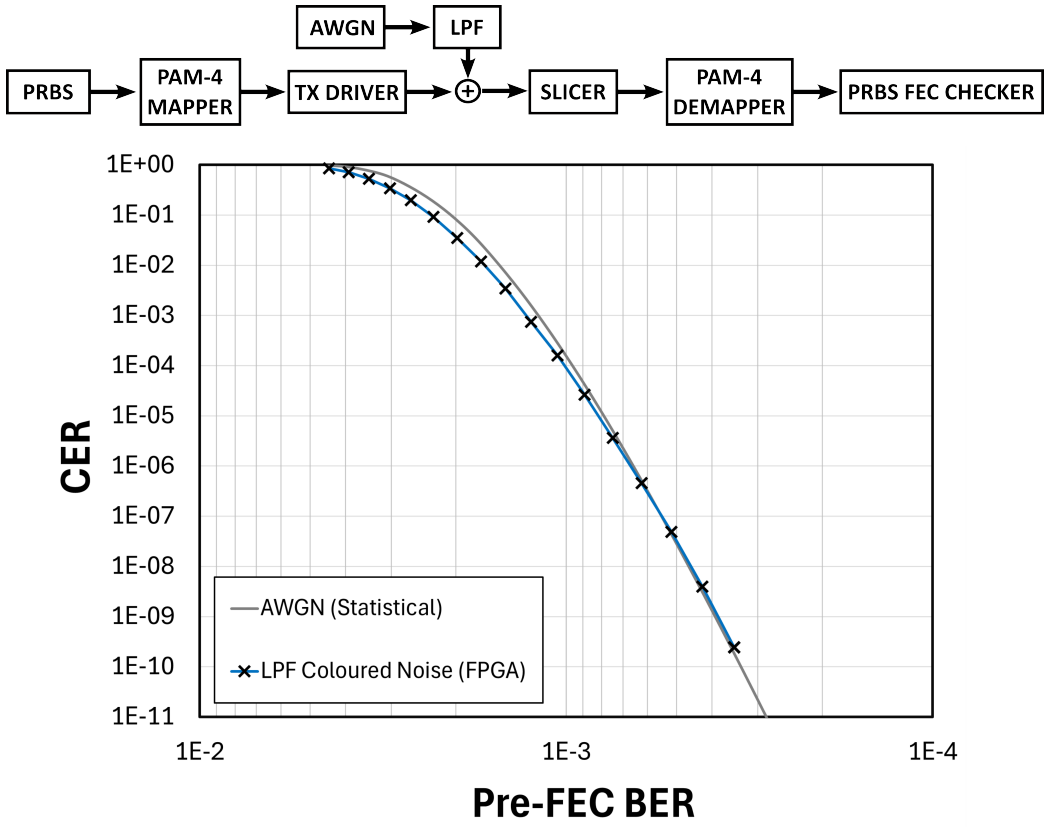

Figure 5.10: Pre-FEC BER vs. CER for PAM-4 Channel with AWGN and LPF coloured noise

Figure 5.10 shows that at higher pre-FEC BER, the coloured noise actually has better performance. There is a point at which the two types of noise have the same CER, at a pre-FEC BER of around 5.5×10^{-4}, and at lower pre-FEC BER, coloured noise shows worse performance. To understand this effect, consider a very high pre-FEC BER of around 3×10^{-3}, where on average 16 bit errors occur every 5440-bit-long KP4 codeword. With AWGN, these 16 bit errors are likely spread out into 16 different 10-bit-long FEC symbols and a codeword error is likely. In contrast, with coloured noise, it is more likely that some bit errors are grouped into one FEC symbol, so the total number of FEC symbol errors in the codeword is less than 16, and the codeword is correctable. However, at much higher SNR, the average codeword has much fewer than 16 bit errors, and codeword errors are extremely unlikely. A KP4 codeword must have at least 16 bit errors to be a candidate for a codeword error. Any correlation between bit errors increases the odds of getting a candidate error codeword with more than 16 bit errors, which is why the noise colouring increases the CER at high SNR.

The simulation results in Figure 5.10 are also run using 60 cores at 150 MHz for a speed of 9 Gb/s. A table showing hardware utilization is left out, as it is almost identical to Table 5.2, with an additional row for the 5-tap FIR filter, which uses 200 look-up tables and 50 flip-flops.

5.3 $1 + \alpha D$ ISI Channel

The FPGA platform also includes a module to implement an "analog" $1 + \alpha D$ channel using the signed decimal signal representation introduced earlier in this chapter. A natural approach to implement the ISI channel on the FPGA platform would be to create a quantized digital version of Figure 5.11 (a) by first mapping PAM-4 symbols $b_k \in \{0, 1, 2, 3\}$ to the analog transmitted signal representation $\in \{-72, -24, 24, 72\}$, and then passing this through an FIR filter. However, implementing the FIR filter requires multiplication and addition computations that can be avoided with a simple look-up-table implementation depicted in Figure 5.11 (b) which requires fewer hardware resources. The $4^2 = 16$ possible channel outputs are pre-computed. The channel module takes in a series of input symbols and directly outputs the corresponding channel output based on the two most recent input PAM-4 symbols b_{k-1} and b_k.

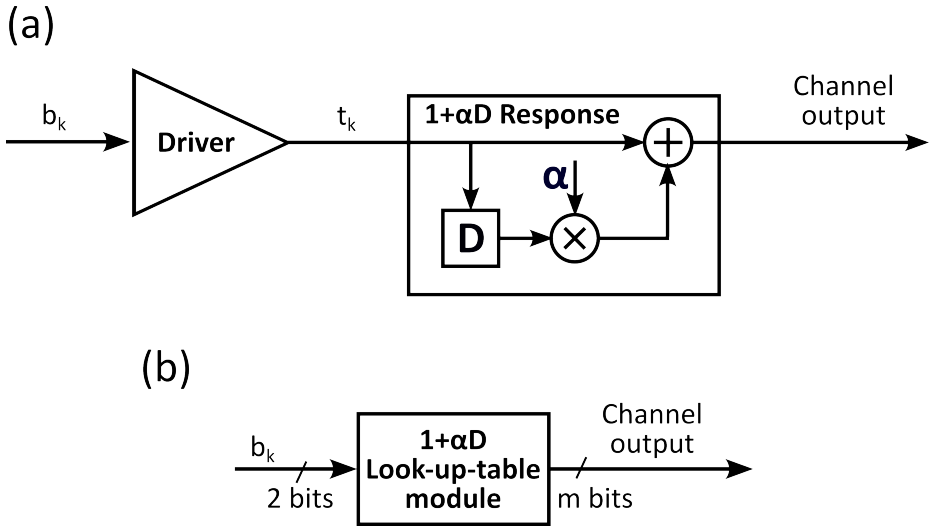

Figure 5.11: (a) Analog $1 + \alpha D$ PAM-4 channel. (b) FPGA platform's LUT implementation of $1 + \alpha D$ PAM-4 channel.

Table 5.3 shows the possible channel output values in signed decimal representation for the $1 + 0.5D$ channel, with scaling factor $s = 24$.

b_{k-1} \ b_k	0	1	2	3
0	-108	-60	-12	36
1	-84	-36	12	60
2	-60	-12	36	84
3	-36	12	60	108

Table 5.3: Possible outputs of $1 + 0.5D$ channel with scaling factor $s = 24$.

Some rounding of α may be required to accommodate 8-bit signal resolution, resulting in inaccuracy compared to a software simulation that uses floating-point numbers to represent the analog signal. No rounding is required in Table 5.3 because the scaling factor $s = 24$ multiplied by the channel ISI $\alpha = 0.5$ is an integer. The hardware implementation of this channel model is given in Algorithm 4.

Algorithm 4 Implementation of analog $1 + \alpha D$ ISI channel.

Initial Information:

Initialize current symbol $cur\_symbol = 0$

Initialize previous symbol $prev\_symbol = 0$

Channel outputs look-up-table

$LUT[b_{k-1}, b_k]$ is channel output given current PAM-4 symbol b_k and previous symbol $b_k - 1$

Algorithm:

for Each incoming transmitted symbol b_k **do**

$prev\_symbol \leftarrow cur\_symbol$

$cur\_symbol \leftarrow b_k$

Output $t_k = LUT[prev\_symbol, cur\_symbol]$

end for

5.4 MLSD

The Viterbi algorithm [27] finds the maximum-likelihood state sequence of a Markov process observed in discrete memoryless noise. This algorithm can be applied to an ISI channel in the presence of AWGN to find the maximum-likelihood sequence of transmitted PAM-4 symbols b_k, given the received noisy samples r_k. This application is known as maximum-likelihood sequence detection (MLSD), and it offers an inherent advantage over a DFE: While the DFE seeks to do symbol-by-symbol detection by subtracting ISI out of each sample r_k, MLSD uses the ISI to inform its symbol detection over a sequence of symbols. MLSD can be applied to channels with any memory length; however, the algorithm complexity increases exponentially with the channel memory length. For 200 Gb/s applications with stringent area and power requirements, MLSD is only practical for a channel with $1 + \alpha D$ having channel memory $\mu = 2$. In this section, the algorithm for MLSD is described for a $1 + \alpha D$ channel, as well as some details on its hardware implementation on the FPGA platform.

PAM-4 Trellis

A PAM-4 channel trellis diagram is shown in Figure 5.12, representing a time-unrolled Markov process. A state $s \in \{0, 1, 2, 3\}$ is assigned for each of the PAM-4 symbols that may be transmitted. The 16 possible state transitions from a timestep to the next are represented with arrows. The notation s_k refers to state s at timestep k. The MLSD algorithm's goal is to find the path through the trellis $\{\dots, s_{k-1}, s_k, s_{k+1}, \dots\}$ corresponding to the most likely sequence of transmitted symbols given the noisy samples $\{\dots, r_{k-1}, r_k, r_{k+1}, \dots\}$.

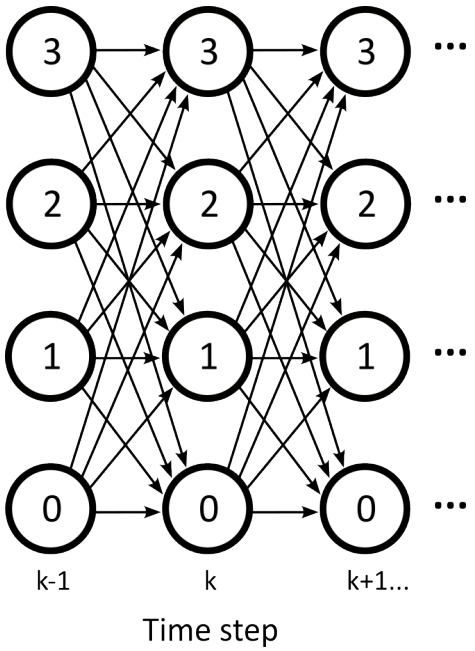

Figure 5.12: PAM-4 MLSD trellis.

Given a noisy channel sample r_k, the maximum-likelihood distance metric for a transition from a source state at time $k-1$ to a sink state at time k is denoted $\Gamma(s_{k-1}, s_k)$. This metric gives the log-likelihood that the transition from s_{k-1} to s_k did *not* occur, so a large distance metric means the transition is unlikely and a small distance metric means that the transition is likely. For a channel with AWGN, the maximum-likelihood distance metric is proportional to the squared difference between r_k and the expected observation for a transition from source to sink with no noise. Consider a transition from the source state $s_{k-1} = 1$ to sink state $s_k = 2$, with a $1 + 0.5D$ channel. This means that the PAM-4 symbols transmitted are $b_{k-1} = 1$ corresponding to an analog level -1, and $b_k = 2$ corresponding to an analog level $+1$. The expected sample with no noise at time k is the inner product between the channel pulse response $h = [1, 0.5]$ and $[+1, -1]$. If we observed a noisy sample of r_k, the maximum-likelihood distance metric for the transition from the source state $s_{k-1} = 1$ to sink state $s_k = 2$ is

$$\Gamma(1,2) = \ (\langle [1, 0.5], [-1, +1] \rangle - r_k)^2.$$

For a path through the trellis visiting state s at time k, we denote the total distance to s as $\Gamma(s_k)$. The total distance to s is the sum of all the state transition distances appearing in the trellis path until time k. For a set of trellis paths terminating in state s at time step k, the most likely path given a set of observations is the path with the lowest distance metric $\Gamma(s_k)$.

The Algorithm

The Viterbi algorithm keeps track of four "survivor" paths through the trellis that terminate at the four different states. The survivor path terminating at state s at time step k is denoted $\hat{u}(s_k)$ and contains the most likely sequence of states terminating in state s at time k. For each time step k,

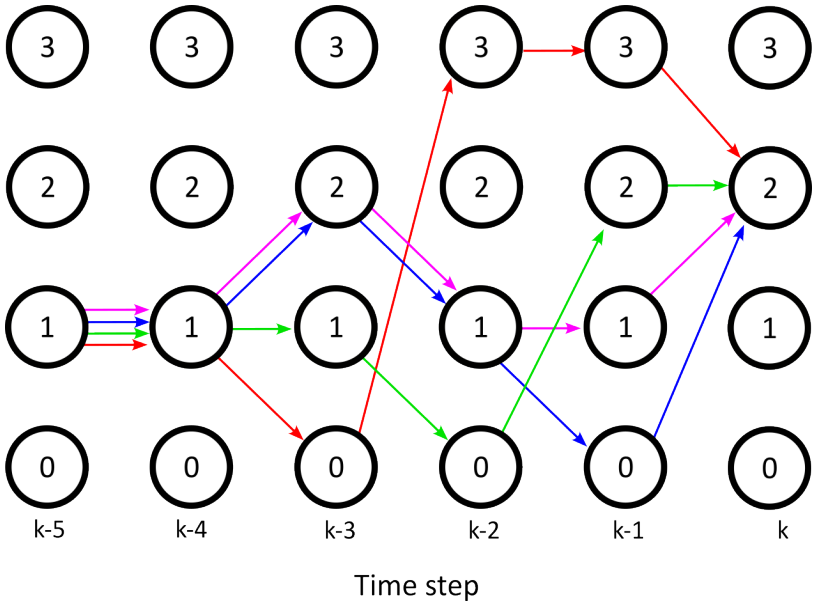

Figure 5.13: Example of paths considered for picking the survivor terminating in state 2 at time step k.

the survivor path to a state s_k is updated by considering all four possible transitions to s_k from the survivor paths at $k-1$ and choosing the one with the smallest maximum-likelihood distance metric. Figure 5.13 shows an example of paths considered when picking the survivor terminating in state 2 at time step k. Each of the four survivor paths at time $k-1$ is shown in a different colour.

