CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Implementation of an adaptive equalizer based on machine learning for real-time coherent optical receivers

Master's thesis in Embedded Electronic System Design

Boyang Zheng

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2021

Master's thesis 2021

Implementation of an adaptive equalizer based on machine learning for real-time coherent optical receivers

Boyang Zheng

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2021

Implementation of an adaptive equalizer based on machine learning for real-time coherent optical receivers

Boyang Zheng

Supervisor: Magnus Karlsson, Microtechnology and Nanoscience
Examiner: Per Larsson-Edefors, Computer Science and Engineering

Master's Thesis 2021
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Gothenburg, Sweden 2021

Implementation of an adaptive equalizer based on machine learning for real-time coherent optical receivers

Boyang Zheng
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

An adaptive equalizer is used to compensate for polarization mode dispersion (PMD) in polarization multiplexed (PM) optical communication systems. The PMD is not only influenced by the optical fiber but also affected by the fiber non-linear effects. Furthermore, the non-linear effects will be more and more severe with the increase of the channel's data throughput. However, traditional algorithms, such as the constant modulus algorithm (CMA) and decision-directed Least Mean Square (DD-LMS), only focus on compensating the PMD, and are not efficient for the non-linear effects. We try to apply machine learning-neural networks (ML-NNs) algorithm in the adaptive equalizer to deal with non-linear effects and more complex PMD. We also compare the ML-NNs and the CMA algorithm both in the software and hardware performance.

In this project, we improve the traditional ML-NNs structure based on the cascade multiple-input multiple-output (MIMO) filter theory so that the ML-NNs adaptive equalizer could be implemented in a real-time system more efficiently. MATLAB simulations are used to compare the software performance between the two algorithms via the effective signal-to-noise ratio (SNR). The two algorithms are re-written to the very-high-speed integrated circuit hardware description language (VHDL) format to compare the field-programmable gate array (FPGA) resource utilization. In addition, we extract information on how the number of taps in the ML-NNs equalizer influences the FPGA resource utilization.

We verified that the two types of equalizers have the same MATLAB software performance in terms of effective SNR. We also conclude that the FPGA is a good choice for implementing the forward propagation of the ML-NNs equalizer. However, it is not suitable for implementing backpropagation because of the high degree of multipliers required and the complexity of calculating derivatives.

Keywords: Adaptive equalizer, CMA, Machine-learning, Matlab, FPGA, VHDL, Coherent optical receivers.

Acknowledgements

I would like to first direct my thanks and gratitude to Professor Magnus Karlsson and Prof Per Larsson-Edefors for their valuable support and advice throughout the Master's Thesis Project.

I would also like to express my gratitude to Jochen Schröder, Christian Häger, and Erik Börjeson at Chalmers for the suggestions and comments about the theory algorithm and VHDL implementation.

Finally, I must express my profound gratitude to all of you at Embedded Electronic System Design (EESD) Master's Program for supporting and teaching in my master study life.

Boyang Zheng, Gothenburg, August 2021

Contents

List of Figures

List of Tables

1

Introduction

With the rapidly increasing demand [1] for data transmission, high capacity and reliable optical fiber communication systems are becoming more critical. Developing a next-generation optical communication system will be a most challenging problem in the next few years. Nowadays, optical communication systems usually use advanced modulation formats [2] and polarization multiplexing (PM) [3] to maximize channel spectral efficiency and increase channel capacity. However, these technologies simultaneously lead to extra system complexity and make the system suffer from impairments in the optical channel.

Advanced modulation formats will increase the complexity of transceivers directly [4], and the tolerance to optical channel impairments will also decrease significantly. The simplest example is that the 16 quadrature amplitude modulation (16QAM) format will require a higher signal-to-noise ratio (SNR) compare to quadrature phase shift keying (QPSK) modulation formats to ensure it can be demodulated [5]. Polarization multiplexing increases channel capacity without significantly increasing the complexity of the transceiver. However, an optical fiber is always slightly elliptical and birefringent which means different index of refraction for the x and y polarizations, and results in a different delay for each polarization, which is also known as polarization-mode dispersion (PMD) [6].

The group velocity dispersion (GVD) and PMD are important limitations for the optical channel capacity. We already have several solutions such as chromatic dispersion (CD) equalizer [7] in the digital signal processing (DSP) or dispersion compensation fiber (DCF) to compensate GVD. For the PMD, we commonly use an adaptive equalizer based on the constant modulus algorithm (CMA) and decision directed least mean square (DD-LMS) [8] to compensate it.

However, the PMD is a more complex problem that is not only influenced by fiber's physical characteristics [9] but also affected by non-linear effects; there has already been some research about the interaction between PMD and nonlinearity [10]. Traditional algorithms such as the CMA and DD-LMS can effectively track polarization changes and compensate polarization mode dispersion with complex calculations and significant hardware resources. However, they are not very efficient for non-linear impairments such as the fiber Kerr effect [11]. In this case, we want to expand the traditional adaptive equalizer so that it can be efficient for PMD and non-linear effects simultaneously.

As one of the most popular technologies, machine learning (ML) has been considered for signal recovery [12]. In this thesis, we consider applying machine learning-neural

networks (ML-NNs) as a potential solution to expand the traditional adaptive equalizer, which has been discussed in the literature [13]. We not only do simulate the software layer and compare it with the CMA equalizer, but we also consider the hardware implementation. Furthermore, utilization of hardware resources is also an aspect that we are interested in. However, we will primarily concentrate on the ML algorithm for compensating the PMD and compare the primary performance and field programmable gate array (FPGA) resource utilization with the CMA. Then we will try to conclude how the ML layers and the number of taps will influence the hardware resource consumption. Finally, as a future direction, we point out that the equalizer could be expanded to contain a non-linear compensation module.

The following report covers the process and results of the master thesis project. The introduction chapter mainly discusses the work background and goals, followed by a chapter introducing the basic concepts of optical transmission, the CMA algorithm, and the traditional ML-NNs algorithm. The next chapter concentrates on the simulation of the CMA adaptive equalizer, the ML-NNs adaptive equalizer, also verifies that the two equalizers have the same software performance based on effective SNR. Then the fourth chapter is used to describe the hardware implementation and compares the FPGA resource utilization between the two types of equalizers. Finally, this report ends with the work results and conclusion.

2

Theory

2.1 Optical transmission

An optical transmission link diagram is shown in Figure 2.1. Ideal polarization multiplexing-quadrature phase shift keying (PM-QPSK) symbols u_t are sent out from the Transmitter (Tx) block. Polarization rotation and PMD will be added to the symbols when the u_t pass the optical channel. An adaptive equalizer is used in the receiver to compensate for the rotation and the PMD. What we expect is that the output u_o from the equalizer would be ideal transmitted symbols plus the added noise.

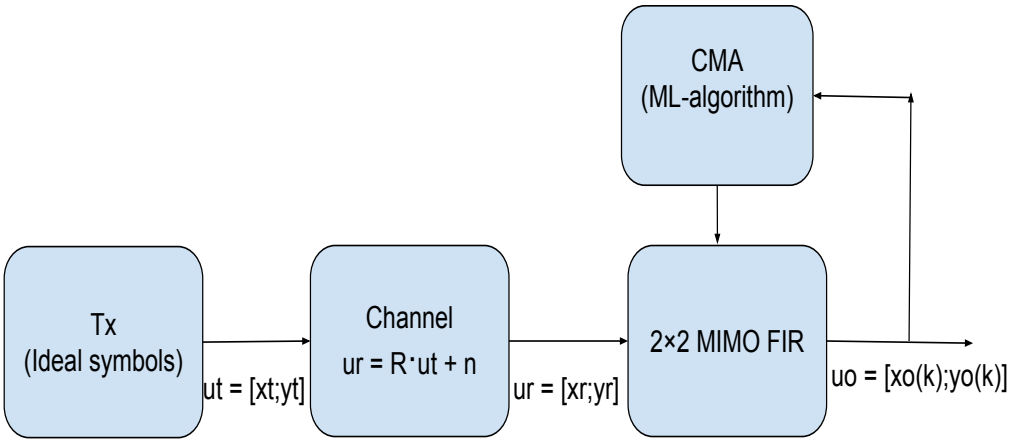

Figure 2.1: Simplified optical transmission system diagram.

2.1.1 Transmitter module

The ideal modulation symbols are transmitted out from the transmitter block. The PM-QPSK modulation format is used in this project. Figure 2.2 shows the comparison between the transmitted PM-QPSK symbols and received symbols with added noise. The signal-to-noise ratio (SNR) is defined by

$$SNR(dB) = 10 \cdot \log_{10} \frac{P_{signal}}{P_{noise}}. \tag{2.1}$$

Here P_{signal} is the signal power and the P_{noise} is the noise power. In Figure 2.2, we

used 10 Gbaud as the rate of the transmitter, which means the number of symbols the transmitter should send out every second is $10 \cdot 10^9$.

