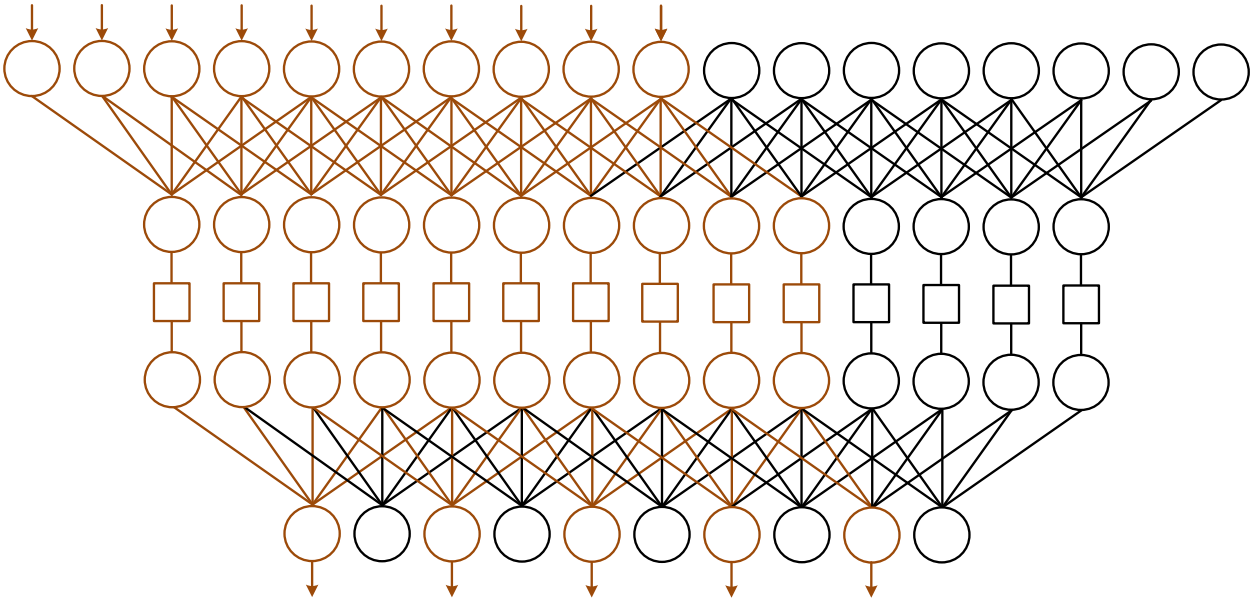
CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG



FPGA Implementation of Machine Learning Based Nonlinear Equalizer with On-Chip Training

Master's thesis in Embedded Electronic System Design

KEREN LIU

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2022

Master's thesis 2022

FPGA Implementation of Machine Learning Based Nonlinear Equalizer with On-Chip Training

KEREN LIU

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2022

FPGA Implementation of Machine Learning Based
Nonlinear Equalizer with On-Chip Training
KEREN LIU

Supervisors:
Christian Häger, Department of Electrical Engineering
Erik Börjesson, Department of Computer Science and Engineering

Examiner:
Per Larsson-Edefors, Department of Computer Science and Engineering

Master's Thesis 2022
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Schematic of the proposed on-chip training.

Typeset in LaTeX
Gothenburg, Sweden 2022

FPGA Implementation of Machine Learning Based
Nonlinear Equalizer with On-Chip Training
KEREN LIU
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

In fiber-optical communication, the linear polarization mode dispersion (PMD) effect and the nonlinear optical Kerr effect have a combined detrimental effect on the transmitted signal, and the time-varying nature of PMD also means that the PMD-Kerr effect can change over time. Therefore, a nonlinear equalizer that compensates for both PMD and Kerr effects and is adaptive to the time-varying PMD-Kerr channel is needed at the receiver side. In recent years, machine learning algorithms, especially neural networks, have been introduced to the fiber-optical field and they offer a promising way to construct a nonlinear equalizer in the digital signal processing (DSP) part in the receiver. In this work, we have implemented an adaptive machine learning based nonlinear equalizer on a field-programmable gate array (FPGA). A model-based machine learning algorithm is adopted and modified for FPGA implementation, and the training of the equalizer is also implemented on the same chip. The equalizer contains multiple layers, which has essentially the same structure as a neural network. The on-chip training realizes the backward propagation to optimize the weights in the equalizer, which allows the equalizer to adapt to the time-varying PMD-Kerr channel in real-time. Logic simulations of a 3-layer equalizer show that its static performance is very close to the ideal upper limit of the original 3-layer software model, and that the 3-layer equalizer can remain stable until the varying speed of each principal state of polarization (PSP) reaches above 1×10^5 rad/s in the same rotation direction. Final FPGA implementation results show that a 3-layer equalizer utilizes around 79.72% of the total DSP resources on a medium size FPGA, and the DSP becomes the bottleneck of FPGA resources. The on-chip training constitutes most of the used resources due to its complexity and intensive arithmetic computation.

Keywords: fiber-optical communication, PMD effect, optical Kerr effect, machine learning, adaptive nonlinear equalizer, FPGA, on-chip training

Acknowledgements

I would like to thank my thesis supervisors, Prof. Christian Häger, for providing me with this thesis opportunity and giving me enlightening supervision and helpful guidance during the thesis; and Erik Börjesson, for his many suggestions and our discussions about digital circuit design, Matlab and RTL coding, and experiment setup. I would also like to thank Prof. Per Larsson-Edefors for his careful examination of this thesis and for providing me with the working environment and equipment.

Keren Liu, Gothenburg, October 2022

Contents

Acronyms

ANN Artificial neural network
ASIC Application specific integrated circuit
AWGN Additive white Gaussian noise
BER Bit error rate
BP Backward propagation
BRAM Block random access memory
DGD Differential group delay
DUT Design under test
FFT Fast Fourier transform
FIR Finite impulse response
FP Forward propagation
FPGA Field-programmable gate array
IC Integrated circuit
IFFT Inverse fast Fourier transform
ISI Intersymbol interference
LUT Look-up table
MF Matched filter
MIMO Multi-input multi-output
ML Machine learning
NN Neural network
QAM Quadrature amplitude modulation
QPSK Quadrature phase-shift keying
RNG Random number generator
RRC Root-raised-cosine
RTL Register-transfer level
SGD Stochastic gradient descent
SNR Signal-to-noise ratio
SR Shift register
SSFM Split-step Fourier method
STD State-transition diagram
VHDL Very high speed integrated circuit hardware description language

1
Introduction

In single-mode dual-polarization optical fiber there exist several impairment effects in the waveguide resulting in bit error rate (BER) loss at the receiver. Among all the impairment effects, polarization mode dispersion (PMD) [1] and optical Kerr effect [2] are studied in this thesis project. PMD is a linear time-varying impairment that requires adaptive equalization. The optical Kerr effect is nonlinear and together with PMD it will have a combined detrimental effect on the transmitted signal [3, 4]. Thus, it is important to develop adaptive nonlinear equalizers to compensate for the PMD-Kerr effect in receivers.

Various algorithms and corresponding nonlinear equalizer designs have been proposed to compensate for the PMD-Kerr effect, and some of them are built as DSP units in receivers. The DSP units are usually implemented on digital integrated circuits (ICs) such as field-programmable gate arrays (FPGAs) and application specific integrated circuits (ASICs) [5]. Therefore, this thesis aims at implementing a machine learning (ML) based nonlinear equalizer on an FPGA for the compensation of the PMD-Kerr effect. A model-based neural network (NN) equalizer [6] originally simulated only in software is chosen as the basis of our work and appropriately modified to facilitate an efficient FPGA implementation. The training of the equalizer is also implemented on the same FPGA and the on-chip training can work in parallel with the inference, which enables the equalizer to adapt to time-varying PMD-Kerr channels in real-time.

Several experiments can be carried out on our proposed equalizer to assess its performance and convergence speed. The experiments can be divided into two categories. The first one aims to find the optimal hyperparameters of the equalizer in terms of performance, convergence speed and hardware utilization using a grid search. The second one aims at testing the implemented equalizer with a time-varying PMD-Kerr channel.

1.1 Related Work

There is a large number of published papers that focus on the digital IC implementations of non-ML-based equalizers. The non-ML-based equalizers are usually only linear equalizers. Some works adopt ASICs [7, 8, 9, 10, 11] and some adopt FPGAs [12, 13, 14, 15] as the platforms to realize non-ML-based linear equalizers.

Machine learning can be used to construct equalizers and these equalizers usually outperform traditional equalizers for their ability of simultaneous equalization of

both linear and nonlinear impairments [16]. There are lots of previous works focusing on the theoretical models of the ML-based equalizers [6, 17, 18, 19, 20, 21, 22]. Like non-ML-based linear equalizers, ML-based nonlinear equalizers can also be implemented on ASICs [23, 24, 25] or FPGAs [26, 27, 28, 29, 30]. Unlike these previous ML-based nonlinear equalizers implemented on FPGAs, this thesis firstly implements an NN-based nonlinear equalizer for PMD-Kerr effect with gradient backward propagation (BP) working in parallel on an FPGA.

Among all the published ML-based equalizer models for PMD-Kerr effect, the equalizer proposed in [6] is chosen as the original model in this thesis for its feasibility of hardware implementation.

1.2 Thesis Contributions

This section briefly introduces what has been accomplished in this thesis and our contributions. Overall, an adaptive ML-based equalizer for the PMD-Kerr effect has been built on an FPGA with on-chip gradient BP, and the equalizer's adaptive function can work in real-time. The detailed contributions can be summarized as:

- The equalizer is able to compensate for both the PMD effect and the optical Kerr effect in single-mode dual-polarization fiber-optical communication.
- The equalizer has an adaptive capability of compensating for time-varying PMD effect in real-time.
- The inference and training of the original model-based NN equalizer in [6] has been thoroughly studied and necessary modifications have been made on it for FPGA implementation.
- Both the inference and training have been implemented on the same FPGA chip and can work in parallel.

This thesis dedicates to answering the following research questions that arise in the process of achieving the above results:

- How should the original equalizer model be modified to be mapped to FPGAs while also keeping its real-time adaptive functionality?
- How should the general architecture be designed and the specific circuits be built to realize the modified equalizer model?
- How should the performance and convergence speed of the FPGA-based equalizer be tested and compared against the original software model?

1.3 Thesis Outline

The contents of each chapter can be summarized as follows: Chapter 2 is an introduction to the basics of fiber-optical communication systems, PMD and optical Kerr effects, neural networks, and supporting works. Chapter 3 explains the general methodology used in this thesis project. Chapter 4 describes the original equalizer model and the modified equalizer model for FPGA implementation. Chapter 5

demonstrates the FPGA implementation of the proposed equalizer, and each hardware component is explained in detail. Chapter 6 describes the design and setup of the experiments performed on the proposed equalizer and presents the results of each experiment. The FPGA synthesis and implementation results are also demonstrated. Chapter 7 discusses possible future work to improve or extend the proposed equalizer design. Chapter 8 summarizes the whole thesis including the main contributions and the final results achieved.

2

Technical Background

This chapter starts by introducing the general structure of fiber-optical communication systems. Then the transmitter, channel and equalizer are described separately. In addition, two supporting works of this thesis are briefly introduced.

2.1 Fiber-Optical Communication System

Fig. 2.1 shows the general and simplified structure of fiber-optical communication system including a transmitter, a channel, and a receiver. In the simplified structure, all the optical and electronic components except for the fiber are ignored, and only the DSP components that are relevant to this thesis are shown.

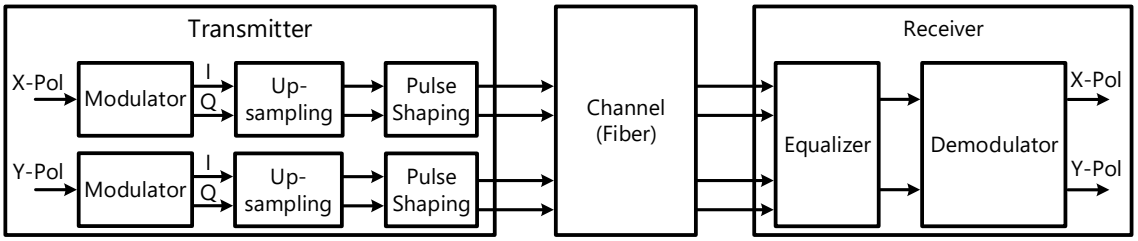


Figure 2.1: The simplified structure of fiber-optical communication system.

The modulators in the transmitter modulate the x-polarization and y-polarization symbols respectively to an in-phase (I) and a quadrature (Q) signals in some format, such as quadrature phase-shift keying (QPSK). The modulated signals are then up-sampled with an up-sampling factor. Two identical pulse-shaping filters are then applied to each up-sampled signal. There are different kinds of pulse-shaping filters and one commonly used is called root-raised cosine filter. The optical components after the modulator convert the electrical signals into optical signals and the optical signals are transmitted through optical fiber. The channel, i.e., the optical fiber, introduces the PMD and the optical Kerr effects as well as other impairments to the optical signals which degrade the quality of the transmitted signals. The optical components after the channel and before the receiver converts the optical signals back to two electrical signals, and following DSP components can be applied to them. In the DSP receiver part, the channel impairments are compensated by the equalizer, and the matched filters and down-sampling are usually part of the equalizer. The MFs restore the signals after pulse shaping and then the down-sampling

is applied. The down-sampled dual-polarization signals are finally demodulated by demodulators.

2.2 Transmitter

The transmitter in Fig. 2.1 includes modulators, up-sampling components and pulse-shaping filters. The basic principle of each component is introduced in this section.

2.2.1 Digital Modulation Formats

The digital modulation converts the binary data stream into multi-bit symbols by mapping the binary stream through an alphabet. There are several different alphabets and the constellation diagrams of two typical examples, QPSK and 16 quadrature amplitude modulation (QAM), are shown in Fig. 2.2. The Gray-coded QPSK has 4 possible discrete symbols from "00" to "11", while the 16QAM has 16 possible discrete symbols from "0000" to "1111".

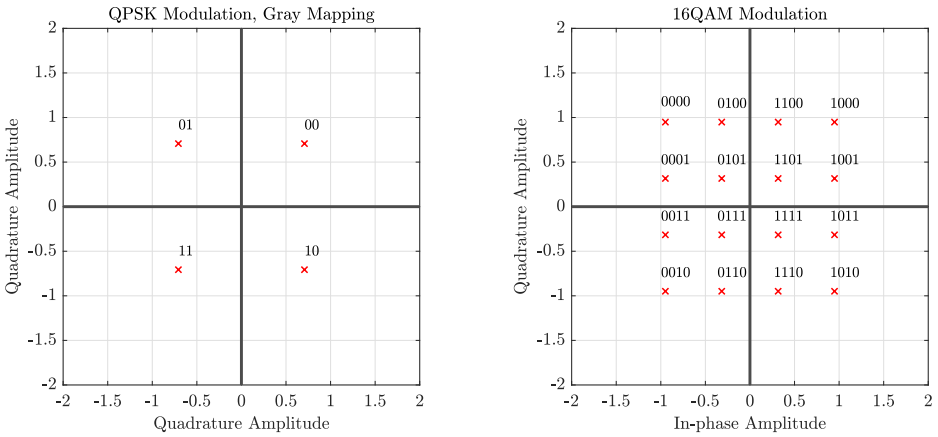


Figure 2.2: The constellation diagram of Gray-coded QPSK and 16QAM. The average power is set to 1 W in both constellations.

2.2.2 Up-Sampling and Pulse Shaping

The pulse-shaping filter is a kind of low-pass filter proposed to reduce the intersymbol interference (ISI) [31] to zero in the transmitter while also limiting the bandwidth of the modulated signal. The root-raised cosine (RRC) filter is one typical kind of pulse-shaping filter, and its impulse response $h(t)$ is

$$h(t) = \frac{\sin(\pi t/T)}{\pi t} \frac{\cos(\pi t\beta/T)}{1 - 4\beta^{n2}t^2/T^2} \tag{2.1}$$

where t is the time, T is the symbol duration and β is the roll-off factor [32]. One important feature of RRC is that its matched filter (MF) is itself. A finite impulse response (FIR) approximation of the RRC filter can be derived from Eq. (2.1) if given the number of samples per symbol and the filter span in number of symbols.

As shown in Fig. 2.1, the modulated signal should firstly be up-sampled with interpolation of 0, and then a pulse-shaping filter can be applied to it. In an FIR realization of RRC, the up-sampling and the RRC filter can usually be merged into one component where the up-sampling and the pulse-shaping can be performed at the same time. Assuming that the up-sampling factor is 2, the data stream is changed from 1 sample per symbol to 2 samples per symbol after going through up-sampling and RRC filter.

2.3 Channel

This thesis focuses on two impairments in the fiber-optical channel: the linear PMD effect and the nonlinear optical Kerr effect. This section briefly introduces the split-step Fourier method (SSFM) based Manakov-PMD model and also the simplified PMD-Kerr model adopted in this thesis.

2.3.1 SSFM Based Manakov-PMD Model

Birefringent fibers can be described by the Manakov-PMD equation which does not have any general analytic solutions [33]. The SSFM can be used to solve the Manakov-PMD equation numerically and the solution is usually used to model and simulate the PMD-Kerr effect [34]. In the solution to model first-order PMD combined with optical Kerr effect, a fiber is divided into multiple concatenated sections and each section is further divided into multiple steps, as shown in Fig. 2.3. There are K sections and each section is divided into S steps noted as "St.". Each section has an equal length L_c, and each step has an equal length L_c/S. The differential group delay (DGD) noted as $\tau^{(k)}$ refers to the difference in propagation time between the two polarizations in section k. The DGD in each step is thus $\tau^{(k)}/S$. There is an ε block following each step which models the optical Kerr effect. The $R^{(k)}$ is the rotation matrix of PSP following the last optical Kerr block in each section.

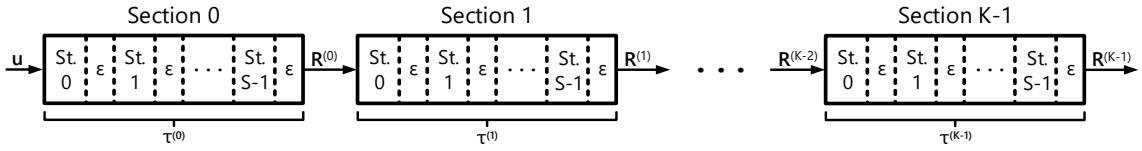


Figure 2.3: The SSFM based Manakov-PMD model consisting of K sections with S steps in each section.

In each step, the DGD is modeled as

$$\boldsymbol{J}^{(k)}(\omega) = \begin{bmatrix} \exp(j\omega\frac{-\tau^{(k)}}{2S}) & 0 \\ 0 & \exp(j\omega\frac{\tau^{(k)}}{2S}) \end{bmatrix} \tag{2.2}$$

in which the DGD is realized in the frequency domain. The x-polarization and the y-polarization are delayed by $-\tau^{(k)}/(2S)$ and $\tau^{(k)}/(2S)$ respectively, which leads to a DGD of $\tau^{(k)}/S$ in each step in section k. The $\boldsymbol{u}$ denotes the Jones vector and we

have $\boldsymbol{u} = [u_x(t,z), u_y(t,z)]^\top$ where t is the propagation time, z is the propagation distance, and u_x and u_y are the two polarizations. The optical Kerr effect after each DGD step in every section can be modeled as

$$\varepsilon(\boldsymbol{u}) = \boldsymbol{u} \exp(\frac{8}{9} j \frac{L_c}{S} \gamma \left\|\boldsymbol{u}\right\|^2) \tag{2.3}$$

where γ denotes the Kerr parameter and 8/9 is the averaging coefficient. The PSP rotation after each section can be modeled as

$$\boldsymbol{R}^{(k)} = \begin{bmatrix} m^{(k)} & n^{(k)} \\ -(n^{(k)})^* & (m^{(k)})^* \end{bmatrix} \quad m^{(k)}, n^{(k)} \in \mathbb{C},\ |m^{(k)}|^2 + |n^{(k)}|^2 = 1 \tag{2.4}$$

2.3.2 Simplified PMD-Kerr Model

To facilitate FPGA implementation, a simplified version of SSFM-based Manakov-PMD model is adopted in this thesis. In this simplified model, there is only one SSFM step and one ε component in each section, in other words, $S = 1$ is substituted in Eq. (2.2) and Eq. (2.3) and we have

$$\boldsymbol{J}^{(k)}(\omega) = \begin{bmatrix} \exp(j\omega\frac{-\tau^{(k)}}{2}) & 0 \\ 0 & \exp(j\omega\frac{\tau^{(k)}}{2}) \end{bmatrix} \tag{2.5}$$

and

$$\varepsilon(\boldsymbol{u}) = \boldsymbol{u} \exp(j\bar{\gamma} \left\|\boldsymbol{u}\right\|^2) \tag{2.6}$$

where the symbol $\bar{\gamma}$ is used to represent $\frac{8}{9} L_c \gamma$ for simplicity[1]. The PSP rotation matrix $\boldsymbol{R}^{(k)}$ is also simplified as

$$\boldsymbol{R}^{(k)} = \begin{bmatrix} \cos(\theta^{(k)}) & \sin(\theta^{(k)}) \\ -sin(\theta^{(k)}) & \cos(\theta^{(k)}) \end{bmatrix} \quad \theta^{(k)} \in \mathbb{R} \tag{2.7}$$

which is actually a special case of Eq. (2.4). Fig. 2.4 shows the simplified PMD-Kerr model. Another difference is that the simplified model has an individual input rotation matrix while the SSFM-based model does not.

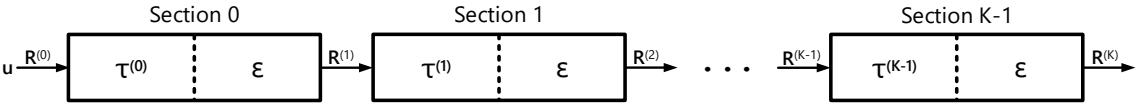


Figure 2.4: The simplified PMD-Kerr model consisting of K sections.

2.4 Equalizer

This section focuses on the basic knowledge of our implemented equalizer including deep learning and model-based neural network.

[1]The symbol $\bar{\gamma}$ will also be referred as "Kerr parameter" in the following.

2.4.1 Deep Learning

This subsection briefly describes the basic principles behind deep learning, which is the basis of the adopted adaptive model-based ML equalizer in this thesis. The description includes two parts, the artificial neural network (ANN) and the batch-based training.

2.4.1.1 Artificial Neural Network

An ANN can be described as a directed computational graph in which each node contains a scalar function $f^{(n)}(\boldsymbol{x}^{(n)}; \boldsymbol{\Psi}^{(n)})$. The $\boldsymbol{x}^{(n)}$ is the input vector and the $\boldsymbol{\Psi}^{(n)}$ is a set of tunable parameters. Such scalar function is usually a linear operation and it is also followed by a nonlinear function in most cases, which is known as an activation function. A typical example is the node used in the fully-connected layers in feedforward ANN which can be described as

$$f^{(n)}(\boldsymbol{x}^{(n)}; \boldsymbol{\Psi}^{(n)}) = \varepsilon((\boldsymbol{w}^{(n)})^{\top}\boldsymbol{x}^{(n)} + b^{(n)}) \tag{2.8}$$

where ε is the activation function, $\boldsymbol{w}^{(n)}$ is known as the weight vector, and $b^{(n)}$ is called a bias. We also have the tunable parameter vector $\boldsymbol{\Psi}^{(n)} = \{\boldsymbol{w}^{(n)}, b^{(n)}\}$. A group of nodes sharing the same inputs is defined as a layer l and each layer can be viewed as a multi-input multi-output (MIMO) function:

$$g^{(l)}(\boldsymbol{x}^{(l)}; \boldsymbol{\Omega}^{(l)}) = [f^{(0)}(\boldsymbol{x}^{(l)}; \boldsymbol{\Psi}^{(0)}), f^{(1)}(\boldsymbol{x}^{(l)}; \boldsymbol{\Psi}^{(1)}), \ldots, f^{(N-1)}(\boldsymbol{x}^{(l)}; \boldsymbol{\Psi}^{(N-1)})] \tag{2.9}$$

where the layer contains N nodes and we have $\boldsymbol{\Omega}^{(l)} = \{\boldsymbol{\Psi}^{(0)}, \boldsymbol{\Psi}^{(1)}, \ldots, \boldsymbol{\Psi}^{(N-1)}\}$.

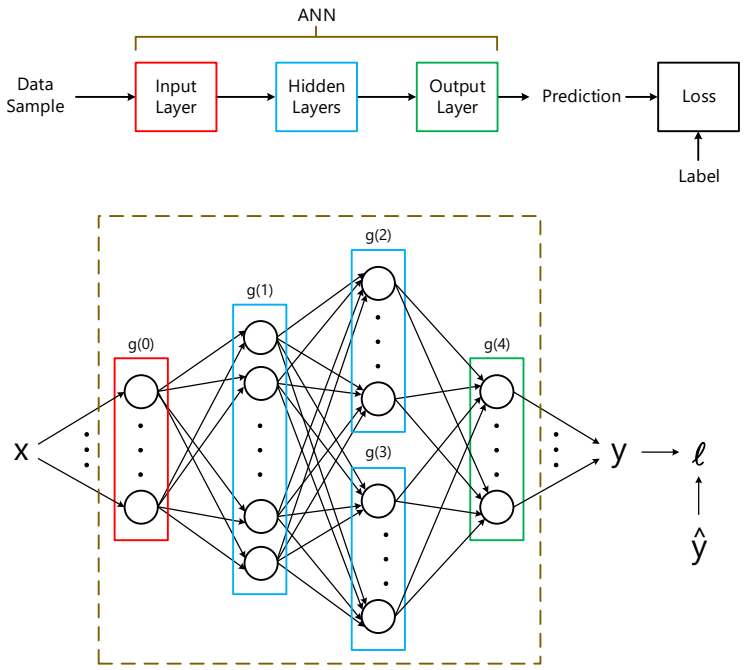


Figure 2.5: The structure of an example of ANN including the fully-connected layers [6].

By implementing L layers and connecting their inputs and outputs, an ANN can be constructed. The directed computational graph of the entire ANN can be abstracted as $\boldsymbol{y} = f(\boldsymbol{x}; \boldsymbol{\Phi})$, where $\boldsymbol{\Phi} = \{\boldsymbol{\Omega}^{(0)}, \boldsymbol{\Omega}^{(1)}, \ldots, \boldsymbol{\Omega}^{(L-1)}\}$ is the vector of all tunable parameters, $\boldsymbol{x}$ denotes the input vector to the whole ANN, and $\boldsymbol{y}$ denotes the output vector of the ANN, i.e., the prediction. Fig. 2.5 shows an example ANN including the fully-connected layers. The layers are divided into an input layer, multiple hidden layers, and an output layer.

2.4.1.2 Batch-Based Training

In neural networks, the process of input samples propagating through the network in the forward direction is called forward propagation (FP). By comparing the FP outputs and the desired results, the tunable parameters in a neural network can be optimized by updating them in an iterative fashion and this process is called learning or training. The training is usually batch based, which means that more than one input sample is used in each iteration and the set of input-output pairs $(\boldsymbol{x}, \boldsymbol{y})$ in an iteration is called a batch. We use a set $\boldsymbol{\mathcal{B}_j}$ to denote a batch and j is the iteration index. For each input-output pair $(\boldsymbol{x}, \boldsymbol{y}) \in \boldsymbol{\mathcal{B}}_j$, the input sample $\boldsymbol{x}$ is propagated through the ANN and a prediction $\boldsymbol{y} = f(\boldsymbol{x}, \boldsymbol{\Phi})$ is computed at the output. We also have a desired output $\hat{\boldsymbol{y}}$ which is known as a label, and the difference between the prediction $\boldsymbol{y}$ and the label $\hat{\boldsymbol{y}}$ can be quantified using a loss function $\ell(\boldsymbol{y}, \hat{\boldsymbol{y}})$. Since the batch is introduced, the batch-based loss function can be written as

$$\mathcal{L}(\boldsymbol{\mathcal{B}}_j; \boldsymbol{\Phi}) = \frac{1}{b} \sum_{(\boldsymbol{x}, \boldsymbol{y}) \in \boldsymbol{\mathcal{B}}_j} \ell(f(\boldsymbol{x}, \boldsymbol{\Phi}), \hat{\boldsymbol{y}}) \tag{2.10}$$

where the b is the batch size in number of samples. The loss function is also depicted in Fig. 2.5.