In Figure 5.13, all paths visit state 1 at time step $k-4$. Because they are survivor paths, they must include the maximum-likelihood path to this state and visit precisely the same states for all time steps before $k-4$. At a timestep in the past where the four survivor paths all visit the same state s at time $k-\delta$, then the maximum-likelihood sequence of states must contain s at time step $k-\delta$, and so the PAM-4 symbol corresponding to state s is chosen as the MLSD output for time step $k-\delta$. Since all survivor paths are the same before $k-\delta$, MLSD only needs to remember the survivor paths at the most recent δ time steps. The parameter δ is known as the "traceback length", and it should be sufficiently long so that the probability that all survivor paths visit the same state at $k-\delta$ is high.

Algorithm 5 states the MLSD algorithm for equalization of a $1+\alpha D$ channel.

Algorithm 5 MLSD algorithm [27].

Storage:

k (time index, modulo $\delta + 1$)

$\hat{u}_k = \{\hat{u}_{k-\delta}(s_k), ..., \hat{u}_k(s_k)\}, \qquad s_k \in \{0, 1, 2, 3\}$ (survivor paths)

$\Gamma(s_k), \qquad s_k \in \{0, 1, 2, 3\}$ (accumulated path metric for survivor paths)

Recursion:

for each state s_k **do**

Compute $\Gamma(s_{k-1}, s_k) = \Gamma(s_{k-1}) + \Gamma(s_{k-1}, s_k)$ for all four possible transitions (s_{k-1}, s_k)

Find $\Gamma(s_k) = \min(\Gamma(s_{k-1}, s_k))$

Store $\Gamma(s_k)$ and corresponding survivor $\hat{u}_k(s_k)$

end for

Hardware Implementation

The FPGA platform has a module that performs a hardware implementation of MLSD. The implementation closely follows Algorithm 5. The traceback length and the number of bits used to store transition and survivor path metrics are customizable Verilog parameters. This thesis uses a traceback length $\delta = 10$ for all simulations, a value that was experimentally determined to have good performance for the $1 + 0.5D$ channel.

In the simulations presented in this thesis, the received noisy samples r_k^{bin} are 8-bit numbers. Since the state transition metrics are generated from squared eight-bit numbers, they are stored in 16-bit numbers to avoid overflow.

The survivor path distance metrics constantly grow in value over the simulation runtime: with each new received sample r_k, a new 16-bit transition metric is added to each survivor path distance metric. These path metrics are stored in 20-bit numbers for the simulations presented in this thesis. To avoid overflow, when all four path metrics are greater than 2^{19} (they have MSB = 1), 2^{19} is subtracted from all metrics (the MSB is set to 0), preserving the relative difference between the path metrics so that the most likely path remains the one with minimum distance metric.

Simulation Results

Figure 5.14 shows SNR vs. pre-FEC BER for a $1 + 0.5D$ channel with AWGN, equalized with MLSD. The same channel equalized with a DFE is included for comparison. The statistical model presented in [8] is used to generate the results for the DFE channel, and the MLSD simulation results are generated on the FPGA platform. Because there are no statistical models capable of modelling MLSD, data points from a time-domain Matlab simulation are included to confirm the accuracy of the FPGA MLSD implementation.

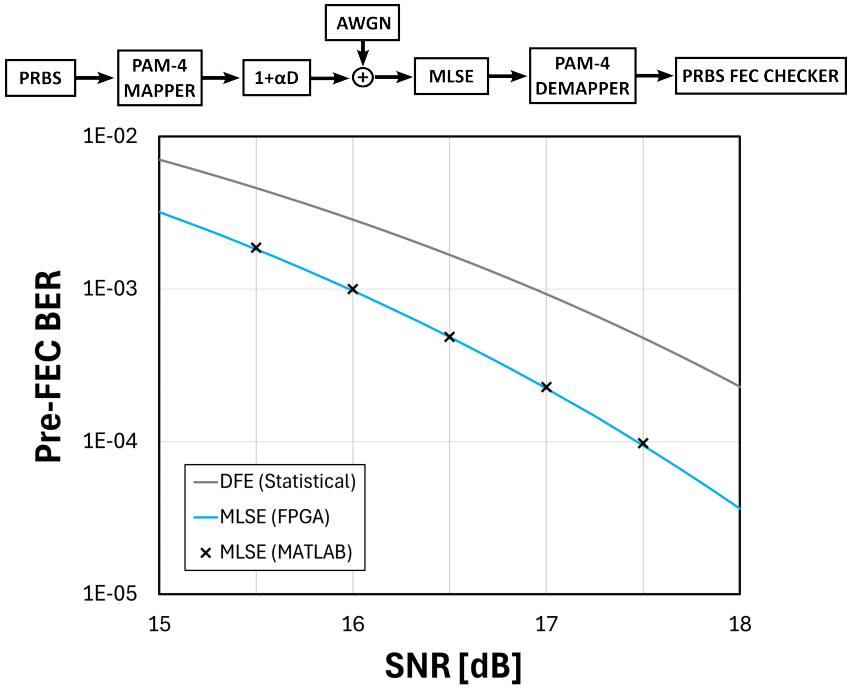

Figure 5.14: SNR vs. pre-FEC BER for $1 + 0.5D$ channel with AWGN equalized with MLSD.

MLSD outperforms the DFE across all SNR levels because only MLSD uses the $1 + 0.5D$ channel's amplified signal power, while the DFE seeks to cancel out the ISI. Figure 5.15 shows FPGA simulation results for SNR vs. CER for the same setup.

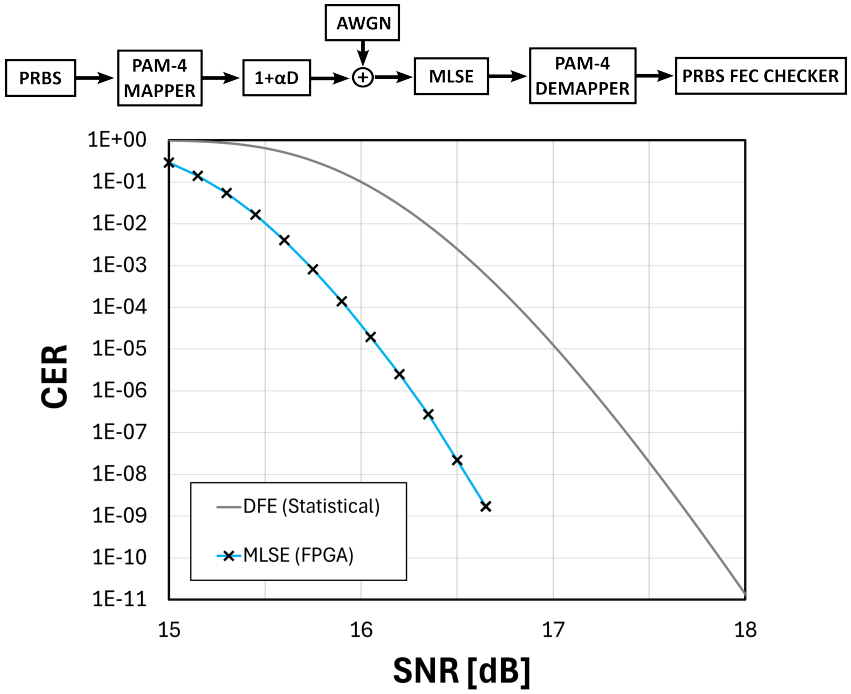

Figure 5.15: SNR vs. CER $1 + 0.5D$ channel with AWGN, equalized with DFE and MLSD.

The MLSD's improvement in pre-FEC BER over the DFE results in an improvement in CER by about 1 dB. Table 5.4 shows the hardware utilization for the FPGA implementation of the analog $1+\alpha D$ channel with MLSD. The MLSD module takes up significant hardware resources to store all the path metrics and to do the add/compare/select operations in Algorithm 5, and uses DSP blocks on the FPGA to reduce the amount of LUT resources required. The simulation is run using 40 cores at 150 MHz for a speed of 6 Gb/s.

Module Name	**# Look-up Tables**	**# Flip Flops**	**# DSPs**
PRBS	32	65	0
PAM-4 Mapper	3	5	0
$1+\alpha D$ Channel	3	4	0
AWGN Generator	1283	4280	0
MLSD	944	161	10
PAM-4 Demapper	3	5	0
FEC Checker	720	831	0
Total	3888	5351	10
× 40 Parallel Cores	156,000	214,000	400
Processor, Driver & Memory	5,000	5,000	400
Grand Total	168,000 (30%)	219,000 (20%)	400 (50%)

Table 5.4: FPGA utilization for analog $1+\alpha D$ channel with MLSD simulation using 40 parallel cores at 150 MHz for a simulation speed of 6 Gb/s.

Chapter 6

Inner FEC Code

6.1 Extended Hamming (128,120) Code

The 200 Gb/s Ethernet standard has adopted a binary Hamming code for the inner FEC protecting optical links. To work well with the KP4 outer FEC code, the inner code's dimension k should be divisible by ten, so the information bits in an inner FEC codeword are divided into an integer number of ten-bit-long outer FEC symbols. This integer division is necessary for the convolutional interleaving algorithm discussed in Section 6.2. A systematic Hamming (127, 120) code satisfies this requirement, with each codeword containing 120 information bits followed by seven parity-check bits.

A drawback of Hamming codes is their low minimum Hamming distance of three. Because of this, codewords with two or more errors risk being decoded to the wrong codeword. These inner-FEC "miscorrections" significantly worsen the concatenated FEC performance, as the added bit error may corrupt an additional outer-FEC symbol. To improve the performance of the Hamming (127,120) code, an additional parity bit can be added to form the extended Hamming (128,120) code which has a minimum Hamming distance of four. This code has been selected in the 200 Gb/s Ethernet Standard [4]. Although the extended Hamming (128,120) code can still only correct one bit per codeword, it will not miscorrect inner codewords with two bit errors, significantly improving the overall concatenated FEC performance. Also, the extended code has eight parity bits, which can fit evenly into four PAM-4 symbols. Even for the extended Hamming code, the probability of miscorrection is non-negligible and significantly impacts the CER [12]. To properly consider miscorrections, the FPGA platform uses a true inner-FEC encoder and decoder rather than the PRBS "FEC checker" approach used for the outer KP4 code.

6.1.1 Encoder

The encoder takes sequences of 120 information bits, denoted $\mathbf{u} \in \mathbb{F}_2^{1\times 120}$, and computes codewords $\mathbf{c} \in \mathbb{F}_2^{1\times 128}$. The notation $\mathbb{F}_2^{i\times j}$ refers to a $i \times j$ vector with entries from $\mathbb{F}_2$ (binary entries).

This codeword $\mathbf{c}$ is obtained by multiplying $\mathbf{u}$ with a generator matrix $\mathbf{G} \in \mathbb{F}_2^{120\times 128}$.

$$\mathbf{c} = \mathbf{uG}$$

Because this code is systematic, the first 120 bits of $\mathbf{c}$ are exactly $\mathbf{u}$, and the generator matrix is of the form:

$$\mathbf{G} = \begin{bmatrix} \mathbf{I}_{120} & \mathbf{P} \end{bmatrix}$$

Where $\mathbf{I}_{120}$ is the 120×120 binary identity matrix and $\mathbf{P}$ is a binary 120×8 parity-check matrix that defines the code. The specific matrix used for the FPGA platform is taken from a recent Ethernet standard contribution.

Hardware Implementation

An increase in bit rate by $\frac{128}{120}$ is required after encoding to keep the same codeword rate with added parity bits. Typically, this would be done by introducing a new clock domain with a higher frequency after encoding. However, implementing this on the FPGA platform introduces unnecessary design complexity. Instead, the PRBS generator is modified to output a sequence of 120 consecutive bits followed by a pause for eight clock cycles, giving time for the encoder to output the additional eight parity bits.

To implement the encoder, it is not necessary to do the full matrix multiplication $\mathbf{c} = \mathbf{uG}$, because $\mathbf{u}$ appears directly as the first 120 bits of $\mathbf{c}$. Instead, the eight parity bits $\mathbf{p} \in \mathbb{F}_2^{1\times 8}$ can be computed with $\mathbf{p} = \mathbf{uP}$ and appended to the end of the information bits, as shown in Figure 6.1. This binary matrix multiplication is a Verilog module that uses XOR gates to perform the binary multiply-accumulate operations between $\mathbf{u}$ and the columns of $\mathbf{P}$. This block introduces one clock cycle of delay from when the encoder module has received the 120 information bits to when the parity bits $\mathbf{p}$ are ready.

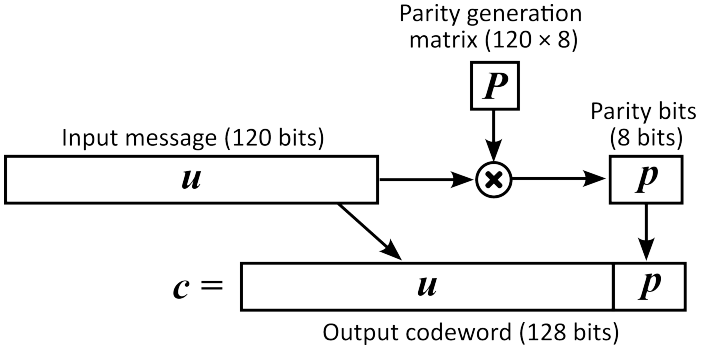

Figure 6.1: Extended Hamming (128,120) encoding.

Because a delay is necessary to calculate the parity check bits for a codeword, the encoder uses two sets of registers to maintain a constant stream of output bits. While one register is filling with incoming data, the parity bits for the other register are calculated, and vice-versa. Two 120-bit-long registers u_1 and u_2 hold information bits for two codewords, and two eight-bit-long registers p_1 and p_2 hold the corresponding parity bits. The algorithm for cycling between the two sets of registers to produce a constant stream of output bits is achieved using a finite state machine described in Table 6.1. The table includes six numbered states from 0 to 5. States 0 and 1 are only visited during start-up during the first codeword. The encoder cycles through states 2-5 in regular operation. In

Table 6.1, the "length" column has the duration of each state in clock cycles, the "input" column has the name of the register that takes in data, the "output" column has the name of the register from which data is output, the "action" column specifies any computations that are done in that state, and the "next state" column specifies the following state.

State	Length	Input	Output	Action	Next State
0	120	u_1	none	none	1
1	8	none	none	calculate $p_1 = Hu_1$	2
2	120	u_2	u_1	none	3
3	8	none	p_1	calculate $p_2 = Hu_2$	4
4	120	u_1	u_2	none	5
5	8	none	u_2	calculate $p_1 = Hu_1$	2

Table 6.1: Hamming (128,120) encoder finite state machine.

6.1.2 Decoder

A block code decoder typically takes in messages $\mathbf{y}$ and outputs the codeword $\mathbf{c}$ that has the smallest Hamming distance to $\mathbf{y}$. A regular, not-extended Hamming code has a minimum Hamming distance between codewords of three. If $\mathbf{y}$ contains zero or one bit error, $\mathbf{y}$ will have the smallest Hamming distance to the correct transmitted codeword. However, if there are two or more bit errors, the $\mathbf{y}$ may be closer to another incorrect codeword. Because Hamming codes fall into a class called "perfect" codes, any $\mathbf{y}$ having two bit errors is guaranteed to be only one bit away from a different codeword, and the so the decoder will always make a miscorrection in this case.