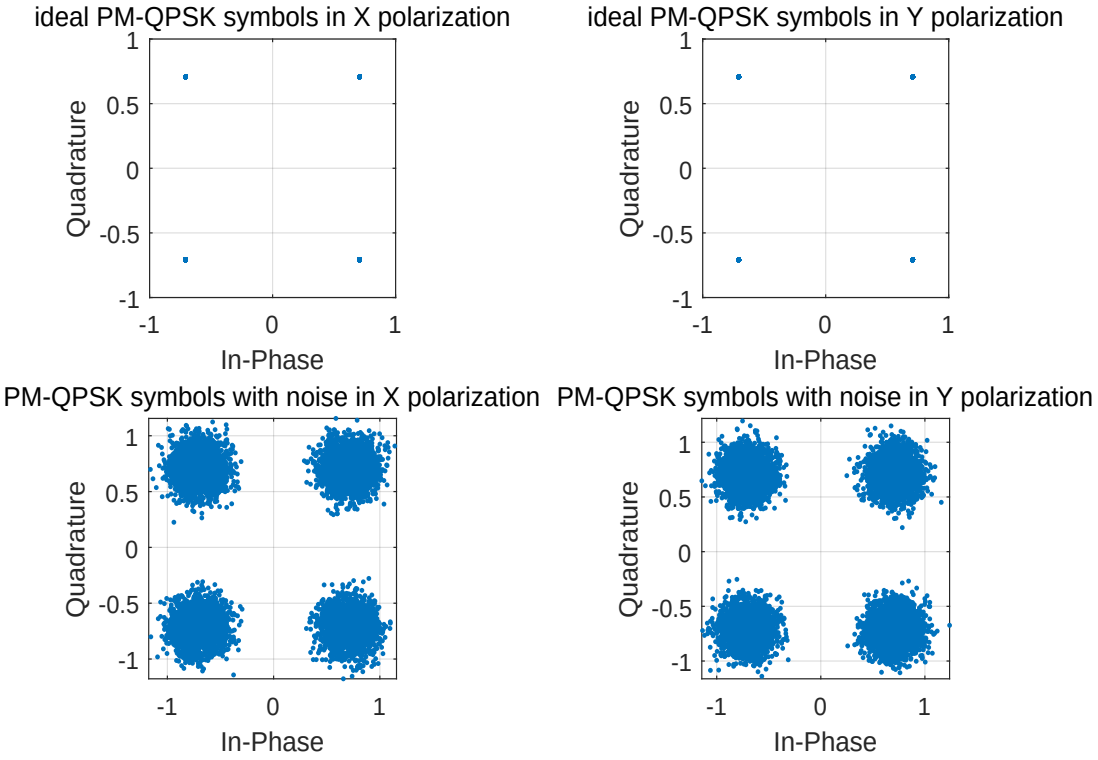

Figure 2.2: Top row: the transmitted PM-QPSK symbols and noise; Bottom row: the received symbols with SNR = 15 dB.

2.1.2 Channel model

This project uses two types of channel models based on the same equation

$$u_r = R \cdot u_t + n. \tag{2.2}$$

The first is the rotation channel which is used to check the accuracy of the calculation process of the algorithms:

$$u_r = U \cdot u_t + n, \tag{2.3}$$

where the $R = U$, which the U is a rotation matrix

$$U = [\cos\alpha \ \ -\sin\alpha;\ \ \sin\alpha \ \ \cos\alpha]. \tag{2.4}$$

Here, α is a rotation angle, n is the additive white Gaussian noise (AWGN), u_t is the transmitted symbols, and u_r is the received symbols. Figure 2.3 shows an example where the constellation contains both the polarization rotation $\alpha = 0.36\pi$ and noise.

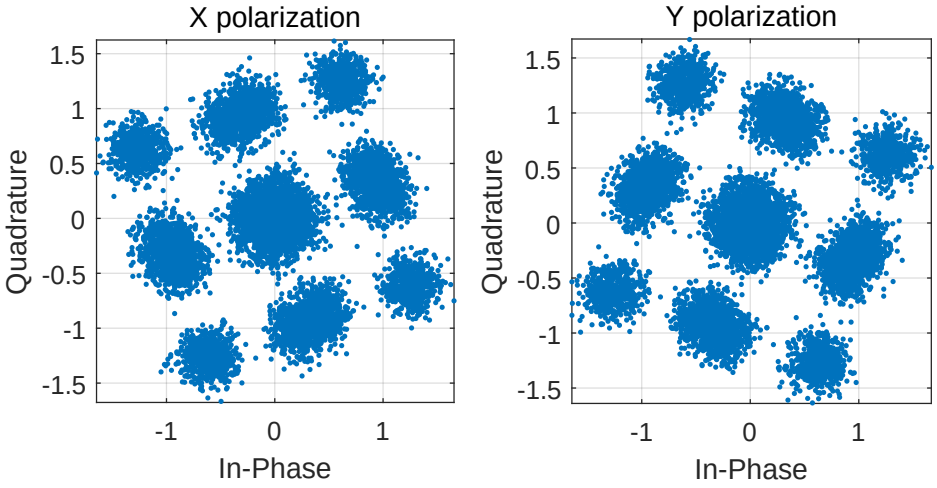

Figure 2.3: Symbols after a polarization rotation channel with SNR = 15 dB.

The PMD can be added to the channel in the frequency domain, as

$$U_r(w) = R(w) \cdot U_t(w) + n, \tag{2.5}$$

where $U_t(w)$ is the Discrete Fourier Transform (DFT) of the transmitted symbols. $U_r(w)$ is the DFT of the received symbols. The $R(w)$ is

$$R(w) = U \cdot D(w) \cdot U^{-1}, \tag{2.6}$$

where U is a rotation matrix and $D(w)$ is

$$D(w) = [\exp(i \cdot w \cdot T_0/2)\ 0; 0\ \ \exp(-i \cdot w \cdot T_0/2)]. \tag{2.7}$$

Here, the T_0 is the differential group delay. Figure 2.4 shows an example, with T_0 = 10 ps, of a constellation diagram containing both PMD and noise.

2.1.3 Equalization module

In optical communication systems, an adaptive equalizer is commonly used to compensate for the PMD. Generally, a two by two multiple-input multiple-output (MIMO) finite impulse response (FIR) filter [14] is used to construct the adaptive equalizer. For simplicity of explanation of the MIMO FIR, the filter process can be described as a matrix calculation.

If we consider the MIMO FIR as an N tap filter, the input matrix should be $u_i(k)$=$[X_i(k)$; $Y_i(k)]$ where

$$X_i(k) = [x_i(k); x_i(k-1); ...; x_i(k-N)] \tag{2.8}$$

$$Y_i(k) = [y_i(k); y_i(k-1); ...; y_i(k-N)], \tag{2.9}$$

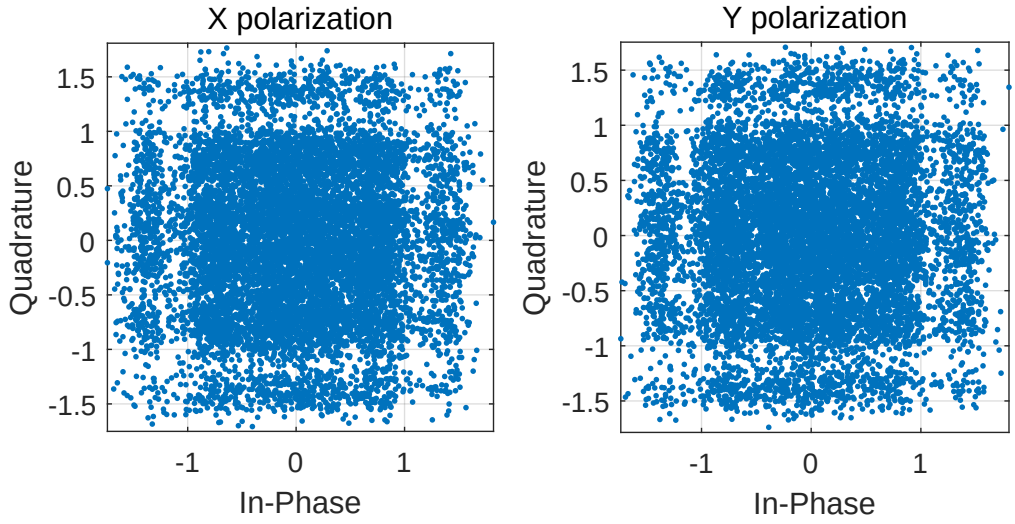

Figure 2.4: Symbols after a polarization PMD channel with with SNR = 15 dB.

$X_i(k)$, $Y_i(k)$ are the input samples for the x, y polarization. As a result, the $u_i(k)$ should be a column vector with a length of $2N$. From now on, we will use k to denote the symbol index.

The output of the MIMO filter is

$$x_o(k) = h_x^H(k) \cdot u_i(k) \tag{2.10}$$

$$y_o(k) = h_y^H(k) \cdot u_i(k), \tag{2.11}$$

where h_x, h_y are column vectors of length $2N$ representing the filter tap weights for the response to x and y polarization, and $()^H$ denotes the conjugate transpose.

2.2 Introduction to the CMA

The constant modulus algorithm (CMA) algorithm attempts to minimize the error in a mean square sense to make all of the symbols distributed close to a circle. For instance, with reference to the equation (2.3), the CMA algorithm aims at finding U's inverse and retrieve u_t.

2.2.1 Error function

The error function is used to estimate the difference of the power between the output symbols and the ideal symbols, and it is given by:

$$\xi_x(k) = A^2 - |x_o(k)|^2 \tag{2.12}$$

$$\xi_y(k) = A^2 - |y_o(k)|^2, \tag{2.13}$$

where A is a real-valued constant given by the amplitude of the transmitted symbol:

$$A = |u_t|, \tag{2.14}$$

where u_t is the transmitted symbols.

2.2.2 Update equation

The coefficients update equation is used to calculate the new coefficients and update for the next input sequence, and it is given by [8]:

$$h_x(k+1) = h_x(k) + \mu \cdot \xi_x(k) \cdot x_o(k)^* \cdot u_i(k) \tag{2.15}$$

$$h_y(k+1) = h_y(k) + \mu \cdot \xi_y(k) \cdot y_o(k)^* \cdot u_i(k), \tag{2.16}$$

where μ is the step size parameter, which will determine whether the CMA algorithm would converge and the speed of the convergence, and $()^*$ denotes the conjugate.