The target of training is to decrease the loss as much as possible. The gradients of the loss with respect to all the tunable parameters, $\nabla_{\boldsymbol{\Phi}} \mathcal{L}(\boldsymbol{\mathcal{B}}_j; \boldsymbol{\Phi})$, can be used to optimize the loss function. In other words, the gradients are used to update all the parameters at each iteration. There are a lot of methods for optimizing the parameters using the gradients, and one typical approach is called stochastic gradient descent (SGD) [35] which is shown as below:

$$\boldsymbol{\Phi_{j+1}} = \boldsymbol{\Phi_j} - \xi \nabla_{\boldsymbol{\Phi_j}} \mathcal{L}(\boldsymbol{\mathcal{B}}_j; \boldsymbol{\Phi_j}) \tag{2.11}$$

where ξ is a scalar known as learning rate. In SGD, all the tunable parameters are updated simply by subtracting the corresponding scaled gradients at each iteration.

The gradients can be calculated numerically by applying the chain rule to the local derivatives of each node. In such process, the data flow is reversed compared to the FP, and the outputs of nodes in each layer now become the local derivatives. The calculation of the local derivatives need the derivatives from the previous layer as inputs, and such process of propagating the derivatives backward to calculate the gradients is called backward propagation (BP).

2.4.2 Model-Based Neural Network

Instead of using the feedforward ANN with fully-connected layers introduced in Section 2.4.1.1, a type of NN called model-based neural network is adopted to construct the equalizer in this thesis. As its name indicates, the model-based neural network is based on the simulation model of the PMD-Kerr effect. In such NN, a component called MIMO-FIR filter is adopted as the entire linear part in each layer and the Kerr model introduced in Section 2.3.2 is used as the activation function in each layer. The RRC filter shown in Section 2.2.2 is used as an additional linear layer in this model-based NN.

Before introducing the MIMO-FIR filter, we need to first introduce the Lagrange fractional delay filter and the MIMO system as the basis for the MIMO-FIR filter.

2.4.2.1 Lagrange Fractional Delay Filter

As introduced in Eq. (2.5), the DGD can be modeled in the frequency domain. However, this frequency domain model is not suited to be directly used in a model-based NN equalizer. This is because the digital signals received in the DSP part in the receiver are defined in the discrete-time domain and the conversion between the frequency domain and the time domain is thus needed. Usually, such conversion is realized by fast Fourier transform (FFT) and inverse fast Fourier transform (IFFT) in digital ICs. The problem is that the digital implementation of FFT/IFFT has a significant overhead in terms of resource utilization. Therefore, the Lagrange fractional delay filter is introduced since it can provide a way to directly realize the DGD in the time domain. Then, instead of using the frequency domain model directly, we can use the Lagrange fractional delay filter in our model-based NN equalizer.

The Lagrange fractional delay filter is based on the Lagrange interpolation and can delay a signal by a fraction of signal period [36]. It is an FIR filter working in the time domain to approximate the fractional delay, which is also know as maximally-flat FIR approximation. Since the filter is completely time-domain based and only requires addition and multiplication operations, it is widely used in digital IC-based DSPs. The FIR coefficients of the Lagrange fractional delay filter can be obtained from Eq. (2.12), and D is the total delay that this filter realizes [36]. For a Lagrange fractional delay filter with an odd number of taps, we have $D = (N-1)/2 + d$ in which N is the number of FIR taps and d is the fractional delay.

$$h(n) = \prod_{k=0,\, k\neq n}^{N} \frac{D-k}{n-k}, \quad n = 0, 1, 2, ..., N \tag{2.12}$$

When the Lagrange fractional delay filter is adopted to realize the DGD, the filter applied to the x-polarization and that on the y-polarization should have the same N, and thus we have $\tau = ((N-1)/2 + d_x) - (N-1)/2 + d_y) = d_x - d_y$ in which d_x and d_y are the fractional delays of the x-polarization and y-polarization respectively.

2.4.2.2 MIMO System

Like the DGD model, the PSP rotation model in Eq. (2.7) is also adopted in the model-based NN equalizer and such model can be implemented as a MIMO system in digital ICs. The MIMO system is a commonly used technique in digital IC-based DSPs and the structure of a complex 2×2 MIMO is shown in Fig. 2.6 as an example, where the H parameters and the x_{in} and y_{in} are all complex numbers. It can be seen that the PSP rotation model in Eq. (2.4) is actually a complex 2×2 MIMO system.

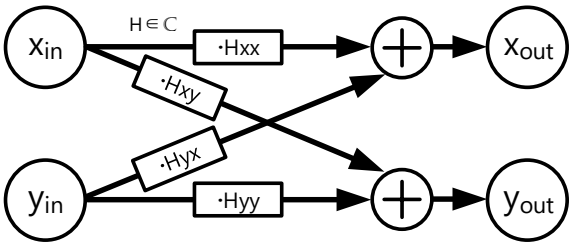


Figure 2.6: The structure of a complex 2×2 MIMO system as an example.

The simplified PSP rotation in Eq. (2.7) can be realized by two real 2×2 MIMO systems, as shown in Fig. 2.7 where all the H parameters are real numbers. The two MIMO systems are identical and in parallel.

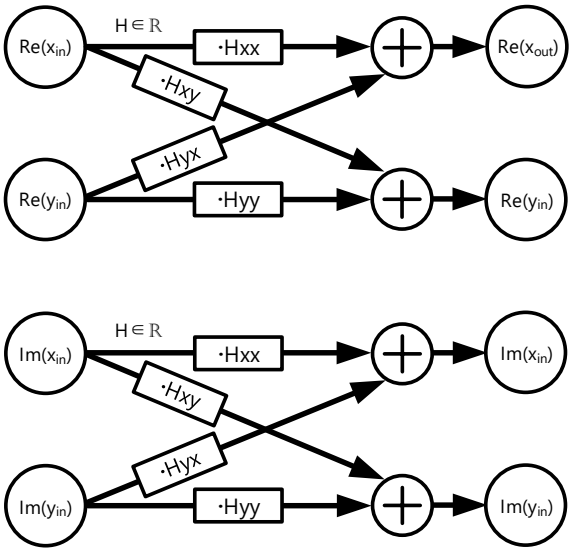


Figure 2.7: The structure of the simplified PSP rotation model implemented by two real 2×2 MIMO systems.

2.4.2.3 MIMO-FIR Filter

It is straightforward to notice that the two real MIMO systems shown in Fig. 2.7 and the Lagrange fraction delay filter can be combined into one component which can be used to model both the DGD and the PSP rotation at the same time. In other words, the PMD effect can be modeled by this combined component which is known

as the MIMO-FIR filter. Furthermore, the inverse of Eq. (2.7) can also be modeled by the two real MIMO systems shown in Fig. 2.7 and the inverse of Eq. (2.5) can be realized using the Lagrange fractional delay filter by simply substituting $-\tau$ for τ. Therefore, the inverse of the PMD effect can also be realized by the MIMO-FIR filter. The linear part in each layer in our model-based NN equalizer is thus based on this MIMO-FIR filter to compensate for the PMD effect. The equation and structure of the MIMO-FIR filter will be shown in Section 4.2 and Section 5.2.1 respectively.

2.4.2.4 Kerr Model Activation Function

The activation function in each layer in our NN equalizer is also model-based and the model shown in Eq. (2.6) is adopted. By simply substituting $-\bar{\gamma}$ for $\bar{\gamma}$, the inverse of the Kerr effect is realized in the NN equalizer. Therefore, the model-based NN equalizer is able to compensate for the optical Kerr effect. Moreover, this Kerr model is essentially a nonlinear function and thus can be used as the activation function in an NN. The equation and structure of the Kerr model activation function will be shown in Section 4.2 and Section 5.2.2 respectively.

2.4.2.5 Matched Filter Linear Layer

The MF of a pulse-shaping filter can restore the shaped pulses and the MF of the RRC filter is itself, as described in Section 2.2.2. Therefore, our model-based NN equalizer includes a linear layer which is based on the RRC filter model. The equation and structure of this MF linear layer will be shown in Section 4.2 and Section 5.2.3 respectively.

2.5 Supporting Works

This section introduces two supporting works of this thesis, and some components in the final implemented equalizer are modified from the components in these works.

2.5.1 FPGA-Based PMD Effect Emulator

An FPGA-based PMD effect emulator is proposed in [37] in which a PMD model is realized on an FPGA. The DGD model and the PSP rotation model are separately implemented in [37], and the DGD is realized by the Lagrange fractional delay filter. Thus, this thesis adopts the Lagrange fractional filter from [37] and modifies it by adding two MIMO systems. Then the MIMO-FIR filters used in the equalizer in our thesis project are constructed. Moreover, a transmitter is also implemented on an FPGA in [37]. This thesis adopts the RRC filter from the transmitter and modifies it to be used as the MF linear layer in our proposed equalizer.

2.5.2 FPGA-Based Optical Kerr Effect Emulator

An FPGA-based optical Kerr effect emulator is proposed in [38] using the Kerr effect model described in Eq. (2.7), and the model is implemented on an FPGA.

The exact same Kerr effect emulator is directly adopted as the activation function in the model-based NN equalizer implementation in this thesis project.

3

Methods

The development flow of this thesis project can be divided into five steps: equalizer design and Python (Tensorflow) simulation, Matlab simulation, model modification and its Matlab re-simulation, VHDL-Matlab co-simulation, and FPGA implementation. Fig. 3.1 shows the overall work flow of this thesis project.

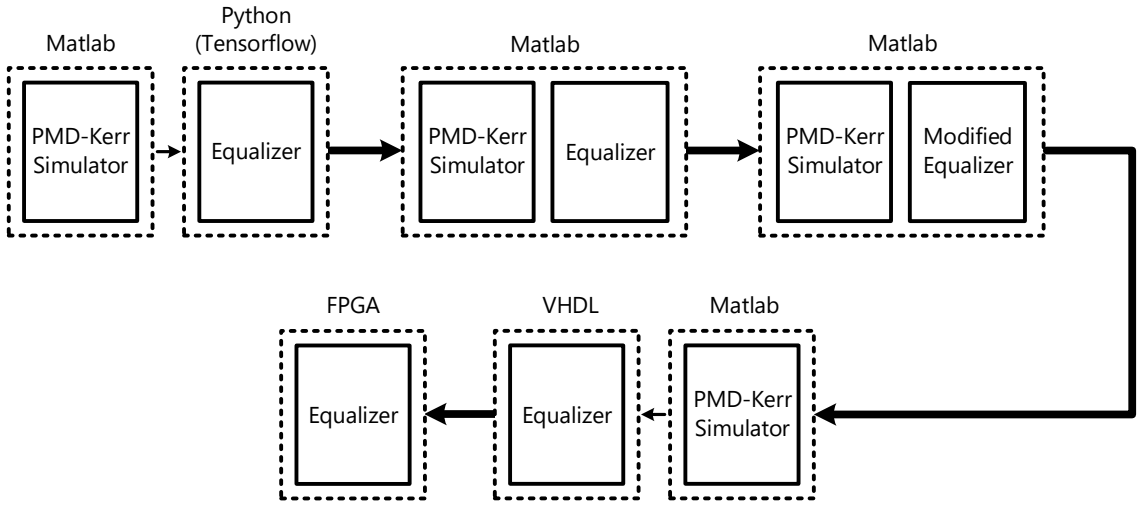


Figure 3.1: The overall development flow of the thesis project.

3.1 Equalizer Design and Python Simulation

In the first step, an equalizer model is designed and it is implemented as software simulation based on Python. More specifically, a widely-used Python-based ML framework, Tensorflow [39], is used to construct the equalizer to provide a reference for the next steps. Only the FP of the equalizer needs to be coded in Tensorflow, while the BP can be generated automatically by the built-in functions without specifying any code about the model and computation of the BP. Besides the equalizer, a PMD-Kerr simulator including a transmitter and a channel is also implemented in Matlab. The simulation of PMD-Kerr effect is involved in the channel simulation. The PMD-Kerr simulator is used to provide test data for the design under test (DUT), i.e., the equalizer. The Matlab-based PMD-Kerr simulator is used for providing test data for every step in this work flow.

3.2 Matlab Simulation

In the previous Tensorflow simulation, the BP is realized automatically, but the full BP model still needs to be mathematically derived for being implemented on an FPGA. In this step, each layer in the BP is explicitly and manually calculated, and Matlab is still used to verify the correctness of the BP calculation. The same equalizer as in the Tensorflow simulation is implemented in Matlab, except that the BP is replaced with the manually derived one. The Matlab simulation results should be compared against the Tensorflow results to verify the correctness of the BP model calculation. If the same input data and equalizer hyperparameters are provided, the results of the Tensorflow simulation should be identical to those of the Matlab simulation. The same PMD-Kerr simulator as in the previous step is implemented in Matlab to provide the same test data.

3.3 Model Modification and Matlab Simulation

Before implementing the equalizer on an FPGA, the original model of the equalizer should be modified to make the FPGA implementation feasible. The modifications are added to the Matlab implementation in the second step while all other components are kept unchanged. Both the FP and BP are calculated and explicitly implemented in the second step, which makes the modifications feasible. In order to verify the impact of the modifications, the new simulation results are compared against the results of the second-step Matlab results to quantify the resulting decreases in performance and convergence speed. Note that the same input data are provided to the modified equalizer as in the previous two steps.

3.4 VHDL-Matlab Co-Simulation

The modified Matlab model is then implemented using very high speed integrated circuit hardware description language (VHDL), and the new VHDL-based equalizer should be verified and tested using VHDL logic simulations. We use ModelSim 10.7 to carry out the logic simulations. The VHDL-Matlab co-simulation means that the Matlab simulation is for generating the test data and the VHDL simulation is for testing the equalizer using the generated data from Matlab. The input data to the VHDL-based equalizer are still the same as in steps and are stored as data files. The data files are then read into Modelsim and thus the VHDL-Matlab co-simulation is realized. Since the VHDL implementation further adds degradation in performance and the convergence speed is also different, the test results from the VHDL logic simulation are compared with the original model and the modified model in Matlab, through which we can get values on the differences. Moreover, a new data representation method and shorter wordlengths are used in VHDL simulations, and thus tests should also be done about them.

3.5 FPGA Implementation

The last step in this thesis project is the implementation of our proposed equalizer on an FPGA, and the synthesis tool is Vivado Design Suite 2020.2. In this step, the register-transfer level (RTL) VHDL design goes through the synthesis and implementation steps in Vivado. A bitstream file is finally generated which can be downloaded to the target FPGA. We mainly focus on the timing and resource utilization information obtained after the implementation step in Vivado in this step.

4

Equalizer Model and Software Simulation

This chapter focuses on the model of the equalizer implemented in this thesis and its software simulation in Tensorflow. This chapter draws the theoretical basis for the hardware implementation which will be thoroughly discussed in Chapter 5. It chapter starts with the channel setup and simulation in Matlab, as the equalizer is based on the PMD-Kerr model. Then the overall structure of the original equalizer model simulated in Tensorflow is proposed. After that, the equalizer model with the calculation of every layer simulated in Matlab is introduced. Finally, a modified model simulated in Matlab with Taylor expansion in the BP is proposed.

4.1 Transmitter-Channel Setup and Simulation

A channel and transmitter simulation should be implemented in Matlab to provide test data for the equalizer. Table 4.1 shows the transmitter and channel parameters, and such setup will be used throughout this thesis. In this setup, the symbol rate, section length, PMD parameter and Kerr parameter are adopted from [6]. The 32 Gbaud symbol rate is up-sampled to 64 Gbaud before it goes through an RRC pulse-shaping filter with a roll-off factor of 0.1. The channel contains 3 fiber sections and each section has a length of 100 km, i.e., L_c=100 km.

Usually, an effective fiber simulation requires that the section be short enough to assume that the PMD effect keeps constant over the short distance [40]. In this thesis, the 100 km section is much longer than a typical value such as 0.1 km and the 3 sections are also much fewer than a typical value of 1000 sections [41]. However, the number of layers in the model-based equalizer should be the same as the number of sections in the channel model, and an FPGA usually cannot host too many layers. Using very few sections with a short length also does not make sense since the PMD-Kerr effect would be very weak and equalization is not even needed under such a scenario. Thus, this thesis adopts a small number of sections with a long length in the channel simulation, which results in the same number of layers in the equalizer. Although such a setup does not necessarily simulate a realistic transmission scenario, this paper aims to verify the feasibility of implementing a model-based NN equalizer on an FPGA and investigate the design trade-offs throughout this process. Moreover, this thesis aims at providing a hardware architecture of such an equalizer and its corresponding digital circuit design, based on which future work can imple-

ment the equalizer on more powerful platforms possible to host more layers, such as ASICs.

In the channel simulation, the DGD is set to be the same for each section. The mean DGD $\bar{\tau}$ is used as the DGD of each section, and we have $\bar{\tau} = \tau\sqrt{3\pi L_c/8}$ where τ is the PMD parameter [41]. Each rotation angle $\theta^{(k)}$ $(k = 0, 1, 2, 3)$ is generated randomly and uniformly on $[-\pi, \pi]$.

Table 4.1: The channel and transmitter parameters.

Parameter	Value
modulation	Gray-coded QPSK
symbol rate	32 Gbaud
sample rate	64 Gbaud (2 samples/symbol)
pulse shaping	0.1 root-raised cosine (51 taps)
number of fiber sections (K)	3
section length (L_c)	100 km
PMD parameter (τ)	$0.2\,\mathrm{ps}/\sqrt{\mathrm{km}}$
mean DGD ($\bar{\tau}$)	2.1708 ps
Kerr parameter (γ)	1.2 rad/W/km
$\bar{\gamma}$	106.6667 rad/W
rotation angle k ($\theta^{(k)}$)	uniform distribution on $[-\pi, \pi]$
AWGN noise power (P_n)	-14 dBm

Fig. 4.1 shows the structure of the transmitter-channel simulation in Matlab, which is divided into two parts: the transmitter part and the channel part. In the transmitter part, a random number generator (RNG) generates a sequence of data symbols over two polarizations, denoted by thin arrows. The QPSK modulation is then applied to the two polarizations. The QPSK modulation is only chosen for testing and the modulation format can be easily changed to others, such as 16QAM. The two polarizations are then up-sampled to 2 samples per cycle, denoted by thick arrows, and the signals go through an RRC filter for pulse shaping after the up-sampling.

The channel part can be divided into several sections based on our proposed model in Section 2.3.2. In this thesis, only an equalizer with 3 layers is studied and thus the channel should also contain 3 sections. However, the number of sections can be increased by duplicating the same section several times for testing the equalizers with more layers. It can be seen that each section is further divided into three components, namely Rotation, Delay and Kerr. The input parameters to each section are also shown in Fig. 4.1. Note that there is an additional Output Rotation component with the rotation angle $\theta^{(3)}$ after all the 3 sections. After the Output Rotation, there is an additive white Gaussian noise (AWGN) component which adds the AWGN with the power P_n to the x and y polarization signals. The equalizer then receives the final outputs from the channel. The equalizer can be either Tensorflow, Matlab or VHDL based, as introduced in Chapter 3.

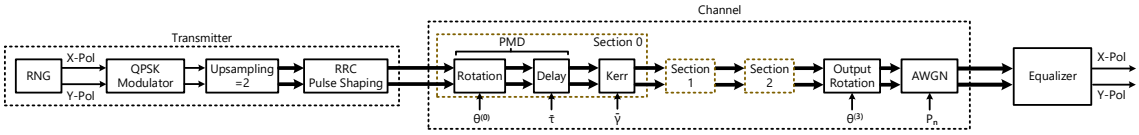


Figure 4.1: The structure of the transmitter and channel simulation in Matlab.

4.2 Forward Propagation Model and Tensorflow Simulation

The original equalizer design in this thesis is based on the model-based nonlinear equalizer proposed in [6]. The equalizer is model-based which means that it has a correspondence with the channel model. The FP of the equalizer alternates linear and nonlinear layers as the linear and nonlinear blocks in the channel model shown in Fig. 4.1. The linear layers are implemented using MIMO-FIR filters which have been briefly introduced in Section 2.4.2.3, and the nonlinear layers are based on the Kerr effect model in Eq. (2.6). The equalizer has essentially the same structure as a neural network, which means it is adaptive to channel variations due to its training.

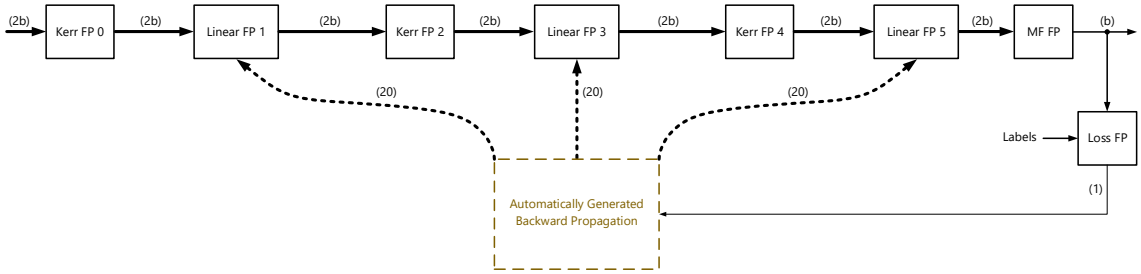


Figure 4.2: The structure of the equalizer model in Tensorflow simulation. The thick arrow represents $2b$ samples, the thin arrow represents b symbols, and the dashed arrow represents 20 gradients.

The equalizer is firstly simulated in Tensorflow, and Fig. 4.2 shows the structure of the equalizer model in Tensorflow simulation. It can be seen that the structure is divided into two parts: the FP and the BP. The number in each block counts the layer regardless of the type of the layer. Generally, a "layer" also refers to the combination of one "Kerr FP" and one "Linear FP", and thus the equalizer contains 3 layers and each layer corresponds to each section in the channel model in Fig. 4.1. The Linear FP layer is a MIMO-FIR filter component, and the Linear FP layers 3 and 5 correspond to the PMD components in sections 1 and 0 in the channel model. The Linear FP layer 1 corresponds to both the PMD component of Section 2 in the channel model and the Output Rotation component in the channel model, and it can compensate for both components simultaneously. The Kerr FP layer is a nonlinear activation layer that compensates for the Kerr effect, and the Kerr FP layers 0, 2 up to 4 correspond to the Kerr components in sections from 2 down to 0 in the channel model. The "MF FP" is short for matched filter forward propagation

which corresponds to the RRC filter in the transmitter in Fig. 4.1. The Loss FP calculates the batch-based loss between the predictions and the labels.

The equalizer is pilot-based, which means that the equalizer is assumed to have information about the labels during training. Moreover, all the Kerr FP layers are assumed to have information about the Kerr parameter $\bar{\gamma}$ in the channel. In Tensorflow, all the BP layers need not be explicitly implemented since they are automatically generated, as shown in Fig. 4.2. The entire flow including the FP and the BP is executed in a loop, and one iteration includes the execution of $2b$ samples, as the batch size is set to b symbols and each symbol is up-sampled to 2 samples per cycle in the transmitter. Therefore, the input and output sizes in all the FP layers except for the MF FP layer are $2b$ samples, denoted by thick arrows. The MF FP layer includes the down-sampling and thus its output size should be b symbols, which is denoted by a medium-sized arrow. The Loss FP layer calculates the differences between the labels and the predictions. Since the loss function is batch-based, the output of the Loss FP layer is a real number which is then forwarded to the BP. A thin arrow is used to represent this real number. The outputs of the BP are no longer data samples but gradients, and each output corresponds to a Linear FP layer in the FP. Since there are 5 taps in each Linear FP layer and each tap contains 4 weights, namely Hxx, Hyx, Hxy and Hyy, there are 20 weights in each layer. Thus, the size of each output of the BP should also be 20. In other words, each dashed arrow from the BP contains 20 gradients. It should also be mentioned that all the weights and gradients are real numbers, and the weights in the Linear FP layers are updated by the corresponding gradients via SGD in each iteration. The sizes of all ports are demonstrated in number of samples (gradients) within the parentheses beside the solid arrows in Fig. 4.2.

Table 4.2 shows all the fixed hyperparameters of the equalizer that will be applied throughout the thesis. Besides the listed hyperparameters, the batch size b and the transmission power P are important and considered variable parameters which will be further studied in Chapter 6. The initialization of the equalizer is shown in Eq. (4.6).

The calculation in each Linear FP layer k in a batch can be written as

$$\begin{cases} xi_j^{(k+1)} = \sum_{d=0}^{4} Hxx_i^{(k,d)} \cdot xi_{j-4+d}^{(k)} + \sum_{d=0}^{4} Hyx_i^{(k,d)} \cdot yi_{j-4+d}^{(k)} \\ xq_j^{(k+1)} = \sum_{d=0}^{4} Hxx_i^{(k,d)} \cdot xq_{j-4+d}^{(k)} + \sum_{d=0}^{4} Hyx_i^{(k,d)} \cdot yq_{j-4+d}^{(k)} \\ yi_j^{(k+1)} = \sum_{d=0}^{4} Hxy_i^{(k,d)} \cdot xi_{j-4+d}^{(k)} + \sum_{d=0}^{4} Hyy_i^{(k,d)} \cdot yi_{j-4+d}^{(k)} \\ yq_j^{(k+1)} = \sum_{d=0}^{4} Hxy_i^{(k,d)} \cdot xq_{j-4+d}^{(k)} + \sum_{d=0}^{4} Hyy_i^{(k,d)} \cdot yq_{j-4+d}^{(k)} \end{cases}, \quad 2ib \leq j < 2(i+1)b \tag{4.1}$$

where the xi, xq, yi and yq denote the I and Q samples in the x and y polarizations. The Hxx, Hyx, Hxy and Hyy are the trainable weights in the Linear FP layers, and each weight contains 5 taps (items). The b is the batch size in number of symbols, and we have 2 samples per symbol. In other words, each batch contains $2b$ samples. The i is the batch index which means that it is the i-th batch. For the

data samples, the superscript and subscript are the layer index and sample index, respectively. E.g., the $xi_j^{(k+1)}$ means that it is the j-th sample and the output from the Linear FP layer k. The $Hxx_i^{(k,d)}$ means that it is the d-th tap in the weight Hxx in Linear FP layer k, and the i is still the batch index.

Table 4.2: The fixed equalizer hyperparameters.

Parameter	Value
number of layers (K)	3
number of MIMO-FIR filter taps (N)	5
matched filter	0.1 RRC with 51 taps
Kerr parameter ($\bar{\gamma}$)	106.6667 rad/W
learning rate (ξ)	32
optimizer	SGD

The calculation in each Kerr FP layer k is the same as the Kerr model in Eq. (2.6) except that the $\bar{\gamma}$ is replaced by $-\bar{\gamma}$. The expanded calculation is shown as below:

$$\begin{cases} \phi_j^{(k)} = \bar{\gamma}((xi_j^{(k)})^2 + (xq_j^{(k)})^2 + (yi_j^{(k)})^2 + (yq_j^{(k)})^2) \\ xi_j^{(k+1)} = xi_j^{(k)} \cos\phi_j^{(k)} - xq_j^{(k)} \sin\phi_j^{(k)} \\ xq_j^{(k+1)} = xi_j^{(k)} \sin\phi_j^{(k)} + xq_j^{(k)} \cos\phi_j^{(k)} \\ yi_j^{(k+1)} = yi_j^{(k)} \cos\phi_j^{(k)} - yq_j^{(k)} \sin\phi_j^{(k)} \\ yq_j^{(k+1)} = yi_j^{(k)} \sin\phi_j^{(k)} + yq_j^{(k)} \cos\phi_j^{(k)} \end{cases} \tag{4.2}$$

where the ϕ is an intermediate value which will also be referred to as the "Kerr angle" in the following. The superscript and subscript represent the sample index and the layer index, independently.

The calculation in the MF FP layer in a batch can be written as

$$u_{2j+1}^{(7)} = \sum_{d=0}^{50} M^{(d)} \cdot u_{2j-49+d}^{(6)}, \quad ib \leq j < (i+1)b \tag{4.3}$$

where the M represents the 51 RRC taps adopted in this thesis. The RRC taps are generated by the FIR approximation derived from Eq. (2.1) since the RRC filter is its own matched filter. The superscript in M denotes the tap index. The u can be substituted by xi, xq, yi or yq. The superscript and subscript in u still denote the the layer index and sample index, respectively. The subscript in u inherently contains the down-sampling of 2. The i is still the batch index and b is still the batch size in number of symbols.