With the extended Hamming code, the minimum Hamming distance between codewords is extended to four. As a result, any $\mathbf{y}$ having two bit errors will be equidistant from the correct codeword and some other incorrect codeword. In this case, the decoder cannot tell which is correct. This is known as a decoding failure, and the decoder outputs $\mathbf{y}$ without correcting.

To decode input messages $\mathbf{y}$, first the syndrome $\mathbf{s} \in \mathbb{F}_2^{1\times 8}$ is calculated using the parity-check matrix $\mathbf{H} \in \mathbb{F}_2^{8\times 128}$.

$$\mathbf{s} = \mathbf{y}\mathbf{H}^T$$

Because the code is systematic, the parity-check matrix $\mathbf{H}$ is of the form

$$\mathbf{H} = \begin{bmatrix} \mathbf{P}^T & \mathbf{I}_8 \end{bmatrix}$$

where $\mathbf{I}_8$ is the is the 8×8 binary identity matrix.

The syndrome can take $2^8 = 256$ possible values. If the syndrome is $0 \in \mathbb{F}_2^{1\times 8}$, $\mathbf{y}$ is a codeword. In this case, the decoder removes the eight parity bits from $\mathbf{y}$ and outputs the remaining 120 information bits. Of the 255 remaining possible syndrome values, 128 can be uniquely mapped to a single bit error index that can be flipped to reach a valid codeword. If one of these syndromes is

calculated, the decoder flips the bit at this index before removing the eight parity bits and outputting the 120 information bits. The remaining 127 possible syndromes correspond to the case where $\mathbf{y}$ is equidistant from two valid codewords, resulting in a decoding failure. In this case, the decoder "knows" that the codeword is incorrect but cannot correct it, and the first 120 bits are output as-is.

Hardware Implementation

Similar to encoding, the binary matrix multiplication to calculate the syndrome is done with a Verilog module using XOR gates. A look-up table maps syndrome values to error locations. The look-up table (LUT), is implemented using a 256-long array of 8-bit values stored in read-only memory on the FPGA. It is loaded with values such that $LUT[\mathbf{s}]$ is the unique error location corresponding to the syndrome $\mathbf{s}$. For the values of $\mathbf{s}$ that correspond to zero errors, or a message that is equidistant from two codewords $LUT[\mathbf{s}] = 255$, a placeholder value that does not correspond to a valid bit index from 0 to 127, and signifies that the decoder should not flip any bits.

Again, similar to encoding, two 128-bit registers denoted y_1 and y_2 store input messages so that y_1 may store input bits while y_2 is being decoded and vice versa. The finite state machine that implements this decoder in hardware is shown in Table 6.2.

State	Length	Input	Output	Action	Next State
0	128	c_1	none	none	1
1	8	$c_2[0:7]$	none	correct c_1	2
2	120	$c_2[8:127]$	$c_1[0:119]$	none	3
3	8	$c_1[0:7]$	none	correct c_2	4
4	120	$c_1[8:127]$	$c_2[0:119]$	none	1

Table 6.2: Hamming decoder finite state machine.

Simulation Results

Figure 6.2 shows SNR vs. CER for an AWGN channel protected by the concatenated extended Hamming (128,120) + KP4 FEC (denoted "CFEC" in the legend). The same channel protected by just the KP4 FEC is included for comparison. The CER reported on the y-axis is the outer KP4 CER rather than the inner code's CER. The statistical model presented in [12] verifies the accuracy of FPGA concatenated FEC simulations. For the remainder of the figures that present simulation results in this thesis, the PAM-4 mapper and de-mapper blocks in the system diagram above the plot are omitted for simplicity.

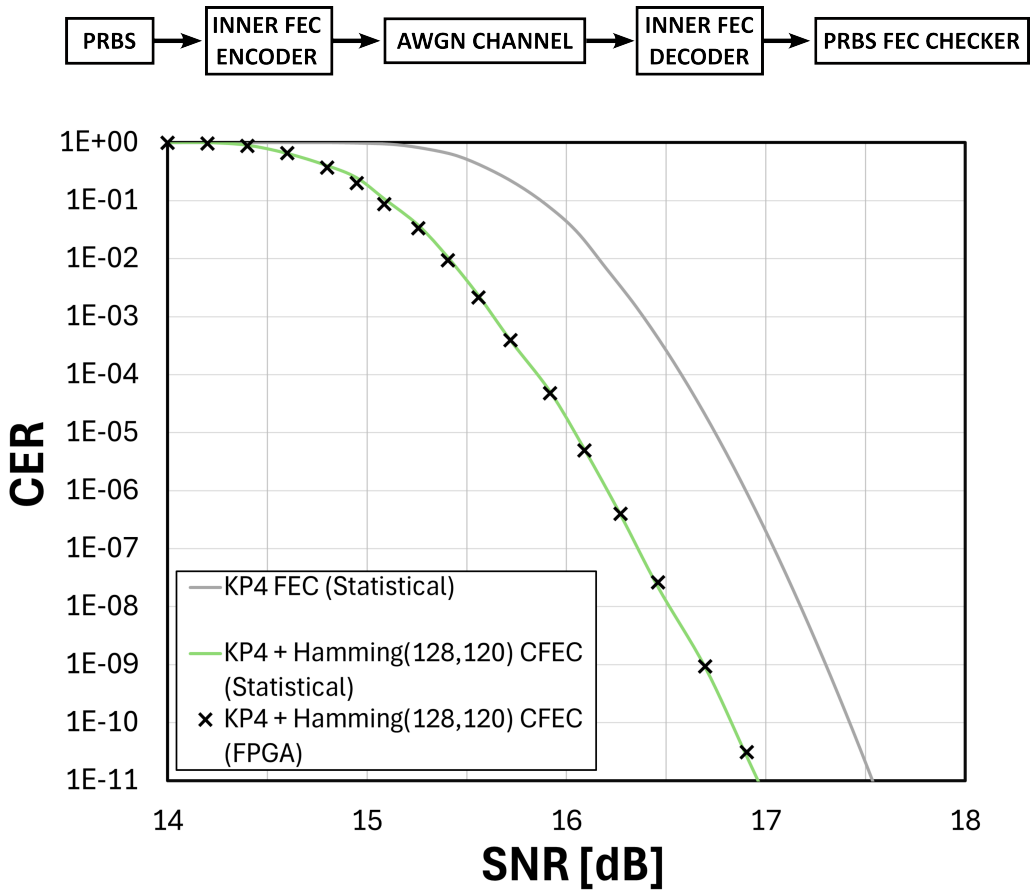

Figure 6.2: AWGN channel protected by concatenated KP4 + Hamming(128,120) FEC.

Figure 6.2 shows that the inner code offers a significant boost in performance of about 0.5 dB across a large range of SNR. Table 6.3 shows the hardware resources used for the simulation results shown in Figure 6.2. The inner FEC encoder and decoders are quite resource intensive, primarily due to the multiple registers that store input and output bits. The decoder uses more resources due to its syndrome look-up-table. The simulation is run on 100 parallel cores at 150 MHz for a simulation speed of 15 Gb/s.

Module Name	**# Look-up Tables**	**# Flip Flops**
PRBS	32	65
Inner FEC Encoder	525	493
PAM-4 Mapper	3	5
AWGN Channel	251	0
PAM-4 De-mapper	3	5
Inner FEC Decoder	1076	536
FEC Checker	720	831
Total	2,610	1,935
× 100 Parallel Cores	261,000	194,000
Processor, Driver & Memory	5,000	5,000
Grand Total	266,000 (49%)	199,000 (18%)

Table 6.3: FPGA utilization for AWGN error-injection channel with concatenated Hamming (128,120) + KP4 FEC using 100 parallel cores at 150 MHz for a simulation speed of 15 Gb/s.

Coloured Noise

For non-binary codes such as the KP4 code that has 10 bits per FEC symbol, Section 5.2 showed that a channel with LPF coloured noise has better CER compared to an AWGN channel for high pre-FEC BERs. Figure 6.3 shows the same LPF coloured noise channel protected with the concatenated Hamming (128,120) and KP4 FEC. The concatenated FEC with AWGN channel, as well as the KP4 FEC with both types of noise are included on the figure for comparison. Pre-FEC BER is reported on the x-axis, which refers to the BER introduced by the channel, without inner or outer FEC decoding.

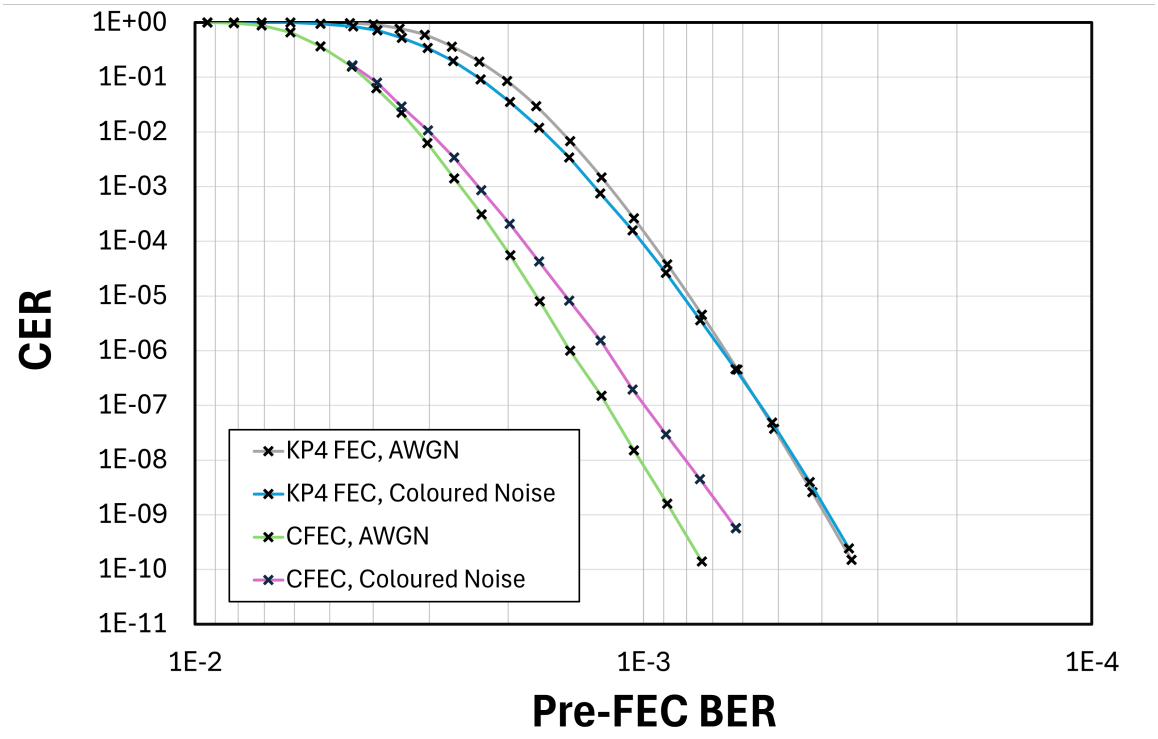

Figure 6.3: LPF coloured noise channel protected by concatenated KP4 + Hamming(128,120) FEC.

Figure 6.3 shows that for the concatenated FEC, the coloured noise degrades performance compared to AWGN across all SNR. This differs from the KP4 FEC, because errors are first corrected by the binary inner code. Since the inner code is binary, there is no benefit from grouping multiple bit errors into one FEC symbol because each bit is its own FEC symbol. Conversely, the coloured noise introduces a higher likelihood of inner-FEC codewords having more than one bit error and being uncorrectable, resulting in a higher BER after inner FEC decoding compared to AWGN, and a worse overall CER.

6.2 Convolutional Interleaving

Although the errors introduced in optical links may be predominantly random, the inner-FEC decoder changes the error distribution by correcting the inner-FEC codewords with only one error and not correcting those with multiple errors. As a result, the majority of the 120-bit long inner FEC messages will be error-free. However, occasionally, a codeword will have two or more errors, effectively introducing correlated errors after inner FEC decoding. To give the outer decoder the best chance of correcting errors from an erroneous 120-bit-long inner-FEC codeword, each of its twelve 10-bit-long outer-FEC symbols should be distributed to different outer-FEC codewords. This could be achieved with 12-way block interleaving, however this is too resource intensive for 200 Gb/s systems, requiring 12 KP4 FEC encoders and decoders on working in parallel on the same chip. Only 2-way and 4-way block interleaving is supported in the Ethernet standard. However, this distribution of symbols *can* be realistically achieved using a combination of block

interleaving and convolutional interleaving.

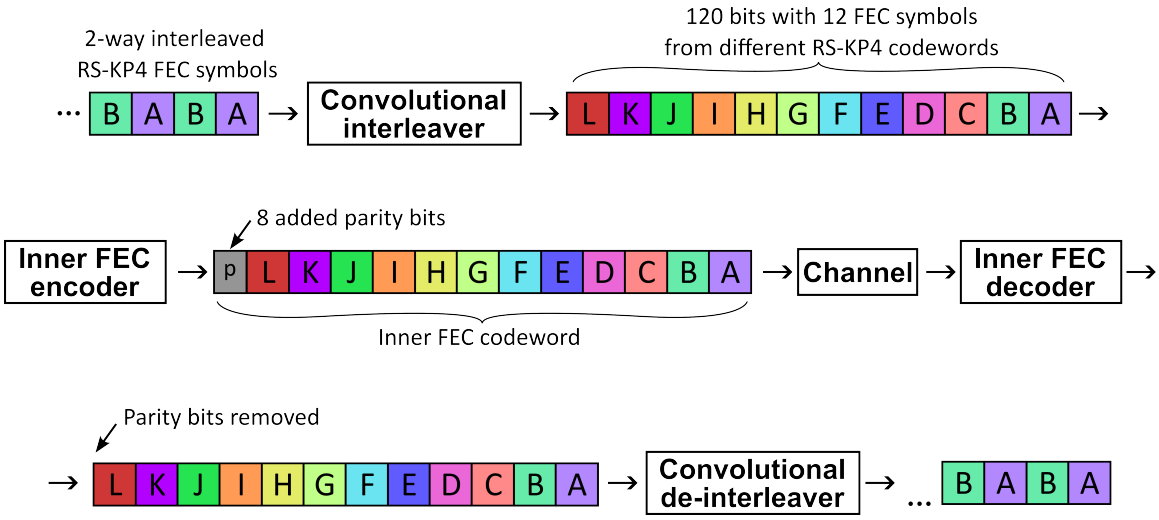

Figure 6.4: Data path with convolutional interleaving.