2.2.3 The CMA equalization process

Each input sequence $u_i(k)$ will be used to calculate one output based on the formulas (2.10) (2.11). The error function uses this output to calculate the corresponding error result. The update formulas will calculate the new coefficients for the next input sequence based on the formulas (2.15) (2.16). When new coefficients have been updated, it means finishing one iteration and this equalization process means the CMA adaptive equalizer will update coefficients for every symbol.

As mentioned before, the CMA adaptive equalizer only tries to find the inverse of U. It means the error signal would always contain noise. We expect the error curve to decrease with the number of iteration; however, the noise will never become zero but fluctuate. When the error curve has fluctuated in a certain range, it could be considered converged.

2.3 Introduction to the ML-NNs

Machine learning-neural networks (ML-NNs) are algorithms that try to mimic the brain. The primary process of ML-NNs contains several steps. Many training samples should be collected first, followed by unique mark labels in the training samples. Finally, training is carried out on all of the coefficients based on the labeled training samples [15]. The main target is trying to extract the feature of training samples and conclude the main features to several coefficient matrices. Forward propagation will calculate the output to be the same as the features based on the coefficient matrices. An example of basic network structure is shown in Figure 2.5.

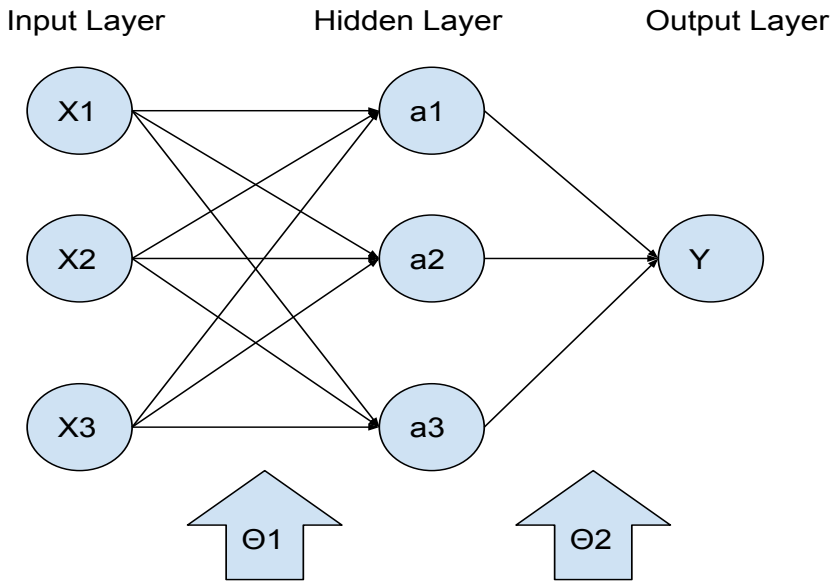

Figure 2.5: An example of basic neural networks structure.

2.3.1 Forward propagation

Data propagation from the input layer to the output layer is called forward propagation. For the hidden layer in our example

$$a_1 = f(\theta_{1\ 11} \cdot X_1 + \theta_{1\ 12} \cdot X_2 + \theta_{1\ 13} \cdot X_3) \tag{2.17}$$

$$a_2 = f(\theta_{1\ 21} \cdot X_1 + \theta_{1\ 22} \cdot X_2 + \theta_{1\ 23} \cdot X_3) \tag{2.18}$$

$$a_3 = f(\theta_{1\ 31} \cdot X_1 + \theta_{1\ 32} \cdot X_2 + \theta_{1\ 33} \cdot X_3), \tag{2.19}$$

where the θ_1 is coefficient matrix with a size of 3×3 and f is the activation function [16].

For the output layer

$$Y = f(\theta_{2\ 11} \cdot a_1 + \theta_{2\ 12} \cdot a_2 + \theta_{2\ 13} \cdot a_3), \tag{2.20}$$

where θ_2 is coefficient matrix with a size of 1×3. The whole process could be simplified to

$$Y = f(\theta_2 \cdot f(\theta_1 \cdot X_i)). \tag{2.21}$$

Here, $X_i = [X_1; X_2; X_3]$ is one input training sample.

2.3.2 Cost function

The cost function is mainly used to estimate the mean square error between the forward propagation outputs and training samples.

$$L(\theta_1, \theta_2) = \frac{1}{m} \cdot \sum_{i=1}^{m} (y_i - Y_i)^2, \tag{2.22}$$

where m is the number of total training samples, y_i is the labeled training sample, and Y_i is the forward propagation output.

2.3.3 Back propagation

The back propagation process is mainly responsible for the gradient calculation and coefficient update. The stochastic gradient descent [17] algorithm, which is a commonly used algorithm in ML-NNs to minimize the cost function, is considered to be the first choice in this project. The cost function is given in formula (2.22), and the gradient is derivative of $L(\theta_1, \theta_2)$

$$d(L)/d(\theta_{1\ i,j}) \tag{2.23}$$

$$d(L)/d(\theta_{2\ i,j}), \tag{2.24}$$

where i and j are the row and column indices. The typical update formula in ML is

$$\theta_{1i,j} = \theta_{1i,j} - \mu \cdot dL/d(\theta_{1i,j}) \tag{2.25}$$

$$\theta_{2i,j} = \theta_{2i,j} - \mu \cdot dL/d(\theta_{2i,j}), \tag{2.26}$$

where μ is the learning rate, also called stepsize, which will influence the convergence behavior of the iteration process.

2.3.4 The ML-NNs calculation process

In the calculation process, the corresponding input samples X and output labels y_i should be prepared first. Moreover, m input samples could obtain m forward propagation outputs Y_i based on section 2.3.1. The next step is to calculate the average cost function via section 2.3.2. Finally, update the new coefficient matrices based on section 2.3.3. When new coefficient matrices are generated, it means that finished one iteration, and each iteration uses the same training samples as before. In general, more number iterations mean higher accuracy of the ML-NNs system. However, it also means more time is consumed.

3

MATLAB Software implementations

3.1 System parameters overview

Table 3.1 shows the overview of the basic system parameters. The PM-QPSK modulation format is chosen in our simulation and verification system and sets the SNR to 15 dB. A step size related to $1/2^p$ (p is a natural number) is used in the system for more efficiency in the very-high-speed integrated circuit hardware description language (VHDL) implementation. We set the CMA adaptive equalizer to have 17 filter taps. 16 bit word lengths are also used in this project.

Table 3.1: Basic System Setup

Modulation format	PM-QPSK
SNR	15 dB
Symbol rate	10 GBaud
Stepsize	0.03125(1/32)
Taps of CMA equalizer	17
Differential group delay	10 ps
Word length	16 bits

3.2 The CMA adaptive equalizer in Matlab

Figure 3.1 shows the error signal of two polarizations as achieved in the equations (2.12-2.13); it is the same as our expected results. The error value will fluctuate in a particular range depending on the noise level. Figures 3.2 and 3.3 are shown for the CMA equalizer's constellation output. It is easy to find that the Figure 3.2 and 3.3 are the same as the Figure 2.2. All of the equalizer's output symbols are distributed to the position of the ideal symbols contains noise. As a result, we could consider that the CMA equalizer works as expected.

3.3 The ML-NNs adaptive equalizer in Matlab

It is challenging to implement the same processes mentioned in section 2.3 in real-time signal processing. The most important problem in a real-time project is la-

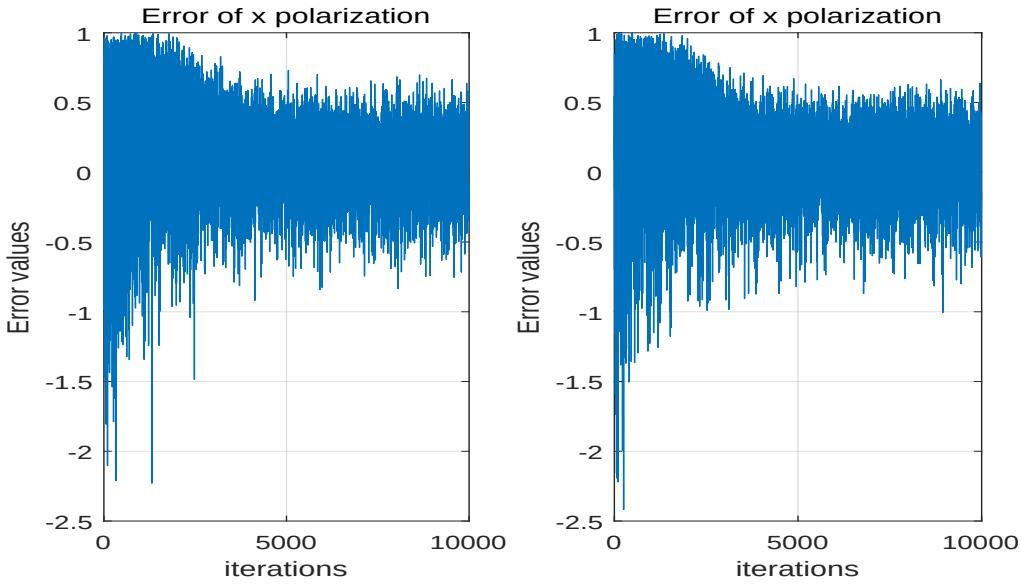

Figure 3.1: Error curve of the CMA adaptive equalizer, Matlab simulation.