The calculation of the Loss FP layer in the i-th batch is

$$\mathcal{L}_i = \frac{\sum_{d=ib}^{(i+1)b-1} ((\hat{xi}_d - xi_{2d+1}^{(7)})^2 + (\hat{xq}_d - xq_{2d+1}^{(7)})^2 + (\hat{yi}_d - yi_{2d+1}^{(7)})^2 + (\hat{yq}_d - yq_{2d+1}^{(7)})^2)}{2b} \tag{4.4}$$

where the $\hat{\square}$ denotes the desired output symbol, i.e., the label. The subscript in the loss $\mathcal{L}$ is the batch index. The subscript in the labels $\hat{xi}$, $\hat{xq}$, $\hat{yi}$ and $\hat{yq}$ is the symbol index, not the sample index since the data stream has already been down-sampled. The b is still the batch size in number of symbols. It can be seen that the loss is not normalized since the normalization causes additional hardware utilization when porting to an FPGA. The lack of normalization in the loss can always be compensated by using a larger learning rate in the training, as shown in Table 4.2.

Although the BP is automatically generated by Tensorflow, the updates of all the weights in all Linear FP layers still need to be specified. This thesis adopts SGD as the method of optimization, and the updates of weights in Linear FP layer k in the i-th batch can be written as

$$H_{i+1}^{(k,d)} = H_i^{(k,d)} - \xi \nabla_{H_i^{(k,d)}} \mathcal{L}_i \,, \quad 0 \leq d < 5 \tag{4.5}$$

where the ξ is the learning rate, and the $\nabla_{H_i^{(k,d)}} \mathcal{L}_i$ denotes the gradient of $\mathcal{L}_i$ with respect to $H_i^{(k,d)}$. The (k, d) in the superscript represents that it is d-th tap in the weight H in Linear FP layer k. The H can be replaced by Hxx, Hyx, Hxy or Hyy. The gradients are automatically generated by Tensorflow, and the updates of all Linear FP layers happen together in each batch.

The initialization scheme of the weights is

$$\begin{cases} \boldsymbol{Hxx}_0^{(k)} = \boldsymbol{Hyy}_0^{(k)} = [0\ 0\ 0\ 0\ 0], & \forall k \in \{1, 3, 5\} \\ \boldsymbol{Hyx}_0^{(k)} = \boldsymbol{Hxy}_0^{(k)} = [0\ 0\ 1\ 0\ 0], & \forall k \in \{1, 3, 5\} \end{cases} \tag{4.6}$$

where the 0 in the subscript denotes that the weights are in the 0-th batch, and the k in the superscript is still the layer index.

4.3 Backward Propagation Model and Matlab Simulation

Although the BP can be automatically generated in Tensorflow, the calculation in the BP should still be mathematically derived for VHDL implementation. We use Matlab to simulate an identical equalizer model where the BP automatically generated in Tensorflow in the past is manually rebuilt, layer by layer. Therefore, we can use the Matlab equalizer to verify whether our BP model is correct by comparing it against the Tensorflow reference. Moreover, it is hard to alter the automatically generated BP in Tensorflow, while we can easily do that in the rebuilt Matlab equalizer.

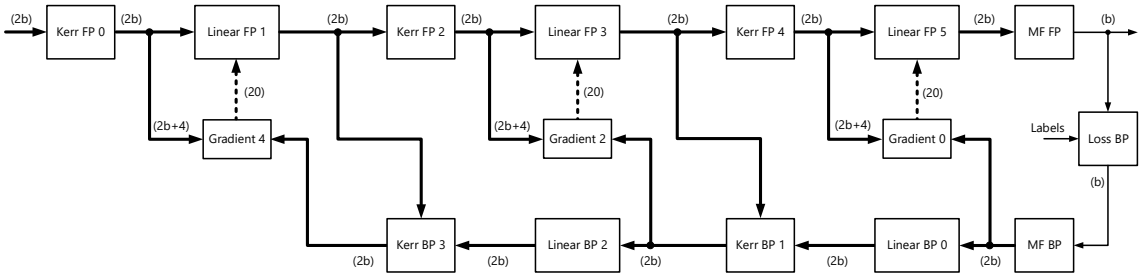


Figure 4.3: The structure of the equalizer model in Matlab simulation. The thick arrow represents $2b$ samples, the thin arrow represents b symbols, and the dashed arrow represents 20 gradients.

Fig. 4.3 shows the structure of the equalizer model in the Matlab simulation. The FP is the same as in Fig. 4.2 except that the Loss FP layer is no longer needed. Both the input and output sizes of the Loss BP layer are b samples, in 1 sample per symbol. The MF BP layer then converts the data stream back to 2 samples per symbol, and the output size becomes $2b$ samples. The input size and the output size are both $2b$ in the Linear BP layers. There are two inputs and both have the size of $2b$ samples in the Kerr BP layers, where one input is from the input of the corresponding Kerr FP layer and the other is from the output of the previous Linear BP layer. In the Gradient layers, the input from the previous Kerr BP layer still has a size of $2b$ samples while the other input from the input of the corresponding Linear FP layer has a size of $2b + 4$. In the $2b + 4$ samples, the $2b$ samples are directly from the input of the Linear FP layer while the 4 more samples are the samples stored in the MIMO-FIR filter in the Linear FP layer. The outputs of the Gradient layers are no longer data samples but gradients, and the output size is 20 gradients in each Gradient layer. In the corresponding Linear FP layer, there are 4 weights Hxx, Hyx, Hxy and Hyy, and each contains 5 taps. Thus, each Gradient layer has an output size of 20 gradients. The port sizes are also noted in the parentheses next to the arrows in Fig. 4.3.

It should be noted that the Loss FP layer is no longer needed in the Matlab equalizer since the loss is not part of the output of the equalizer. The information of the loss function is already involved in the Loss BP layer since the Loss BP layer is essentially the derivative of the loss function. The loss function is needed only in the Tensorflow equalizer since Tensorflow needs the loss function as an input to its built-in functions to generate the BP automatically. It should also be noted that the input and output size are both b symbols in the Loss BP layer, unlike the 1 loss output in the Loss FP layer.

In the Matlab model, the calculation in the Loss BP layer is

$$\bar{\mathcal{L}}_{u,j} = \frac{u^{(7)}_{2j+1} - \hat{u}_j}{b}, \quad ib \le j < (i+1)b \tag{4.7}$$

where the $\bar{\square}$ represents the outputs of the Loss BP layer, MF BP layer, Linear BP layers and Kerr BP layers. The subscript j in $\bar{\mathcal{L}}$ is the symbol index, not the sample index since the data stream has already been down-sampled. The u can be either

xi, xq, yi or yq. The superscript and subscript in u are the layer index and sample (symbol) index, respectively. The i is still the batch index and b is still the batch size in number of symbols.

The calculation in the MF BP layer depends on the value of the batch size b. When the number of matched filter taps is 51 and $b < 26$, the calculation is

$$\begin{cases} \bar{u}_{2j+1}^{(0)} = \sum_{d=26-(i+1)b+j}^{25} M^{(2d)} \cdot \bar{\mathcal{L}}_{u,25-d+j} \\ \bar{u}_{2j}^{(0)} = \sum_{d=26-(i+1)b+j}^{25} M^{(2d-1)} \cdot \bar{\mathcal{L}}_{u,25-d+j} \end{cases}, \quad ib \le j < (i+1)b, \;\; b < 26 \tag{4.8}$$

and when the batch size $b \geq 26$, the calculation becomes

$$\begin{cases} \bar{u}_{2j+1}^{(0)} = \sum_{d=0}^{25} M^{(2d)} \cdot \bar{\mathcal{L}}_{u,25-d+j} \\ \bar{u}_{2j}^{(0)} = \sum_{d=1}^{25} M^{(2d-1)} \cdot \bar{\mathcal{L}}_{u,25-d+j} \end{cases}, \quad ib \le j < (i+1)b - 25, \;\; b \geq 26 \tag{4.9}$$

$$\begin{cases} \bar{u}_{2j+1}^{(0)} = \sum_{d=26-(i+1)b+j}^{25} M^{(2d)} \cdot \bar{\mathcal{L}}_{u,25-d+j} \\ \bar{u}_{2j}^{(0)} = \sum_{d=26-(i+1)b+j}^{25} M^{(2d-1)} \cdot \bar{\mathcal{L}}_{u,25-d+j} \end{cases}, \quad (i+1)b-25 \le j < (i+1)b, \;\; b \geq 26 \tag{4.10}$$

where the j is the symbol index and i is the batch index. The superscript in the output u is reset to 0, which denotes the layer index in the BP. The subscript in u is the sample index. The u can be either xi, xq, yi or yq. The subscript in $\bar{\mathcal{L}}$ is the sample index in the output of the previos Loss BP layer. It can be seen that the data stream is restored to two samples per symbol and the output size is $2b$ samples.

The calculation in the Linear BP layer k can be divided into two parts in a batch:

$$\begin{cases} \bar{xi}_j^{(k+1)} = \sum_{d=0}^{4} Hxx_i^{(5-k,4-d)} \cdot \bar{xi}_{j+d}^{(k)} + \sum_{d=0}^{4} Hxy_i^{(5-k,4-d)} \cdot \bar{yi}_{j+d}^{(k)} \\ \bar{xq}_j^{(k+1)} = \sum_{d=0}^{4} Hxx_i^{(5-k,4-d)} \cdot \bar{xq}_{j+d}^{(k)} + \sum_{d=0}^{4} Hxy_i^{(5-k,4-d)} \cdot \bar{yq}_{j+d}^{(k)} \\ \bar{yi}_j^{(k+1)} = \sum_{d=0}^{4} Hyx_i^{(5-k,4-d)} \cdot \bar{xi}_{j+d}^{(k)} + \sum_{d=0}^{4} Hyy_i^{(5-k,4-d)} \cdot \bar{yi}_{j+d}^{(k)} \\ \bar{yq}_j^{(k+1)} = \sum_{d=0}^{4} Hyx_i^{(5-k,4-d)} \cdot \bar{xq}_{j+d}^{(k)} + \sum_{d=0}^{4} Hyy_i^{(5-k,4-d)} \cdot \bar{yq}_{j+d}^{(k)} \end{cases}, \quad 2ib \le j < 2(i+1)b - 4 \tag{4.11}$$

$$\begin{cases} \bar{xi}_j^{(k+1)} = \sum_{d=0}^{2(i+1)b-j-1} Hxx_i^{(5-k,4-d)} \cdot \bar{xi}_{j+d}^{(k)} + \sum_{d=0}^{2(i+1)b-j-1} Hxy_i^{(5-k,4-d)} \cdot \bar{yi}_{j+d}^{(k)} \\ \bar{xq}_j^{(k+1)} = \sum_{d=0}^{2(i+1)b-j-1} Hxx_i^{(5-k,4-d)} \cdot \bar{xq}_{j+d}^{(k)} + \sum_{d=0}^{2(i+1)b-j-1} Hxy_i^{(5-k,4-d)} \cdot \bar{yq}_{j+d}^{(k)} \\ \bar{yi}_j^{(k+1)} = \sum_{d=0}^{2(i+1)b-j-1} Hyx_i^{(5-k,4-d)} \cdot \bar{xi}_{j+d}^{(k)} + \sum_{d=0}^{2(i+1)b-j-1} Hyy_i^{(5-k,4-d)} \cdot \bar{yi}_{j+d}^{(k)} \\ \bar{yq}_j^{(k+1)} = \sum_{d=0}^{2(i+1)b-j-1} Hyx_i^{(5-k,4-d)} \cdot \bar{xq}_{j+d}^{(k)} + \sum_{d=0}^{2(i+1)b-j-1} Hyy_i^{(5-k,4-d)} \cdot \bar{yq}_{j+d}^{(k)} \end{cases}, \quad 2(i+1)b - 4 \le j < 2(i+1)b \tag{4.12}$$

The subscript in $\bar{xi}$, $\bar{xq}$, $\bar{yi}$ and $\bar{yq}$ now becomes the sample index. The superscript in $\bar{xi}$, $\bar{xq}$, $\bar{yi}$ and $\bar{yq}$ is the layer index in the BP, while the superscripts in Hxx, Hyx, Hxy and Hyy are the layer index in the FP and the tap index. The subscript in Hxx, Hyx, Hxy and Hyy is the batch index.

The calculation in the Kerr BP layer k is complicated and can be written in two steps. The first step contains Eq. (4.13) to Eq. (4.17):

$$\begin{cases} R_j^{(5-k)} = \bar{\gamma} \cdot \cos(\bar{\gamma}((xi_j^{(5-k)})^2 + (xq_j^{(5-k)})^2 + (yi_j^{(5-k)})^2 + (yq_j^{(5-k)})^2)) \\ S_j^{(5-k)} = \bar{\gamma} \cdot \sin(\bar{\gamma}((xi_j^{(5-k)})^2 + (xq_j^{(5-k)})^2 + (yi_j^{(5-k)})^2 + (yq_j^{(5-k)})^2)) \end{cases} \tag{4.13}$$

$$\begin{cases} \dfrac{\partial xi_j^{(6-k)}}{\partial xi_j^{(5-k)}} = -2S_j^{(5-k)} \cdot xi_j^{(5-k)} \cdot xi_j^{(5-k)} + R_j^{(5-k)} + 2R_j^{(5-k)} \cdot xi_j^{(5-k)} \cdot xq_j^{(5-k)} \\ \dfrac{\partial xq_j^{(6-k)}}{\partial xi_j^{(5-k)}} = -2S_j^{(5-k)} \cdot xi_j^{(5-k)} \cdot xq_j^{(5-k)} - S_j^{(5-k)} - 2R_j^{(5-k)} \cdot xi_j^{(5-k)} \cdot xi_j^{(5-k)} \\ \dfrac{\partial yi_j^{(6-k)}}{\partial xi_j^{(5-k)}} = -2S_j^{(5-k)} \cdot xi_j^{(5-k)} \cdot yi_j^{(5-k)} + 2R_j^{(5-k)} \cdot xi_j^{(5-k)} \cdot yq_j^{(5-k)} \\ \dfrac{\partial yq_j^{(6-k)}}{\partial xi_j^{(5-k)}} = -2S_j^{(5-k)} \cdot xi_j^{(5-k)} \cdot yq_j^{(5-k)} - 2R_j^{(5-k)} \cdot xi_j^{(5-k)} \cdot yi_j^{(5-k)} \end{cases} \tag{4.14}$$

$$\begin{cases} \dfrac{\partial xi_j^{(6-k)}}{\partial xq_j^{(5-k)}} = -2S_j^{(5-k)} \cdot xq_j^{(5-k)} \cdot xi_j^{(5-k)} + S_j^{(5-k)} + 2R_j^{(5-k)} \cdot xq_j^{(5-k)} \cdot xq_j^{(5-k)} \\ \dfrac{\partial xq_j^{(6-k)}}{\partial xq_j^{(5-k)}} = -2S_j^{(5-k)} \cdot xq_j^{(5-k)} \cdot xq_j^{(5-k)} + R_j^{(5-k)} - 2R_j^{(5-k)} \cdot xq_j^{(5-k)} \cdot xi_j^{(5-k)} \\ \dfrac{\partial yi_j^{(6-k)}}{\partial xq_j^{(5-k)}} = -2S_j^{(5-k)} \cdot xq_j^{(5-k)} \cdot yi_j^{(5-k)} + 2R_j^{(5-k)} \cdot xq_j^{(5-k)} \cdot yq_j^{(5-k)} \\ \dfrac{\partial yq_j^{(6-k)}}{\partial xq_j^{(5-k)}} = -2S_j^{(5-k)} \cdot xq_j^{(5-k)} \cdot yq_j^{(5-k)} - 2R_j^{(5-k)} \cdot xq_j^{(5-k)} \cdot yi_j^{(5-k)} \end{cases} \tag{4.15}$$

$$\begin{cases} \dfrac{\partial xi_j^{(6-k)}}{\partial yi_j^{(5-k)}} = -2S_j^{(5-k)} \cdot yi_j^{(5-k)} \cdot xi_j^{(5-k)} + 2R_j^{(5-k)} \cdot yi_j^{(5-k)} \cdot xq_j^{(5-k)} \\ \dfrac{\partial xq_j^{(6-k)}}{\partial yi_j^{(5-k)}} = -2S_j^{(5-k)} \cdot yi_j^{(5-k)} \cdot xq_j^{(5-k)} - 2R_j^{(5-k)} \cdot yi_j^{(5-k)} \cdot xi_j^{(5-k)} \\ \dfrac{\partial yi_j^{(6-k)}}{\partial yi_j^{(5-k)}} = -2S_j^{(5-k)} \cdot yi_j^{(5-k)} \cdot yi_j^{(5-k)} + R_j^{(5-k)} + 2R_j^{(5-k)} \cdot yi_j^{(5-k)} \cdot yq_j^{(5-k)} \\ \dfrac{\partial yq_j^{(6-k)}}{\partial yi_j^{(5-k)}} = -2S_j^{(5-k)} \cdot yi_j^{(5-k)} \cdot yq_j^{(5-k)} - S_j^{(5-k)} - 2R_j^{(5-k)} \cdot yi_j^{(5-k)} \cdot yi_j^{(5-k)} \end{cases} \tag{4.16}$$

$$\begin{cases} \dfrac{\partial xi_j^{(6-k)}}{\partial yq_j^{(5-k)}} = -2S_j^{(5-k)} \cdot yq_j^{(5-k)} \cdot xi_j^{(5-k)} + 2R_j^{(5-k)} \cdot yq_j^{(5-k)} \cdot xq_j^{(5-k)} \\ \dfrac{\partial xq_j^{(6-k)}}{\partial yq_j^{(5-k)}} = -2S_j^{(5-k)} \cdot yq_j^{(5-k)} \cdot xq_j^{(5-k)} - 2R_j^{(5-k)} \cdot yq_j^{(5-k)} \cdot xi_j^{(5-k)} \\ \dfrac{\partial yi_j^{(6-k)}}{\partial yq_j^{(5-k)}} = -2S_j^{(5-k)} \cdot yq_j^{(5-k)} \cdot yi_j^{(5-k)} + S_j^{(5-k)} + 2R_j^{(5-k)} \cdot yq_j^{(5-k)} \cdot yq_j^{(5-k)} \\ \dfrac{\partial yq_j^{(6-k)}}{\partial yq_j^{(5-k)}} = -2S_j^{(5-k)} \cdot yq_j^{(5-k)} \cdot yq_j^{(5-k)} + R_j^{(5-k)} - 2R_j^{(5-k)} \cdot yq_j^{(5-k)} \cdot yi_j^{(5-k)} \end{cases} \tag{4.17}$$

where the R, the S and all the derivatives are introduced as intermediate values. The second step is

$$\begin{cases} \bar{xi}_j^{(k+1)} = \bar{xi}_j^{(k)} \cdot \dfrac{\partial xi_j^{(6-k)}}{\partial xi_j^{(5-k)}} + \bar{xq}_j^{(k)} \cdot \dfrac{\partial xq_j^{(6-k)}}{\partial xi_j^{(5-k)}} + \bar{yi}_j^{(k)} \cdot \dfrac{\partial yi_j^{(6-k)}}{\partial xi_j^{(5-k)}} + \bar{yq}_j^{(k)} \cdot \dfrac{\partial yq_j^{(6-k)}}{\partial xi_j^{(5-k)}} \\ \bar{xq}_j^{(k+1)} = \bar{xi}_j^{(k)} \cdot \dfrac{\partial xi_j^{(6-k)}}{\partial xq_j^{(5-k)}} + \bar{xq}_j^{(k)} \cdot \dfrac{\partial xq_j^{(6-k)}}{\partial xq_j^{(5-k)}} + \bar{yi}_j^{(k)} \cdot \dfrac{\partial yi_j^{(6-k)}}{\partial xq_j^{(5-k)}} + \bar{yq}_j^{(k)} \cdot \dfrac{\partial yq_j^{(6-k)}}{\partial xq_j^{(5-k)}} \\ \bar{yi}_j^{(k+1)} = \bar{xi}_j^{(k)} \cdot \dfrac{\partial xi_j^{(6-k)}}{\partial yi_j^{(5-k)}} + \bar{xq}_j^{(k)} \cdot \dfrac{\partial xq_j^{(6-k)}}{\partial yi_j^{(5-k)}} + \bar{yi}_j^{(k)} \cdot \dfrac{\partial yi_j^{(6-k)}}{\partial yi_j^{(5-k)}} + \bar{yq}_j^{(k)} \cdot \dfrac{\partial yq_j^{(6-k)}}{\partial yi_j^{(5-k)}} \\ \bar{yq}_j^{(k+1)} = \bar{xi}_j^{(k)} \cdot \dfrac{\partial xi_j^{(6-k)}}{\partial yq_j^{(5-k)}} + \bar{xq}_j^{(k)} \cdot \dfrac{\partial xq_j^{(6-k)}}{\partial yq_j^{(5-k)}} + \bar{yi}_j^{(k)} \cdot \dfrac{\partial yi_j^{(6-k)}}{\partial yq_j^{(5-k)}} + \bar{yq}_j^{(k)} \cdot \dfrac{\partial yq_j^{(6-k)}}{\partial yq_j^{(5-k)}} \end{cases} \tag{4.18}$$

In all the Kerr BP calculations from Eq. (4.13) to Eq. (4.18), the xi, xq, yi and yq are outputs from the FP and the superscript in them represents the layer index in the FP. The $\bar{xi}$, $\bar{xq}$, $\bar{yi}$ and $\bar{yq}$ are outputs from the BP and the superscript in them represents the layer index in the BP. As shown in Fig. 4.1, the Kerr BP layer has an input from the corresponding Kerr FP layer and another input directly from the previous Linear BP layer, which corresponds to the calculations shown here. Moreover, all the subscripts represent the sample index.

The calculation of the Gradient layer k in a batch can be summarized as

$$\begin{cases} \nabla_{Hxx_i^{(5-k,d)}} \mathcal{L}_i = \sum_{j=2ib}^{2(i+1)b-1} \bar{xi}_j^{(k)} \cdot xi_{j-4+d}^{(5-k)} + \sum_{j=2ib}^{2(i+1)b-1} \bar{xq}_j^{(k)} \cdot xq_{j-4+d}^{(5-k)} \\ \nabla_{Hyx_i^{(5-k,d)}} \mathcal{L}_i = \sum_{j=2ib}^{2(i+1)b-1} \bar{xi}_j^{(k)} \cdot yi_{j-4+d}^{(5-k)} + \sum_{j=2ib}^{2(i+1)b-1} \bar{xq}_j^{(k)} \cdot yq_{j-4+d}^{(5-k)} \\ \nabla_{Hxy_i^{(5-k,d)}} \mathcal{L}_i = \sum_{j=2ib}^{2(i+1)b-1} \bar{yi}_j^{(k)} \cdot xi_{j-4+d}^{(5-k)} + \sum_{j=2ib}^{2(i+1)b-1} \bar{yq}_j^{(k)} \cdot xq_{j-4+d}^{(5-k)} \\ \nabla_{Hyy_i^{(5-k,d)}} \mathcal{L}_i = \sum_{j=2ib}^{2(i+1)b-1} \bar{yi}_j^{(k)} \cdot yi_{j-4+d}^{(5-k)} + \sum_{j=2ib}^{2(i+1)b-1} \bar{yq}_j^{(k)} \cdot yq_{j-4+d}^{(5-k)} \end{cases}, \quad 0 \le d < 5 \tag{4.19}$$

where the output of Gradient layer k is the gradient of $\mathcal{L}_i$ with respect to $H_i^{(5-k)}$, i.e., the weights in Linear FP layer $5-k$. This agrees with the structure shown in

Fig. 4.1. The superscript $5-k$ and d in Hxx, Hyx, Hxy and Hyy are the FP layer index and tap index, respectively. The subscript in Hxx, Hyx, Hxy. The i still denotes the batch index and b still denotes the batch size in number of symbols. The superscript in xi, xq, yi or yq is the layer index in FP, while the superscript in $\bar{xi}$, $\bar{xq}$, $\bar{yi}$ and $\bar{yq}$ is the layer index in BP. The subscripts j and $j-4+d$ are both the sample indices.

The updates of all weights in Matlab simulation are the same as in the Tensorflow model shown in Eq. (4.5).

4.4 Modified Equalizer Model and Matlab Simulation

After the equalizer model is rebuilt in Matlab, modifications can be done on the Matlab equalizer since the BP is explicitly implemented from scratch. In this thesis, the only modification we made is on the Kerr BP layers, while all the other FP and BP layers as well as the entire structure are the same as in Fig. 4.3. The hardware friendly first-order Taylor expansion [42, 43] is applied to the Kerr FP model, and the new Kerr BP layer is calculated from the Kerr FP model with Taylor expansion. However, the Kerr FP layer itself is not changed and only the Kerr BP layer is modified.

The original Kerr FP model can be written as

$$\varepsilon(\boldsymbol{u}) = \boldsymbol{u}\exp(-j\bar{\gamma}\left\|\boldsymbol{u}\right\|^2) \tag{4.20}$$

and after applying the first-order Taylor expansion, the Kerr FP becomes

$$\varepsilon(\boldsymbol{u}) = \boldsymbol{u}(\,1-j\bar{\gamma}\left\|\boldsymbol{u}\right\|^2) \tag{4.21}$$

and the model can be further written into

$$\begin{cases} \phi_j^{(5-k)} = -\bar{\gamma}((xi_j^{(5-k)})^2 + (xq_j^{(5-k)})^2 + (yi_j^{(5-k)})^2 + (yq_j^{(5-k)})^2) \\ xi_j^{(6-k)} = xi_j^{(5-k)} - xq_j^{(5-k)} \cdot \phi_j^{(5-k)} \\ xq_j^{(6-k)} = xq_j^{(5-k)} + xi_j^{(5-k)} \cdot \phi_j^{(5-k)} \\ yi_j^{(6-k)} = yi_j^{(5-k)} - yq_j^{(5-k)} \cdot \phi_j^{(5-k)} \\ yq_j^{(6-k)} = yq_j^{(5-k)} + yi_j^{(5-k)} \cdot \phi_j^{(5-k)} \end{cases} \tag{4.22}$$

where the superscript is the layer index in the FP and k is the layer index in the BP. The subscript is the sample index. Thus, the new Kerr BP layer with Taylor expansion can be derived based Eq. (4.22).