Convolutional interleaving is a technique that permutes the order of input symbols at the cost of added latency [22]. It delays symbols by differing amounts, resulting in a shuffled transmission order. This shuffling is reversed on the receiving side by a convolutional de-interleaver, shown in Figure 6.4. A recent IEEE standard proposal defines the convolutional interleaver architecture to achieve an inner-FEC codeword with outer FEC symbols from 12 distinct KP4 codewords [22]. This interleaver architecture, shown in Figure 6.5, is parameterized by W (symbol length), P (number of lanes), and D (delay constant). It has P different parallel lanes that contain first-in-first-out shift registers. Data is input to the lanes in blocks of length W KP4 FEC symbols in a round-robin fashion. For example, if $W = 4$, the first 4 KP4 FEC symbols go to lane 0, where there is no delay, and they are immediately output. The following four symbols are input to lane 1 where they are stored in the first W-length block of the shift register. The next four symbols are input to lane 2, and so on. After lane $P - 1$ is reached, the order circles back to lane 0. The output data is taken in blocks of W FEC symbols from the same lane as the input. Upon start-up, the input data symbols must cycle through all the lanes multiple times before all the shift registers are full. Filling up these shift registers introduces a significant amount of latency.

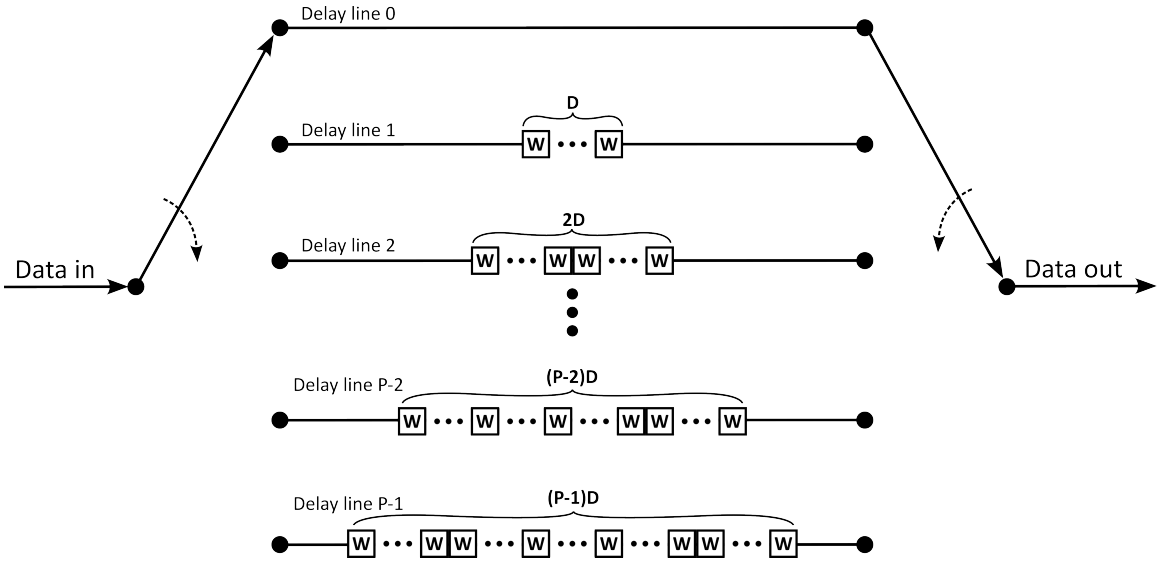

Figure 6.5: Convolutional interleaver with parameters W, P, D.

The convolutional de-interleaver architecture is shown in Figure 6.6. It works the same way as the interleaver, but the delay length per lane is flipped so that the first lane has a long delay and the last lane has no delay. The data which got the maximally delayed in the interleaver gets no delay in the de-interleaver and vice-versa, resulting in the original order of transmission being restored.

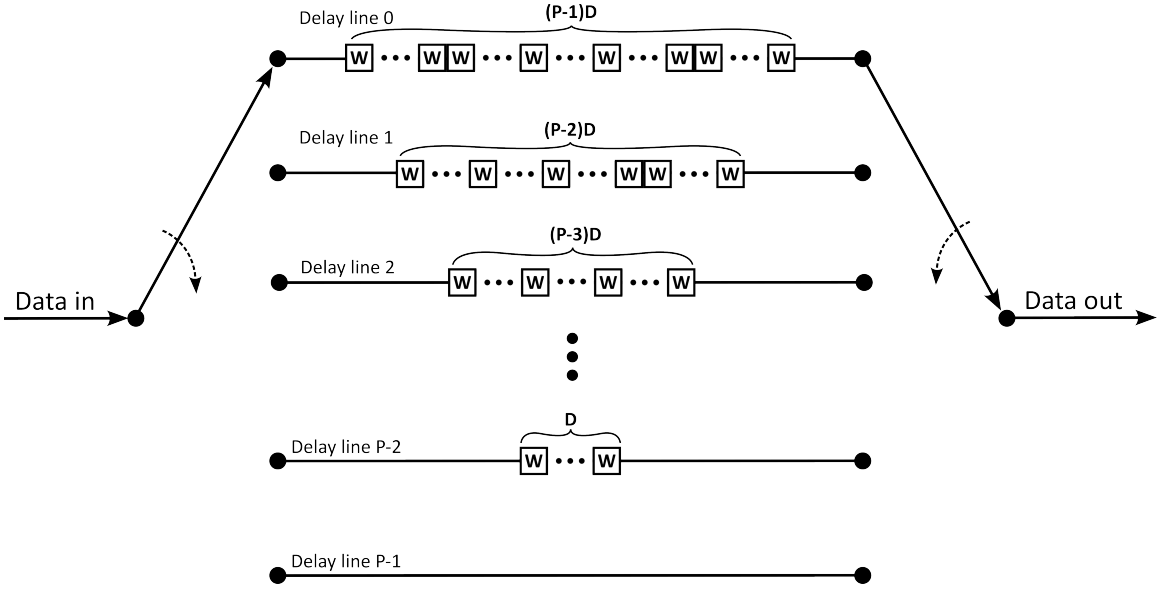

Figure 6.6: Convolutional de-interleaver with parameters W, P, D.

The symbol length W is set to match the block interleaving scheme. For example, if 4-way block interleaving is used, $W = 4$ and each W-length block contains four 10-bit-long KP4 FEC symbols (40 bits). These four FEC symbols are kept together because they already originate from 4 different

KP4 codewords, so they do not need to be shuffled relative to each other. The number of lanes P is then set as $P = \frac{12}{W}$ so that taking one W-length symbol from each of the P lanes makes up a full 120-bit inner-FEC message. For example, if $W = 4$, then $P = \frac{12}{4} = 3$. This allows for the 12 FEC symbols to originate from 12 different codewords, given that the last symbol from each convolutional interleaver lane is sufficiently spaced so that no FEC symbols come from the same 544-FEC-symbol-long codeword. To ensure this spacing condition, the delay constant D must be greater than $544/P$. The Ethernet standard also requires D to be an integer multiple of 12 so that the convolutional interleaver holds an even number of inner FEC messages in the shift registers. For this reason, D is chosen to be the smallest multiple of 12 greater than $544/P$. For $W = 4$ and $P = 3$, $D = 192$.

The total latency introduced by the interleaver and de-interleaver is determined by the total number of bits held in the interleaver and de-interleaver combined. This is calculated by summing the number of bits in each lane. Lane i's shift register holds $i \times D \times W$ 10-bit-long KP4 FEC symbols, and so the total number of bits for both interleaver and de-interleaver can be computed with

$$\text{Latency[bits]} = 2 \times \sum_{i=0}^{P} i \times D \times W \times 10.$$

For example, with the parameters $W = 4$, $P = 3$, and $D = 192$, the convolutional interleaving and de-interleaving results in 46080 bits of latency. At a data rate of 200 Gb/s, this equals 230.4 ns of added delay. For latency-sensitive applications, Ethernet standard contributions consider a half-depth interleaver where the delay D is reduced by half [28]. This cuts latency in half at the cost of a slight decrease in performance due to worse error distribution than the full interleaver. Table 6.4 shows the convolutional interleaving parameters and latency for full-depth and half-depth convolutional interleavers using the KP4 outer code and Hamming (128,120) inner code.

Depth	**W**	**P**	**D**	**Latency (bits)**	**Latency (ns)**
half	2	6	48	28800	144
half	4	3	96	23040	115.2
full	2	6	96	57600	288
full	4	3	192	46080	230.4

Table 6.4: Convolutional interleaver parameters and latency for KP4 + Hamming (128,120) concatenated FEC.

Hardware Implementation

Verilog parameters are used for W, P, D, and the number of bits per FEC symbol so that the convolutional interleaver module can potentially be used in a concatenated FEC architecture with different inner and outer FEC codes. The shift registers are initially loaded with all-zeros. This results in the wireline system's outputting a long sequence of zeros before the PRBS data is output. To accommodate this, the PRBS checker has an additional Verilog parameter specifying the convolutional interleaver latency in number of bits. Upon receiving valid data, the FEC

checker ignores the first sequence of zeros that come from the convolutional interleaver's initial shift register values.

Simulation Results

Figure 6.2 shows SNR vs. CER for an AWGN channel protected by the concatenated extended Hamming (128,120) + KP4 FEC with no interleaving, 4-way block interleaving, 4-way block interleaving + half-depth convolutional interleaving, and 4-way block interleaving + full-depth convolutional interleaving.

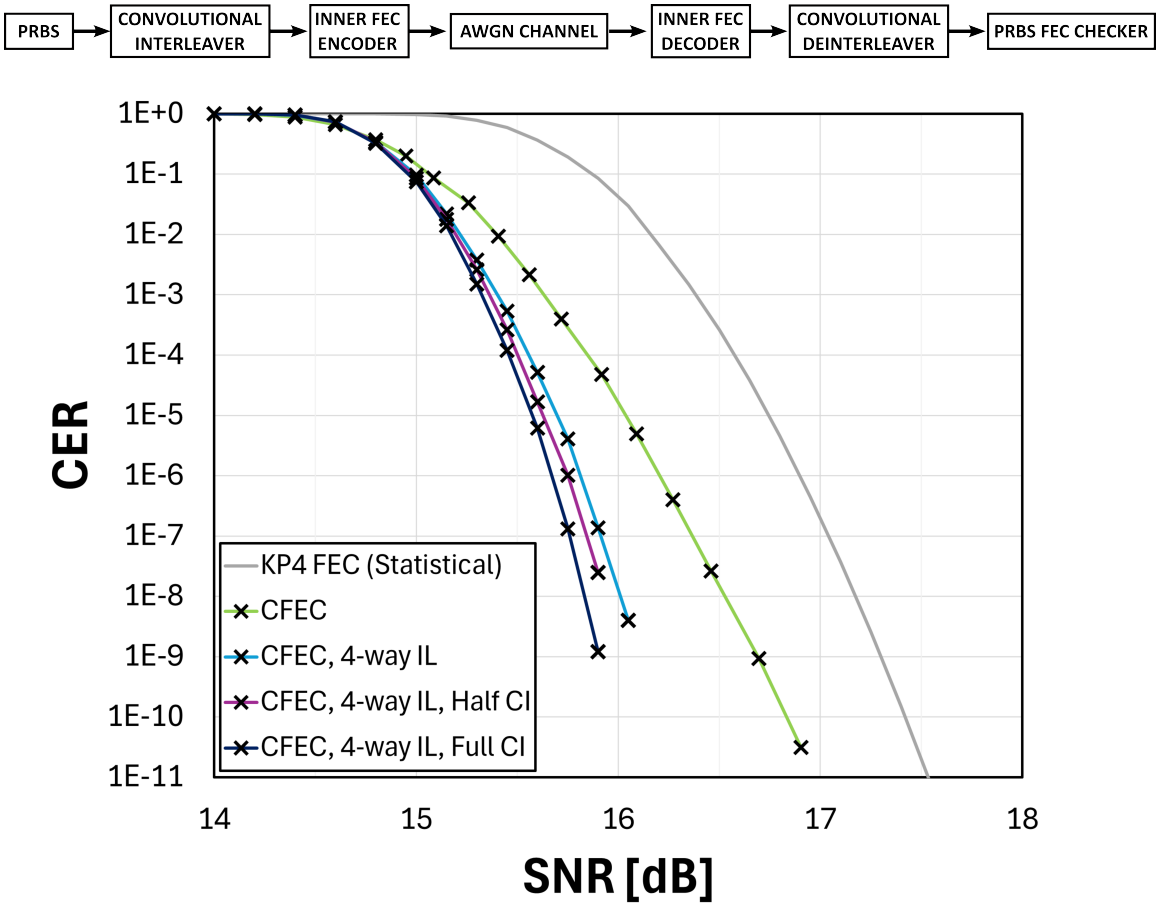

Figure 6.7: AWGN channel protected by concatenated RS-KP4 + Hamming (128,120) FEC codes with 4-way block interleaving + half-depth and full-depth convolutional interleaving.

Despite the AWGN channel producing random errors, the interleaving improves CER by breaking up the error correlation introduced by inner-FEC decoding. The channel protected by 4-way block interleaving and full convolutional interleaving shows about 0.5 dB improvement over the concatenated FEC with no interleaver. However, diminishing gains are seen with the increasing amounts of interleaving. A recent Ethernet standard contribution argues that the latency introduced by the convolutional interleaver may not be worth the marginal improvement in

performance [28]. Table 6.5 shows the hardware resources used for the simulation results shown in Figure 6.2. The simulation is run on 40 parallel cores at 150 MHz for a simulation speed of 6 Gb/s.

Module Name	**# Look-up Tables**	**# Flip Flops**
PRBS	32	65
Convolutional Interleaver (full)	932	698
Inner FEC Encoder	525	493
PAM-4 Mapper	3	5
AWGN Channel	251	0
PAM-4 Demapper	3	5
Inner FEC Decoder	1076	536
Convolutional De-interleaver (full)	903	697
FEC Checker	720	831
Total	4,445	3,330
× 40 Parallel Cores	178,000	133,000
Processor, Driver & Memory	5,000	5,000
Grand Total	183,000 (34%)	138,000 (13%)

Table 6.5: FPGA utilization for AWGN error-injection channel with concatenated Hamming (128,120) + KP4 FEC and convolutional interleaver using 40 parallel cores at 150 MHz for a simulation speed of 6 Gb/s.

Chapter 7

Soft-Decision Receivers and Inner-FEC Decoding

Soft-decision decoding can improve the inner FEC code's performance by using channel reliability information as well as the hard-decided PAM-4 symbols to find the correct codeword. This boost in performance comes at the cost of added complexity: Firstly, a soft-output receiver is required to produce this reliability information, and secondly, a soft-decision inner-FEC decoder uses a more complex decoding algorithm. This chapter details a modified inner FEC code, soft-decision inner FEC decoder, and soft-output receivers included in the FPGA platform.

Figure 7.1 shows the architecture for soft-decision inner-FEC decoding used on the FPGA platform. The receiver not only produces a sequence of hard-decision PAM-4 symbols $d_k \in \{0, 1, 2, 3\}$, but also a series of reliability values α_k. The reliability values should have the property that if $\alpha_k < \alpha_j$, then $P(d_k = b_k) < P(d_j = b_j)$. In other words, symbols with higher α are more likely to be correct.