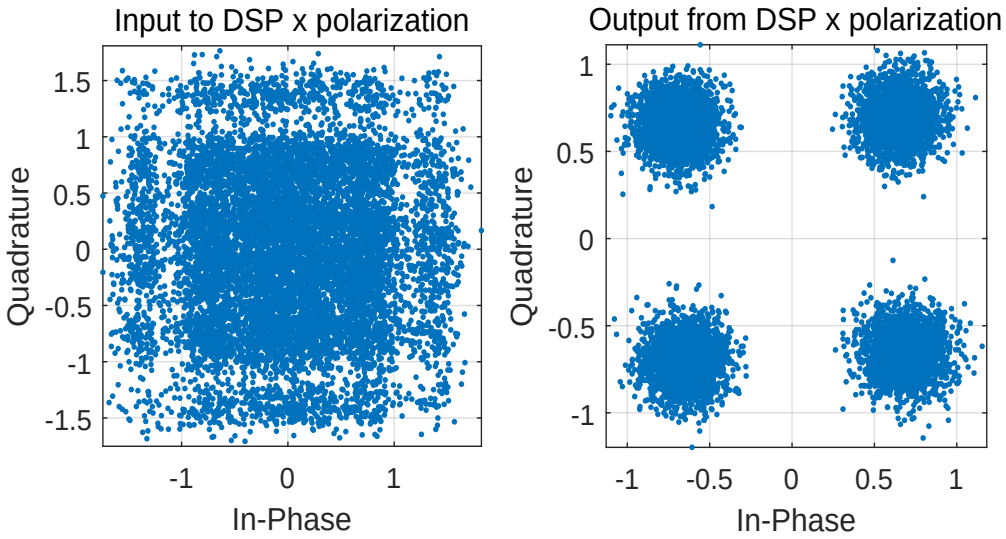

Figure 3.2: The CMA equalizer's output in x polarization, Matlab simulation.

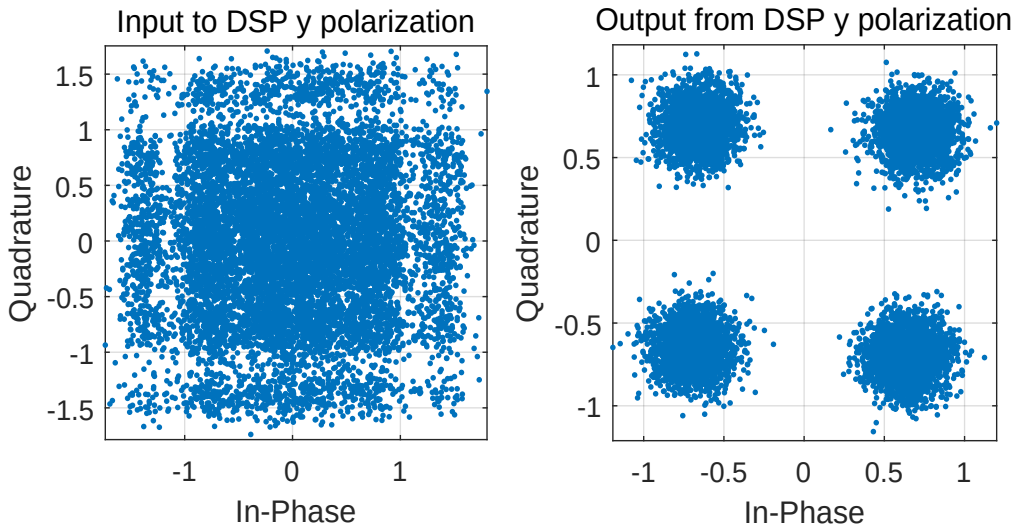

Figure 3.3: The CMA equalizer's output in y polarization, Matlab simulation.

tency [18]. For instance, the ML-NNs system needs some time to collect samples and label the samples. Moreover, the number of training samples would influence the time consumption of the above two steps. At the same time, it also hard to estimate how many training samples and iterations are needed for each training process. All of the above steps and problems will result in significant delays and affect hardware requirements.

An expansion of the CMA equalizer as a potential ML-NNs is investigated in this project. The rough design diagram is shown in Figure 3.4. The H_1 and H_2 are the coefficient matrices of size $2\times2N$. It should be noticed that we first remove the activation function f to check if the new structure could compensate for the PMD, which is a linear effect.

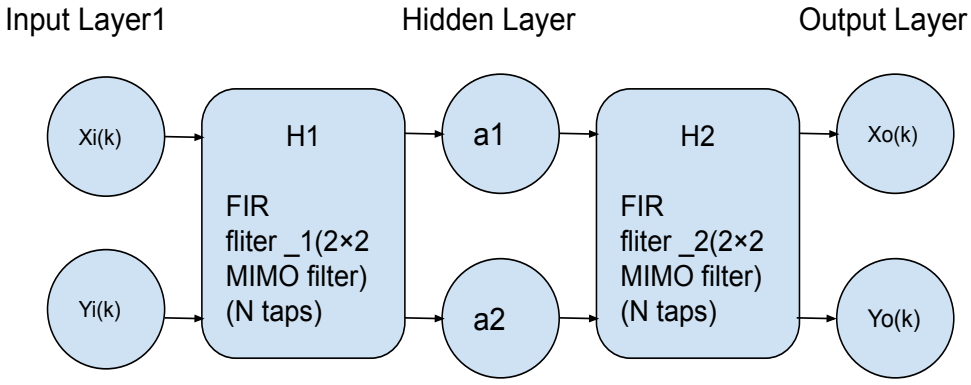

Figure 3.4: Design structure of the ML-NNs adaptive equalizer.

3.3.1 Forward propagation

For the forward propagation, the main calculation process is the same as traditional ML-NNs. First, we define every input training sample just as for the input of the

CMA equalizer as given in the formulas (2.8) (2.9).

Then, by matrix multiplication we calculate the hidden layer:

$$A_1 = H_1 \cdot u_i(k), \tag{3.1}$$

where $A_1 = [a_1; a_2]$ is a column vector of length 2.

It should be noticed that every input sequence with length $2N$ will obtain an output vector with length 2 for the first FIR filter. However, we still need an input sequence with length $2N$ for the second FIR filter. This case could be easily considered as it should run the first FIR filter N times to collect $2N$ outputs as the input to the second FIR filter. As a result, N number of A_1 should be reshaped to a column vector A_1' with size $2N$. Finally, for the output layer, we use the same calculation process:

$$u_o(k) = H_2 \cdot A_1', \tag{3.2}$$

where $u_o(k) = [X_o(k); Y_o(k)]$ is a column vector of length 2.

3.3.2 Cost function

The cost function is built based on the error function of the CMA and equation (2.22). It will try to distribute all of the symbols close to a circle.

$$L_{H_1,H_2}(k) = (A^2 - |X_o(k)|^2)^2 + (A^2 - |Y_o(k)|^2)^2, \tag{3.3}$$

where A is the same as the CMA shows in equation (2.14), the optimum H_1 and H_2 are obtained via minimizing the cost function based on the stochastic gradient descent algorithm.

In this part, we have several improvements from the traditional ML-NNs structure. If we compare formula 3.3 to formula 2.22, we set m to 1, which means every iteration will only use a single training vector. This improvement could avoid the latency which because of collecting training samples. Moreover, we also use the same label A as in the CMA instead of the labeled samples y_i. This improvement could eliminate the latency which results from labeling the training samples.

3.3.3 Back propagation

The backpropagation process is responsible for the gradient calculation and coefficient update. This project uses the same equations which are shown in (2.23-2.26) to construct the backpropagation process. The cost function is L_{H_1,H_2} as defined in equation (3.3), and we need to calculate the derivative of each of the elements in the coefficient matrices.

3.3.4 Simulation results

In the simulation step, we could notice that two cascaded FIR filter with N taps would have the same channel memory as one $2N-1$ tap FIR filter. As Table 3.1 shows, we have set the CMA equalizer to 17 taps. We need to set the taps of the ML-NNs equalizer to 9 taps that could make sure the ML-NNs equalizer has the same channel memory as the CMA equalizer. All of the other parameters should be kept the same as in Table 3.1.

Figure 3.5 shows the cost curve of the ML-NNs adaptive equalizer; it is the same as our expected results. The cost value will fluctuate in a certain range depending on the noise level. Figures 3.6 and 3.7 show the constellation output from the ML-NNs equalizer, compare with Figure 2.2 what means the ML-NNs equalizer could work as we expected.

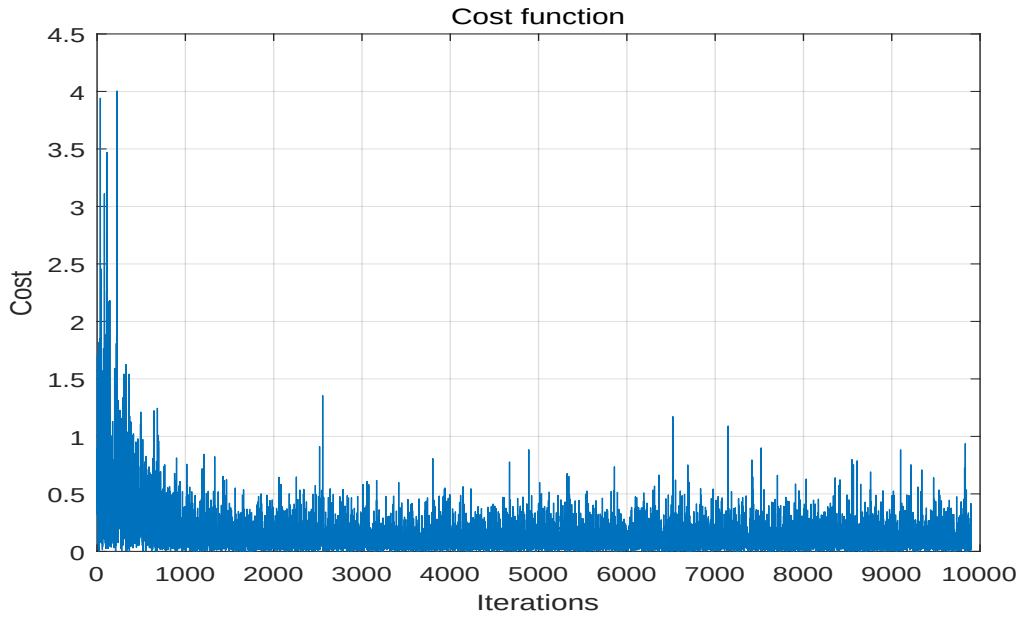

Figure 3.5: Cost curve of the ML-NNs equalizer, Matlab simulation.