The calculation in the modified Kerr BP layer can be divided into two steps. The

first step contains equations from Eq. (4.23) to Eq. (4.26):

$$\begin{cases} \dfrac{\partial xi_j^{(6-k)}}{\partial xi_j^{(5-k)}} = 1 + 2\bar{\gamma} \cdot xi_j^{(5-k)} \cdot xq_j^{(5-k)} \\ \dfrac{\partial xq_j^{(6-k)}}{\partial xi_j^{(5-k)}} = \bar{\gamma}((xi_j^{(5-k)})^2 + (yi_j^{(5-k)})^2 + (yq_j^{(5-k)})^2) + 3\bar{\gamma} \cdot xq_j^{(5-k)} \cdot xq_j^{(5-k)} \\ \dfrac{\partial yi_j^{(6-k)}}{\partial xi_j^{(5-k)}} = 2\bar{\gamma} \cdot xq_j^{(5-k)} \cdot yi_j^{(5-k)} \\ \dfrac{\partial yq_j^{(6-k)}}{\partial xi_j^{(5-k)}} = 2\bar{\gamma} \cdot xq_j^{(5-k)} \cdot yq_j^{(5-k)} \end{cases} \tag{4.23}$$

$$\begin{cases} \dfrac{\partial xi_j^{(6-k)}}{\partial xq_j^{(5-k)}} = -\bar{\gamma}((xq_j^{(5-k)})^2 + (yi_j^{(5-k)})^2 + (yq_j^{(5-k)})^2) - 3\bar{\gamma} \cdot xi_j^{(5-k)} \cdot xi_j^{(5-k)} \\ \dfrac{\partial xq_j^{(6-k)}}{\partial xq_j^{(5-k)}} = 1 - 2\bar{\gamma} \cdot xi_j^{(5-k)} \cdot xq_j^{(5-k)} \\ \dfrac{\partial yi_j^{(6-k)}}{\partial xq_j^{(5-k)}} = -2\bar{\gamma} \cdot xi_j^{(5-k)} \cdot yi_j^{(5-k)} \\ \dfrac{\partial yq_j^{(6-k)}}{\partial xq_j^{(5-k)}} = -2\bar{\gamma} \cdot xi_j^{(5-k)} \cdot yq_j^{(5-k)} \end{cases} \tag{4.24}$$

$$\begin{cases} \dfrac{\partial xi_j^{(6-k)}}{\partial yi_j^{(5-k)}} = 2\bar{\gamma} \cdot xi_j^{(5-k)} \cdot yq_j^{(5-k)} \\ \dfrac{\partial xq_j^{(6-k)}}{\partial yi_j^{(5-k)}} = 2\bar{\gamma} \cdot xq_j^{(5-k)} \cdot yq_j^{(5-k)} \\ \dfrac{\partial yi_j^{(6-k)}}{\partial yi_j^{(5-k)}} = 1 + 2\bar{\gamma} \cdot yi_j^{(5-k)} \cdot yq_j^{(5-k)} \\ \dfrac{\partial yq_j^{(6-k)}}{\partial yi_j^{(5-k)}} = \bar{\gamma}((xi_j^{(5-k)})^2 + (yi_j^{(5-k)})^2 + (yi_j^{(5-k)})^2) + 3\bar{\gamma} \cdot yq_j^{(5-k)} \cdot yq_j^{(5-k)} \end{cases} \tag{4.25}$$

$$\begin{cases} \dfrac{\partial xi_j^{(6-k)}}{\partial yq_j^{(5-k)}} = -2\bar{\gamma} \cdot xi_j^{(5-k)} \cdot yi_j^{(5-k)} \\ \dfrac{\partial xq_j^{(6-k)}}{\partial yq_j^{(5-k)}} = -2\bar{\gamma} \cdot xq_j^{(5-k)} \cdot yi_j^{(5-k)} \\ \dfrac{\partial yi_j^{(6-k)}}{\partial yq_j^{(5-k)}} = -\bar{\gamma}((xi_j^{(5-k)})^2 + (yq_j^{(5-k)})^2 + (yq_j^{(5-k)})^2) - 3\bar{\gamma} \cdot yi_j^{(5-k)} \cdot yi_j^{(5-k)} \\ \dfrac{\partial yq_j^{(6-k)}}{\partial yq_j^{(5-k)}} = 1 - 2\bar{\gamma} \cdot yi_j^{(5-k)} \cdot yq_j^{(5-k)} \end{cases} \tag{4.26}$$

where the derivatives are introduced as intermediate values. The second step is

$$\begin{cases} \bar{xi}_j^{(k+1)} = \bar{xi}_j^{(k)} \cdot \dfrac{\partial xi_j^{(6-k)}}{\partial xi_j^{(5-k)}} + \bar{xq}_j^{(k)} \cdot \dfrac{\partial xq_j^{(6-k)}}{\partial xi_j^{(5-k)}} + \bar{yi}_j^{(k)} \cdot \dfrac{\partial yi_j^{(6-k)}}{\partial xi_j^{(5-k)}} + \bar{yq}_j^{(k)} \cdot \dfrac{\partial yq_j^{(6-k)}}{\partial xi_j^{(5-k)}} \\ \bar{xq}_j^{(k+1)} = \bar{xi}_j^{(k)} \cdot \dfrac{\partial xi_j^{(6-k)}}{\partial xq_j^{(5-k)}} + \bar{xq}_j^{(k)} \cdot \dfrac{\partial xq_j^{(6-k)}}{\partial xq_j^{(5-k)}} + \bar{yi}_j^{(k)} \cdot \dfrac{\partial yi_j^{(6-k)}}{\partial xq_j^{(5-k)}} + \bar{yq}_j^{(k)} \cdot \dfrac{\partial yq_j^{(6-k)}}{\partial xq_j^{(5-k)}} \\ \bar{yi}_j^{(k+1)} = \bar{xi}_j^{(k)} \cdot \dfrac{\partial xi_j^{(6-k)}}{\partial yi_j^{(5-k)}} + \bar{xq}_j^{(k)} \cdot \dfrac{\partial xq_j^{(6-k)}}{\partial yi_j^{(5-k)}} + \bar{yi}_j^{(k)} \cdot \dfrac{\partial yi_j^{(6-k)}}{\partial yi_j^{(5-k)}} + \bar{yq}_j^{(k)} \cdot \dfrac{\partial yq_j^{(6-k)}}{\partial yi_j^{(5-k)}} \\ \bar{yq}_j^{(k+1)} = \bar{xi}_j^{(k)} \cdot \dfrac{\partial xi_j^{(6-k)}}{\partial yq_j^{(5-k)}} + \bar{xq}_j^{(k)} \cdot \dfrac{\partial xq_j^{(6-k)}}{\partial yq_j^{(5-k)}} + \bar{yi}_j^{(k)} \cdot \dfrac{\partial yi_j^{(6-k)}}{\partial yq_j^{(5-k)}} + \bar{yq}_j^{(k)} \cdot \dfrac{\partial yq_j^{(6-k)}}{\partial yq_j^{(5-k)}} \end{cases} \tag{4.27}$$

where the superscript in xi, xq, yi or yq is the layer index in the FP while the superscript in $\bar{xi}$, $\bar{xq}$, $\bar{yi}$ or $\bar{yq}$ is the layer index in BP. All subscripts represent the sample index. It can be seen that the complexity of the calculations is drastically decreased compared to the original Kerr BP layer shown from Eq. (4.13) to Eq. (4.18).

5

Hardware Implementation

In this chapter, the digital circuit implementation of our proposed equalizer model is demonstrated and explained in detail, layer by layer. The introduction starts with the overall structure of the hardware equalizer, as shown in Fig. 5.1. Such a structure is basically the same as the modified software model shown in Fig. 4.3 where each component is replaced with the hardware version and new shift registers (SRs) are added.

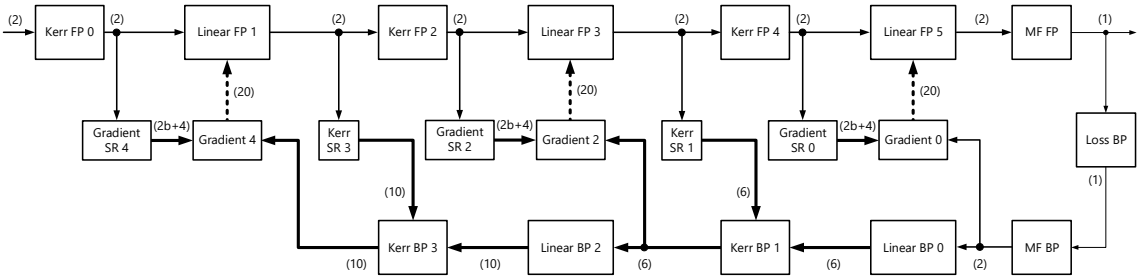


Figure 5.1: The system structure of the hardware implementation.

In the hardware equalizer, the throughput is 2 samples per clock cycle, which means that the equalizer circuit implementation can process 2 samples every clock cycle. In the software implementation, the calculation of all the $2b$ samples in a batch can be seen as finished in one iteration. In the hardware implementation, the throughput is 2 samples per cycle, but it takes $2b$ samples (b clock cycles) for each Gradient layer to produce one gradient output, and then one weight update can happen. Therefore, the batch size in the hardware equalizer is still $2b$ samples.

When porting the original software model to the hardware, the whole system is pipelined both in the FP and in the BP, which means that registers are inserted in all layers. On the one hand, the system is fully pipelined, which means that the throughput becomes two samples per cycle after the pipeline is full. On the other hand, the pipeline introduces latency to each layer. Table 5.1 shows the latency of each layer in the equalizer. This thesis only implements 3 layers and Section 7.1.1 further discusses the latency in a system with more than 3 layers in the future. The second modification is that SR components are added to the system. There are two inputs in the linear BP layer and the Kerr BP layer, one from the output of the previous BP layer and the other from the input of the corresponding FP layer, as shown in Fig. 5.1. Since there is latency in each layer, the input from

the corresponding FP layer is not synchronized with the input from the previous BP layer. The SRs are introduced to solve this issue. For Kerr SR, the latency equals the sum of all latencies on the right side. Take Kerr SR 1 as an example, the latency $b + 12$ is equal to the latency of Kerr FP 4, Linear FP 5, MF FP, Loss BP, MF BP and Linear BP 0 added together, and thus the two inputs are aligned. The latency of each Gradient SR layer is equal to the overall latency on the right side plus 2 clock cycles. This is because the input from the corresponding Linear FP layer should include the 4 previous samples stored inside the MIMO-FIR filter in the Linear FP layer. As the throughput is 2 samples per cycle, 2 more clock cycles should be delayed in each Gradient SR layer.

Table 5.1: The latency in each layer in the hardware implementation.

Layer/Component	Latency (Clock Cycles)
Kerr FP	4
Linear FP	1
MF FP	3
Loss BP	1
MF BP	$b + 1$
Linear BP	2
Kerr BP	2
Gradient	b
Gradient SR 0	$b + 6$
Gradient SR 2	$b + 15$
Gradient SR 4	$b + 24$
Kerr SR 1	$b + 12$
Kerr SR 3	$b + 21$

The input size and output size of each component in the hardware equalizer in Fig. 5.1 are not b or $2b$ samples like the software model, and the port sizes in number of samples are noted in parentheses next to each arrow. The thinnest arrow represents 1 sample (symbol), the medium-sized arrow represents 2 samples, the thick arrow represents a port size larger than 2 samples, and the dashed arrow still represents 20 weights. In the FP, from layer Kerr 0 to layer Linear FP 5, both the input size and output size are 2 samples. The down-sampling is included in the MF BP layer so that its output size is one symbol. In the BP, the MF BP layer converts the data stream back to 2 samples per cycle, and thus it has an output size of 2 samples. For the remaining layers in the BP, all input sizes or output sizes that are larger than 2 samples are represented by thick arrows. These port sizes can be different in each layer which will be further discussed in the following introduction of each layer.

Although the port sizes in the BP are larger than 2 samples, the throughput of the entire equalizer is still 2 samples per cycle. In each batch in the BP, the number of input samples is 0 in some clock cycles, and the number of input samples is larger than 2 in other clock cycles, and thus the throughput averaged on a whole batch in the BP is still 2 samples per cycle. As a result, the weights can still be updated every $2b$ cycles.

5.1 Batch-based Forward Propagation and Backward Propagation

In order to explain the hardware structure of the equalizer more clearly, the batch-based FP and BP should be outlined first. Fig. 5.2 shows a simplified example including one Linear FP layer, one Kerr FP layer and one MF FP layer. The t denotes the discrete time index, and the superscript denotes the layer index. The inputs to the Linear FP layer and the MF FP layer are 2 samples per cycle, and thus the sample is denoted as "0" or "1" to represent the first sample or the second sample in the 2 samples. The Linear FP layer and the MF FP layer are simplified to contain only 5 taps. As an example, the batch size is set to 10 samples.

The circles denote the inputs and outputs of each layer. The arrows denote the inputs and outputs in a batch. The circles denote the inputs and outputs of each layer. Each line in the Linear FP layer represents a MIMO-FIR filter tap multiplied by an input, and the results are added together as 5 lines that converge to each output. Each line represents an RRC tap in the MF FP layer, and the multiplication at each input and the addition at each output are the same as in the Linear FP layer. The single line in each Kerr FP block represents multiplication by 1.

In the MF FP layer, one out of two samples is selected as the final output and the other is discarded, by which the down-sampling of 2 is realized. The flow from top to bottom is the FP. When the directions of the input and output arrows are reversed, the flow from bottom to top becomes the BP, which is colored in Fig. 5.2.

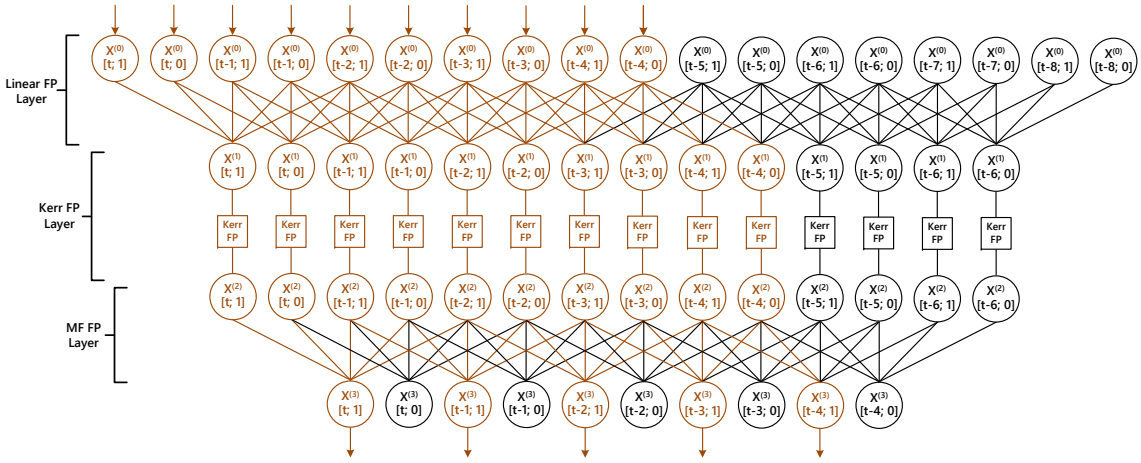


Figure 5.2: The batch-based FP and BP shown in an example case including one linear FP layer, one Kerr FP layer and one MF FP layer.

In the Linear FP layer in, only the 6 newest samples with the time index from $t-2$ to t are stored in the MIMO-FIR filter since the number of taps is set to 5 and the 2-sample parallelism is adopted. In the MF FP layer, only the 6 newest samples with the time index from $t-2$ to t are stored in the FIR filter since the number of RRC taps is set to 5. However, in the BP, as shown colored in Fig. 5.2, the already discarded samples in the FP are used as inputs in the BP, such as the

samples with $t-3$ and $t-4$ in the Linear FP layer. Thus, these samples should be temporarily stored until reused in the BP. The SRs shown in Fig. 5.1 are responsible for storing these discarded FP values, in addition to their function of compensating for hardware latency.

5.2 Forward Propagation

This section focuses on introducing the hardware structure of each component in the FP by using block diagrams.

5.2.1 Linear FP Layer

The Linear FP layer in the FP is based on the MIMO-FIR filter, a combination of the MIMO structure and the FIR structure. Fig. 5.3 shows the detailed hardware architecture of the Linear FP layer. In the top-level architecture shown at the far right, there are SRs with a step of 2 samples corresponding to the input 2 samples per cycle. There are two MIMO-FIR filters implemented and working in parallel, and thus two output samples can be generated at each cycle. The symbol S denotes one sample including one X_i, one X_q, one Y_i, and one Y_q. The boxes denote registers. In the brackets of each sample, the first item is the discrete time index and the second item denotes whether this is the first or second sample in the two samples per cycle. At the far left, two 2×2 real MIMO systems are shown and the inputs to such MIMO structure are $X_i[t-2;0]$, $X_q[t-2;0]$, $Y_i[t-2;0]$, and $Y_q[t-2;0]$. This MIMO structure is duplicated 5 times for each input of the MIMO-FIR. The output in the shown MIMO structure, $X_i[t-2;0]$, is added with the outputs of other MIMOs namely $X_i[t-2;1]$, $X_i[t-1;0]$, $X_i[t-1;1]$ and $X_i[t;0]$. The result of the addition shown in the middle is $X_i[t;0]$, which is part of the final output. The same addition as X_i is duplicated 4 times for X_q, Y_i and Y_q, as shown in the middle. It should also be noticed that the output is delayed for 1 clock cycle because of the pipelining.

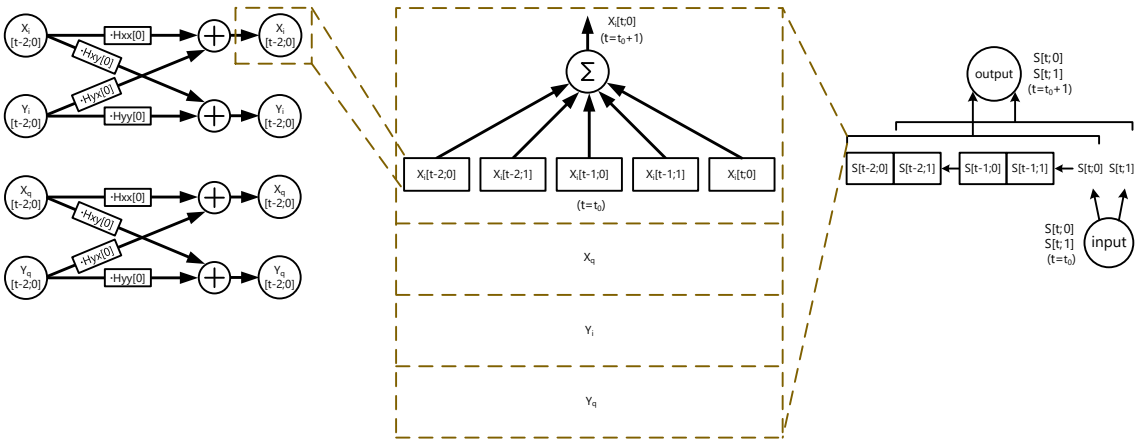


Figure 5.3: The hardware structure of the Linear FP layer. The ***S*** denotes the sample. The first item in brackets of ***S*** is the time index and the second item denotes whether this is the first or second sample in the 2 samples per clock cycle.

5.2.2 Kerr FP Layer

The Kerr FP layer used in this equalizer is based on a previous work [38], with the modification that the ϕ is replaced with $-\phi$ to compensate for the optical Kerr effect. Fig. 5.4 shows the hardware structure of the Kerr FP layer. The critical path is pipelined by 4 registers and thus the latency in this layer is 4 clock cycles. The power of the transmitted signal is firstly calculated and stored in a register. The power is then multiplied with $\bar{\gamma}$ to be converted to a rotation angle ϕ, which is also stored in a register. Then a Limit block can convert the ϕ to the range of $[0, \pi/2]$. A Sine look-up table (LUT) stores all the sine values which are indexed by the input angle with the range $[0, \pi/2]$, and the values of both $\sin\phi$ and $\cos\phi$ can be read out and stored in registers. Finally, the boxed block shows the rotation of the x-polarization in the Kerr FP layer, while the y-polarization is a duplication that is not shown. The rotate block rotates the input signal with the angle $-\phi$, and the inputs X_i and X_q are also delayed by 3 registers to be synchronized with the other inputs $\sin\phi$ and $\cos\phi$.

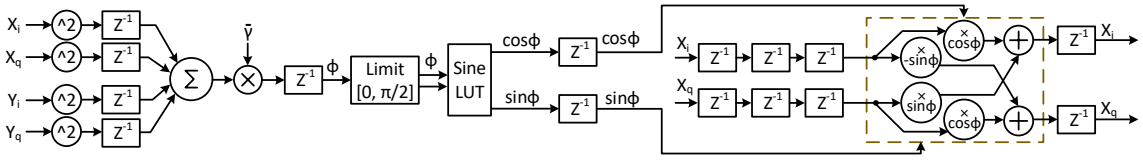


Figure 5.4: The hardware structure of the MF FP layer [38]. The inputs X_i and X_q to the boxed block are delayed by 3 registers.

5.2.3 MF FP Layer

The hardware structure of the MF layer used in this thesis project is based on the pulse-shaping RRC filter in the transmitter proposed by [37], but modified to be used in the equalizer as a matched filter. Fig. 5.5 shows the hardware structure of the MF FP layer, and it should be noticed that only the matched filter with 51 taps is constructed. The xi can be either xq, yi and yq. The square box represents a register, and a 26×26 matrix of registers is constructed. There is also a set of 26 SRs implemented which is shown under the matrix. The SRs shift with a step of 2 samples, which correspond to the input of 2 samples per cycle. There is also a multiplier implemented and accompanying each item in the SRs, and each stored sample is multiplied with the RRC taps from $M[0]$ to $M[25]$. The register matrix is also a set of SRs, and each entire row can be viewed as one item in SRs. The entire bottom row is shifted upwards with a step of 1 to the top row, and the calculated multiplication results are shifted upwards instead of the samples stored in the registers. The content stored in each register is also shown in Fig. 5.5. It should be noticed that the multipliers are only implemented once at the bottom SRs, while the "$\times$" operator in the register matrix from row 0 to row 25 is just used to denote the already calculated value.

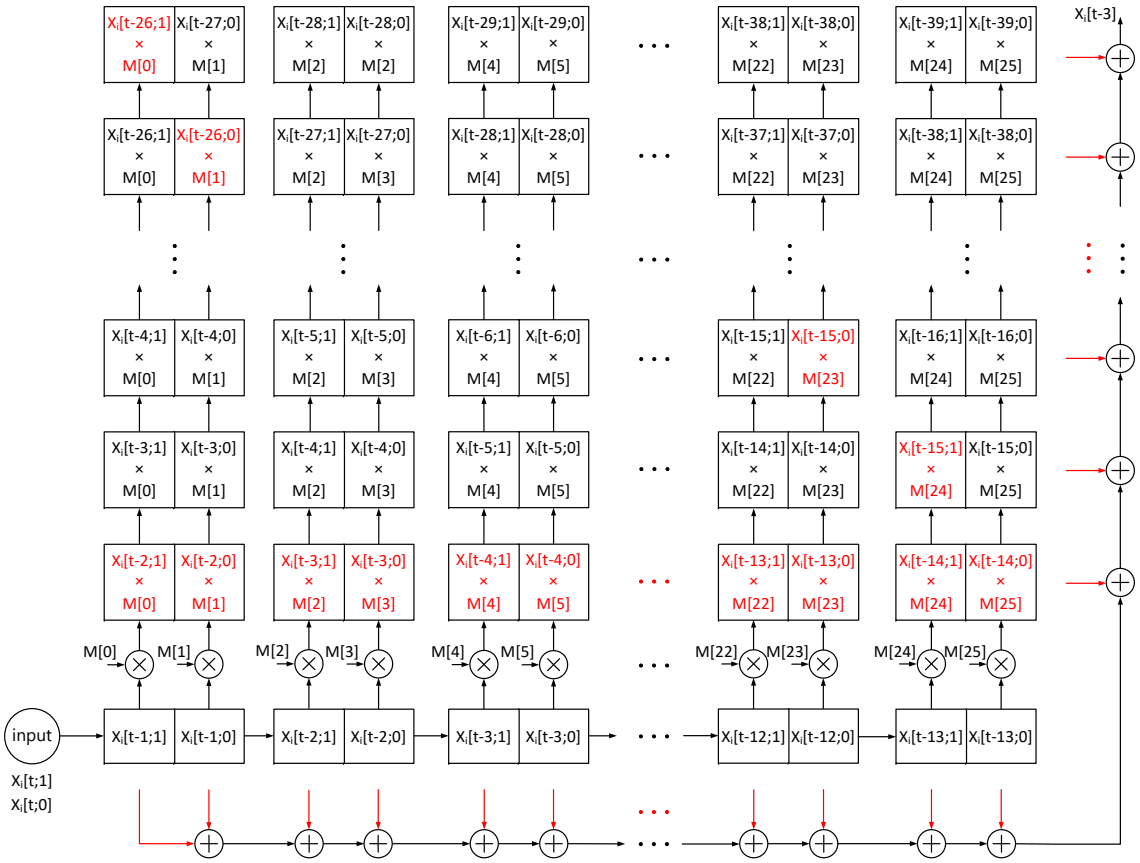


Figure 5.5: The hardware structure of the MF FP layer with 51 taps. The xi can be either xq, yi and yq. The first item the in brackets is the time index. The second item denotes the first or the second sample in the 2 samples per clock cycle.

The result of the MF FP layer is calculated from the diagonal samples and the samples in the bottom row, which are denoted red in Fig. 5.5. The final result is calculated by adding all the "red" samples. The adder chain is shown on the right and the bottom sides. A critical feature of the proposed MF FP layer architecture is that the SRs at the bottom row shift with a step of 2, and thus the down-sampling of 2 is also implemented. Also, it can be seen that due to pipelining, the final output is delayed by 3 clock cycles, one cycle from the bottom SRs, one cycle from the register matrix, and one cycle from the pipelined adder chain.

5.3 Backward Propagation

In this section, the hardware structure of each block in the BP is demonstrated in detail with block diagrams. The state machines in the layers are also thoroughly explained by their input/output mechanism and state-transition diagram (STD).

5.3.1 Loss BP Layer

The loss BP layer serves as the first layer in the BP, and the outputs from the FP combined with the aligned labels are sent to the loss BP layer.

5.3.1.1 Calculation

The loss function defined in Eq. (4.4) does not need to be implemented in the hardware since the BP does not need the results of the loss function itself. Only the backward propagation of the loss function needs to be implemented which is shown in Eq. (4.7). This equation can be rewritten in a more abstract and general way for hardware implementation:

$$\bar{xi} = \frac{(xi - \hat{xi})}{b} \tag{5.1}$$

where only xi is shown but the calculation is the same for xq, yi and yq. The xi denotes the output of FP, i.e., prediction. The $\hat{xi}$ denotes the desired FP output, i.e., label. The $\bar{xi}$ denotes the output of this layer.

5.3.1.2 Hardware Implementation

Fig. 5.6 shows the block diagram of the x-polarization in the Loss BP layer. The x-polarization label $\hat{X}$ is modulated to QPSK since it is stored as a 2-bit baseband symbol in the transmission in hardware. Then two parallel subtractors calculate the difference between the data symbol and the label. In the Loss BP layer, the division is approximated by right shifts to save the on-board DSP resources. We use a batch size of 20 and a data symbol wordlength of either 9, 10 or 11 as an example in Fig. 5.6 (see Section 6.4 about the wordlengths). Thus, we have $A >> 5 + A >> 6 + A >> 8 + A >> 10 = A/19.3208$. There exists a mechanism of VHDL Generics to ensure that at most 6 right-shift operators are implemented for different batch sizes and wordlengths when $16 \leq b \leq 22$. There is one register inserted for pipelining, and the y-polarization has the same hardware as the shown x-polarization implementation.

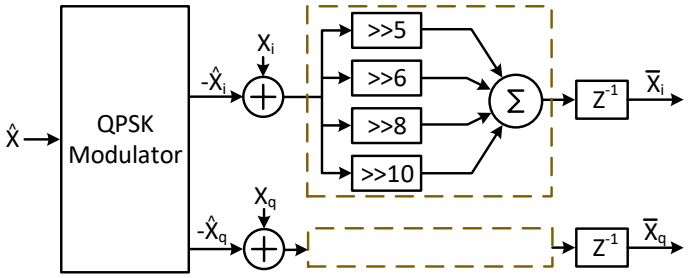


Figure 5.6: The block diagram of the x-polarization in the Loss BP layer. As an example, the batch size is set to 20 and the data symbol wordlength is either 9, 10 or 11.

5.3.2 MF BP layer

The input to the MF BP layer is the output from the Loss BP layer in one symbol per cycle, and the MF BP layer converts the data stream to 2 samples per cycle.

The high-level architecture of the MF BP layer can be summarized as a state machine plus so-called "operators". The operators can cover all the calculations in this layer by assigning different inputs and outputs to them. This enables the re-usage of the

basic operators in different states in the state machine, which can save hardware utilization.

5.3.2.1 Calculation

The calculation in the MF BP layer has already been shown in Eq. (4.8). Such equation is abstracted into basic operators

$$R = R + M \cdot \bar{u}_j \quad ib \leq j < (i+1)b \tag{5.2}$$

where R can be viewed as a register storing the value, and it is reset to 0 when a new batch starts. The M can be any of the 51 taps, the j is the symbol index and i is the batch index. The u represents the input symbol and can be either $\bar{X}_i$, $\bar{X}_q$, $\bar{Y}_i$ or $\bar{Y}_q$.

5.3.2.2 Basic Operators

Before introducing the basic operators themselves, how the input and the output work in the MF BP layer should be first introduced, as shown in Fig. 5.7. The input is one symbol per cycle, while the output is two samples per cycle. The squares $R[0]$, $R[1]$, ..., $R[2b-1]$ are registers. The arrow accompanied by the RRC tap represents the basic operator, as shown in Eq. (5.2). At $t = 0$, only the first 2 operators $R[0]$ and $R[1]$ as well as the connected basic operators are activated. At $t = 1$, the first 4 operators $R[0]$, $R[1]$, $R[2]$ and $R[3]$ and the accompanied operators are activated. The same pattern continues until the end of the batch, and all $2b$ operators are activated at $t = b - 1$.