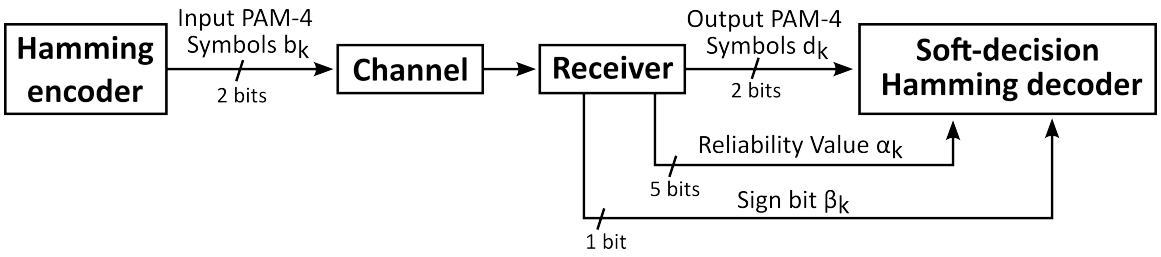

Figure 7.1: Architecture for soft-decision inner-FEC decoding.

Each of the four possible output PAM-4 symbols can be rank-ordered from most to least likely. Typically, the third and fourth most likely PAM-4 symbols have a negligible probability. In this case, the log-likelihood ratio of the first and second most likely symbols can be used as α that satisfies the condition above. For example, if at timestep k the most likely symbol is 0 and the second most likely symbol is 1, then

$$\alpha_k = \log\left(\frac{P(b_k = 0)}{P(b_k = 1)}\right).$$

By definition, $\alpha \geq 0$, with equality when the first and second most likely symbols are equally likely. Although, due to Gray coded PAM-4 symbols, the second and first most likely symbols only differ by one bit, the location of this lower-reliability bit could be either in the most-significant bit (MSB) or least-significant bit (LSB) of the PAM-4 symbol depending on the actual value of the first and second most likely symbols. The location of the lower-reliability bit in each PAM-4 symbol is crucial for the soft-decision decoder to decide which bit to correct. The receiver passes on this information with an additional single "sign bit" β_k. If $\beta_k = 0$, this signifies that the low-reliability bit is in the LSB of the PAM-4 symbol at time index k, and if $\beta_k = 1$, the low-reliability bit is in the MSB.

7.1 Shortened Hamming (68,60) Code

Since each PAM-4 symbol contains only one low-reliability bit and the soft-output receiver passes on this information, the decoder should only consider correcting bits in the 64 low-reliability positions in a 128-bit codeword. From this perspective, the Hamming (128,120) code is inefficient because the eight parity bits protect the entire 128-bit codeword rather than just the 64 bits that are low-reliability. This motivates a new shortened (68,60) Hamming code that has been adopted by the Ethernet standard for use with a soft-decision decoder that protects PAM-4 symbols rather than individual bits [23]. The (68,60) Hamming code is achieved by puncturing the (128,120) code in 60 places.

7.1.1 Encoder

To encode a message, first, each pair of bits that form a PAM-4 symbol in the message $\mathbf{u}$ are XORed together to create a new 60-bit long message $\mathbf{u}'$. A sequence of eight parity bits that protects $\mathbf{u}'$ is generated with $\mathbf{p} = \mathbf{u}'\mathbf{P}$, where $\mathbf{P} \in \mathbb{F}_2^{60\times 8}$ is a new punctured parity generation matrix. Then, the original 120-bit message is transmitted with the eight parity bits, as shown in Figure 7.2.

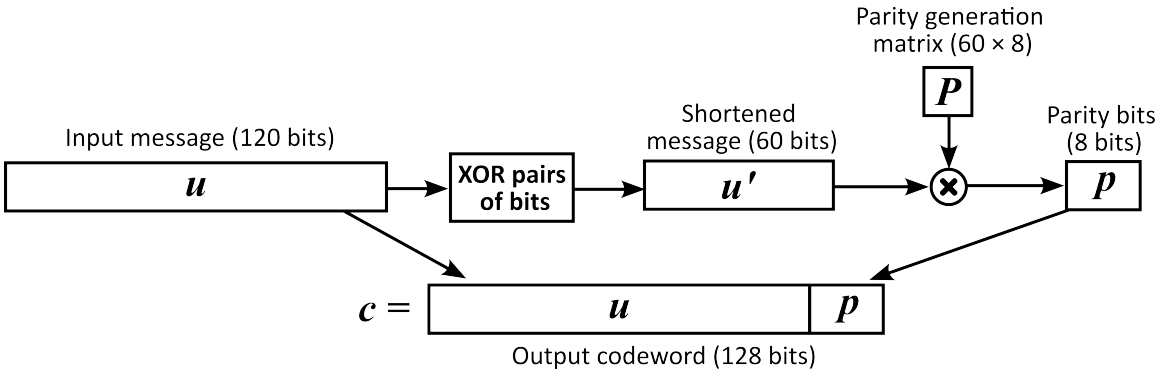

Figure 7.2: Encoding 120 information bits using the shortened Hamming (68,60) code.

7.1.2 Decoder

Although this (68,60) code is typically decoded with the Chase soft-decision decoding algorithm described in Section 7.2, this subsection explains how hard-decision decoding would be done, which is a crucial building block for the Chase algorithm.

The decoder receives the hard-decision PAM-4 symbols d_k and sign bits β_k. The decoder then forms a 68-bit-long received message $\mathbf{y}$. The first 60 bits are produced by XORing pairs of bits in the first 60 PAM symbols together, and the last eight are produced using both bits from the 4 last PAM-4 symbols. The eight-bit syndrome is calculated with the binary matrix multiplication $\mathbf{s} = \mathbf{y}\mathbf{H}^T$, where $\mathbf{H} \in \mathbb{F}_2^{8\times 68}$ is the new shortened Hamming parity check matrix. The eight-bit syndrome can now be uniquely mapped to the location of either a PAM-4 symbol error in the 60 information PAM-4 symbols or the location of a bit error in the eight parity bits. With the shortened code, only 68 of the 256 possible syndromes point to a valid error location, $\mathbf{s} = 0$ indicates no error, and the remaining 196 non-zero syndrome values indicate decoding failure. If a syndrome points to a valid PAM-4 symbol error location in the information bits, the value of β for that PAM-4 symbol determines whether the decoder should flip the MSB or LSB.

7.2 Chase Algorithm for Soft-Decision Decoding

Soft-decision decoding makes use of channel reliability information to improve decoding performance. In his 1972 paper, Chase presented an algorithm for soft-decision decoding of block codes [29]. This algorithm is a popular choice for practical implementations of the inner-FEC decoder in 200 Gb/s Ethernet [28]. This section describes the algorithm and some details on it's hardware implementation on the FPGA platform.

A "complete" hard-decision decoder for a binary (n, k) code takes in a received sequence $\mathbf{y} \in \mathbb{F}_2^{1\times n} = \{y_1,\ y_2, ..., y_n\}$, and finds the error sequence $\mathbf{z}_m$ of minimum binary weight such that $\mathbf{y}+\mathbf{z}_m = \{y_1+z_{m1},\ y_2+z_{m1}, ...y_n+z_{m1}\}$ is a codeword (+ represents modulo-2 addition). In reality, hard-decision binary decoders can only find error patterns of maximum weight t. For example, the decoder for the extended Hamming code with $t = 1$ presented in Section 6.1.2 can only determine an error pattern of binary weight one. If a valid error pattern of weight one cannot be found, it experiences a decoding failure.

Similarly, a "complete" soft-decision decoder finds an error pattern $\mathbf{z}_m$ of minimum *analog* weight such that $\mathbf{y}+\mathbf{z}_m$ is a codeword. The analog weight of an error pattern is defined as $\sum_{i=1}^{N} \alpha_i Z_{mi}$, and it corresponds to the likelihood of an error pattern. An error pattern with a lower analog weight is more likely than one with a higher analog weight. Hence, the "complete" soft-decision decoder finds the most likely codeword given a received sequence. Chase's algorithm for soft-decision decoding approaches the performance of the "complete" soft-decision decoder.

Chase's algorithm finds a set of possible error patterns and picks the one with the lowest analog weight. An error pattern $\mathbf{z}_T$ is found by perturbing $\mathbf{y}$ by a test pattern $\mathbf{T}$, and then doing traditional hard decision decoding on $\mathbf{y} + \mathbf{T}$. If the decoding is successful, it gives a new error pattern $\mathbf{z}'$. The error pattern relative to $\mathbf{y}$ is $\mathbf{z}_T = \mathbf{z}' + \mathbf{T}$.

Let q and w be integer parameters. The test patterns are generated by taking all combinations of w or fewer 1s in the least reliable q positions of the $\mathbf{Y}$, including the all-0s test pattern. This gives a total of $\binom{q}{0} + \binom{q}{1} + \binom{q}{2} + ... + \binom{q}{w}$ test patterns. The Chase algorithm with q, w can potentially

extend the decoder's error-correcting capability from t to $t + w$. By increasing q and/or w, a larger number of test patterns is considered, giving a better chance of finding the error pattern with the lowest analog weight. However, this also increases the complexity of the algorithm.

The Chase soft-decision decoding algorithm is formally stated below:

Algorithm 6 Chase algorithm for soft-decision decoding [29].

Received information:
$\mathbf{y} = \{y_1,\ y_2, ..., y_n\}$ (Received sequence)
$\{\alpha_1,\ \alpha_2, ..., \alpha_n\}$ (Reliability values for received sequence)

Algorithm:
Generate a set of test patterns
for each test pattern $\mathbf{T}$ **do**
  Decode $\mathbf{y} + \mathbf{T}$ with a hard-decision decoder
  if an error pattern $\mathbf{z}'$ is found **then**
    Calculate the analog weight of $\mathbf{z}_T = \mathbf{z}' + \mathbf{T}$
    if this is the lowest analog weight so far **then**
      Store $\mathbf{z}_{min} \leftarrow \mathbf{z}_T$ and its analog weight
    end if
  end if
end for
if an error pattern has been stored **then**
  Output the codeword $\mathbf{y} + \mathbf{z}_{min}$
else
  Output $\mathbf{Y}$ and declare a decoding failure
end if

Hardware Implementation

The hardware implementation of the soft-decision Chase decoder has Verilog parameters for the q and w that determine the number of test patterns considered to decode each codeword. In this thesis, $q = 6$, $w = 3$ is chosen. This choice of parameters is taken from a recent Ethernet standard contribution [28], and results in 42 test patterns.

First, the decoder takes in the hard-decided PAM-4 symbols d_k, reliability values α_k, and sign bits β_k for the 64 PAM-4 symbols that constitute a received message $\mathbf{y}$ (a codeword from the Hamming (68,60) code described in Section 7.1 that may include errors). The decoder performs a partial sorting of the reliability values as they come in so that the index of the least reliable $q = 6$ PAM-4 symbols is immediately available when the complete message has been received. Also, $\mathbf{y}$ is formed using the XOR of each of the first 60 PAM-4 symbols, followed by the eight parity bits from the remaining 4 PAM-4 symbols.

Then, each of the 42 test patterns, including the all-zeros test pattern, is sequentially applied to $\mathbf{y}$ by flipping bits of $\mathbf{y}$ in the positions indicated by the test pattern. For each test pattern, a new syndrome is calculated. If the syndrome corresponds to an error location, the analog weight corresponding to the test pattern is calculated by summing the values of α for each of the bits flipped

by the test pattern and the additional error location corresponding to the syndrome. During this process, the bit error locations for the test pattern that results in a valid codeword with minimum analog weight are stored in memory. At the end of this process, the 60 information PAM-4 symbols are corrected using the PAM-symbol error locations from the minimum-weight test pattern, along with the corresponding values of β_k, which determine if the MSB or LSB should be flipped for each PAM-4 symbol error. Finally, the 120 information bits from the soft-decoded codeword are output.

Sequentially applying the 42 test patterns takes a significant amount of time. Each test pattern takes four clock cycles to apply, perform hard-decision decoding, and calculate the analog weight. Hence, the final soft-decoded codeword is only available $42 \times 4 = 168$ clock cycles after it is input. Three parallel soft-decision decoders are used in a cyclic fashion to maintain a continuous stream of output data, as described in Table 7.1. This parallel rotation introduces a 3-codeword or 384-clock cycle latency to the output of the soft-decision decoder.

Timestep k	**Decoder 1**	**Decoder 2**	**Decoder 3**
0 - 127	Input 1st codeword		
128 - 255	Start decoding 1st codeword	Input 2nd codeword	
256 - 383	Finish decoding 1st codeword	Start decoding 2nd codeword	Input 3rd codeword
384 - 511	Output 1st codeword and input 4th codeword	Finish decoding 2nd codeword	Start decoding 3rd codeword
512 - 639	Start decoding 4th codeword	Output 2nd codeword and input 5th codeword	Finish decoding 3rd codeword
640 - 768	Finish decoding 4th codeword	Start decoding 5th codeword	Output 3rd codeword and input 6th codeword
⋮	⋮	⋮	⋮

Table 7.1: Three-way cycling for continuous output of Chase soft-decision decoder.

7.3 Soft-Output Receiver Models

This chapter presents two types of soft-output receiver algorithms: a simple soft-output slicer for the AWGN channel and the soft-output Viterbi algorithm (SOVA) for the $1 + \alpha D$ ISI channel.

7.3.1 Soft-Output Slicer

The soft-output slicer's purpose is to take in noisy samples at the output of the PAM-4 AWGN channel r_k and produce three outputs: The hard-decision value d_k is the most likely symbol, the reliability measure α_k is the log-likelihood ratio of the first and second most likely symbols, and the sign bit β_k determines the location of the lower-reliability bit in the PAM-4 symbol.