3.4 The effective SNR of the ML-NNs equalizer and the CMA equalizer

The effective SNR(SNR_{eff}) will be used to represent the performance of the adaptive equalizer. To verify that the ML-NNs equalizer has the same performance as the CMA equalizer, the effective SNR is calculated for the two equalizers and plotted in the same diagram for comparison. The formula for calculating the effective SNR is

$$SNR_{eff} = \frac{|u_o|^2}{|u_t - u_o|^2}, \tag{3.4}$$

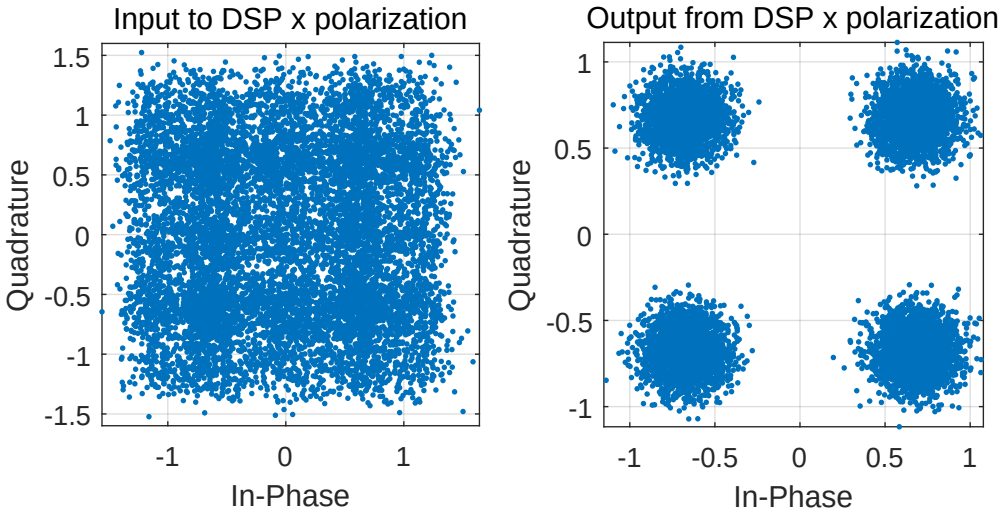

Figure 3.6: The ML-NNs equalizer in x polarization, Matlab simulation.

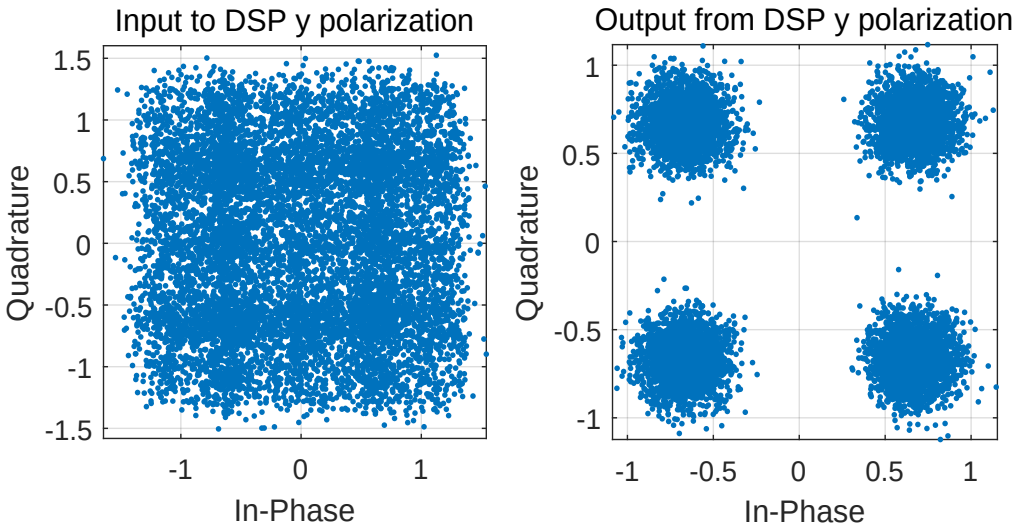

Figure 3.7: The ML-NNs equalizer in y polarization, Matlab simulation.

where u_o is the output symbol from the equalizer, and u_t is the ideal transmitted symbols. Figure 3.8 shows the effective SNR curve for the 17 taps CMA equalizer and the two 9 taps ML-NNs equalizer based on the same system setup. Moving average calculation is used in the diagram to keep the curve more readable, and the moving window size is 300.

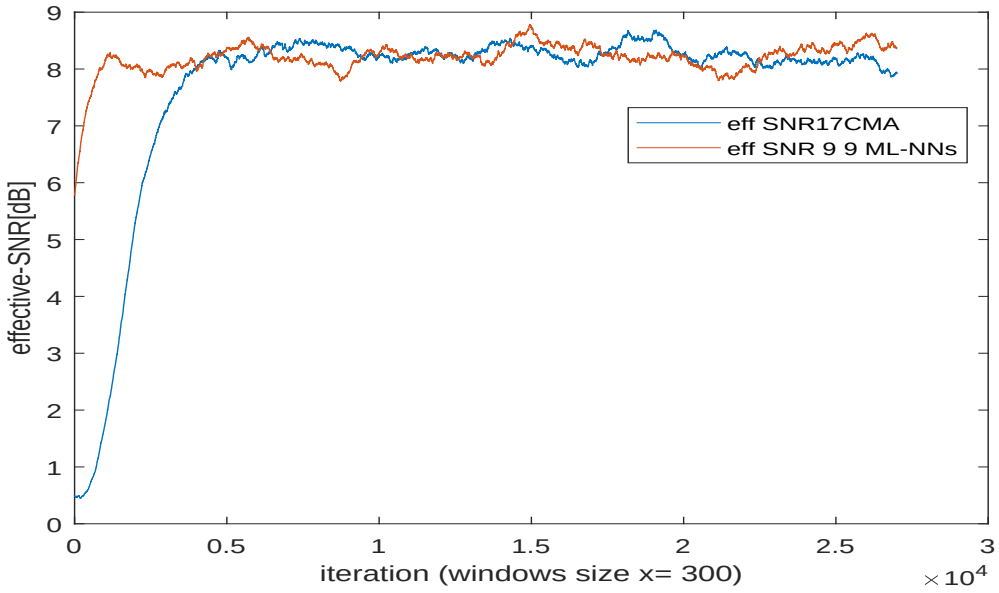

Figure 3.8: Comparison of the SNR_{eff} between the ML-NNs and CMA.

In the diagram, we compare the 17 tap CMA equalizer with two 9 tap ML-NNs equalizer. As the results curve shows, the two equalizers have the same maximum effective SNR, which means that they have the same software performance. However, the curve also shows that the ML-NNs have a faster convergence than the CMA equalizer. It should be noticed that the start SNR of the ML-NNs equalizer is higher than the CMA equalizer in the diagram. This is because the curve is built on moving average calculation; the two starting points are the same when setting the window size to 0.

4

Hardware implementations

4.1 Verification methods

4.1.1 Verification process

The verification structure diagram is shown in Figure 4.1. In the verification process, Matlab and VHDL simulators are combined to use for checking verification simulation results. In the first step, the test symbols u_r are generated from Matlab and provided as input ".vec" files. Then, a specific testbench is written with VHDL for scanning input ".vec" files as the adaptive equalizer's input and writes the equalizer's output u_o to output ".vec" files. Finally, the output ".vec" files will be imported into Matlab, where the results are plotted and results are checked.

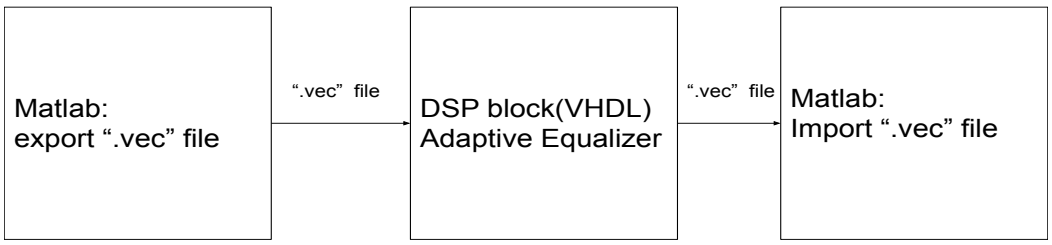

Figure 4.1: Verification process diagram.

4.1.2 Generation of test symbols from Matlab

The fractional number multiplication calculation will be used in the whole verification process, where all of the multipliers will only deal with fractional numbers.

As a result, we need to ensure that all input data are less than 1. The transmitted symbols u_t which are shown in Figure 2.2 only contain fractional parts. The symbols in Figure 2.3 only contain a polarization rotation channel, and Figure 2.4 contains a PMD channel. It is easy to find that the symbols u_r contain integer and fractional parts at the same time. However, the integer part of the symbols u_r in Figures 2.3 and 2.4 are still not greater than 2. For keeping that all of the test symbols only contain fractional parts, a scale with 1/2.4 is used to scale down the symbols u_r.