At $t = b$, the first two results from the previous finished batch stored in $R[0]$ and $R[1]$ are read out and sent to the output. The two new values from the second batch are stored in $R[0]$ and $R[1]$ at the same time. In other words, the outputs at $t = b$ are the first and the second output sample in the previous batch. Then at $t = 1$, the output is shifted to $R[2]$ and $R[3]$. The pattern continues until $t = 2b - 1$ when the output is shifted to $R[2b-2]$ and $R[2b-1]$. The previous batch is completely output to the next layer at $t = 2b - 1$. To summarize this pattern, the results of a batch will be read out from the registers in the next batch, and the values in the registers will be replaced by the intermediate values in the next batch after being read out.

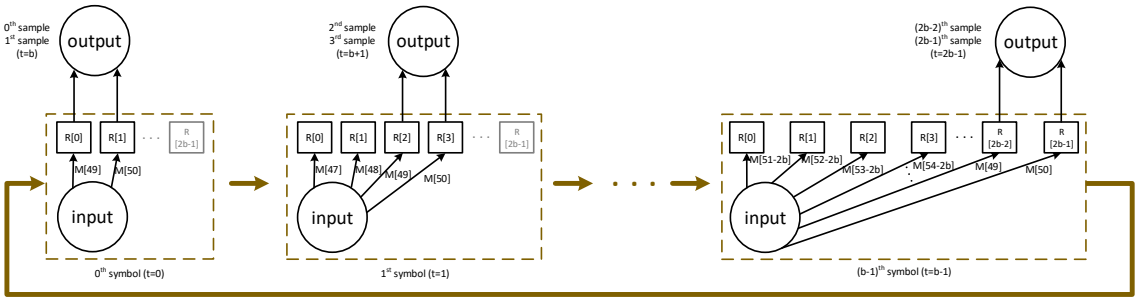


Figure 5.7: The input and output of the MF BP layer.

As shown in Fig. 5.7, every MF BP layer has $2b$ operators working in parallel in each channel, and thus every MF BP layer has $8b$ parallel operators in total. Fig. 5.8 shows the block diagram of an individual operator and how all the operators are organized into the MF BP layer.

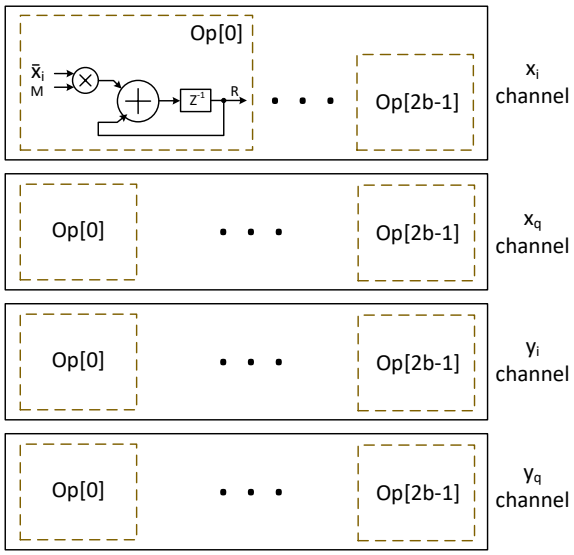


Figure 5.8: The block diagram of the basic operator in the MF BP layer, and all parallel basic operators in a layer. Each layer contains 4 parallel channels and each channel contains $2b$ identical basic operators.

5.3.2.3 State Machine

The entire MF BP layer is controlled by a state machine, and Fig. 5.7 can also be seen as an STD. There are b states in total when $b \leq 25$, and each block in Fig. 5.7 can be seen as a single state. At $t = 0$ the state machine shifts to State 0 (S0), and $t = 1$ to S1, etc. At each state, one symbol in a batch is processed, and the b states correspond to the b symbols in a batch. At State $b-1$ a batch is finished and the state machine returns back to S0 to process a new batch. The state machine controls all the inputs and outputs of the basic operators, and different inputs and outputs are selected in different states. The hardware utilization in the MF BP layer increases as the batch size increases since more basic operators working in parallel need to be implemented, as shown in the last state in Fig. 5.8. This increase of hardware utilization stops when the batch size reaches 26 for this particular 51-tap MF. Moreover, the state machine and the input and output pattern of the MF BP layer are also changed when $b > 25$. See Section 7.1.2 about the discussion of the MF BP layer with $b > 25$.

5.3.3 Linear BP Layer

The Linear BP layers have different input sizes and output sizes at different layer indices k. The MF BP layer converts the data stream from 1 symbol per cycle to 2 samples per cycle. Thus, the Linear BP layer right after the MF BP layer has an input size of 2 samples. However, the output size is actually 6 samples. For

the second Linear BP layer, the input size becomes 6 samples and the output size becomes 10 samples.

The Linear BP layer follows the same high-level architecture as the MF BP layer, which is a state machine plus basic operators.

5.3.3.1 Calculation

The calculation of the Linear BP layer can be divided into two parts which are shown in Eq. (4.11) and Eq. (4.12). The basic operators can be extracted as

$$\begin{cases} \bar{xi}^{(k+1)} = \sum_{d=0}^{4} Hxx^{(4-d)} \cdot \bar{xi}_{j+d}^{(k)} + \sum_{d=0}^{4} Hxy^{(4-d)} \cdot \bar{yi}_{j+d}^{(k)} \\ \bar{xq}^{(k+1)} = \sum_{d=0}^{4} Hxx^{(4-d)} \cdot \bar{xq}_{j+d}^{(k)} + \sum_{d=0}^{4} Hxy^{(4-d)} \cdot \bar{yq}_{j+d}^{(k)} \\ \bar{yi}^{(k+1)} = \sum_{d=0}^{4} Hyx^{(4-d)} \cdot \bar{xi}_{j+d}^{(k)} + \sum_{d=0}^{4} Hyy^{(4-d)} \cdot \bar{yi}_{j+d}^{(k)} \\ \bar{yq}^{(k+1)} = \sum_{d=0}^{4} Hyx^{(4-d)} \cdot \bar{xq}_{j+d}^{(k)} + \sum_{d=0}^{4} Hyy^{(4-d)} \cdot \bar{yq}_{j+d}^{(k)} \end{cases}, \quad 2ib \le j < 2(i+1)b - 4 \tag{5.3}$$

and

$$\begin{cases} \bar{xi}^{(k+1)} = \sum_{d=0}^{2(i+1)b-j-1} Hxx^{(4-d)} \cdot \bar{xi}_{j+d}^{(k)} + \sum_{d=0}^{2(i+1)b-j-1} Hxy^{(4-d)} \cdot \bar{yi}_{j+d}^{(k)} \\ \bar{xq}^{(k+1)} = \sum_{d=0}^{2(i+1)b-j-1} Hxx^{(4-d)} \cdot \bar{xq}_{j+d}^{(k)} + \sum_{d=0}^{2(i+1)b-j-1} Hxy^{(4-d)} \cdot \bar{yq}_{j+d}^{(k)} \\ \bar{yi}^{(k+1)} = \sum_{d=0}^{2(i+1)b-j-1} Hyx^{(4-d)} \cdot \bar{xi}_{j+d}^{(k)} + \sum_{d=0}^{2(i+1)b-j-1} Hyy^{(4-d)} \cdot \bar{yi}_{j+d}^{(k)} \\ \bar{yq}^{(k+1)} = \sum_{d=0}^{2(i+1)b-j-1} Hyx^{(4-d)} \cdot \bar{xq}_{j+d}^{(k)} + \sum_{d=0}^{2(i+1)b-j-1} Hyy^{(4-d)} \cdot \bar{yq}_{j+d}^{(k)} \end{cases}, \quad 2(i+1)b - 4 \le j < 2(i+1)b \tag{5.4}$$

5.3.3.2 Basic Operators

Fig. 5.9 shows how the input and the output work in the first Linear BP layer, and the brackets with an arrow is the basic operator. For simplicity, only the calculation in the xi channel is shown. There are $6 + 2k$ operators implemented, and note that the first Linear BP layer has an index $k = 0$ and the second Linear BP layer has $k = 2$. The operators can be divided into two types: $2 + 2k$ operators with an input width of 10 samples corresponding to Eq. (5.3), and 4 colored operators with an input width shorter than 10 samples corresponding to Eq. (5.4). We will refer to the 10-sample operator as $Op1$ and the other colored operators with shorter than 10 samples as $Op2$. In the first k clock cycles, i.e., $2k$ samples, the entire Linear BP layer is deactivated. At $t = k$ and $t = k + 1$, only the inputs are activated while the outputs and the basic operators still stay idle. The incoming samples are forwarded to registers represented as squares. At $t = k + 2$, two input ports are activated and the registers start shifting with a step of two samples. The outputs at $t = k + 2$ are the first and the second output sample in this batch. Meanwhile, two $op1$ operators and two output ports are also activated. Then in the following cycles, the registers continue shifting and two results are continually sent to the output at each cycle

until $t = b-2$. At $t = b-1$, all the $2+2k$ operators are activated and all the results are sent to the output ports at the same clock cycle. Then this batch ends and the state machine returns to the first state to start accepting the next batch.

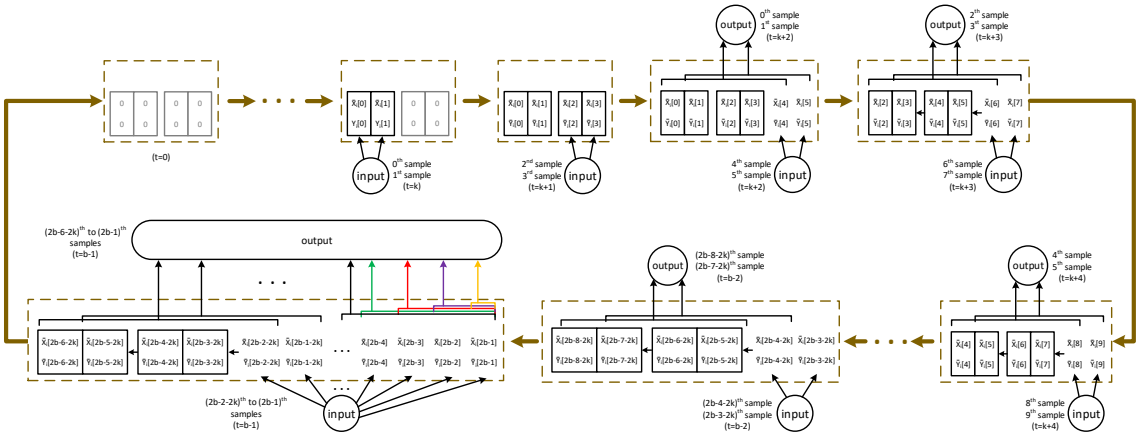


Figure 5.9: The input and output of the Linear BP layer k. For simplicity, only the calculation in the xi channel is shown.

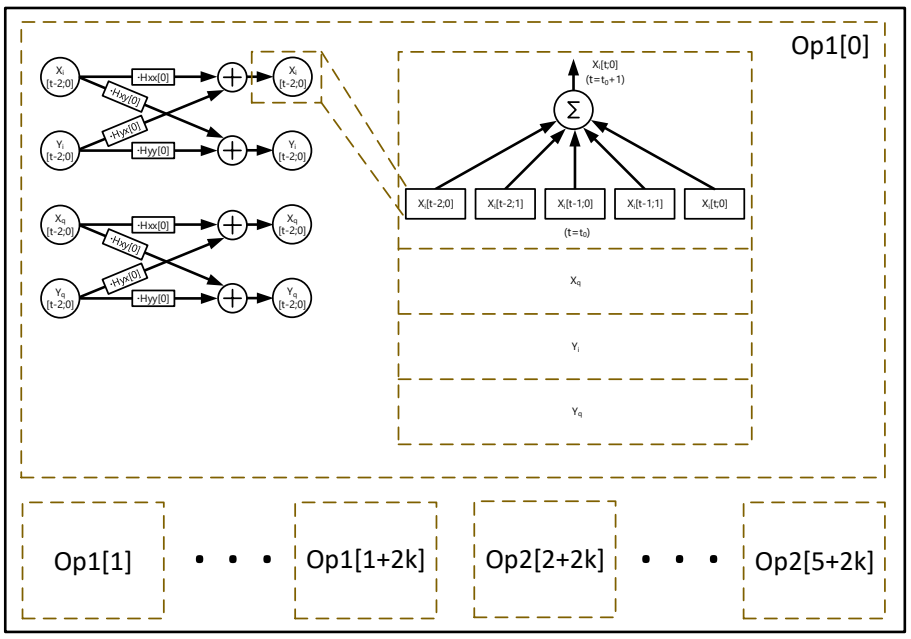


Figure 5.10: The block diagram of the basic operator in the Linear BP layer and all parallel basic operators in a layer. Each operator covers all 4 channels. Only the structure of the $Op1$ is shown while the structure of the $Op2$ is not specifically illustrated. Each layer contains $6 + 2k$ operators in total.

Note that the basic operator defined in Eq. (5.3) and Eq. (5.4) includes all four channels xi, xq, yi and yq, and thus every Linear BP layer contains $6 + 2k$ parallel operators including $2 + 2k$ $Op1$ operators and 4 $Op2$ operators. The number of operators and the level of parallelism increase as the layer index increases. Such

increase stops when reaches the layer index reaches $k = b-2$ if b is even or $k = b-3$ if b is odd. See Section 7.1.3 about the discussion of the Linear BP layer when the number of layers is above 3.

Fig. 5.10 shows the block diagram of the basic operator in the Linear BP layer and how all basic operators form into a Linear BP layer.

5.3.3.3 State Machine

Each Linear BP layer is controlled by a state machine. It controls all the inputs and outputs of the basic operators, and different inputs and outputs are selected in different states. Fig. 5.11 shows the STD of the state machine in the first layer. The state S0 corresponds to $t = 0$ in Fig. 5.9, the state S1 corresponds to $t = 1$, the state S2 covers from $t = 2$ to $t = b - 2$, and the state S3 corresponds to $t = b - 1$. Then the state machine returns to the initial state S0. The state machine is activated by a valid signal from the previous BP layer. There is a counter counting from 0 to $b - 1$ running in parallel with the state machine, and it starts counting after being activated by the valid signal. When the counter reaches $b - 2$, the state is changed to S3.

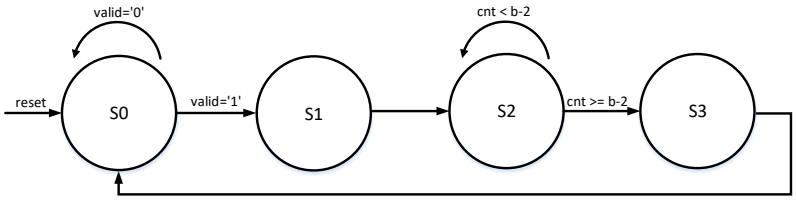


Figure 5.11: The STD of the state machine in the Linear BP layer $k = 0$.

Fig. 5.12 shows the STD of the state machine in the second linear BP layer. It can be seen that S0 corresponds to $t = 0$, S1 corresponds to $t = 1$, S2 corresponds to $t = 2$, S3 corresponds to $t = 3$, S4 covers from $t = 4$ to $t = b - 2$, and S5 corresponds to the last $t = b - 1$ in a batch where all the operators and output ports are activated. Then transition from S4 to S5 happens when the counter reaches $b - 2$.

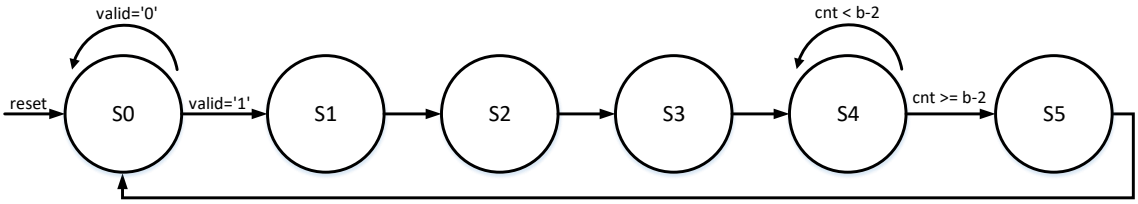


Figure 5.12: The STD of the state machine in the Linear BP layer $k = 2$.

5.3.4 Kerr BP Layer

The Kerr BP layer is the backward propagation of the Kerr FP layer. There is no basic operator or state machine in this layer. The calculation is already shown from Eq. (4.23) to Eq. (4.27).

5.3.4.1 Hardware Implementation

Fig. 5.13 shows the block diagram of the Kerr BP layer. The hardware is pipelined by inserting a layer of registers and can be divided into three parts. The two parts on the left are aligned by output registers, while the outputs from these two blocks serve as the input to the part at the far right. The left two parts correspond to Eq. (4.23) until Eq. (4.26), while the rightmost part corresponds to Eq. (4.27).

All the intermediate values are reused as much as possible in the leftmost part to save the DSP resources. The multiplication of 2 is also implemented by a shift left operator to save hardware resources.

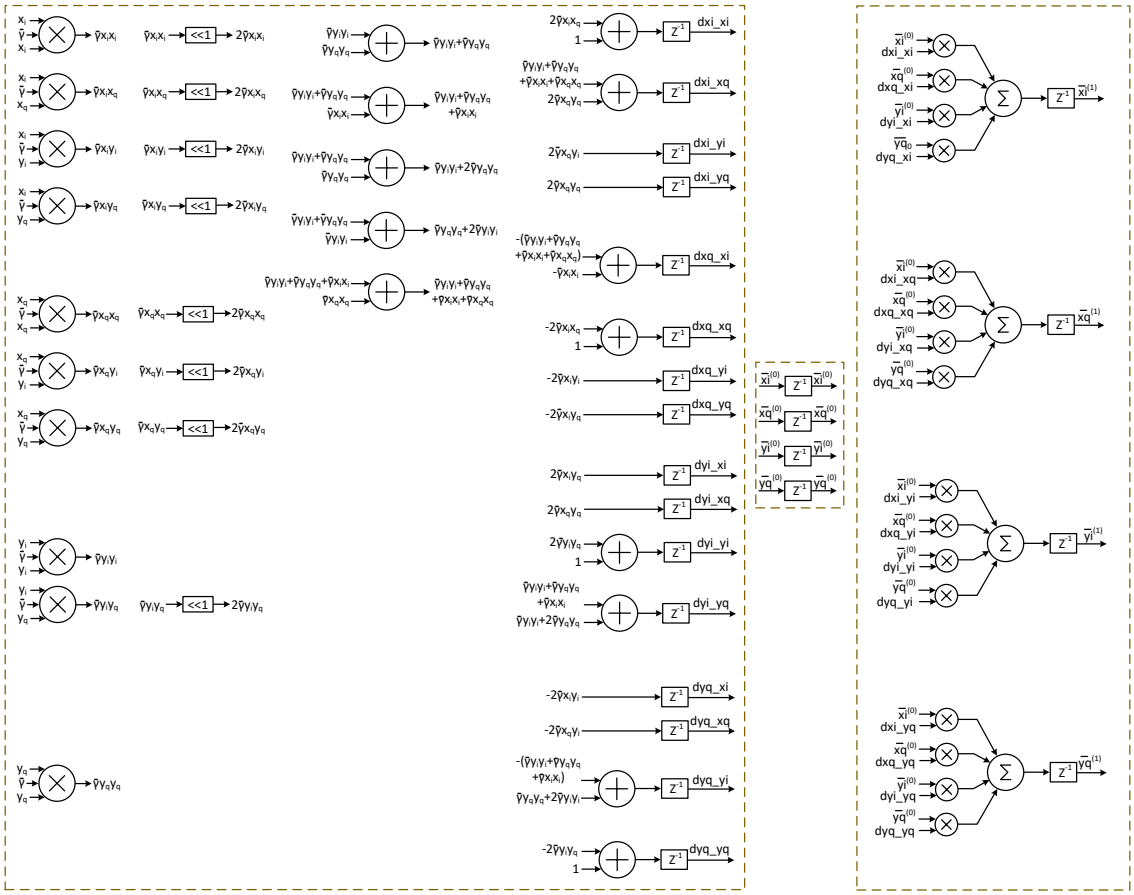


Figure 5.13: The block diagram of the Kerr BP layer. The superscript 0 in $\overline{xi}^{(0)}$ denotes it is the input, and the superscript 1 denotes it is the output. The "dxi_xq" denotes the $(\partial xi)/(\partial xq)$, etc.

5.3.5 Gradient Layer

Like the Linear BP layer, the Gradient layer also has a different input size when the layer index k differs. Moreover, like the Kerr BP layer, the inputs are from both the FP and the BP. In our implemented 3-layer equalizer, the input size is 2 samples when $k = 0$, 6 samples when $k = 2$, and 10 samples when $k = 4$.

The Gradient BP layer follows the same high-level architecture where basic operators

are implemented and a state machine controls the inputs and outputs of the basic operators.

5.3.5.1 Calculation

The calculation of the Gradient layer is shown in Eq. (4.19), from which the basic operators can be extracted in a more general and simpler fashion as

$$\begin{cases} R^{(Hxx,d)} = R^{(Hxx,d)} + \bar{xi}_j \cdot xi_{j-4+d} + \bar{xq}_j \cdot xq_{j-4+d} \\ R^{(Hyx,d)} = R^{(Hyx,d)} + \bar{xi}_j \cdot yi_{j-4+d} + \bar{xq}_j \cdot yq_{j-4+d} \\ R^{(Hxy,d)} = R^{(Hxy,d)} + \bar{yi}_j \cdot xi_{j-4+d} + \bar{yq}_j \cdot xq_{j-4+d} \\ R^{(Hyy,d)} = R^{(Hyy,d)} + \bar{yi}_j \cdot yi_{j-4+d} + \bar{yq}_j \cdot yq_{j-4+d} \end{cases}, \quad 2ib \leq j < 2(i+1)b \tag{5.5}$$

where R can be viewed as a register to temporarily store the intermediate value of either $\nabla Hxx^{(d)}$, $\nabla Hyx^{(d)}$, $\nabla Hxy^{(d)}$ or $\nabla Hyy^{(d)}$ (in a simplified format of symbol). The d refers to the tap index in the weights, and j denotes the sample index in the batch i. The $\bar{xi}$ denotes the input from the BP, and xi denotes the input from the FP, and the same goes for xq, yi and yq.

5.3.5.2 Basic Operators

There are three Gradient layers implemented, namely $k = 0$, $k = 2$ and $l = 4$ in our equalizer. Fig. 5.14 shows the input and the output of the Gradient layers, and it only shows the calculation of tap $Hxx^{(d)}$ as an example. The input port of the Gradient layer k has two parts: the $2 + 2k$ samples from the output of the previous BP layer and the $2b + 4$ samples from the intput of the corresponding Linear FP layer. The output port size is fixed as 5 taps from $\nabla Hxx^{(0)}$ to $\nabla Hxx^{(4)}$. Fig. 5.14 shows only the hardware of one item $Hxx^{(d)}$ and it can be seen from Eq. (5.5) that the operand shown in Fig. 5.14 should be duplicated for 20 times and working in parallel in each Gradient layer since each Linear FP layer contains 4 weights Hxx, Hyx, Hxy and Hyy, and each weight contains 5 taps. The arrow accompanied by an xi or an xq sample denotes a multiplication operator, and all the multiplied values are added to the value stored in the register $R[d]$.

From $t = 0$ to $t = k - 1$, the whole Gradient layer stays idle, which means that the inputs, the operators and the outputs are not activated. Note that these states do not exist when $k = 0$. From $t = k$ to $t = b - 2$, only 4 inputs are activated and all other inputs still stay idle by applying 0 to them. The $\bar{xi}$ or $\bar{xq}$ is directly from the input ports at each clock cycle with 2 samples per cycle. All the $2b + 4$ xi and xq inputs are only valid at $t = k$ and they are stored in registers at $t = k$, which is not shown in Fig. 5.14. At $t = b - 1$, all the $2 + 2k$ $\bar{xi}$ and $\bar{xq}$ inputs are activated as well as all the operators. The result $\nabla Hxx[d]$ finally finishes calculation and is sent to the output port at $t = b - 1$. Then a batch finishes, and the layer returns to the first state.

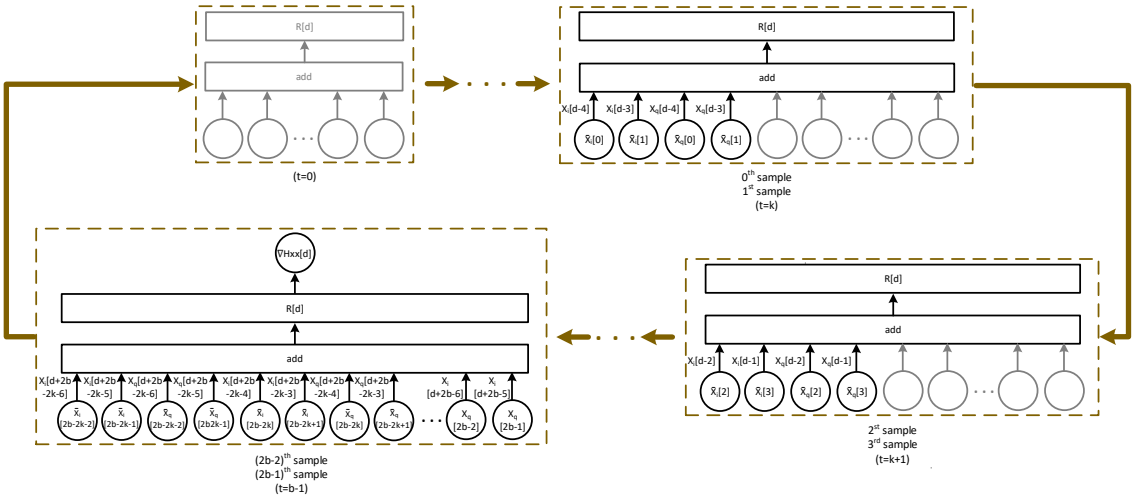


Figure 5.14: The input and output of the Gradient layer when $k = 0,\ 2$ or 4. It only shows the hardware of calculating of the d-th item in ∇Hxx, $\nabla Hxx^{(d)}$, as an example. The $\nabla Hyx^{(d)}$, $\nabla Hxy^{(d)}$ and $\nabla Hyy^{(d)}$ are duplications of the shown block diagram.

The basic operator shown in Eq. (5.5) contains all the four channels xi, xq, yi and yq. Thus, every Linear BP layer contains 5 operators working in parallel for the 5 taps. From Fig. 5.14, it can be seen that the number of multiplier-adders increases as the layer index increases, and thus the hardware consumption increases. Such increase stops when $k > 2\lfloor\frac{b-2}{2}\rfloor$ where "$\lfloor\ \rfloor$" is the floor operation. Section 7.1.4 discusses the hardware structure of the Gradient layer when the number of layers is above 3.

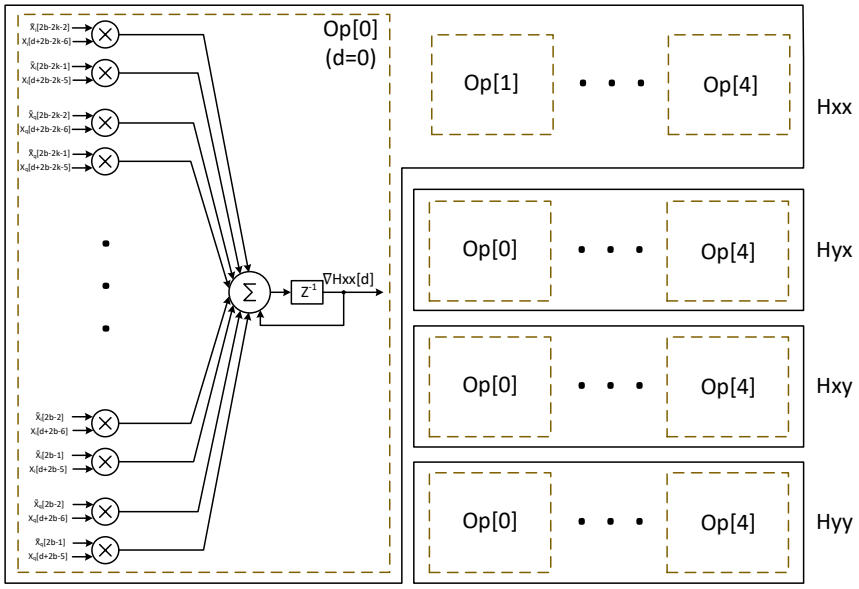


Figure 5.15: The block diagram of the basic operators and the structure of the entire Gradient layer ($k = 0$). Each layer contains 4 parallel gradients Hxx, Hyx, Hxy and Hyy. Each gradient contains 5 basic operators.