Given the received sample r_k, the slicer seeks to determine the first and second most likely values of b_k to have been transmitted. Applying Bayes rule,

$$P(b_k|r_k) = \frac{P(r_k|b_k)P(b_k)}{P(r_k)}$$

For the AWGN channel with PAM-4 signalling, the PDF for a received sample r_k at time k, given that b_k was transmitted, is

$$P(r_k|b_k) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}(\frac{r_k - t_k}{\sigma})^2}$$

where t_k is the transmitted signal level corresponding to b_k. Assuming all PAM-4 symbols are equally likely to be transmitted,

$$P(b_k) = \frac{1}{4}$$

Substituting,

$$P(b_k|r_k) = \frac{\frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}(\frac{r_k - t_k}{\sigma})^2} \frac{1}{4}}{P(r_k)}.$$

Lumping the factors that are common to all four possible symbols b_k into a constant denoted λ gives

$$P(b_k|r_k) = \lambda e^{-\frac{1}{2}(\frac{r_k - t_k}{\sigma})^2}$$

Therefore, the log-likelihood that a symbol was transmitted is proportional to the squared difference between r_k and the corresponding transmitted signal level t_k. The most likely symbol denoted b_k^* has the lowest squared distance, and the second most likely transmitted symbol denoted b_k^{**} has the second lowest squared distance. The corresponding most and second most likely transmitted signal levels are denoted t_k^* and t_k^{**}. The log-likelihood ratio is then calculated as

$$\begin{aligned}
&\log\left(\frac{\lambda e^{-\frac{1}{2}(\frac{r_k - t_k^*}{\sigma})^2}}{\lambda e^{-\frac{1}{2}(\frac{d_k - t_k^{**}}{\sigma})^2}}\right) \\
&= -\frac{1}{2}\left(\frac{r_k - t_k^*}{\sigma}\right)^2 + \frac{1}{2}\left(\frac{r_k - t_k^{**}}{\sigma}\right)^2 \\
&= \frac{1}{2\sigma^2}((r_k - t_k^{**})^2 - (r_k - t_k^*)^2).
\end{aligned}$$

The hardware implementation only requires this log-likelihood ratio up to a scale factor, and so the factor $\frac{1}{2\sigma^2}$ is disregarded. Finally, the sign bit β is determined directly from the values of the most likely and second most likely symbols. If these two symbols differ in the LSB, then the LSB is the low-likelihood bit, and $\beta = 0$; if the symbols differ in the MSB, then the MSB is the low-likelihood

bit and $\beta = 1$.

Hardware Implementation

The number of bits used to represent the noisy received samples r_k, the number of bits used to represent the reliability values α_k, and the scale factor s are all customizable Verilog parameters. The simulations presented in this thesis use 5-bit values to represent α_k^{bin}. For good soft-decision decoding performance, the hardware values of α_k^{bin} are scaled so that typical values fall into the range of unsigned decimal numbers that can be represented with 5 bits (0-31).

For each noisy sample, the squared error term $(r_k^{bin} - t_k^{bin})^2$ is calculated for each possible symbol t_k^{bin}. For an 8-bit representation of t_k^{bin} and r_k^{bin}, these error terms are stored in 16-bit registers. The four possible error terms are partially sorted, determining the minimum and second-minimum terms corresponding to the most likely and second-most likely symbols. The difference between these two error terms is a 16-bit value that represents the log-likelihood ratio up to a scale factor, $\tilde{\alpha} \in \mathbb{F}_2^{1\times 16}$. Instead of scaling this 16-bit reliability down to 5 bits by simply dividing by 2^{11}, the following heuristic is used:

$$\alpha^{bin} = \min\left(31, \left\lfloor \frac{\tilde{\alpha}}{2^6} \right\rfloor\right)$$

Dividing by only 2^6 using right shift operation keeps an appropriate amount of separation between the 5-bit reliability values. However, this means the result may be greater than 31 for very reliable symbols, so the α value is clipped at 31 to maintain a 5-bit representation. This increases the resolution of α for the less-reliable symbols at the cost of saturating the α values at 31 for the higher-reliability symbols. Because the Chase algorithm for soft-decision decoding is only concerned with the six least-reliable PAM-4 symbols in a 64-symbol-long codeword, increasing the resolution for the low-reliability α values is beneficial because it allows the soft-decision decoder to distinguish between low-reliability levels, and the saturation of high-reliability symbols at $\alpha = 31$ is irrelevant to the soft-decision decoder's performance.

The full algorithm for the soft-output slicer is detailed in Algorithm 7:

Algorithm 7 Implementation of soft-output slicer, assuming 8-bit r_k, 5-bit α, $s = 24$.

Initial Information:

Algorithm:

for Each input noisy sample r_k **do**

Calculate squared error terms $e_k = (r_k^{bin} - t_k^{bin})^2$ from each $t_k^{bin} \in \{-72, -24, 24, 72\}$

Partially sort squared error terms to determine first and second most likely t_k^{*bin} and t_k^{**bin}

Calculate unscaled reliability $\tilde{\alpha} = (r_k^{bin} - t_k^{**bin})^2 - (r_k^{bin} - t_k^{*bin})^2$

Output $\alpha_k^{bin} = \min\left(31, \left\lfloor \frac{\tilde{\alpha}}{2^6} \right\rfloor\right)$

Output d_k corresponding to t_k^{*bin}

Output β_k corresponding to t_k^{*bin} and t_k^{**bin}

end for

Simulation Results

Figure 7.3 shows SNR vs. CER for the AWGN channel protected by the Hamming (68,60) inner FEC with soft-decision (SD) decoding and KP4 outer FEC. Simulation results with no interleaving, 4-way block interleaving, and 4-way block interleaving + full convolutional interleaving are shown. The same channel protected by a concatenated KP4 + hard-decision (HD) Hamming (128,120) FEC, and only a KP4 FEC, are included for comparison.

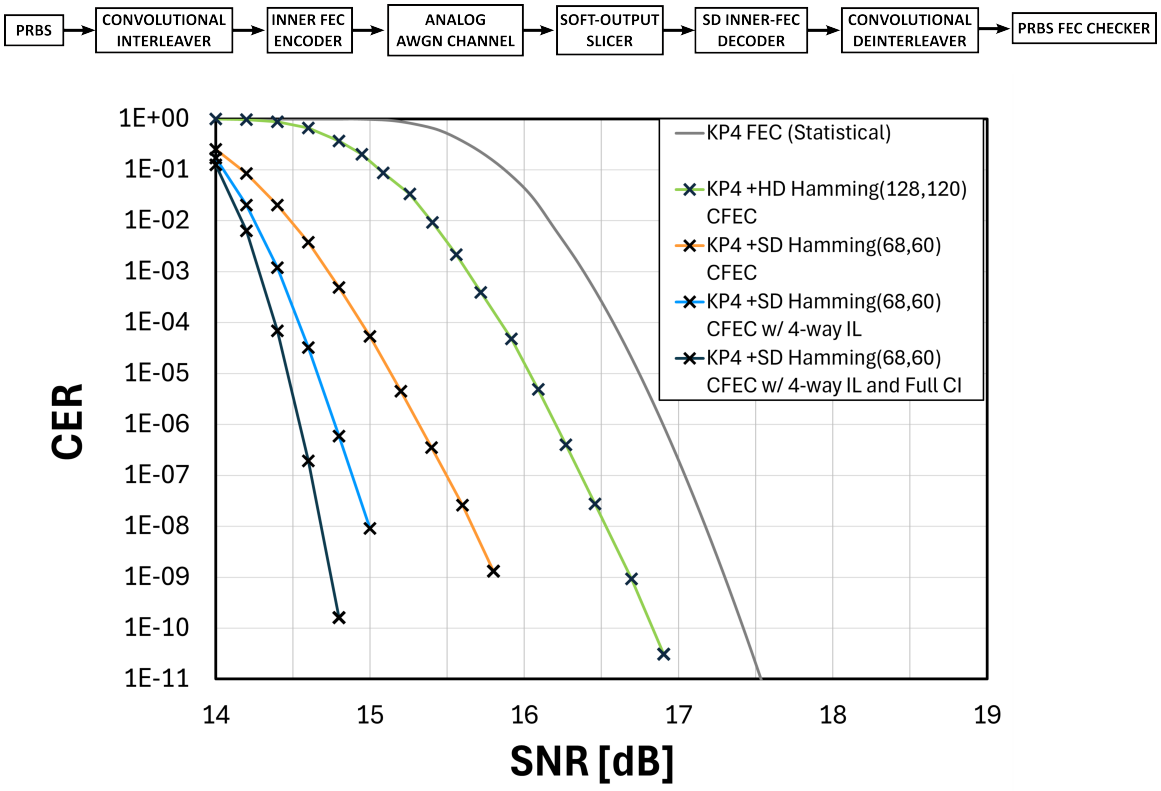

Figure 7.3: SNR vs. CER for the AWGN channel with soft-decision Hamming (68,60) inner code + KP4 outer code and various interleaving schemes.

Soft-decision decoding provides significant boost in performance over the concatenated FEC with hard-decision decoding, and interleaving provides another large improvement. With 4-way block interleaving and full convolutional interleaving, the system shows a 2 dB SNR improvement over the same link protected only by the KP4 FEC. Table 7.3 shows the hardware resources used for this simulation. The soft-decision slicer requires significantly more resources than the hard-decision slicer. Also, the soft-decision decoder requires substantial hardware resources because of the large storage and computation requirements to implement the Chase algorithm. This utilization is tripled due to the three-way cycling between parallel decoders to maintain a constant stream of output data. 20 parallel cores are used at 150 MHz for a simulation speed of 3 Gb/s.

Module Name	**# Look-up Tables**	**# Flip Flops**
PRBS	32	65
Convolutional Interleaver (full depth)	923	698
Inner FEC Encoder	562	375
TX Driver	3	5
AWGN Generator	2,183	4,280
Soft-Output slicer	163	16
Soft-Decision Inner FEC Decoder	12,269	1,813
Convolutional De-interleaver	903	697
FEC Checker	270	831
Total	17,667	8,780
× 20 Parallel Cores	353,000	176,000
Processor, Driver & Memory	5,000	5,000
Grand Total	358,000 (67%)	181,000 (17%)

Table 7.2: FPGA utilization for analog AWGN channel concatenated soft-decision Hamming (68,60) + KP4 FEC with convolutional interleaver using 20 parallel cores at 150 MHz for a simulation speed of 3 Gb/s.

7.3.2 Soft-Output Viterbi Algorithm

In their 1989 paper, Hagenauer and Hoeher proposed the soft-output Viterbi algorithm (SOVA) [30]. SOVA is a modified Viterbi algorithm that not only outputs the maximum likelihood state sequence for a Markov process, but also a reliability value for each state (soft decisions). Their original paper only applies to equalizing links with PAM-2 modulation. Later, in 1999, Gong et al. proposed a generalized SOVA that can be applied to PAM-4 links [31]. This algorithm has become popular for 200 Gb/s optical receivers with soft-decision inner FEC decoding. This section describes the SOVA algorithm, specifically for the soft-output equalization of a PAM-4 signal over a noisy $1 + \alpha D$ channel.

SOVA builds on the MLSD algorithm presented in Section 5.4, with some extra computation used to produce the reliability values α_k. Recall that the four survivor paths $\hat{u}(s_k)$ contain the most likely sequence of states starting from timestep $k - \delta$, and terminating in the four possible states at timestep k, $s_k \in \{0, 1, 2, 3\}$. For SOVA, an additional $\delta \times 4$ matrix of reliability values is maintained for each survivor path, denoted $\hat{L}(s_k)$. The entry in $\hat{L}(s_k)[j, \mu]$ represents the reliability difference (in log-likelihood ratio format) between the survivor path terminating in s_k and the most likely path terminating at state s at timestep k that visits state μ at timestep j, for some $k - \delta \leq j < k$.

Assuming that all survivor paths have merged at time step $k-\delta$ (they all visit some state $s^*_{k-\delta}$), this state s^* is on the most likely sequence of symbols, and it is chosen as the output of the MLSD algorithm for timestep $k-\delta$. Also, each of the four reliability matrices $\hat{L}(s_k)[j,\mu]$ will be equal for $j=\delta$ [30]. The reliability difference between the most likely symbol $s^*_{k-\delta}$ and some other symbol $s^{'}_{k-\delta}$ is $\hat{L}(s_k)[j=\delta, s^{'}]$. The state $s^{**}_{k-\delta}$ that minimizes this difference is the second most likely symbol at time step $k-\delta$, and so SOVA can output $\alpha_{k-\delta} = \hat{L}(s_k)[j=\delta, s^{**}]$. The sign bit $\beta_{k-\delta}$ is determined similarly to the soft-output slicer by simply looking at the values of the most and second-most reliable symbols.

The key to SOVA is updating the reliability matrices for the next timestep $\hat{L}(s_{k+1})[j,\mu]$, given the current reliability matrix $\hat{L}(s_k)[j,\mu]$ and new incoming sample r_{k+1}. To do this, the quantity Δ_m is defined for each state s_{k+1} as

$$\Delta_m = \Gamma(m_k, s_{k+1}) - \min_{l\in\{0,1,2,3\}} \Gamma(l_k, s_{k+1})$$

$$l, m \in \{0,1,2,3\}.$$

That is, Δ_m measures the difference between the minimum transition metric to a state s_{k+1} and the transition from state m to state s_{k+1}. For the most likely transition $m=l$ and $\Delta_m = 0$. The reliability matrix for $j=k$, $\hat{L}(s_{k+1})[j=k,\mu]$, is set to Δ_μ, and the following rule is used to update $\hat{L}(s_{k+1})[j,\mu]$ for $j=k-\delta,\ldots,k-1$:

$$\hat{L}(s_{k+1})[j,\mu] = \min_{m\in\{0,1,2,3\}} \{\hat{L}(m)[j,\mu] + \Delta_m\}$$

Algorithm 8 shows the SOVA algorithm. It is largely the same as the MLSD algorithm with the additions that produce the reliability matrix shown in red.

Algorithm 8 Soft-Output Viterbi Algorithm for PAM-4 $1 + \alpha D$ channel. [31]

Storage:

k (time index, modulo $\delta + 1$)

$\hat{u}_k = \{\hat{u}_{k-\delta}(s_k), ..., \hat{u}_k(s_k)\}, \quad s_k \in \{0, 1, 2, 3\}$ (survivor paths)

$\Gamma(s_k), \quad s_k \in \{0, 1, 2, 3\}$ (accumulated path metric for survivor paths)

$\hat{L}(s_k)[j, \mu] \quad s_k \in \{0, 1, 2, 3\}, j \in k - \delta, \ldots, k, \mu \in \{0, 1, 2, 3\}$ (reliability matrix for each survivor path)

Recursion:

for each state s_k **do**

Compute $\Gamma(s_{k-1}, s_k) = \Gamma(s_{k-1}) + \Gamma(s_{k-1}, sk)$ for all four transitions (s_{k-1}, s_k)

Find $\Gamma(s_k) = \gtrdot\sqsupset\ltimes(\Gamma(s_{k-1}, s_k))$

Store $\Gamma(s_k)$ and corresponding survivor $\hat{u}_k(s_k)$

Store $\Delta_m = \Gamma(m_k, s_{k+1}) - \min_{l \in \{0,1,2,3\}} \Gamma(l_k, s_{k+1})$

Update $\hat{L}(s_{k+1})_{k+1,mu} = \Delta_\mu$

for $j = k - \delta$ to $j = k$ **do**

Update $\hat{L}(s_{k+1})[\mu, j] = \min_{m \in \{0,1,2,3\}} \{\hat{L}(m)[\mu, j] + \Delta_m\}$

end for

end for

Hardware Implementation

The hardware implementation of the SOVA receiver closely matches Algorithm 8. Similar to MLSD, traceback length, the value of α for the $1 + \alpha D$ response, and number of bits used to represent reliability values are customizable Verilog parameters. For the simulations shown in this thesis, the traceback length is set to $\delta = 10$ and 5-bit values represent α_k.