A word length of 16 bits is used in this project. The most significant bit means a sign bit, the remaining 15 bits mean a fractional number. In general, 15 bits binary data should be representative of an integer. However, in VHDL, we could consider that the 15 bits binary integer is equal to a fractional number, and the value of the fractional number is

$$V_f = \frac{int(B_{15\ bits})}{2^{15}}, \tag{4.1}$$

where V_f is the value of fractional number, and the $int(B_{15\ bits})$ means to transfer the 15 bits binary data to an integer. Test data with integer format can be generated from Matlab based on this theory. After the u_r has been scaled down with the factor 1/2.4, we can ensure that all of the u_r are fractional numbers. Then, change the format of (4.1)

$$int(B_{15\ bits}) = V_f \cdot 2^{15}, \tag{4.2}$$

where $V_f = u_r$. Finally, the test data with integer format will be written into the ".vec" files.

4.2 The CMA adaptive equalizer in VHDL

4.2.1 Block diagram of the CMA equalizer

A pipeline design is employed to have parallelization implemented more conveniently. The design block diagram is shown in Figure 4.2. The "FIR Filter" block is used as a filter process. The "Error" module is in charge of calculating the error signals based on the formulas (2.12) (2.13). The "Gradient_calculation" block will calculate the coefficients weight with the formula (2.15) (2.16), which will be utilized for updating the filter coefficients. The "coefficient_update" only finishes the process of updating coefficients. In the Figure 4.2, we use [15:0] to represents the 16-bit data used throughout the design.

There should be noticed that for corresponding to the coherent receiver, we use the I and Q components separately as the input and output. As a result, in the design, the adaptive equalizer should be a 4×4 MIMO filter. In addition, we added the error value as the output port for more convenience to check the error curve.

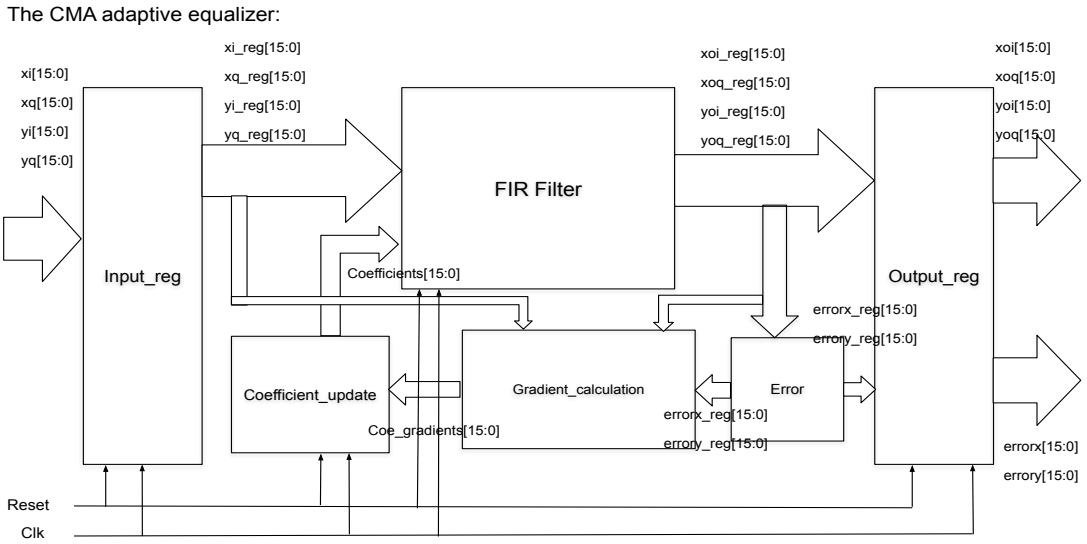

Figure 4.2: Block diagram of the CMA equalizer VHDL design.

4.2.2 Implementation of the CMA equalizer

As shown in the block diagram, the combinational logic circuit would be the primary choice for implementing the CMA equalizer. However, the "FIR Filter" and "Coefficients_update" blocks are also dependent on the previous values. These two modules should be built on the sequential logic circuit.

In VHDL, the fractional number multiplication could be calculated directly. However, if one of the fractional constant is close to a particular format such as $1/2^p$, where p is a natural number, the bits shift operation in VHDL would be more efficient. E.g., the stepsize in the project is $0.03125(1/32 = 1/2^5)$, 5 bits right shift would consume less resources than a direct multiplication.

4.2.3 Verification of the CMA equalizer

The output ".vec" files from the VHDL simulator will be imported into Matlab which then plots the results diagram directly. Figure 4.3 shows the error curve of the CMA adaptive equalizer. The result diagrams show that the error curve converges as expected.

The resulting constellation of x and y polarization is shown in Figure 4.4 and Figure 4.5. The CMA adaptive equalizer in the VHDL appear to work as expected because it does find the output symbols distributed similarly as in Figures 2.6 and 2.7.

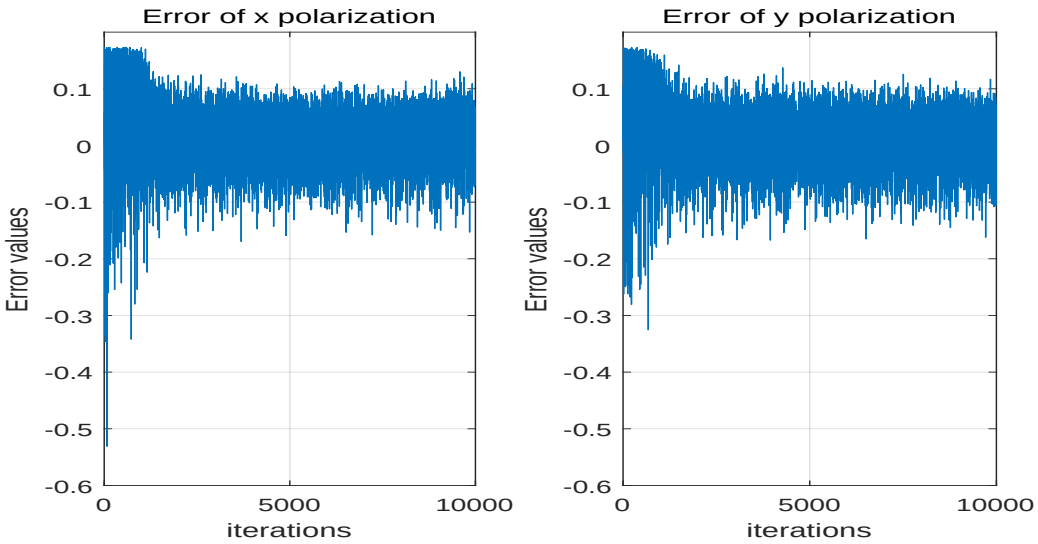

Figure 4.3: Error curve of the CMA adaptive equalizer, VHDL verification.

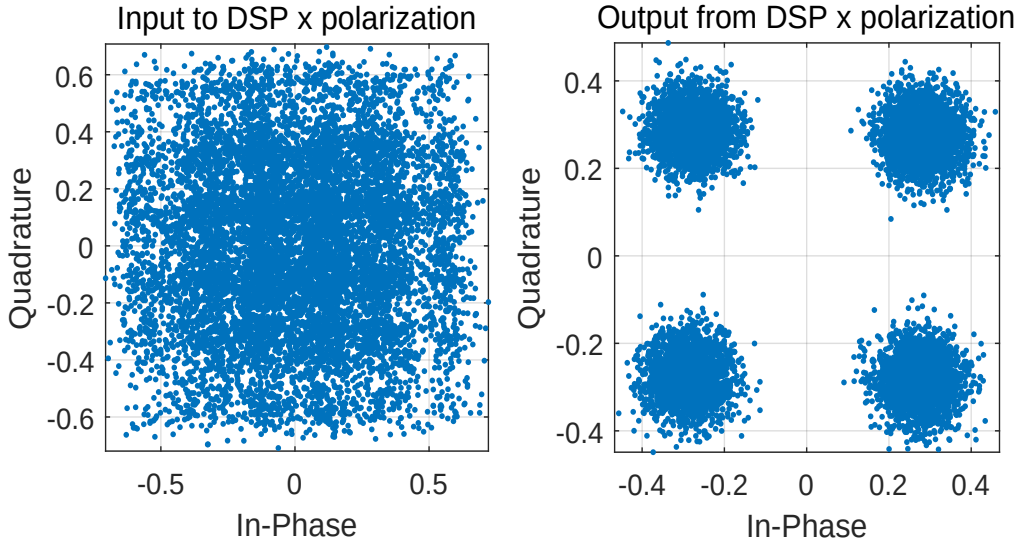

Figure 4.4: The CMA equalizer in x polarization, VHDL verification.