Fig. 5.15 shows the block diagram of the basic operators in the Gradient layer $k = 0$. Each Gradient layer contains 4 parallel gradients Hxx, Hyx, Hxy and Hyy, and each gradient contains 5 taps. The basic operator is for calculating 1 tap, and thus each Gradient layer contains 20 basic operators in total.

5.3.5.3 State Machine

Fig. 5.16 shows the STD of the state machine in Gradient layer $k = 0$. A counter counting from 0 to $b - 1$ runs in parallel with the state machine. The function of each state can be summarized as follows:

- S0: The state machine and the counter are activated if the valid signal becomes "1", and this state corresponds to $t = k$ in Fig. 5.14. Note that there are no idle states from $t = 0$ to $= k - 1$ in Fig. 5.14 when $k = 0$.
- S1: This state corresponds to the states from $t = k+1$ to $t = b-2$ in Fig. 5.14. The state machine remains in this state when cnt $< b - 2$.
- S2: This state corresponds to the last state $t = b - 1$.

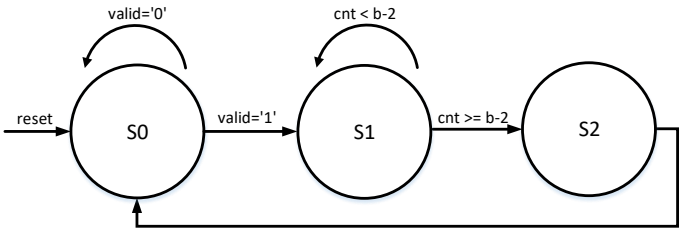


Figure 5.16: The STD of the state machine in Gradient layers $k = 0$.

Fig. 5.17 shows the STD of the state machine in Gradient layer $k = 2$ or $k = 4$. The function of each state can be summarized as follows:

- S0: The state machine and the counter are activated if the valid signal becomes "1", and this state corresponds to $t = 0$ in Fig. 5.14.
- S1: This state corresponds to $t = 1$ to $t = k$ in Fig. 5.14. The state machine remains in this state when cnt $< k - 1$.
- S2: This state corresponds to $t = k + 1$ to $t = b - 2$ in Fig. 5.14. The state machine remains in this state when when cnt $< b - 2$.
- S3: This state corresponds to the last state $t = b - 1$.

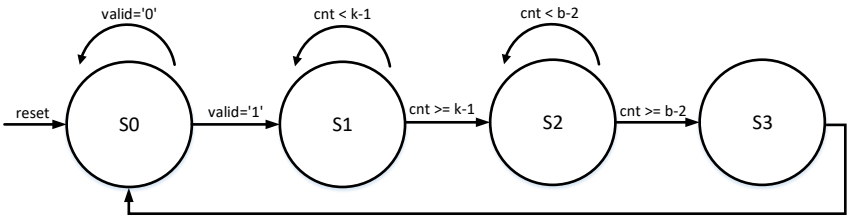


Figure 5.17: The STD of the state machine in the Gradient layer $k = 1$ or $k = 2$.

5.3.6 Weight Update Block

The weight update blocks are used to update the trainable weights in Linear FP layers, and therefore each Linear FP layer has a corresponding Weight Update block. Since the training is batch-based, the update is executed only once in each batch. This means that the weight update happens every b clock cycles, after the process of the $2b$ samples in a batch is finished.

The inputs to each Weight Update block are the trainable weights and the calculated gradients from the corresponding Gradient layer. The Weight Update block returns the updated weights. There is no state machine or basic operator in this block. The calculation of the Weight Update layer is shown in Eq. (4.5).

5.3.6.1 Hardware Implementation

Fig. 5.18 shows the block diagram of the Weight update block. In order to save the DSP resources, the learning rate ξ is implemented by left shifting the gradient with p bits. Thus the learning rate is limited to $2^p, p \in \mathbb{N}$. Such a learning rate is enough for this design.

There are 4 different weights in each Linear FP layer Hxx, Hyx, Hxy and Hyy, and each weight contains 5 taps. Fig. 5.18 shows only the weight update of one tap d, and thus the shown hardware should be duplicated 5 times to form a complete Weight Update layer. Moreover, it should be noticed that the left shift operations are implemented in the corresponding Gradient layer. In the Gradient layer, there are intermediate values whose wordlengths are longer than the outputs. Thus, these intermediate values should be right-shifted to be converted to the wordlength of the outputs. Thus, by subtracting the number of right shift bits with p, the left shift of p bits is realized.

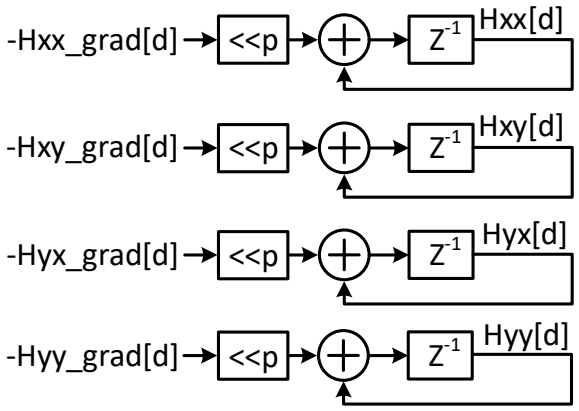


Figure 5.18: The block diagram of the Weight Update block. The same hardware is duplicated for the indices $d = 0, 1, 2, 3$, and 4. The symbol "Hxx_grad[d]" represents the gradient $\nabla_{Hxx^{(d)}}\mathcal{L}$, etc. The left shift operations are implemented in the Gradient layers but shown here for simplicity.

5.3.6.2 Weight Update Pattern

It should be noted that the timing when the weights are updated in the hardware implementation is different from the software. Table 5.2 shows how all the weights are updated in the software implementation, while Table 5.3 shows the hardware implementation. Each item in the tables is the symbol index. The $\boldsymbol{H}$ denotes the weights, and the subscript in $\boldsymbol{H}$ increases 1 when a weight update occurs. Thus, the tables can show at which symbol the weights in different layers are updated. In the software implementation, the three layers are simultaneously updated every b symbols, and there is no initial interval. In the hardware implementation, on the contrary, every layer has an initial interval when updating, and different layer has a different initial interval. As a result, all the layers are not updated at the same time. However, the hardware implementation is fully pipelined which means that the weight update still happens every b symbols after the pipeline is full.

Table 5.2: The pattern of weight update in the software implementation.

	$\boldsymbol{H_0}$	$\boldsymbol{H_1}$	$\boldsymbol{H_2}$	$\boldsymbol{H_3}$	$\boldsymbol{H_4}$	$\cdots$
Linear FP layer 0	0	$b-1$	$2b-1$	$3b-1$	$4b-1$	$\cdots$
Linear FP layer 1	0	$b-1$	$2b-1$	$3b-1$	$4b-1$	$\cdots$
Linear FP layer 2	0	$b-1$	$2b-1$	$3b-1$	$4b-1$	$\cdots$

Table 5.3: The pattern of weight update in the hardware implementation.

	$\boldsymbol{H_0}$	$\boldsymbol{H_1}$	$\boldsymbol{H_2}$	$\boldsymbol{H_3}$	$\boldsymbol{H_4}$	$\cdots$
Linear FP layer 0	0	$2b+28$	$3b+28$	$4b+28$	$5b+28$	$\cdots$
Linear FP layer 1	0	$2b+24$	$3b+24$	$4b+24$	$5b+24$	$\cdots$
Linear FP layer 2	0	$2b+20$	$3b+20$	$4b+20$	$5b+20$	$\cdots$

6

Experiments and Results

In this chapter, all experiments carried out in this paper and their setups are introduced, and all the experimental results are presented. The experiments are performed on both the software equalizer models and the VHDL hardware equalizer.

Firstly, the metrics used in this thesis are introduced. Secondly, the experiments on the optimal set of parameters carried out in the software equalizers and the hardware equalizer are presented. Thirdly, the performance of the software equalizer and that of the hardware equalizer are compared. Fourthly, the experiments on the hardware equalizer with a transient or time-varying channel are demonstrated. Finally, the FPGA synthesis and implementation results are shown.

6.1 Metrics

The metric adopted in this thesis to assess the performance of the equalizer is effective signal-to-noise ratio (SNR), which is defined as

$$\text{effective SNR} = \frac{||\hat{s}||}{||\hat{s} - s||} \tag{6.1}$$

where s is the original transmitted symbol and $\hat{s}$ is the symbol after equalization. In the rest of thesis, "SNR" and "effective SNR" both refer to the mean effective SNR which is calculated on a window of 4,096 symbols. The number 4,096 is deliberately chosen for porting the calculation of the mean effective SNR to the same FPGA in the future. The averaging window is sliding with a step of 1 symbol in order to more accurately illustrate how the SNR changes over time.

Two metrics are defined as below to evaluate the performance of the equalizers in both the software model and hardware implementation:

- *performance*: the final achieved effective SNR, which is calculated on the last 4,096 symbols.
- *convergence time*: the number of symbols before the equalizer converges to within 1% of its final effective SNR.

6.2 Model Verification

In order to verify that the Matlab equalizer proposed in Section 4.3 is an identical to the Tensorflow version proposed in Section 4.2, 20 fiber realizations are used

to generate the test data. Each fiber realization has a different set of $\theta^{(k)}$ $\forall k \in \{0, 1, 2, 3\}$ that are generated uniformly and randomly on $[-\pi, \pi]$, while all the other parameters are kept the same as Table 4.1. The same test data are used as the input to the Tensorflow equalizer and the Matlab equalizer, and the training curves from both of them are compared. The transmission power is set to 10 dBm, and there are 184,095 symbols used in the training. The setup of the channel simulation can be seen in Section 4.1 and Table 4.1, and the equalizer setup is shown in Table 4.2. The effective SNR is averaged on 4,096 symbols. Fig. 6.1 shows the training curves on one of the fiber realizations, and the two curves completely overlap each other. All training curves on the remaining 19 fiber realizations are also identical, which confirm that our mathematical derivation of the BP is correct.

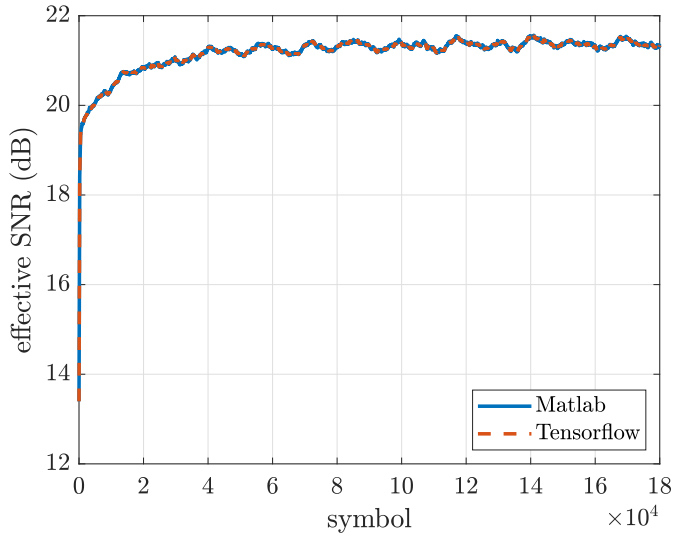


Figure 6.1: The training curves from the Tensorflow and the Matlab equalizer on one of the 20 fiber realizations.

6.3 Batch Size and Transmission Power in Software Equalizers

Before testing the VHDL hardware equalizer, the software equalizer is investigated firstly to provide a reference to the hardware implementation. The batch size b and the transmission power P are two important system-level hyperparameters that can strongly impact the performance and convergence time. To find the optimal batch size and transmission power, these two parameters are varied respectively and each parameterization is tested on the 20 fiber realizations. In each test, 184,095 symbols are used and the performance is calculated on the last 4,096 symbols.

Fig. 6.2 shows the performance with respect to the batch size and transmission power. The batch size is in number of symbols and thus the actual number of samples in a batch should be two times the number of symbols. For example, if the batch size is 20, the actual number of samples in a batch is 40. At each combination of batch size and transmission power, the performance is tested using the 20 different

fiber realizations, and the performance is averaged. The "without PMD and Kerr effect" line shows an ideal channel without any PMD or Kerr effect. The "channel inverse" means that the equalizer is initialized as the inverse of the PMD-Kerr effect in the channel, and that the BP is not activated. The "channel inverse" curve shows the upper limit of performance regardless of the batch size used. It can be seen that the performance decreases as the batch size decreases in general, and there is an optimal transmission power for each batch size.

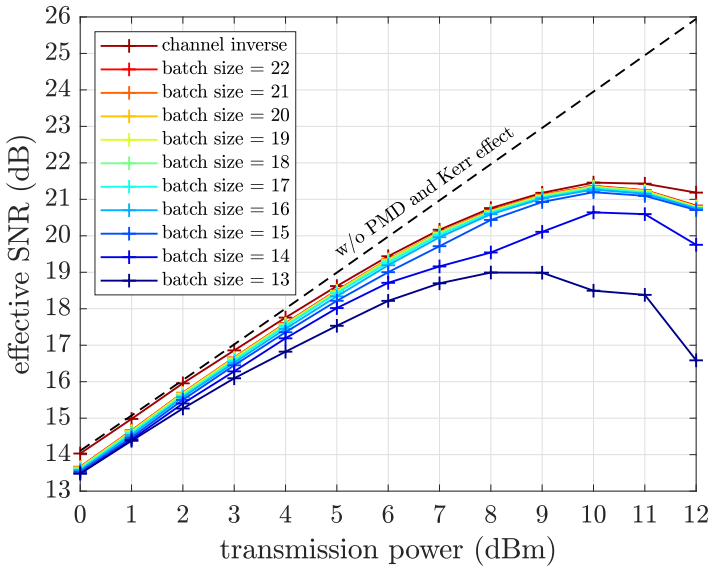


Figure 6.2: The performance with respect to the batch size and transmission power. The Taylor expansion is not applied in the BP.

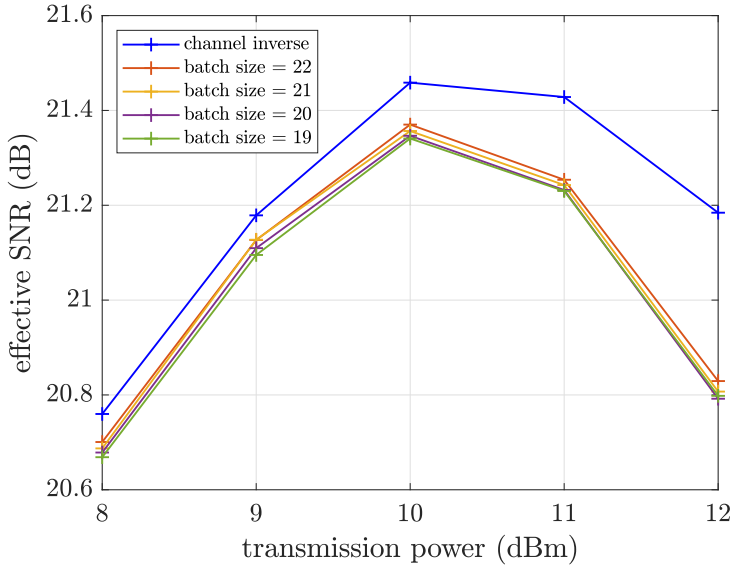


Figure 6.3: The performance with respect to the batch size ranging from 19 to 22 and the transmission power ranging from 8 dBm to 12 dBm. The Taylor expansion is not applied in the BP.

To have a clearer view of the parameterizations that have higher performance, Fig. 6.3 shows only the batch sizes between 19 and 22, for a transmission power between 8 dBm and 12 dBm.

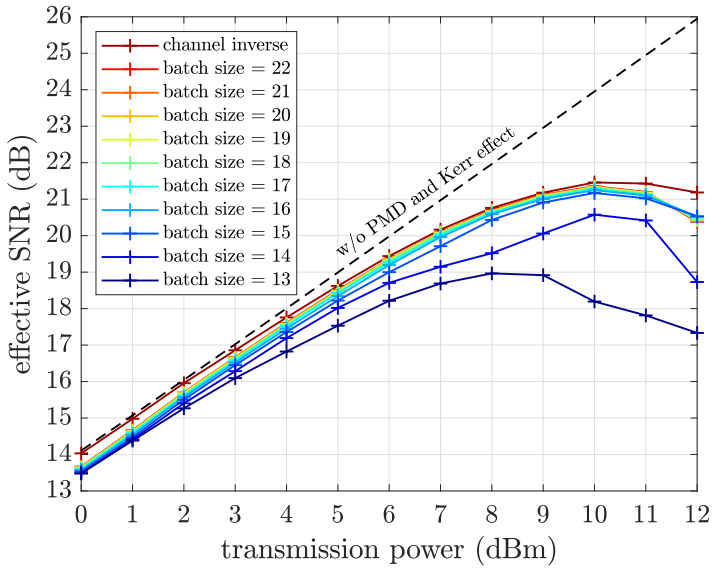


Figure 6.4: The performance with respect to the batch size and transmission power. The Taylor expansion is applied in the BP.

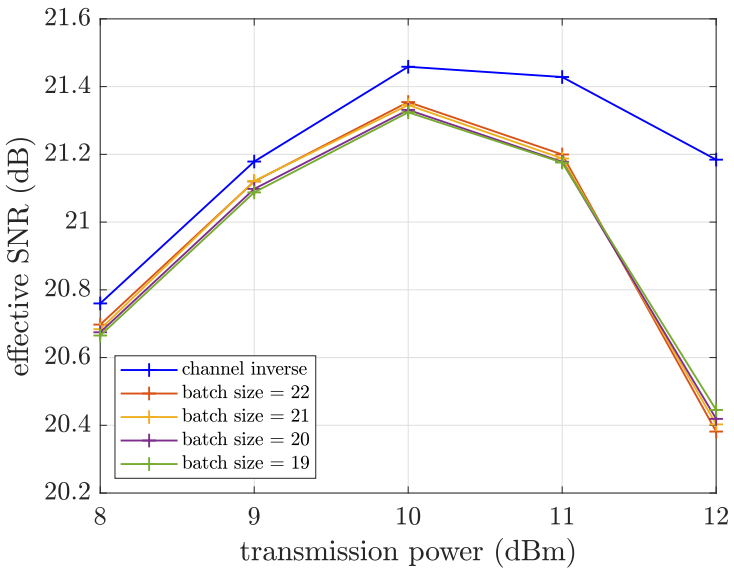


Figure 6.5: The performance with respect to the batch size and transmission power. The Taylor expansion is applied in the BP.

Then the Taylor expansion is applied to the BP of the equalizer, and the same test of performance is conducted on the modified equalizer model. Fig. 6.4 shows all the tested batch sizes and transmission power, and Fig. 6.5 shows only the batch sizes

ranging from 19 to 22 and the transmission power ranging from 8 dBm to 12 dBm. It can be seen that the performance still follows the same pattern that it increases as the batch size increases, and that there is always an optimal transmission power for each batch size. However, there is a degradation in the performance compared to the original equalizer when the transmission power is 12 dBm.

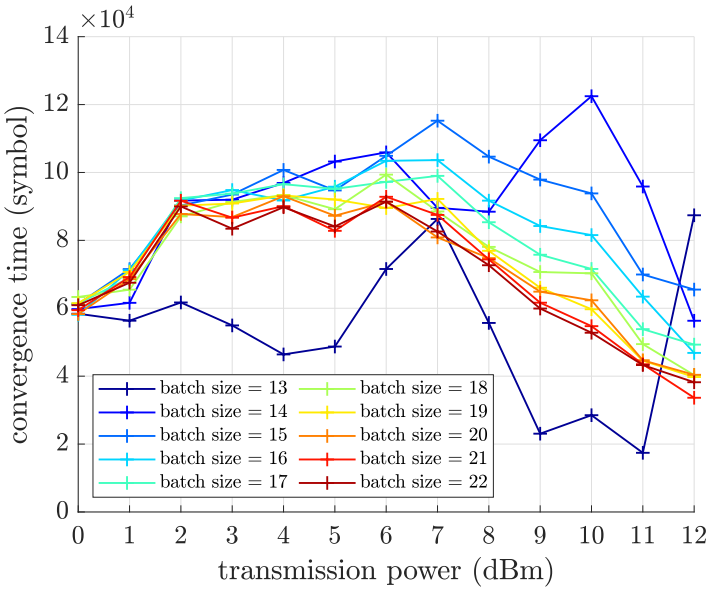


Figure 6.6: The convergence time with respect to the batch size and transmission power. The Taylor expansion is applied in the BP.

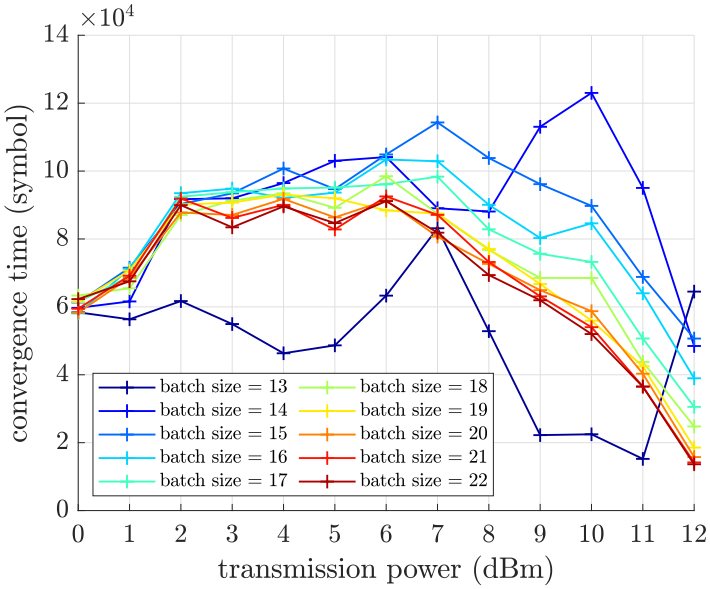


Figure 6.7: The convergence time with respect to the batch size and transmission power. The Taylor expansion is applied in the BP.

The convergence time is also measured in this test, and it is also averaged on all the

20 different fiber realizations. Fig. 6.6 shows the convergence time in relation to the batch size and the transmission power. The convergence time follows the pattern that it generally decreases as the batch size increases, except for batch sizes 13, 14, 19 and 20.

Fig. 6.7 shows the convergence time when the Taylor expansion is applied in the BP. It can be seen that, although the values are different from the equalizer without Taylor expansion, the general pattern that the convergence time decreases as the batch size increases still holds.

Based on all the test data obtained from the above tests, a batch size of 21 and a a transmission power of 10 dBm can be selected as the optimal operating point in terms of performance, convergence time and resource utilization. When the batch size is smaller than 26 symbols, the resource utilization increases as the batch size increases (see Section 7.1.2). Thus, bath size 22 is not chosen although it has the best performance and convergence time. However, this optimal point only works in the software models. When the equalizer is ported to FPGAs, there will be latency introduced to each layer when pipelining is used and there will be a performance deterioration caused by the fixed-point approximations.

6.4 Wordlengths in VHDL Hardware

In the software equalizer models, all the variables and constants are represented in double-precision floating-point format. However, the VHDL equalizer adopts fixed-point format with an alterable wordlength for each variable and constant. Normally, due to the limited DSP resources on an FPGA, the fixed-point format should have shorter wordlengths than the double format in software equalizers. Thus, there are two errors caused by the fixed-point representation:

- *fixed-point error*: the error caused by the fixed-point representation itself due to the difference between floating point format and fixed-point format.
- *rounding error*: the error caused by the shorter wordlengths in the VHDL equalizer compared to the 64-bit double format in the software equalizers.

In the VHDL equalizer, all the parameters/variables/constants have 5 different wordlengths which are denoted and summarized as below:

- *dat*: the wordlength of the symbol/sample, and the wordlength of the inputs and outputs of all BP layers except for the output of the Gradient layers.
- *ker*: the wordlength of the Kerr parameter $\bar{\gamma}$.
- *ang*: the wordlength of the Kerr angle ϕ in the Kerr FP layers.
- *wei*: the wordlength of each tap in the weights in the Linear FP layers, and the wordlength of the gradients in the Gradient layers.
- *tap*: the wordlength of the RRC taps in the MF FP layer and in the MF BP layer.

It should be noticed that the wordlength of the intermediate signals in each FP or BP layer may differ from the 5 wordlengths listed and they are actually combinations

of these wordlengths, but all the parameters/variables/constants listed above must have the assigned wordlengths.

There is a degradation in the performance and the convergence speed due to the fixed-point error and the rounding error. Since the FPGA resource utilization increases as the wordlengths increase, the wordlengths cannot be extended without limitation. Thus, tests should be carried out on the wordlengths to find the optimal set $\{dat, ker, ang, wei, tap\}$ within a certain range.

The tests are performed under the optimal operating point in the software implementation, i.e., a batch size of 21 symbols and a transmission power of 10 dBm. An important assumption should also be made that the optimal wordlengths do not change for different combinations of batch size and transmission power. This assumption ensures that testing the wordlengths at only one operating point is enough, thus avoiding excessive simulation runs.

In the experiments on the wordlengths, the *dat*, *ker*, *ang*, *wei* and *tap* are all varied independently from 12 to 16 with a step of 2 bits, which results in 243 tests in total. In each test, 20 simulations are performed on the 20 fiber realizations, and the performance and convergence time from the 20 simulations are both averaged, as in previous tests. We use VHDL logic simulation to conduct the tests and such simulation is performed in QuestaSim 2021.2. The number of symbols used in each simulation is 184,095 symbols, and the performance is calculated on the last 4,096 symbols.

Then the performance and the convergence speed are sorted in descending order respectively, and Table 6.1 shows the sets $\{dat, ker, ang, wei, tap\}$ with the highest performance, lowest performance, highest convergence speed, and lowest convergence speed. The set $\{14, 16, 12, 14, 12\}$ is chosen as the optimal operating point and will be used in all the following experiments, which is also shown in the table. A longer wordlength generally results in more hardware resource utilization, and thus a trade-off needs to be made on the performance, convergence time and resource utilization. The chosen operating point meets the criteria that it has relatively high performance, high convergence speed, and limited resource utilization.

Table 6.1: The results from the experiments on wordlengths in the VHDL equalizer.

Metric Type	Wordlengths	Perform. (Rank)	Conv. Time (Rank)
highest perform.	$\{14, 14, 12, 16, 12\}$	21.2456 dB (1)	5.3041×10^4 symbs (122)
lowest perform.	$\{12, 12, 14, 12, 16\}$	21.1840 dB (243)	5.2038×10^4 symbs (216)
highest conv. speed	$\{16, 14, 12, 12, 12\}$	21.2204 dB (165)	4.9091×10^4 symbs (1)
lowest conv. speed	$\{14, 12, 16, 16, 16\}$	21.2219 dB (138)	5.7920×10^4 symbs (243)
operating point	$\{14, 16, 12, 14, 12\}$	21.2443 dB (17)	5.4967×10^4 symbs (23)

It should be noticed that all the sets of wordlengths tested in this experiment perform very well and have relatively small difference in performance and convergence time, if comparing the highest and lowest values. This indicates that the set of shortest wordlengths $\{12, 12, 12, 12, 12\}$ is already located on the plateau of the function of

performance and the function of convergence time. Thus, more tests need to be done in the future to test wordlengths shorter than $\{12, 12, 12, 12, 12\}$.

6.5 Batch Size and Transmission Power in VHDL Equalizer

In Section 6.3, the optimal batch size and transmission power are surveyed in the software equalizer models. However, when porting the equalizer model to VHDL hardware, modifications are introduced so that the optimal batch size and transmission power might drift to a different point. In this section, the same experiment done in Section 6.3 is applied to the VHDL equalizer but using VHDL logic simulation in QuestaSim 2021.2. The batch size is varied from 19 to 22 with a step size of 1 and the transmission power is varied from 8 dBm to 12 dBm with a step size of 1 dBm. The wordlengths are fixed using the selected operating point $\{14, 16, 12, 14, 12\}$ in Section 6.4.