Entries in the reliability matrices $\hat{L}(s_k)$ are stored in an unsigned 16-bit representation. Since each of these matrices has $\delta \times 4$ values, this introduces an additional 160 16-bit values to be stored, along with the appropriate hardware to add, compare and update. There is a significant increase in hardware resources over MLSD. Practical implementations of the SOVA algorithm for 200 Gb/s receivers may reduce complexity at the cost of a minor performance penalty by using fewer bits to represent the path metrics and reliability matrices. However, the implementation in the FPGA platform uses no optimizations and keeps the full bit resolution needed to accommodate the 8-bit noisy samples r_k^{bin} without any simplification or rounding. In this way, the FPGA platform's results represent an upper bound on the performance of practical SOVA-based receivers.

Simulation Results

Figure 7.4 shows SNR vs. CER for the $1 + 0.5D$ equalized with SOVA and protected by the concatenated Hamming (68,60) + KP4 FEC codes. Simulation results are shown for this architecture with no interleaving, 4-way block interleaving, and with 4-way block interleaving and full convolutional interleaving. For comparison, several other architectures are shown to protect the same $1 + 0.5D$ channel. The worst performance is from a DFE equalizer protected by the KP4

FEC only. An MLSD equalizer and KP4 FEC improves on the DFE by about 1 dB. The MLSD + concatenated hard-decision Hamming (128,120) + KP4 FEC improves further by about 0.4 dB. This is a relatively small improvement compared to the AWGN channel, which shows an improvement of around 0.7 dB when using the hard-decision inner FEC. This is because the correlated errors reduce the effectiveness of the Hamming (128,120) code with $t = 1$. The use of SOVA and the soft-decison inner FEC code improves over the hard decision inner-FEC code by about 0.5 dB. Due to correlated PAM-4 symbol errors, this is also a smaller improvement than what is measured on the AWGN channel, which has an 0.8 dB improvement with the soft-decision inner FEC code over the hard-decision inner-FEC code. Finally, the interleaving schemes show another significant boost in performance by breaking up the correlated 120-bit-long inner-FEC codewords.

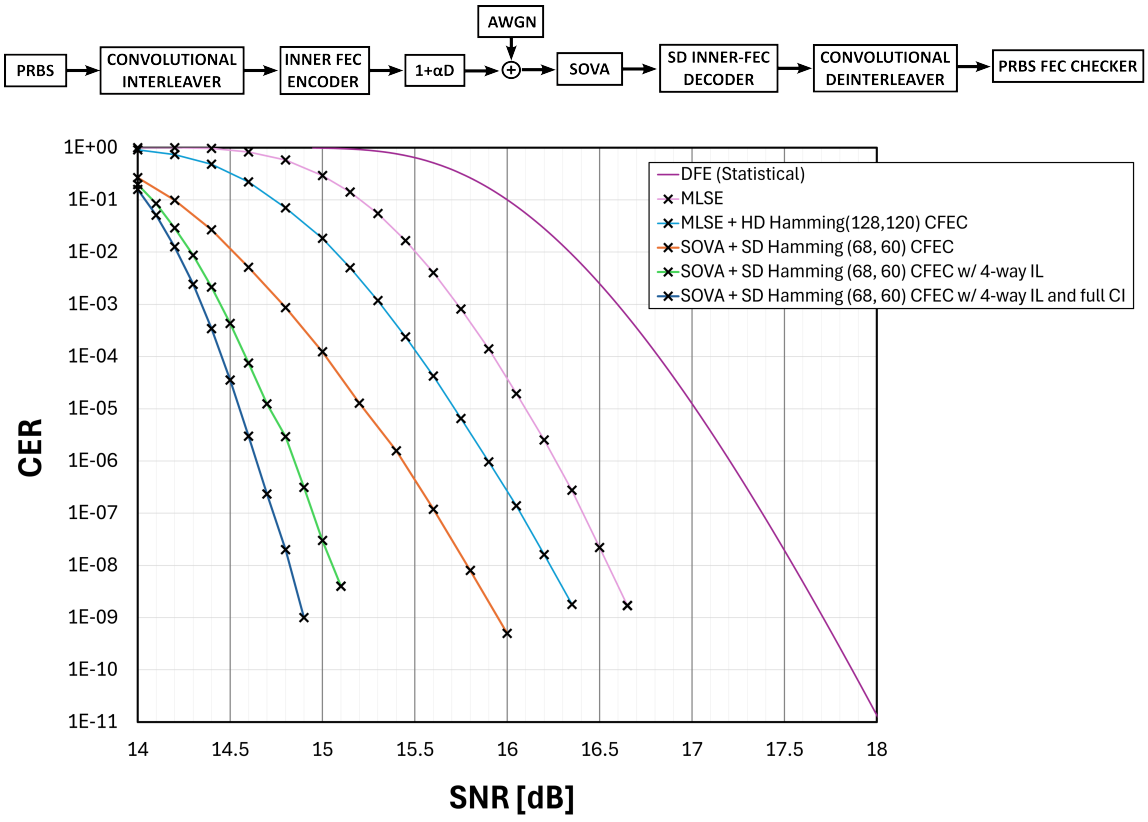

Figure 7.4: SNR vs. CER for $1+0.5D$ equalized with SOVA and protected by concatenated Hamming (68,60) + KP4 FEC.

This simulation was run on 10 parallel cores at 150 MHz for a total simulation speed of 1.5 Gb/s. The hardware utilization is summarized in Table 7.3. The SOVA module is the system's most hardware-intensive module, taking up large amounts of look-up tables, flip-flips, and DSPs. The utilization is significantly larger than the MLSD module because of the large memory and computation requirements to maintain and update the reliability matrices for the four survivor paths.

Module Name	**# Look-up Tables**	**# Flip Flops**	**# DSPs**
PRBS	32	65	0
Convolutional Interleaver (full depth)	923	698	0
Inner FEC Encoder	562	375	0
$1 + \alpha D$ response	3	4	0
AWGN Generator	2,196	4,279	0
Soft-Output Viterbi Algorithm	25,114	2,862	60
Soft-Decision Inner FEC Decoder	12,269	1,813	0
Convolutional De-interleaver (full depth)	903	690	0
FEC Checker	372	831	0
Total	42,618	11,590	60
$\times$ **10 Parallel Cores**	426,000	116,000	600
Processor, Driver & Memory	5,000	5,000	0
Grand Total	431,000 (80%)	121,000 (11%)	600 (75%)

Table 7.3: FPGA utilization for analog $1 + 0.5D$ channel with SOVA equalizer and concatenated soft-decision Hamming (68,60) + KP4 FEC with convolutional interleaver using 10 parallel cores at 150 MHz for a simulation speed of 1.5 Gb/s.

Chapter 8

Case Study

Chapters 4-7 introduced the custom Verilog modules that can be used to build a wireline system model for simulation on the FPGA platform. In this chapter, all these pieces are put together to demonstrate how the FPGA platform can be used in practice. A topic of ongoing discussion for 200 Gb/s multi-part links is the BER allocation between optical and electrical links, i.e., how high of a BER can be allowed in each of the three links while maintaining a sufficiently low post-FEC BER for the entire system? Correctly setting the BER allocation is essential to maintain interoperability for hosts and modules from different vendors. This is particularly important for 200 Gb/s applications because circuit designers are pushing the limits of available semiconductor technology to achieve such a high data rate, and so they do not have the margin to set conservative BER requirements. Recent IEEE standard contributions have proposed target BERs [32], but until now there has not been a reliable method to validate the proposals rigorously.

This chapter presents a model of a realistic multi-part link on the FPGA platform. The BER introduced in the electrical links is assumed to be at a known level. The optical link's SNR is swept to find the level at which the overall system's CER is sufficiently low. This target CER level would ideally be 5.5×10^{-11}, the Ethernet Standard's specification. However, the hardware complexity of this model limits the FPGA's simulation speed to 1.2 Gb/s. At this speed, it would take around 20 days to run a simulation demonstrating a CER of 5.5×10^{-11} with reasonable accuracy. An array of 20 parallel FPGAs could bring this time down to only one day. However, since only one FPGA is available for this research, this chapter assumes a target CER level of 5.5×10^{-9} so that simulations may be performed in one day.

Figure 8.1 shows the system-level model of a multi-part wireline system on the FPGA platform. This model represents the system detailed in Section 2.1, which does communication between data centres over a three-part electrical-optical-electrical link at 200 Gb/s per lane using the Ethernet protocol.

Model of Electrical Links

The electrical links are modelled with the error propagation factor model presented in Section 4.5.2. The initial error probability (IEP) is 2.67×10^{-5}, and the error propagation factor is 0.75. Precoding is used, introducing a BER of 4×10^{-5} by each of the electrical links. This model for host-to-module 200 Gb/s electrical links is taken from a recent IEEE Ethernet contribution [28].

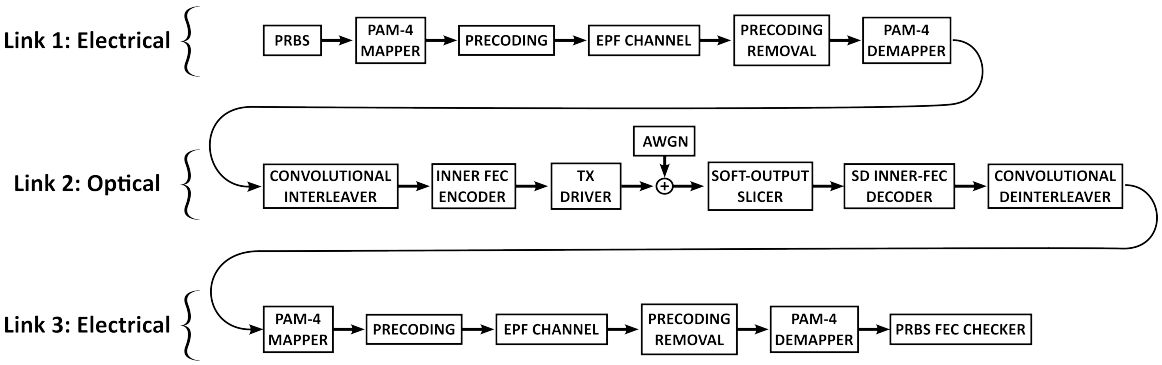

Figure 8.1: System-level architecture for FPGA model of wireline system with AWGN optical link.

Model of Optical Link

The optical link is modelled using the analog AWGN channel presented in Section 5.1 with the soft-output slicer from Section 7.3.1.

FEC architectures

Three FEC architectures are compared in this analysis:

1. **Full protection** mode uses the concatenated soft-decision Hamming (68,60) and KP4 FEC codes. 4-way block interleaving and full convolutional interleaving are used to achieve the optimal inner-FEC codeword with 12 FEC symbols from distinct outer-FEC codewords.

2. **Convolutional interleaver bypass** mode disables the convolutional interleaver to reduce the system's latency with a cost to performance. This mode still uses the concatenated soft-decision Hamming (68,60) + KP4 FEC code with 4-way block interleaving.

3. **Outer-FEC** mode bypasses the inner FEC completely, leaving the three-part link protected only by the outer KP4 code. This mode significantly reduces the power and area requirements of the system by eliminating the inner FEC, at a large cost to performance.

Figure 8.2 shows the optical link's SNR on the x-axis and the corresponding overall KP4 CER for the entire system on the y-axis for the three different FEC architectures.

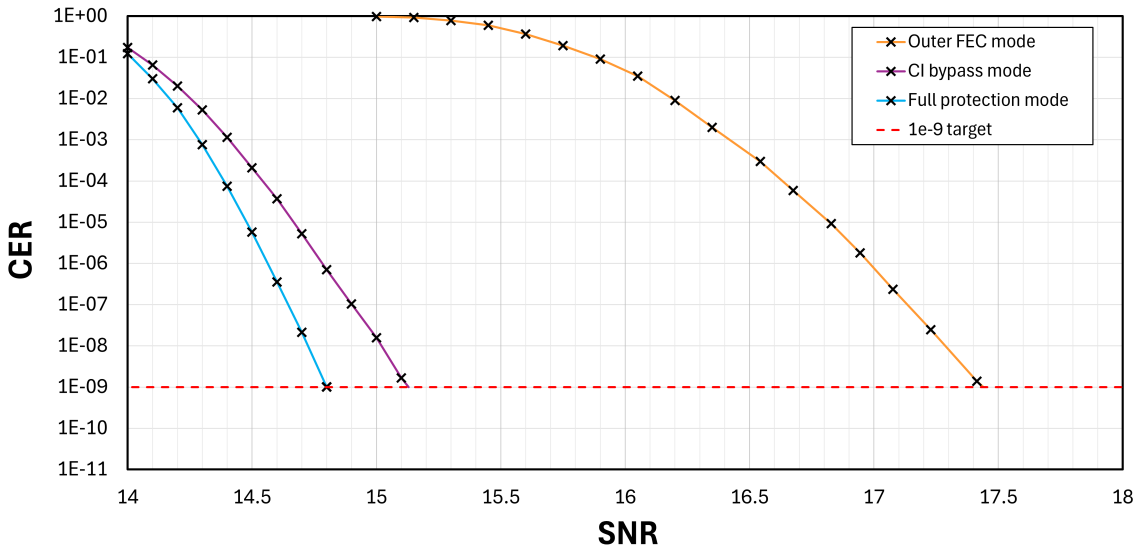

Figure 8.2: Optical link SNR vs. CER for multi-part link system.

Outer-FEC mode shows a significant performance gap from the other two candidates, with around 2 dB degradation at the target CER level. Because of this, bypassing the inner FEC is only practical for an optical link with a very high SNR. Of the other two candidates, the convolutional interleaver bypass mode shows around 0.3 dB degradation compared to the architecture with full protection. Table 8.1 lists the optical SNR required for this system to reach the target CER of 1×10^{-9} using each of the three FEC architectures. Since the Ethernet standard typically lists a BER allocation rather than an SNR allocation, this SNR level is also translated to a pre-FEC BER level in the third column of the table.

FEC Architecture	**Required Optical Link SNR**	**Equivalent Optical BER**
Full Protection	14.8 dB	5.2×10^{-3}
Bypass Convolutional Interleaver	15.13 dB	4.0×10^{-3}
Outer FEC	17.45 dB	3.2×10^{-4}

Table 8.1: Optical SNR and equivalent pre-FEC BER required to meet 10^{-9} CER requirement for multi-part link system.

$1 + 0.5D$ Optical Channel

Next, the same system is analyzed with a $1 + 0.5D$ channel to represent the optical link rather than an AWGN channel. The $1 + 0.5D$ channel is equalized with SOVA for the two architectures that include a soft-decision inner FEC. For the outer-FEC mode, which bypasses the inner FEC entirely, the channel is equalized with MLSD, equivalent to SOVA without the soft reliability values. The system-level diagram for this model is shown in Figure 8.3.