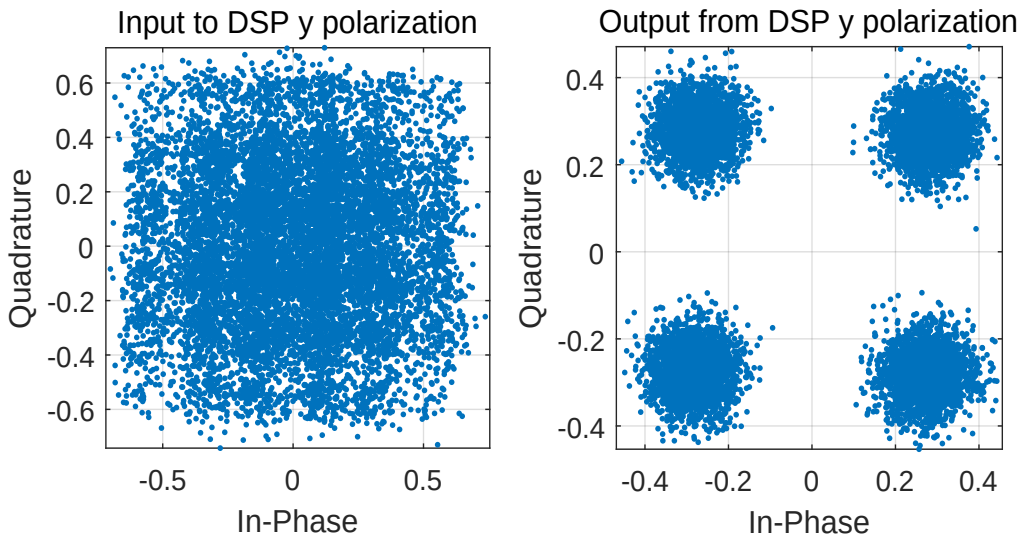

Figure 4.5: The CMA equalizer in y polarization, VHDL verification.

4.2.4 Evaluation of FPGA resource utilization of the CMA equalizer

The FPGA named "XC7VX690TFFG1761-1 [19]" was used to build the Vivado project based on the CMA adaptive equalizer code and finished the synthesis process. Table 4.1 shows some basic hardware information about the chip that was used in the project.

Table 4.1: Basic resource information of XC7VX690TFFG1761-1

Slice LUTs	433200
DSPs	3600
BUFGCTRL	32
Slice Registers	866400
Slice	108300
LUT as Logic	433200

Table 4.2 shows the hardware consumption information about the CMA adaptive equalizer. This report is extracted from the "Utilization report" when the Vivado project finished the "implementation" process.

Table 4.2: FPGA utilization information of the 17 taps CMA adaptive equalizer

Name	Topfile	FIR Filter	Gradient_cal	Coe_update	Error
Slice LUTs(433200)	1.54%	0.52%	0.75%	0.26%	<0.01%
DSPs(3600)	15.33%	7.56%	7.67%	0%	0.11%
BUFGCTRL(32)	3.13%	0%	0%	0%	0%
Slice Registers(866400)	0.39%	0%	0%	0.25%	0%
Slice(108300)	1.92%	0.53%	0.75%	0.53%	<0.01%
LUT as Logic(433200)	1.54%	0.52%	0.75%	0.26%	<0.01%

From the table, we see that all of the resource utilization is lower than 20%. The "DSPs" resource should be focused on because it has the highest consumption and has reached 15.33%. However, this value is still reasonable. The utilizations for all other resources are lower than 5%, which is a good situation. If any resource utilization is above 20%, it means that maybe we need to consider how to improve our design.

4.3 The ML-NNs adaptive equalizer in the VHDL

4.3.1 Block diagram of the ML-NNs equalizer

The design block diagram for the ML-NNs adaptive equalizer is shown in Figure 4.6. The most significant difference with the CMA adaptive equalizer is that the one 2N-1 taps FIR filter is divided into two FIR filters with N taps; the rest of the modules are the same as the CMA structure.

4.3.2 Implementation of the ML-NNs equalizer

The implementation principle is the same as the CMA adaptive equalizer mentioned in section 4.2.2. However, two N-taps FIR filters have the same channel memory as the single $2N-1$ taps FIR filter. Even though the input sequence for each polarization of the first FIR filter is N-length, a vector for each polarization with a length of $2N-1$ is still needed to be collected for the gradient calculation process.

4.3.3 Verification of the ML-NNs equalizer

The same system parameters as given in table 2.1 are used in the VHDL ML-NNs equalizer to avoid extra variable difference. Figure 4.7 shows the cost signal generated from the VHDL simulator. It is not exactly the same as shown in Figure 3.2 because of the lower resolution of the VHDL hardware model. However, the primary convergence behavior is similar.

Figure 4.8 and 4.9 show the result constellation diagram that was generated from VHDL simulator. The constellation diagram could be considered the same as a simulation in Matlab. As an early observation, the ML-NNs equalizer in VHDL could work for compensating the PMD.

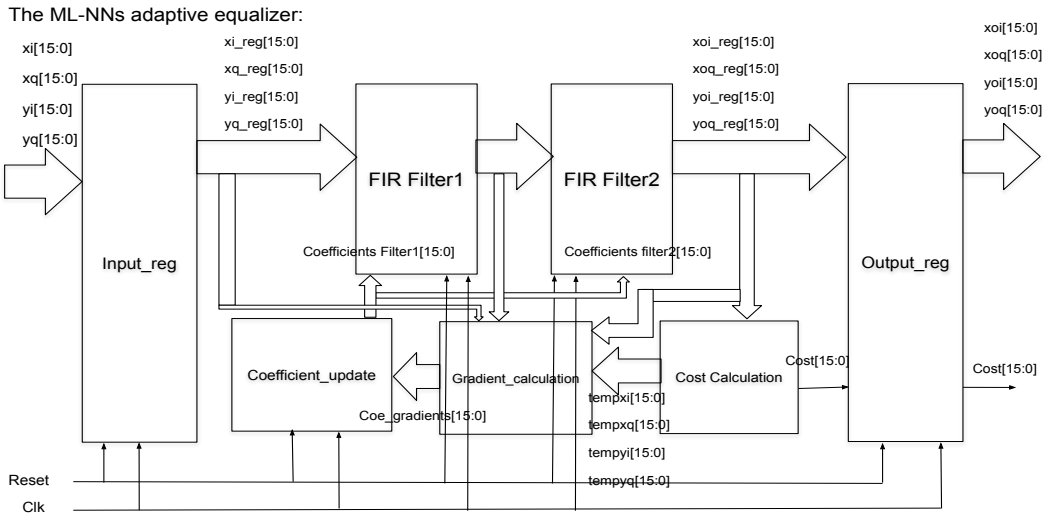

Figure 4.6: Design diagram of the ML-NNs adaptive equalizer

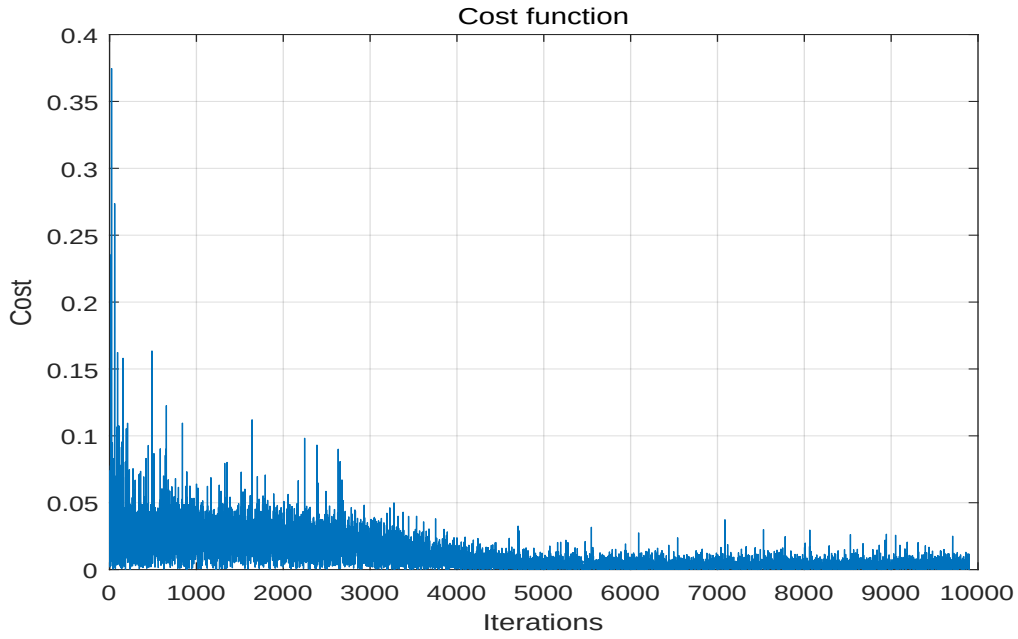

Figure 4.7: Cost curve of ML-NNs adaptive equalizer, VHDL verification.

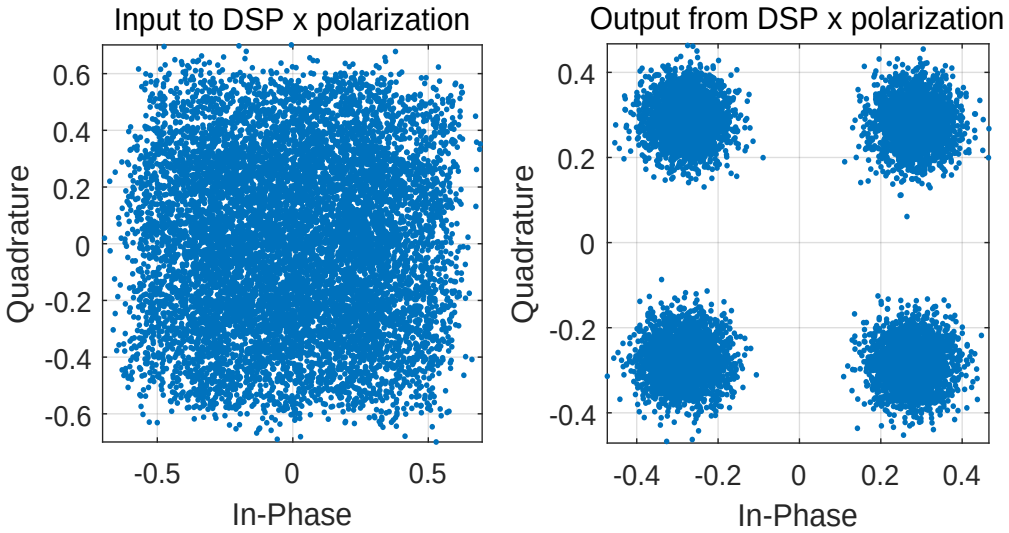

Figure 4.8: The ML-NNs equalizer in x polarization, VHDL verification.