Fig. 6.8 shows the performance with respect to the transmission power when different batch sizes are used. Unlike the software model shown in Fig. 6.5, there is now a considerable decrease in the performance when the transmission power becomes 12 dBm regardless of the batch sizes. Fig. 6.9 shows more clearly about the transmission power from 8 dBm to 11 dBm.

Fig. 6.10 shows the convergence time with respect to the transmission power when different batch sizes are used. A trade-off should be made when selecting the optimal operating point and thus we choose the batch size 21 symbols and the transmission power 10 dBm.

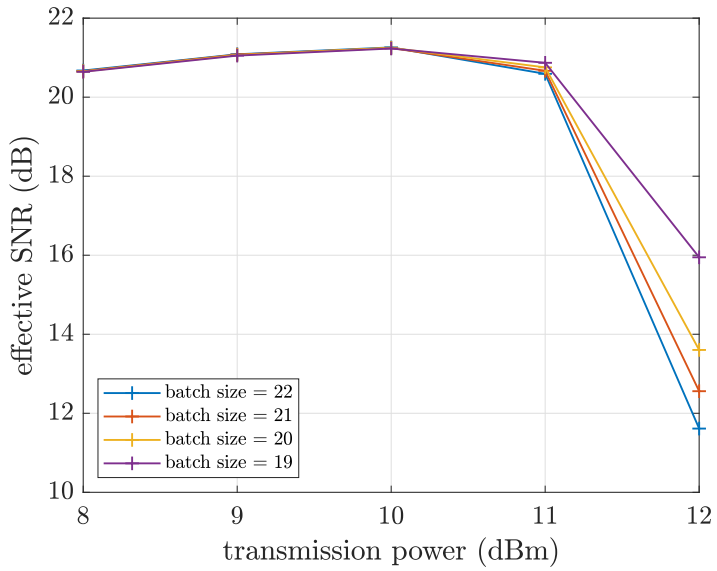


Figure 6.8: The performance with respect to the batch size and transmission power in the hardware implementation.

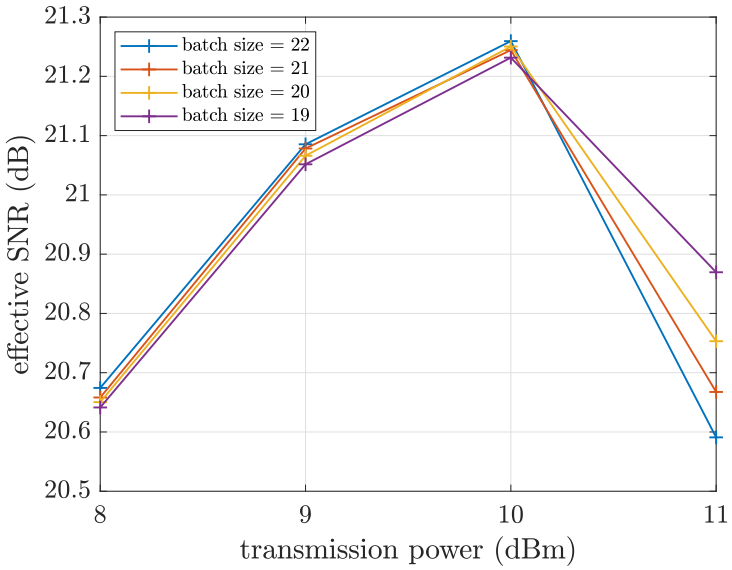


Figure 6.9: The zoomed-in view of the performance with respect to the batch size and transmission power in the VHDL equalizer.

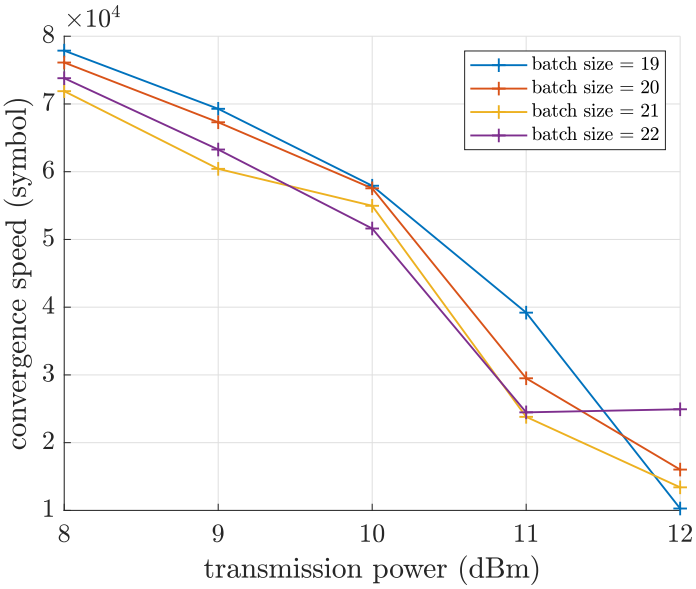


Figure 6.10: The convergence time with respect to the batch size and transmission power in the VHDL equalizer.

6.6 Software Equalizer and Hardware Equalizer Comparison

This section focuses on the comparison between the software model equalizer and the VHDL hardware equalizer based on the results obtained in the previous tests. Table 6.2 compares the chosen operating point in the software equalizer and that

in the VHDL equalizer, with the corresponding equalizer parameters. It can be concluded that the VHDL equalizer has a performance and a convergence time that are both very close to the original equalizer model, which means that our proposed digital implementation of adaptive nonlinear equalizer is successful.

Table 6.2: The comparison between the chosen operating point in the software equalizer model and that in the VHDL equalizer.

Equalizer Type	Batch Size	Trans. Power	Perform.	Conv. Time
channel inverse	–	10 dB	21.4586 dB	–
Matlab w/o Taylor expansion	21 symbs	10 dB	21.3566 dB	5.4732×10^4 symbs
Matlab w/ Taylor expansion	21 symbs	10 dB	21.3456 dB	5.4021×10^4 symbs
VHDL	21 symbs	10 dB	21.2443 dB	5.4967×10^4 symbs

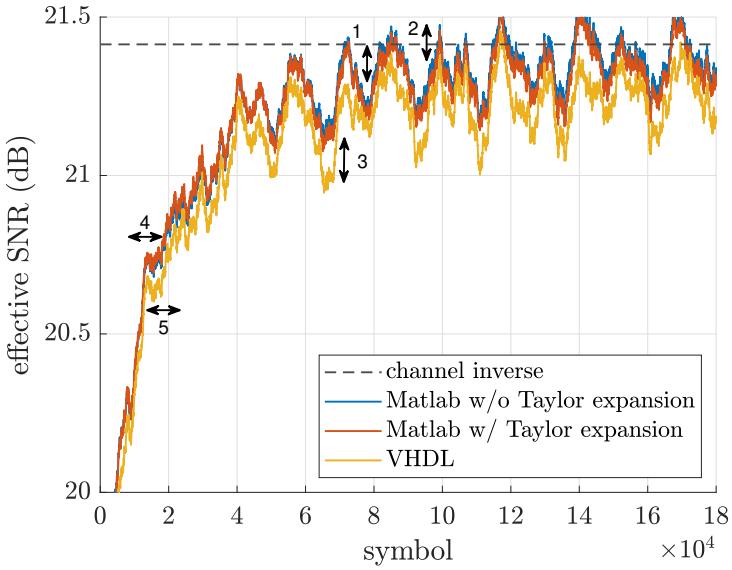


Figure 6.11: The comparison between the training curve of the original Matlab equalizer without Taylor expansion, the training curve of the modified Matlab equalizer with Taylor expansion, and the training curve of the VHDL implementation.

Fig. 6.11 shows the training curve of the original software equalizer, the training curve of the modified software equalizer with Taylor expansion in the BP, and the training curve of the VHDL equalizer. All the three curves are obtained on one fiber realization in the 20 realizations. Like previous tests, 184,095 symbols are used and the SNR is averaged on a window of 4,096 symbols in each test. The channel inverse here refers to the original equalizer model initialized with the inverse of the channel, which serves as the upper-bound of performance of the proposed equalizer model. It can be seen that there is a performance degradation noted as 1 between the ideal upper-bound performance and the original equalizer model, which is due to the limitation of the equalizer model itself such as the MF design and the number of MIMO-FIR filter taps in the Linear FP layers. There is a further

performance degradation between the original equalizer model and the modified model with Taylor expansion in the BP, which is noted as 2. This degradation is caused by the Taylor expansion introduced since the Taylor expansion is just an approximation of the original function. A third degradation is between the modified equalizer model and the VHDL equalizer, and it is noted as 3. Such degradation is caused by the fixed-point error and the rounding error.

Another point is that the convergence speeds are different in all the three cases. The first difference noted as 4 is that the model with Taylor expansion has a faster convergence than the original model. This difference is due to the introduced Taylor expansion resulting in lower performance, so fewer symbols are required for the final convergence. The second difference is that the VHDL equalizer has a lower convergence speed compared with the modified equalizer model, which is reflected in the fact that the VHDL equalizer requires more symbols to converge in Table 6.2. The training curve of the VHDL equalizer can be viewed as approximately a bottom-right shift of the training curve of the modified model. The lower performance of the VHDL equalizer cannot compensate for the right-shift of the training curve, and therefore the VHDL equalizer still needs more symbols to reach convergence. There are three contributors to such right-shift. Firstly the 18-sample latency between the input and output of the equalizer, as shown in Table 5.1. The second contributor is the 4-sample latency between the weight updates in two adjacent Gradient layers, as shown in Table 5.3. The third contributor is the initial latency in the weight update which can be obtained by comparing Table 5.2 and Table 5.3. It can be seen that the hardware equalizer has initial latencies $b + 29$, $b + 25$ and $b + 21$ in the weight update of each layer when compared to the software equalizer.

6.7 Instantaneous PMD Change

After the optimal operating point in the VHDL equalizer is decided, the adaptive capability of the VHDL equalizer can be tested. The first test is to introduce an instantaneous PMD change in the channel after the equalizer has already converged, and to verify that the equalizer can retrain itself afterwards. Table 6.3 shows the PMD angles before and after the instantaneous PMD change, and all the angles are randomly and uniformly generated on $[\pi, \pi]$. The channel with a transient PMD change is simulated using Matlab and the results are used as inputs to the VHDL equalizer. The VHDL logic simulation is then carried out on the VHDL equalizer.

Table 6.3: The transient PMD change in the channel.

PMD Angle $\theta^{(k)}$	Before	After
$\theta^{(0)}$	1.9775 rad	0.8316 rad
$\theta^{(1)}$	2.5497 rad	−2.5287 rad
$\theta^{(2)}$	−2.3437 rad	−1.3917 rad
$\theta^{(3)}$	2.5973 rad	0.2946 rad

As shown in Fig. 6.12, there is an initial convergence, and then a transient PMD

change happens at the 184,096th symbol when a total number of 364,095 symbols are used. The SNR is averaged on 4,096 symbols as before. It can be seen that the VHDL equalizer can quickly recover from the PMD change to its original performance level, just as the software models.

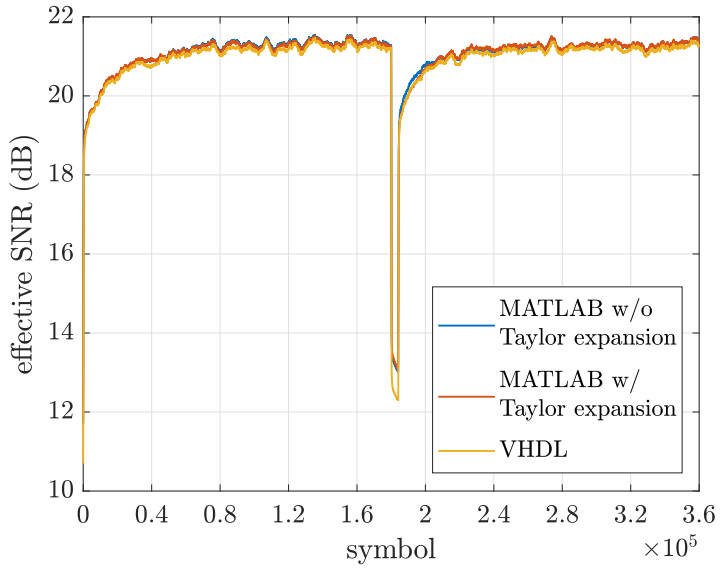


Figure 6.12: The initial training curve and its recovery when an instantaneous PMD change happens. The Matlab equalizers without and with Taylor expansion, and the VHDL equalizer are all shown.

6.8 Time Varying Channel

Due to the nonstationary nature of PMD, one important aspect of an equalizer's adaptive performance is compensating for the time-varying PMD in the channel [44]. The time-variation of PMD is normally a very slow process that takes hours or even days to accumulate a considerable change in real systems [44]. However, we use a much faster PMD variation for simulation purpose. The time-varying channel is simulated in Matlab and the results are used as input data to the VHDL equalizer. The VHDL logic simulation is then conducted on the VHDL equalizer to obtain the equalization outputs.

For our 3-section channel simulation, there are 4 rotation angles $\theta^{(k)}$ $(k = 0, 1, 2, 3)$ including the 3 angles in the 3 sections plus the output rotation angle. In the first test, the varying speeds of all the 4 rotation angles are kept the same and increased from 0 rad/s up to 1×10^7 rad/s. All the 4 rotation angles are initialized randomly and uniformly on $[-\pi, \pi]$, and the initial values are 0.8316 rad, -2.5287 rad, -1.3917 rad and 0.2946 rad from $\theta^{(0)}$ to $\theta^{(3)}$.

Fig. 6.13 shows the training curves from VHDL logic simulation when the varying speeds are 0 rad/s, 1×10^5 rad/s, 1×10^6 rad/s, 1×10^6 rad/s, and 1×10^7 rad/s with the same direction. A total number of 364,095 symbols are used in each test and the averaging window is 4,096 in SNR calculation. It can be seen that, when the

varying speed is 1×10^5 rad/s, the VHDL equalizer has basically the same level of performance and convergence speed as the stationary case 0 rad/s. For the speeds 1×10^6 rad/s and 1×10^7 rad/s, the equalizer cannot keep its original performance anymore and becomes unstable.

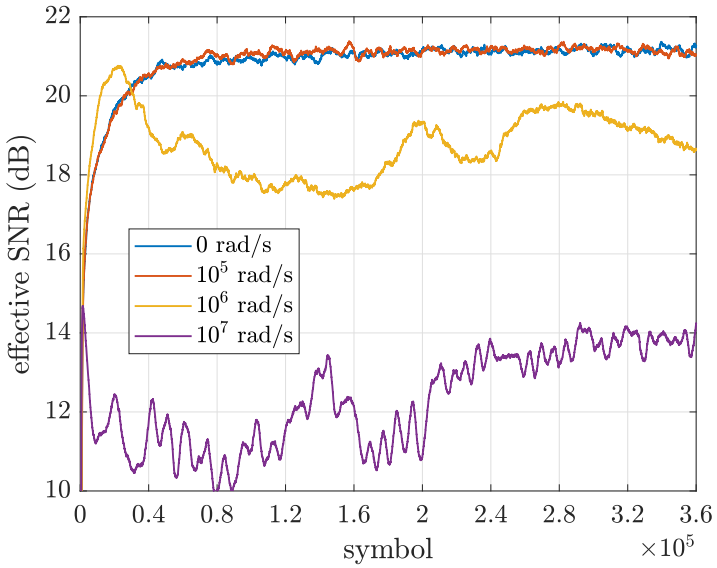


Figure 6.13: The training curves from VHDL logic simulation when different varying speeds of the rotation angles are adopted. All the 4 rotation angles in the channel have the same varying speed and direction.

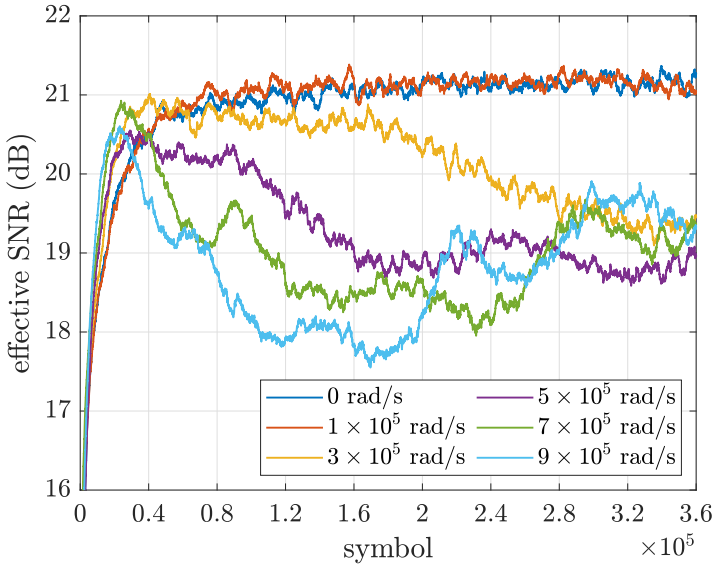


Figure 6.14: The training curves from VHDL simulation when the varying speeds vary between 1×10^5 rad/s and 1×10^6 rad/s.

In the second test, the varying speeds between 1×10^5 rad/s and 1×10^6 rad/s are further tested to acquire an upper bound of the varying speed, and the step is set to

2×10^5 rad/s. Like before, the number of symbols used is also 364,095 in each test and the averaging window of the performance is 4,096. Fig. 6.14 shows the results from VHDL logic simulation. It can be seen that the equalizer becomes unstable when the varying speed reaches 3×10^5 rad/s.

In the third test, the varying speeds of the 4 rotation angles become different and Fig. 6.15 shows the results. The varying speeds of the rotation angles from $\theta^{(0)}$ to the $\theta^{(3)}$ are respectively shown in a set. All the rotation angles have the same direction of varying, and all the initial angles are the same as the first test. The number of symbols in each test is 364,095 and the effective SNR is averaged on a window of 4,096 symbols. We can conclude from the results shown in Fig. 6.14 and Fig. 6.15 that when the VHDL equalizer contains 3 layers, to ensure that the equalizer remains stable, the upper limit on the varying speed of each rotation angle is 1×10^5 rad/s.

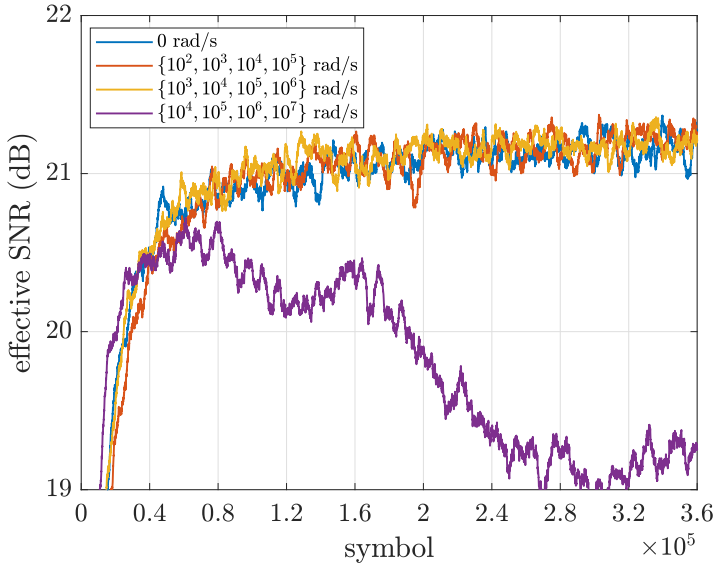


Figure 6.15: The training curves from VHDL simulation when the varying speeds of all rotation angles are different.

6.9 FPGA Implementation

The 3-layer equalizer with the batch size 21 and the wordlengths $\{14, 16, 12, 14, 12\}$ is implemented on a Xilinx Virtex-7 VC709 FPGA development board, and the FPGA model number is Virtex-7 XC7VX690T. The synthesis and analysis tool is Vivado Design Suite 2020.2. The clock frequency is set to 50 MHz. The resource utilization after Vivado implementation is shown in Table 6.4, where the "FP" and "BP" entries refer to the resource utilization in the entire FP and BP, respectively. The "Total" entry refers to the resource utilization of the entire equalizer. The sum of the utilized slice LUTs in the FP and BP is not equal to the total utilization of slice LUTs in the whole system. So do the slice registers. This is because the system-level control logic also consumes LUTs and slice registers, which is not shown as a

separate entry in the table. It should be noticed that the equalizer does not utilize any block random access memory (BRAM) since the whole VHDL equalizer is fully pipelined and runs strictly at the speed of 2 samples per clock cycle. As a result, the data samples do not need to be stored temporarily in any specific on-board memory.

Table 6.4: The Break-down resource utilization on Xilinx VC709 with 50 MHz clock.

Module	Slice LUTs	Slice Registers	DSPs	F7 Muxes	F8 Muxes	BRAM
Kerr FP 0	3,893 (0.90%)	282 (0.03%)	26 (0.72%)	866 (0.40%)	166 (0.15%)	0 (0.00%)
Kerr FP 1	3,893 (0.90%)	282 (0.03%)	26 (0.72%)	866 (0.40%)	166 (0.15%)	0 (0.00%)
Kerr FP 2	3,899 (0.90%)	310 (0.04%)	26 (0.72%)	866 (0.40%)	166 (0.15%)	0 (0.00%)
Linear FP 0	492 (0.11%)	112 (0.01%)	96 (2.67%)	0 (0.00%)	0 (0.00%)	0 (0.00%)
Linear FP 1	492 (0.11%)	112 (0.01%)	96 (2.67%)	0 (0.00%)	0 (0.00%)	0 (0.00%)
Linear FP 2	492 (0.11%)	112 (0.01%)	96 (2.67%)	0 (0.00%)	0 (0.00%)	0 (0.00%)
MF FP	7,028 (1.62%)	7,172 (0.83%)	0 (0.00%)	0 (0.00%)	0 (0.00%)	0 (0.00%)
Loss BP	2,920 (0.67%)	62 (<0.01%)	0 (0.00%)	0 (0.00%)	0 (0.00%)	0 (0.00%)
MF BP	5,724 (1.32%)	1,078 (0.12%)	132 (3.67%)	188 (0.09%)	94 (0.09%)	0 (0.00%)
Linear BP 0	2,439 (0.56%)	1,142 (0.13%)	160 (4.44%)	0 (0.00%)	0 (0.00%)	0 (0.00%)
Linear BP 1	4,591 (1.06%)	1,854 (0.21%)	320 (8.89%)	0 (0.00%)	0 (0.00%)	0 (0.00%)
Kerr BP 0	13,643 (3.15%)	338 (0.04%)	372 (10.33%)	0 (0.00%)	0 (0.00%)	0 (0.00%)
Kerr BP 1	23,488 (5.42%)	562 (0.06%)	620 (17.22%)	0 (0.00%)	0 (0.00%)	0 (0.00%)
Gradient 0	9,478 (2.19%)	3,327 (0.38%)	100 (2.78%)	2,128 (0.98%)	336 (0.31%)	0 (0.00%)
Gradient 1	15,090 (3.48%)	3,375 (0.39%)	300 (8.33%)	1,554 (0.72%)	364 (0.34%)	0 (0.00%)
Gradient 2	20,349 (4.70%)	3,421 (0.39%)	500 (13.89%)	2,506 (1.16%)	616 (0.57%)	0 (0.00%)
FP	20,189 (4.66%)	8,382 (0.97%)	366 (10.17%)	2,598 (1.20%)	498 (0.46%)	0 (0.00%)
BP	97,722 (22.56%)	15,159 (1.75%)	2,504 (69.56%)	6,376 (2.94%)	1,410 (1.30%)	0 (0.00%)
Total	120,423 (27.80%)	32,935 (3.80%)	2,870 (79.72%)	8,974 (4.14%)	1,908 (1.76%)	0 (0.00%)

Table 6.5: The summary of the resource utilization on Xilinx VC709 with 50 MHz clock.

Module	Slice LUTs	DSPs
Kerr FPs	11,685 (2.70%)	78 (2.17%)
Linear FPs	1,476 (0.34%)	288 (8.00%)
MF FP	7,028 (1.62%)	0 (0.00%)
Loss BP	2,920 (0.67%)	0 (0.00%)
MF BP	5,724 (1.32%)	132 (3.67%)
Linear BPs	7,030 (1.62%)	480 (13.33%)
Kerr BPs	37,131 (8.57%)	992 (27.56%)
Gradients	44,917 (10.37%)	900 (25.00%)
FP	20,189 (4.66%)	366 (10.17%)
BP	97,722 (22.56%)	2,504 (69.56%)
Total	120,423 (27.80%)	2,870 (79.72%)

Table 6.5 summarizes the utilization of the resources of interest to us. The repeated items are summarized into one item in the table. It can be seen that the BP utilizes more resources than the FP, which indicates that the on-chip training brings a significant resource overhead to the equalizer. The Gradient layers utilize the most slice LUTs, and the Kerr BP layers utilize the most DSPs.

It can also be seen that the bottleneck of hardware utilization is the DSP. This 3-layer equalizer utilizes 79.72% of the total DSPs, which still leaves enough room for further adding the PMD-Kerr emulator from [38] on the same FPGA in the future. By implementing both the PMD-Kerr emulator and the equalizer on the same FPGA, an entire test platform running on board can be formed in the future.

7

Discussion and Future Work

In this chapter, we focus on some specific aspects of this thesis project that we believe can be improved, and possible future work can be based on them.

7.1 Batch Size and Number of Layers

This section discusses the hardware structure of each BP layer when more than 3 layers are used and a batch size larger than 22 symbols is adopted. These extensions can be implemented in future work to facilitate an equalizer with more than 3 layer implemented on a larger FPGA, which can compensate for more realistic PMD-Kerr effects in the channel.

7.1.1 Hardware System Structure

The same hardware structure shown in Fig. 5.1 can still be applied when more than 3 layers are used in the equalizer. Fig. 7.1 shows our proposed general hardware structure with more than 3 layers. There are L layers in total, and layer 1 to layer $L-2$ are duplications. The "layer" here refers to the general layer including one Kerr FP,one Linear FP, one Linear BP, one Kerr BP, one Gradient, one Kerr SR and one Gradient SR. The layer 0 and layer $L-1$ have different structures which are also shown. We also illustrate the layer index k in each specific layer (Linear FP, Kerr FP, Linear BP, etc.) calculated from L. The port sizes are denoted in parentheses in Fig. 7.1.

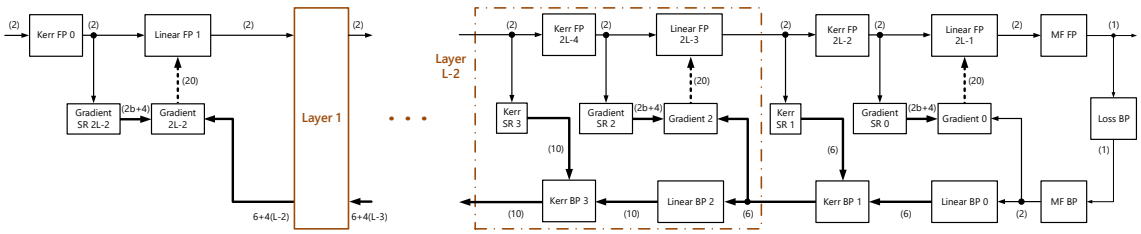


Figure 7.1: The proposed general hardware architecture of the entire equalizer when the number of layers $L > 3$.

Table 7.1 shows the latency in each layer when $L > 3$. It can be seen that only the latency in the Gradient SR component and that in the Kerr SR component grows

linearly with the layer index l. This means that more LUT resources will be utilized in the Gradient SR with $k = 2L - 2l - 2$ and in the Kerr SR with $k = 2L - 2l - 1$ (i.e, the Gradient SR and the Kerr SR in Layer l), as l increases. It can also be seen that the the latency of the Gradient SR, Kerr SR, Gradient layer and MF BP layer grows linearly with the batch size b.