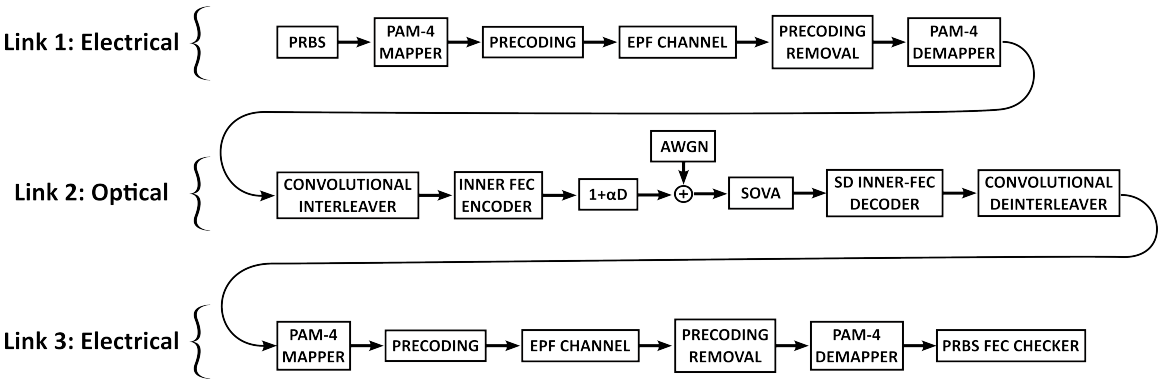

Figure 8.3: System-level architecture for FPGA model of wireline system with $1+0.5D$ optical link.

Figure 8.2 shows the optical link's SNR on the x-axis and the corresponding overall KP4 CER for the entire system on the y-axis for the three different FEC architectures.

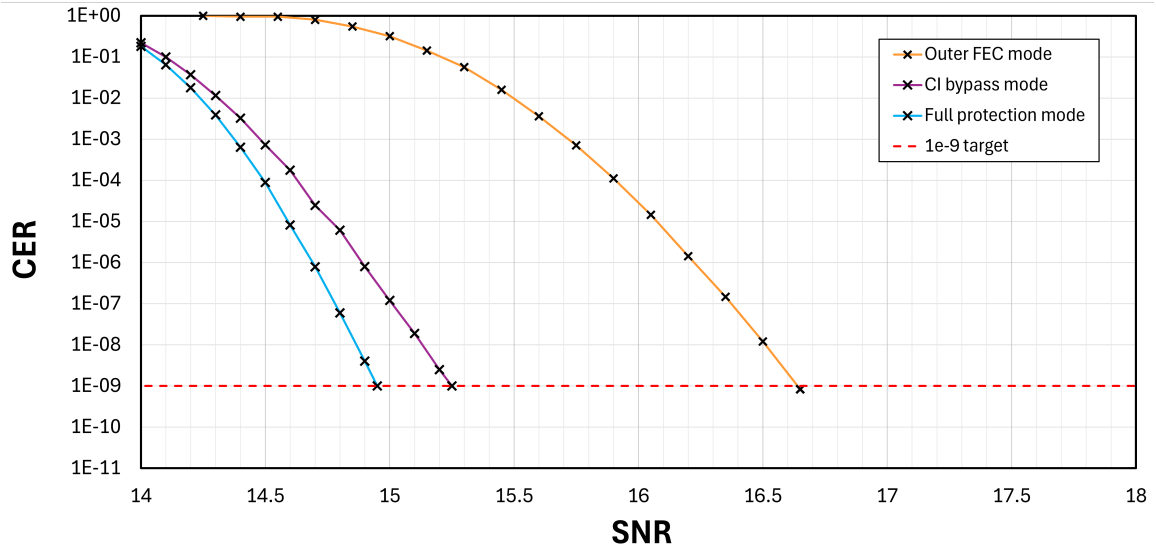

Figure 8.4: Optical link SNR vs. CER for multi-part link system with $1+0.5D$ optical link.

The outer-FEC mode actually shows a better performance compared to Figure 8.2. This is because MLSD benefits from a signal power that is amplified from the $1+0.5D$ response. Despite this increased signal power, the optical channel now suffers from correlated PAM-4 symbol errors, reducing the effectiveness of the inner FEC code. As such, the other two modes show an overall degradation in performance by about 0.1 dB compared to Figure 8.2. Table 8.2 lists the optical SNR and corresponding pre-FEC BER (assuming MLSD equalization of the optical link) required for this system to reach the target CER of 1×10^{-9} using each of the three FEC architectures.

FEC Architecture	**Required Optical Link SNR**	**Equivalent Optical BER**
Full Protection	14.95 dB	3.3×10^{-3}
Bypass Convolutional Interleaver	15.25 dB	2.4×10^{-3}
Outer FEC	16.65 dB	3.9×10^{-4}

Table 8.2: Optical SNR and equivalent BER (assuming MLSD equalization) required in optical $1 + 0.5D$ channel to meet 10^{-9} CER requirement for multi-part link system.

This analysis demonstrates how the FPGA platform may be used to develop BER allocations for the 200 Gb/s Ethernet standard. Another application of this analysis could be for system-level transceiver designers. Consider an engineer evaluating two candidate transimpedance amplifier designs for a 200 Gb/s PAM-4 optical receivers: One design is severely bandwidth-limiting and results in a $1 + 0.5D$ response, and the other has much higher bandwidth and achieves performance close to an AWGN channel, at the cost of a higher power and area. This FPGA platform can be an essential tool to measure how this difference in bandwidth would impact the overall CER of a complex wireline system.

Chapter 9

Conclusion

Tools to analyze the CER of 200 Gb/s wireline systems are needed to design reliable high-speed links and the standards to which they comply. These tools should be fast and flexible to estimate low CER levels for various channel types in a reasonable amount of time. Existing time-domain software platforms are prohibitively slow, and statistical methods lack the complexity required for realistic systems. This thesis presented an FPGA-accelerated platform to address this demand. The platform allows for the modelling of complex wireline systems, including multi-part links with concatenated FEC, soft-decision decoding, block interleaving, convolutional interleaving, and precoding, with the flexibility to explore different channel types and FEC architectures. It provides a significant speed increase of around 10,000 $\times$ over comparable time-domain software simulations, allowing for demonstration of the Ethernet standard's target CER of 5.5×10^{-11} within one day of simulation time for a simple PAM-4 AWGN channel. Although the platform can only demonstrate a CER of 10^{-9} for the full multi-part link model presented in Chapter 8, it can be scaled up using parallel FPGAs to demonstrate even lower CERs. This platform is specifically tailored for architectures relevant to 200 Gb/s Ethernet. I aim to use this platform to contribute to the upcoming Ethernet standard and for the system-level design of future Ethernet-compliant products.

Bibliography

[1] A. Ran and M. Brown, *Error Probability Thoughts*, IEEE standard 802.3dj, 2024. [Online]. Available: https://www.ieee802.org/3/dj/public/adhoc/electrical/24_0418/ran_3dj_elec_01a_240418.pdf.

[2] C. A. Belfiore and J. H. Park, "Decision Feedback Equalization," *Proceedings of the IEEE*, vol. 67, no. 8, pp. 1143–1156, 1979. DOI: 10.1109/PROC.1979.11409.

[3] H. Shakiba, *Error Propagation Analysis of MLSE*, IEEE standard 802.3dj, 2023. [Online]. Available: https://www.ieee802.org/3/dj/public/adhoc/electrical/23_0420/shakiba_3dj_elec_02_230420.pdf.

[4] K. Wu, G. Liga, J. Riani, and A. Alvarado, "Low-Complexity Soft-Decision Detection for Combating DFE Burst Errors in IM/DD Links," *Journal of Lightwave Technology*, vol. 42, no. 5, pp. 1395–1408, 2024. DOI: 10.1109/JLT.2023.3324602.

[5] "Draft Standard for Ethernet Amendment: Media Access Control Parameters for 1.6 Tb/s and Physical Layers and Management Parameters for 200 Gb/s, 400 Gb/s, 800 Gb/s, and 1.6 Tb/s Operation," *IEEE Std 802.3dj-2024*, 2024.

[6] "IEEE Standard for Ethernet," *IEEE Std 802.3-2022 (Revision of IEEE Std 802.3-2018)*, pp. 1–7025, 2022. DOI: 10.1109/IEEESTD.2022.9844436.

[7] P. Anslow, *BER and FER for 100GBASE-SR4*, IEEE standard 802.3bm, 2012. [Online]. Available: https://www.ieee802.org/3/bm/public/mmfadhoc/meetings/nov29_12/anslow_01a_1112_mmf.pdf.

[8] M. Yang, S. Shahramian, H. Shakiba, H. Wong, P. Krotnev, and A. C. Carusone, "Statistical BER Analysis of Wireline Links With Non-Binary Linear Block Codes Subject to DFE Error Propagation," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 67, no. 1, pp. 284–297, 2020. DOI: 10.1109/TCSI.2019.2943569.

[9] R. Barrie, *serdespy*, https://github.com/richard259/serdespy, 2022.

[10] M. Yang, S. Shahramian, H. Wong, P. Krotnev, and A. C. Carusone, "Pre-FEC and Post-FEC BER as Criteria for Optimizing Wireline Transceivers," in *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2021, pp. 1–5. DOI: 10.1109/ISCAS51556.2021.9401125.

[11] M. Yang, "Analysis and Optimization of Wireline Transceivers for Post-FEC BER," PhD thesis, University of Toronto, Toronto, ON, 2024.

[12] R. Barrie, M. Yang, and A. C. Carusone, "Statistical BER Analysis of Concatenated FEC in Multi-Part Links," in *DesignCon 2023*, 2023.

[13] J. Riani, B. Smith, A. Farhood, and L. Patra, *Impact of Burst Errors on Concatenated FEC scheme*, IEEE 802.3dj Task Force, 2023. [Online]. Available: https://www.ieee802.org/3/dj/public/23_03/riani_3dj_01a_2303.pdf.

[14] E. Lord, M. Devlin, N. Harold, and C. Sanderson, "Accelerating BER analysis using an FPGA based processing platform," in *MILCOM 2002. Proceedings*, vol. 1, 2002, 218–223 vol.1. DOI: 10.1109/MILCOM.2002.1180443.

[15] B. P. Smith, "Error-Correcting Codes for Fibre-Optic Communication Systems," PhD thesis, University of Toronto, Toronto, ON, 2011.

[16] W. Wang, Z. Long, W. Qian, *et al.*, "Real-Time FPGA Investigation of Potential FEC Schemes for 800G-ZR/ZR+ Forward Error Correction," *Journal of Lightwave Technology*, vol. 41, no. 3, pp. 926–933, 2023. DOI: 10.1109/JLT.2022.3218957.

[17] D. Zou, K. Song, Z. Chen, C. Zhu, T. Wu, and Y. Xu, "FPGA-Based Configurable and Highly Flexible PAM4 SerDes Simulation System," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 31, no. 9, pp. 1294–1307, 2023. DOI: 10.1109/TVLSI.2023.3286803.

[18] C. J. Clopper and E. S. Pearson, "The Use of Confidence or Fiducal Limits Illustrated in the Case of the Binomial," *Biometrika*, vol. 26, no. 4, pp. 404–413, Dec. 1934, ISSN: 0006-3444. DOI: 10.1093/biomet/26.4.404. eprint: https://academic.oup.com/biomet/article-pdf/26/4/404/823407/26-4-404.pdf. [Online]. Available: https://doi.org/10.1093/biomet/26.4.404.

[19] P. Alfke, *Efficient Shift Registers, LFSR Counters, and Long Pseudo-Random Sequence Generators*, Xilinx Application Note, 1996. [Online]. Available: https://docs.amd.com/v/u/en-US/xapp052.

[20] S. Bhoja, W. Bliss, C. Chen, V. Parthasarathy, J. Wang, and Z. Wang, *Precoding Proposal for PAM-4 Modulation*, IEEE standard 802.3bj, 2011. [Online]. Available: https://www.ieee802.org/3/bj/public/sep11/parthasarathy_01_0911.pdf.

[21] R. McEliece and L. Swanson, "On the decoder error probability for Reed - Solomon codes (Corresp.)," *IEEE Transactions on Information Theory*, vol. 32, no. 5, pp. 701–703, 1986. DOI: 10.1109/TIT.1986.1057212.

[22] G. C. Clark and J. B. Cain, *Error-correction coding for digital communications*. Plenum Press, 1988.

[23] A. Farhood, W. Bliss, S. Ramesh, and D. Cassan, *Concatenated FEC baseline proposal for 200Gb/s per lane IM-DD Optical PMD*, IEEE standard 802.3dj, 2023. [Online]. Available: https://grouper.ieee.org/groups/802/3/dj/public/23_01/23_0206/farhood_3dj_01a_230206.pdf.

[24] R. C. Tausworthe, "Random Numbers Generated by Linear Recurrence Modulo Two," *Mathematics of Computation*, vol. 19, pp. 201–209, 1965. [Online]. Available: https://api.semanticscholar.org/CorpusID:264663238.

[25] D.-U. Lee, J. Villasenor, W. Luk, and P. Leong, "A hardware Gaussian noise generator using the Box-Muller method and its error analysis," *IEEE Transactions on Computers*, vol. 55, no. 6, pp. 659–671, 2006. DOI: 10.1109/TC.2006.81.

[26] A. J. Walker, "An Efficient Method for Generating Discrete Random Variables with General Distributions," *ACM Trans. Math. Softw.*, vol. 3, no. 3, pp. 253–256, Sep. 1977, ISSN: 0098-3500. DOI: 10.1145/355744.355749. [Online]. Available: https://doi.org/10.1145/355744.355749.

[27] G. Forney, "The Viterbi Algorithm," *Proceedings of the IEEE*, vol. 61, no. 3, pp. 268–278, 1973. DOI: 10.1109/PROC.1973.9030.

[28] H. Xiang, K. Huang, M. Dudek, and L. Patra, *Low Latency Mode for Inner FEC*, IEEE 802.3dj Task Force, 2023. [Online]. Available: https://www.ieee802.org/3/dj/public/24_01/he_3dj_01_2401.pdf.

[29] D. Chase, "Class of algorithms for decoding block codes with channel measurement information," *IEEE Transactions on Information Theory*, vol. 18, no. 1, pp. 170–182, 1972. DOI: 10.1109/TIT.1972.1054746.

[30] J. Hagenauer and P. Hoeher, "A Viterbi Algorithm with Soft-Decision Outputs and its Applications," in *1989 IEEE Global Telecommunications Conference and Exhibition 'Communications Technology for the 1990s and Beyond'*, 1989, 1680–1686 vol.3. DOI: 10.1109/GLOCOM.1989.64230.

[31] L. Gong, W. Xiaofu, and Y. Xiaoxin, "On SOVA for nonbinary codes," *IEEE Communications Letters*, vol. 3, no. 12, pp. 335–337, 1999. DOI: 10.1109/4234.809527.

[32] A. Ran, *Proposal for BER budget allocation for AUIs in Type 1 and Type 2 PHYs*, IEEE standard 802.3dj, 2023. [Online]. Available: https://www.ieee802.org/3/dj/public/23_05/ran_3dj_01a_2305.pdf.