Table 7.1: The latency in each layer in the proposed general hardware structure when the number of layers $L > 3$.

layer/component	latency (clock cycles)
Kerr FP	4
Linear FP	1
MF FP	3
Loss BP	1
MF BP	$b + 1$
Linear BP	2
Kerr BP	2
Gradient	b
Gradient SR	$b + 6 + 9l$
Kerr SR	$b + 12 + 9l$

7.1.2 MF BP Layer

When a 51-tap MF FP layer is used, the architecture of the MF BP layer shown in Fig. 5.7 does not work for $b > 25$. Thus, a new hardware architecture can be designed and implemented in the future, which targets at equalizers with a batch size larger than 25 symbols. Fig. 7.2 shows such an architecture. It can be seen that the number of parallel operators is a constant 51 and has nothing to do with the batch size any more.

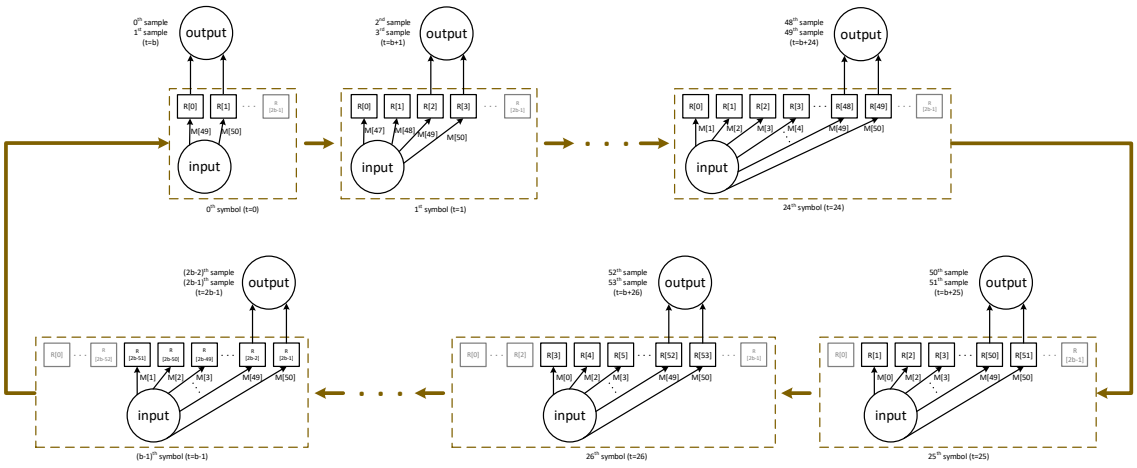


Figure 7.2: The hardware architecture of the MF BP layer when $b > 25$.

From $t = 0$ to $t = 24$, two more inputs/operators/registers are activated at each clock cycle. From $t = 25$ to $t = 2b - 1$, there are constant 51 inputs/operators/registers

activated and not increasing as t increases, while the 51-sample activation window shifts with a step of 2 samples per clock cycle.

7.1.3 Linear BP Layer

Although we have only implemented a 3-layer equalizer including two Linear BP layers $k = 0$ and $k = 2$, the same introduced hardware architecture of Linear BP layer in Section 5.3.3 can be applied for the equalizer with more than 3 layers. In this section we will discuss how this architecture can be applied to the Linear BP layers with $k \geq 4$ in 4 different cases:

1). The layers with $k < 2\lfloor\frac{b-2}{2}\rfloor$[1]:

- The input size should be $2 + 2k$ samples and the output size should be $6 + 2k$ samples.
- There are $6 + 2k$ basic operators implemented in each layer which are divided into two types, $2 + 2k$ $Op1$ operators with a length of 5 samples and 4 $Op2$ operators with lengths of 1, 2, 3 or 4 samples.
- From $t = 0$ to $t = k - 1$, the input ports, operators and output ports all stay idle.
- At $t = k$ and $t = k + 1$, only two input ports are activated and the input samples are stored in the SR, while the outputs and the operators still stay idle.
- At $t = k+2$, two input ports are activated and the registers start shifting with a step size of two samples. Two $Op1$ operators as well as two output ports are activated. The registers continue shifting and two results are continually being forwarded to two output ports at each cycle until $t = b - 2$.
- At $t = b-1$, all the $2+2k$ input ports and all the $6+2k$ operators are activated, and there are $6 + 2k$ results generated. These $6 + 2k$ samples altogether are forwarded to the $6+2k$ output ports at the same clock cycle. Then this batch ends and the state machine returns to the first state to start accepting the next batch.

2). The layer with $k = b - 2$ if b is even,:

- The input size should be $2b - 2$ samples and the output size should be $2b$ samples.
- The number of basic operators is a constant $2b$ including $2b-4$ $Op1$ operators with a length of 5 samples and 4 $Op2$ operators with lengths of 1, 2, 3, or 4 samples.
- From $t = 0$ to $t = b - 3$, the input ports, operators and output ports all stay idle.
- At $t = b - 2$, only two input ports are activated and the input samples are stored in a register, while the outputs and the operators still stay idle.

[1]The symbol "$\lfloor\ \rfloor$" represents the floor operation.

- At $t = b - 1$, all the $2b - 2$ inputs and all the $2b$ operators are activated, and there are $2b$ results generated. These $2b$ samples altogether are forwarded to the $2b$ output ports at the same clock cycle. Then this batch ends and the state machine returns to the first state to start accepting the next batch.

3). The layer with $k = b - 3$ if b is odd:

- The input size should be $2b - 4$ samples and the output size should be $2b$ samples.
- The number of basic operators is a constant $2b$ including $2b - 4$ $Op1$ operators with a length of 5 samples and 4 $Op2$ operators with lengths of 1, 2, 3 or 4 samples.
- From $t = 0$ to $t = b - 4$, the input ports, operators and output ports all stay idle.
- At $t = b - 3$ and $t = b - 2$, only two input ports are activated and the input samples are stored in a register, while the outputs and the operators still stay idle.
- At $t = b - 1$, all the $2b - 4$ inputs and all the $2b$ operators are activated, and there are $2b$ results generated. These $2b$ samples altogether are sent to the $2b$ output ports at the same clock cycle. Then this batch ends and the state machine returns to the first state to start accepting the next batch.

4). The layers with $k > b - 2$ if b is even, or the layers with $k > b - 3$ if b is odd:

- The input size should be $2b$ samples and the output size should be $2b$ samples.
- The number of basic operators is a constant $2b$ including $2b - 4$ $Op1$ operators with a length of 5 samples and 4 $Op2$ operators with lengths of 1, 2, 3, or 4 samples.
- From $t = 0$ to $t = b - 2$, the input ports, operators and output ports all stay idle.
- At $t = b - 1$, all the $2b$ inputs and all the $2b$ operators are activated, and there are $2b$ results generated. These $2b$ samples altogether are sent to the $2b$ output ports at the same clock cycle. Then this batch ends and the state machine returns to the first state to start accepting the next batch.

7.1.4 Gradient Layer

This thesis work only implements the Gradient layers $k = 0$, 2 and 4. However, the same hardware structure shown in Section 5.3.5 can still be applied to the Gradient layers $k > 4$ if an equalizer with more than 3 layers will be designed in the future. There are two different cases of the Gradient layer when $k > 4$ and the following describes the procedure in each case:

1). The layers with $4 < k \leq 2\lfloor \frac{b-2}{2} \rfloor$:

- The input size of $\bar{xi}$, $\bar{xq}$, $\bar{yi}$, or $\bar{yq}$ should be fixed as $2b$ samples, and the input size of x_i, x_q, y_i, or y_q should still be $2b + 4$ samples.

- The number of multiplier-adders is $2k + 2$.
- From $t = 0$ to $t = k - 1$, the input ports, operators and output ports all stay idle.
- From $t = k$ to $t = b - 2$, 4 input ports and 4 operators are activated while the others all stay idle.
- At $t = b - 1$, all the $2k + 2$ inputs and all the $2k + 2$ multiplier-adders are activated, and the results $H_{xx}[d]$, $H_{yx}[d]$, $H_{xy}[d]$ and $H_{yy}[d]$ are calculated. Then this batch ends and the state machine returns to the first state to start accepting the next batch.

2). The layers with $k > 2\lfloor\frac{b-2}{2}\rfloor$:

- The input size of $\bar{xi}$, $\bar{xq}$, $\bar{yi}$, or $\bar{yq}$ is $2k + 2$ samples, and the input size of x_i, x_q, y_i, or y_q is $2b + 4$ samples.
- The number of multiplier-adders is fixed as $2b$.
- From $t = 0$ to $t = b - 2$, the input ports, operators and output ports all stay idle.
- At $t = b - 1$, all the $2b$ inputs and all the $2b$ multiplier-adders are activated, and the results $H_{xx}[d]$, $H_{yx}[d]$, $H_{xy}[d]$ and $H_{yy}[d]$ are calculated. Then this batch ends and the state machine returns to the first state to start accepting the next batch.

7.2 Other Hyperparamters

This section discusses possible future work about all the system-level hyperparameters apart from the batch size and the number of layers already discussed in Section 7.1.

7.2.1 Wordlengths

In the experiments about fixed-point wordlengths in Section 6.4, we only focus on the wordlengths 12, 14 and 16 bits for each item in $\{dat, ker, ang, wei, tap\}$. However, this range falls on the plateau of the function of performance and the function of convergence time, and the results turn out to be very close to each other. In future work, tests on wordlengths below 12 should be carried out to further explorer the relationship between wordlengths, performance and convergence time. The future experiment can be divided into two steps. In the first step, we keep the wordlengths the same for $\{dat, ker, ang, wei, tap\}$ and decrease them from 11 to lower in 1-bit steps. In the second step, all the 5 wordlengths in $\{dat, ker, ang, wei, tap\}$ can be varied independently in steps of 1 bit, and this variation can be done around a point selected in the first step.

7.2.2 Learning Rate and Optimizer

In this thesis, a grid search is carried out on transmission power and batch size while the learning rate is not varied at all. Therefore, the variation of learning rate can

be included in the experiments in future. In other words, transmission power, batch size, number of layers and learning rate can all be varied independently to find the optimal combination of these 4 parameters.

The fixed learning rate used in this thesis project is 32, as shown in Table 4.2, which is larger than the commonly used learning rates in neural networks. This is because the loss used in this thesis project is not normalized as shown in Eq. (4.4). The calculation in the Loss BP layer shown in Eq. (4.7) is consequently also not normalized. The reason for removing the normalization is to save hardware resource utilization. As a result, a large learning rate is adopted to compensate for the unnormalized loss. However, whether the above method has an impact on the equalizer performance and converge speed is not studied in this thesis project. In future work, normalization can be added to the Loss BP layer to thoroughly study how normalization can affect the equalizer performance/convergence speed and how it can impact hardware resource utilization.

The optimizer used in this thesis project is SGD which is very simple in terms of computational complexity and hardware resource utilization. Other types of optimizers may also be possibly used in future work, which may have an impact on both the equalizer performance/convergence speed and the hardware resource utilization. The type of optimizer can also be added to the grid search on hyperparameters.

7.2.3 RRC Filter

The RRC pulse-shaping filter used in the channel simulation in this thesis adopts a roll-off factor of 0.1 and a total number of 51 RRC taps. The same parameters are used for the MF FP layer and the MF BP layer in the equalizer accordingly. These two parameters can have an impact on the performance and convergence speed of the equalizer. It should also be noticed that the number of RRC taps can also have an impact on the hardware resource utilization as discussed in Section 7.1.2. Therefore, the roll-off factor and the numbers of RRC taps should be studied and varied in future work. Possible variation of the roll-off factor and FIR filter taps can also be added to the grid search on hyperparameters.

7.2.4 MIMO-FIR Filter

The number of taps in the MIMO-FIR filter used in this thesis is fixed as 5. A MIMO-FIR filter with more taps can have better compensation of the DGD in the PMD effect but also results in larger resource utilization. In future work, the number of MIMO-FIR filter taps should be studied and varied to be added to the grid search on hyperparameters.

7.3 Loss BP Layer

The VHDL implementation of the Loss BP layer in this thesis project is parameterized as shown in Section 5.3.1. More specifically, different wordlengths and batch sizes can result in different hardware implementation in Fig. 5.6, and VHDL Generics

are used to automatically decide the hardware implementation based on the batch size and wordlengths. However, such VHDL Generics only support the batch size $16 \leq b \leq 22$. Thus, future work is needed to expand the range of the batch size in the VHDL Generics.

Moreover, the division approximated by right shifts can have an impact on both the equalizer performance/convergence speed and the hardware resource utilization. Generally, more shift right operators can have a better approximation of the division but can result in more hardware resource utilization. Thus, the relationship between the number of right shift operators and the equalizer performance/convergence speed and hardware resource utilization should be studied in future work. Moreover, we should also implement the division directly by DSPs in the future and compare its resulting performance, convergence speed and hardware resource utilization to the right shift solution.

7.4 Resource Utilization

Only one FGPA implementation is carried out in this thesis project. In other words, the resource utilization of only one set of hyperparameters is tested on an FPGA. Thus, future work should include the study of the relationship between different hyperparameter values and FPGA resource utilization. The FPGA resource utilization should be obtained for each set of the hyperparameters. Moreover, a trade-off between resource utilization and equalizer performance/convergence speed should also be taken into consideration in future work.

7.5 Evaluation Platform

This thesis project adopts a common research method of FPGA-based equalizers, namely Matlab-VHDL co-simulation. In Matlab-VHDL co-simulation, the test samples are generated using Matlab simulation, sent to the VHDL simulation of the equalizer, and finally sent back from the VHDL simulation to Matlab to process the results. This conventional way requires a complex procedure, and the conversion between the data types of matlab and those of VHDL should also be taken into consideration. Usually, a large number of test samples are needed to test an equalizer's performance and convergence speed, which can cause long runtime in both Matlab and VHDL simulation. Moreover, the time-varying PMD effect can bring further challenges to the Matlab-based test data generation.

The work [37] provides VHDL code for FPGA-based emulation of optical fiber communication systems, including pseudo random number generator, modulator, RRC filter, AWGN generator, PMD emulator and demodulator. This work provides a way to move the simulation of the transmitter and the PMD channel to hardware, forming an FPGA-based PMD effect emulator. Moreover, another work [38] further extends the PMD effect emulator in [37] by implementing and adding a Kerr Emulator to the original FPGA-based PMD emulator, which is called an FPGA-based PMD-Kerr effect emulator.

In future work, the PMD-Kerr emulator from [38] and the VHDL equalizer from this thesis can be implemented together on the same FPGA, forming an entire evaluation platform according to the Fiber-on-Chip approach [45]. For data recording and analysis, a BER-based error counter and a data recorder proposed in [46] can be further added to this evaluation platform. Moreover, the SNR calculation can also be ported to FPGA in the future and we have already adopted an averaing window of 4,096 in thesis to make the SNR on-board calculation feasible. This evaluation platform does not need any Matlab-based test data generator since the test samples are also generated on the same FPGA. The data after equalization are demodulated and the errors are also counted on board. The computer is only used to receive the error information from the FPGA and no calculations need to be performed on the computer. Such an evaluation platform will drastically decrease the runtime of testing an FPGA-based equalizer. Meanwhile, the performance of the FPGA-based PMD-Kerr emulator, i.e., the accuracy of the PMD-Kerr effect emulation, should be kept high enough to generate valid test data for the equalizer on the same FPGA.

8

Conclusion

In this thesis, an ML-based adaptive nonlinear equalizer is implemented on an FPGA, and the equalizer can compensate for the PMD-Kerr effect in the fiber-optical channel in real-time. The training of the equalizer is implemented on the same FPGA and works in parallel with the inference, and thus the on-chip training is realized. The on-chip training enables the equalizer to be adaptive to a time-varying or transient PMD change.

In this thesis, we first evaluate the original equalizer model with only the FP and simulate it in Tensorflow. Secondly, the BP model is deduced, and the equalizer model with the BP manually implemented is simulated in Matlab. Then the Taylor expansion is introduced into the BP, and we simulate the modified model again in Matlab. Then a hardware model of the modified equalizer is proposed and implemented as digital circuits using VHDL, and we use VHDL logic simulation to test the VHDL equalizer. Finally, the VHDL equalizer is synthesized and implemented on a Xilinx VC709 FPGA board.

Thorough experiments are carried out in this thesis to test the performance and convergence speed of the VHDL equalizer. Results show that the VHDL equalizer has a performance close to the theoretical upper limit of the original equalizer model and a convergence time slightly longer than the original equalizer model. Experiment results also show that the converged VHDL equalizer can retrain itself to reach convergence again when a transient PMD change occurs. When the PMD effect in the channel is time-varying, the VHDL equalizer remains stable until the varying speed per each PMD rotation angle exceeds 1×10^5 rad/s in the same varying direction.

Bibliography

[1] J. Gordon and H. Kogelnik, "PMD fundamentals: Polarization mode dispersion in optical fibers," *Proceedings of the National Academy of Sciences*, vol. 97, no. 9, pp. 4541–4550, 2000.

[2] R. Stolen and A. Ashkin, "Optical Kerr effect in glass waveguide," *Applied Physics Letters*, vol. 22, no. 6, pp. 294–296, 1973.

[3] A. Yang, X. Li, A. Xu, and D. Wu, "Combined impacts of group velocity dispersion, Kerr effect and polarization mode dispersion in optical fibers," *Optics communications*, vol. 214, no. 1-6, pp. 133–139, 2002.

[4] C. Menyuk and B. Marks, "Interaction of polarization mode dispersion and nonlinearity in optical fiber transmission systems," *Journal of Lightwave Technology*, vol. 24, no. 7, pp. 2806–2826, 2006.

[5] S. J. Savory, "Digital coherent optical receivers: Algorithms and subsystems," *IEEE Journal of Selected Topics in Quantum Electronics*, vol. 16, no. 5, pp. 1164–1179, 2010.

[6] R. M. Bütler, C. Häger, H. D. Pfister, G. Liga, and A. Alvarado, "Model-Based machine learning for joint digital backpropagation and PMD compensation," *Journal of Lightwave Technology*, vol. 39, no. 4, pp. 949–959, 2020.

[7] D. E. Crivelli, M. R. Hueda, H. S. Carrer, M. del Barco, R. R. López, P. Gianni, J. Finochietto, N. Swenson, P. Voois, and O. E. Agazzi, "Architecture of a single-chip 50 Gb/s DP-QPSK/BPSK transceiver with electronic dispersion compensation for coherent optical channels," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 61, no. 4, pp. 1012–1025, 2014.

[8] H. F. Haunstein, K. Sticht, A. Dittrich, W. Sauer-Greff, and R. Urbansky, "Design of near optimum electrical equalizers for optical transmission in the presence of PMD," in *Optical Fiber Communication Conference and International Conference on Quantum Information*, 2001, paper WAA4.

[9] L. E. Nelson, S. L. Woodward, S. Foo, X. Zhou, M. D. Feuer, D. Hanson, D. McGhan, H. Sun, M. Moyer, M. O. Sullivan, and P. D. Magill, "Performance of a 46-Gbps dual-polarization QPSK transceiver with real-time coherent equalization over high PMD fiber," *J. Lightwave Technol.*, vol. 27, no. 3, pp. 158–167, Feb. 2009.

[10] H. Sun, K.-T. Wu, and K. Roberts, "Real-Time measurements of a 40 Gb/s coherent system," *Optics Express*, vol. 16, no. 2, pp. 873–879, 2008.

[11] B. C. Thomsen, R. Maher, D. S. Millar, and S. J. Savory, "Burst mode receiver for 112 Gb/s DP-QPSK with parallel DSP," *Optics Express*, vol. 19, no. 26, pp. B770–B776, 2011.

[12] T. Tanimura, Y. Aoki, H. Nakashima, T. Hoshida, J. Li, Z. Tao, and J. C. Rasmussen, "FPGA-Based 112Gb/s coherent DP-QPSK receiver and multi-stage PMD-PDL emulator for fast evaluation of digital signal processing algorithms," in *36th European Conference and Exhibition on Optical Communication*, 2010, paper Tu.S.A.3.

[13] E. Dutisseuil, J.-M. Tanguy, A. Voicila, J. Renaudier, and G. Charlet, "21 Gb/s polarization switched-QPSK real-time coherent FPGA-based receiver," in *2013 Optical Fiber Communication Conference and Exposition and the National Fiber Optic Engineers Conference (OFC/NFOEC)*, 2013, paper OW1E.2.

[14] N. Kaneda and A. Leven, "Coherent polarization-division-multiplexed QPSK receiver with fractionally spaced CMA for PMD compensation," *IEEE Photonics technology letters*, vol. 21, no. 4, pp. 203–205, 2008.

[15] E. Dutisseuil, J.-M. Tanguy, A. Voicila, R. Laube, F. Bore, H. Takeugming, F. de Dinechin, F. Cérou, and G. Charlet, "34 Gb/s PDM-QPSK coherent receiver using SiGe ADCs and a single FPGA for digital signal processing," in *Optical Fiber Communication Conference*, 2012, paper OM3H–7.

[16] F. Musumeci, C. Rottondi, A. Nag, I. Macaluso, D. Zibar, M. Ruffini, and M. Tornatore, "An overview on application of machine learning techniques in optical networks," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 2, pp. 1383–1408, 2018.

[17] E. Ip and J. M. Kahn, "Compensation of dispersion and nonlinear impairments using digital backpropagation," *Journal of Lightwave Technology*, vol. 26, no. 20, pp. 3416–3425, 2008.

[18] D. Rafique, M. Mussolin, M. Forzati, J. Mårtensson, M. N. Chugtai, and A. D. Ellis, "Compensation of intra-channel nonlinear fibre impairments using simplified digital back-propagation algorithm," *Optics Express*, vol. 19, no. 10, pp. 9453–9460, 2011.

[19] T. S. R. Shen and A. P. T. Lau, "Fiber nonlinearity compensation using extreme learning machine for DSP-based coherent communication systems," in *16th Opto-Electronics and Communications Conference*, 2011, pp. 816–817.

[20] M. A. Jarajreh, E. Giacoumidis, I. Aldaya, S. T. Le, A. Tsokanos, Z. Ghassemlooy, and N. J. Doran, "Artificial neural network nonlinear equalizer for coherent optical OFDM," *IEEE Photonics Technology Letters*, vol. 27, no. 4, pp. 387–390, 2014.

[21] S. Gaiarin, X. Pang, O. Ozolins, R. T. Jones, E. P. Da Silva, R. Schatz, U. Westergren, S. Popov, G. Jacobsen, and D. Zibar, "High speed PAM-8 optical interconnects with digital equalization based on neural network," in *Asia Communications and Photonics Conference*, 2016, paper AS1C–1.

[22] S. T. Ahmad and K. P. Kumar, "Radial basis function neural network nonlinear equalizer for 16-QAM coherent optical OFDM," *IEEE Photonics Technology Letters*, vol. 28, no. 22, pp. 2507–2510, 2016.

[23] C. Fougstedt, C. Häger, L. Svensson, H. D. Pfister, and P. Larsson-Edefors, "ASIC implementation of time-domain digital backpropagation with deep-learned chromatic dispersion filters," in *2018 European Conference on Optical Communication (ECOC)*, 2018, pp. 1–3.

[24] S. M. Ranzini, V. E. Parahyba, T. Vilela, E. O. Schneider, J. B. Tardelli, J. C. Oliveira, E. P. Lopes, T. C. Lima, J. D. Reis, J. R. Oliveira *et al.*, "Digital back-propagation ASIC design for high-speed coherent optical system," in *2015 SBMO/IEEE MTT-S International Microwave and Optoelectronics Conference (IMOC)*, 2015, pp. 1–5.

[25] L. Li, Z. Tao, L. Liu, W. Yan, S. Oda, T. Hoshida, and J. C. Rasmussen, "Nonlinear polarization crosstalk canceller for dual-polarization digital coherent receivers," in *Optical Fiber Communication Conference*, 2010, paper OWE3.

[26] E. Giacoumidis, Y. Lin, M. Blott, and L. P. Barry, "Real-Time machine learning based fiber-induced nonlinearity compensation in energy-efficient coherent optical networks," *APL Photonics*, vol. 5, no. 4, pp. 041 301–1–041 303–5, 2020.

[27] N. Kaneda, Z. Zhu, C.-Y. Chuang, A. Mahadevan, B. Farah, K. Bergman, D. Van Veen, and V. Houtsma, "FPGA implementation of deep neural network based equalizers for high-speed PON," in *Optical Fiber Communication Conference*, 2020, paper T4D–2.

[28] M. Li, W. Zhang, Q. Chen, and Z. He, "High-Throughput hardware deployment of pruned neural network based nonlinear equalization for 100-Gbps short-reach optical interconnect," *Optics Letters*, vol. 46, no. 19, pp. 4980–4983, 2021.

[29] D. A. Ron, P. J. Freire, J. E. Prilepsky, M. Kamalian-Kopae, A. Napoli, and S. K. Turitsyn, "Experimental implementation of a neural network optical channel equalizer in restricted hardware using pruning and quantization," *Scientific Reports*, vol. 12, no. 1, pp. 1–14, 2022.

[30] S. B. Amado, F. P. Guiomar, N. J. Muga, and A. N. Pinto, "Assessment of nonlinear equalization algorithms for coherent optical transmission systems using an FPGA," in *2015 17th International Conference on Transparent Optical Networks (ICTON)*, 2015, paper Mo.C1.4.

[31] W. J. Dally, W. J. Dally, and J. W. Poulton, *Digital systems engineering*. Cambridge University Press, 1998.

[32] M. Joost, "Theory of root-raised cosine filter," Dec. 2010. [Online]. Available: http://www.michael-joost.de/rrcfilter.pdf

[33] P.-K. A. Wai, C. R. Menyuk, and H. H. Chen, "Stability of solitons in randomly varying birefringent fibers," *Optics Letters*, vol. 16, no. 16, pp. 1231–1233, 1991.

[34] O. V. Sinkin, R. Holzlohner, J. Zweck, and C. R. Menyuk, "Optimization of the split-step Fourier method in modeling optical-fiber communications systems," *Journal of Lightwave Technology*, vol. 21, no. 1, pp. 61–68, 2003.

[35] D. Saad, *On-line learning in neural networks*. Cambridge University Press, 1999.

[36] V. Valimaki and T. Laakso, "Principles of fractional delay filters," in *Proceedings of 2000 IEEE International Conference on Acoustics, Speech, and Signal Processing*, vol. 6, 2000, pp. 3870–3873.

[37] H. Kan, H. Zhou, E. Börjeson, M. Karlsson, and P. Larsson-Edefors, "Digital emulation of time-varying PMD for real-time DSP evaluations," in *Asia Communications and Photonics Conf.*, Oct. 2021, paper M4H.4.

[38] K. Liu, E. Börjeson, C. Häger, and P. Larsson-Edefors, "FPGA-Based optical Kerr effect emulator," in *Signal Processing in Photonic Communications*, Jul. 2022, paper SpTh1I–2.

[39] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-Scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/

[40] C. B. Czegledi, *Modeling and compensation of polarization effects in fiber-optic communication systems*. Chalmers Tekniska Högskola (Sweden), 2018.

[41] G. Liga, C. B. Czegledi, and P. Bayvel, "A PMD-adaptive DBP receiver based on SNR optimization," in *Optical Fiber Communication Conference*, 2018, paper W2A–53.

[42] C. Fougstedt, M. Mazur, L. Svensson, H. Eliasson, M. Karlsson, and P. Larsson-Edefors, "Time-Domain digital back propagation: Algorithm and finite-precision implementation aspects," in *Optical Fiber Communications Conference and Exhibition (OFC)*, 2017, paper W1G.4.

[43] C. Fougstedt, L. Svensson, M. Mazur, M. Karlsson, and P. Larsson-Edefors, "ASIC implementation of time-domain digital back propagation for coherent receivers," *IEEE Photonics Technology Letters*, vol. 30, no. 13, pp. 1179–1182, 2018.

[44] D. E. Crivelli, H. Carter, and M. R. Hueda, "Adaptive digital equalization in the presence of chromatic dispersion, pmd, and phase noise in coherent fiber optic systems," in *IEEE Global Telecommunications Conference, 2004. GLOBECOM'04.*, vol. 4, 2004, pp. 2545–2551.

[45] E. Börjeson and P. Larsson-Edefors, "Fiber-on-Chip: Digital emulation of channel impairments for real-time DSP evaluation," *Journal of Lightwave Technology*, 2022.

[46] E. Börjeson, C. Fougstedt, and P. Larsson-Edefors, "Towards FPGA emulation of fiber-optic channels for deep-BER evaluation of DSP implementations," in *Signal Processing in Photonic Communications*, 2019, paper SpTh1E–4.