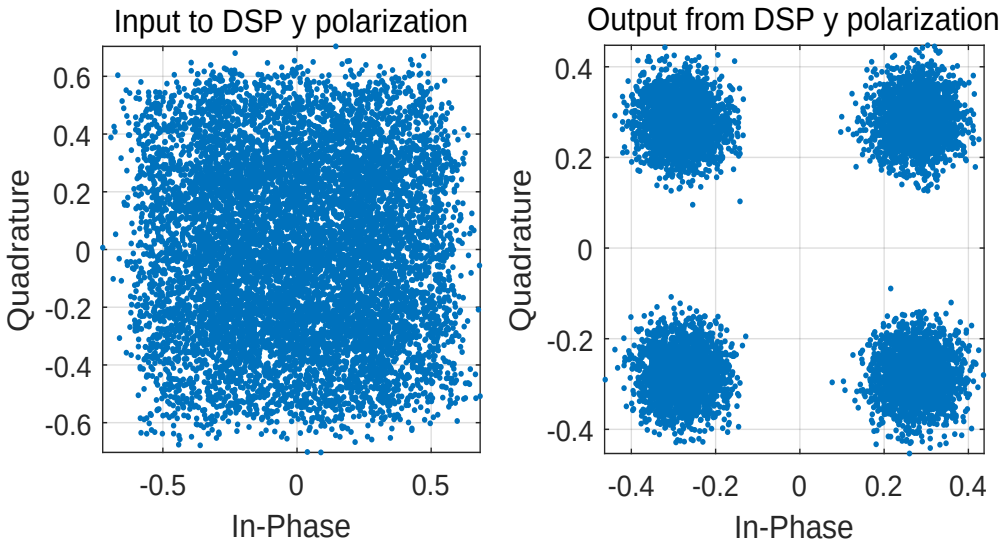

Figure 4.9: The ML-NNs equalizer in y polarization, VHDL verification.

4.3.4 Evaluation of FPGA utilization of the ML-NNs equalizer

Table 4.3 shows the FPGA resource utilization for each module. It should be realized that more multiplication means more "DSPs" resources are occupied.

Table 4.3: FPGA utilization information of the 9 taps ML-NNs equalizer

Name	Topfile	Filter1	Filter2	Cost_cal	Gradient_cal	Coe_update
Slice LUTs(433200)	7.54%	0.39%	0.39%	0.01%	6.06%	0.54%
DSPs(3600)	90.28%	4.00%	4.00%	0.28%	82.00%	0%
BUFGCTRL(32)	3.13%	0%	0%	0%	0%	0%
Slice Registers(866400)	0.57%	0.22%	0.07%	0%	0%	0.27%
Slice(108300)	9.22%	1.26%	0.61%	0.01%	6.57%	0.56%
LUT as Logic(433200)	7.54%	0.39%	0.39%	0.01%	6.06%	0.54%

The FPGA utilization of the forward propagation is the same as the CMA equalizer shown in the table. It means that FPGA should be an excellent choice to implement the complex NNs equalizer if we only focused on the forward propagation.

However, the "DSPs" resource consumption should be noticed in the project. In the results table, the "DSPs" resource of the "gradient calculate" module is in the highest consumption degree because the calculation uses many multiplications. The main serious problems are not only the consumption of "DSPs" resources but also the "Gradient calculate" module is the most difficult to implement because it needs to calculate the derivative of the cost function.

In addition, more hidden layers mean more complex derivatives. However, no existing function or IP core could be used to calculate the derivative automatically in VHDL. That means the derivative formula needs to be calculated by other tools or ourselves first and then written in the VHDL format.

4.4 FPGA utilization comparison between the CMA and ML-NNs

Table 4.4 shows the total resource utilization in the FPGA for the CMA equalizer and the ML-NNs equalizer. From the result table, the "DSPs" resource are significantly different. The ML-NNs consume six times more "DSPs" resources than the CMA equalizer, even though the ML-NNs equalizer only contains one hidden layer. However, the other kinds of resources are not significantly higher in the CMA equalizer than the ML-NNs equalizer.

Table 4.4: FPGA utilization of The 17 taps CMA equalizer vs 9 taps ML-NNs equalizer

Name	The CMA equalizer(17 taps)	The ML-NNs equalize(9 taps)
Slice LUTs(433200)	1.54%	7.54%
DSPs(3600)	15.34%	90.28%
BUFGCTRL(32)	3.13%	3.13%
Slice Registers(866400)	0.39%	0.57%
Slice(108300)	1.90%	9.27%
LUT as Logic(433200)	1.54%	7.54%

4.4.1 The number of taps vs FPGA utilization

Figure 4.10 shows the results of how the number of taps influences the FPGA resource utilization. It shows that only the "DSPs" resource has a quick increase with the taps. The other kinds of resources do not increase much totally, and could be considered stable.

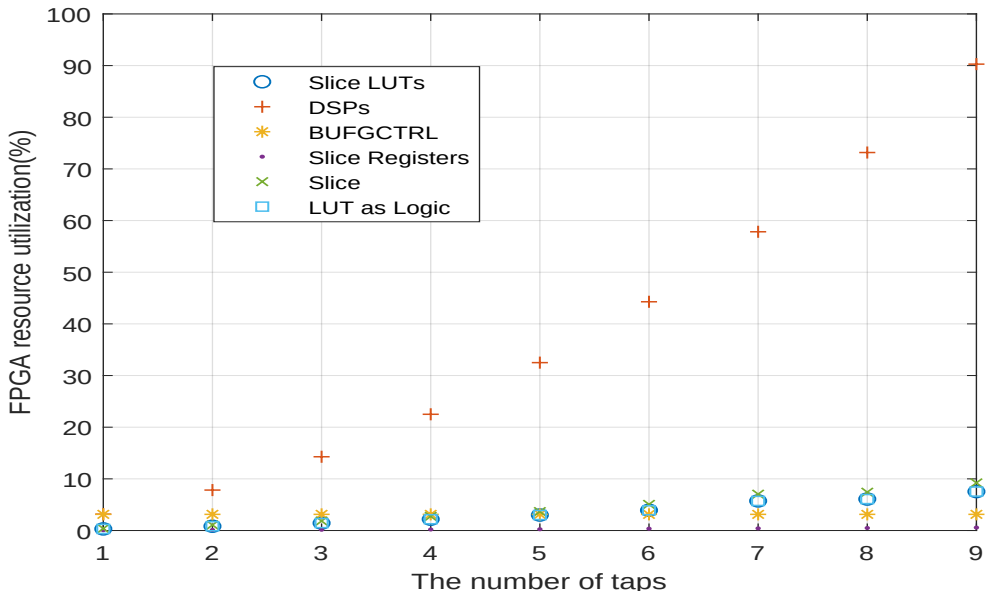

Figure 4.10: The number of taps vs FPGA resource utilization

5

Conclusions and outlook

We have proposed a machine-learning structure that could compensate for the PMD in optical links. We verified that the two types of equalizers have the same software performance based on comparing the effective SNR results. The hardware implementation for the two types of adaptive equalizers are based on FPGA and extraction of the consumed FPGA resources allows a side-by-side comparison. The forward propagation of the two equalizers have the same hardware consumption, as we verified from the utilization report.

The back propagation of the two equalizers has a significantly difference in the "DSPs" resource consumption. The gradient calculation module of the ML-NNs equalizer consumes most of the "DSPs" resource ; this module consumes much more resources that the corresponding module of the CMA equalizer. Another problem in the gradient calculation is that no existing function or IP core can automatically calculate the derivative in the VHDL design. this means each derivative needs to be calculated by other tools or externally. The complexity of this calculation will also increase with the number of hidden layers.

In conclusion, if we focus on "DSPs" resource consumption and calculation complexity, FPGA could be a very good choice to implement forward propagation of the ML-NNs adaptive equalizer. However, it is not suitable for back propagation because of too high "DSPs" resource utilization, and it can not calculate the derivative automatically.

As an expanded work of the ML-NNs equalizer in the future, the activation function could be replaced by the non-linear effects equation so that the equalizer could compensate also for the non-linear effects.

Bibliography

[1] R. Essiambre and R. W. Tkach, "Capacity Trends and Limits of Optical Communication Networks," in Proceedings of the IEEE, vol. 100, no. 5, pp. 1035-1055, May 2012, doi: 10.1109/JPROC.2012.2182970.

[2] P. Winzer, D. Neilson, and A. Chraplyvy, "Fiber-optic transmission and networking: the previous 20 and the next 20 years [Invited]," Opt. Express 26, 24190-24239 (2018).

[3] N.Badraoui and T.Berceli, "Enhancing capacity of optical links using polarization multiplexing," Opt Quant Electron 51, 310 (2019). https://doi.org/10.1007/s11082-019-2017-3

[4] J. Wei, Q. Cheng, R. Penty, I. White, and D. Cunningham, "Analysis of Complexity and Power Consumption in DSP-Based Optical Modulation Formats," in Advanced Photonics for Communications, OSA Technical Digest (online) (Optical Society of America, 2014), paper SM2D.5.

[5] P. Winzer, "High-Spectral-Efficiency Optical Modulation Formats," J. Lightwave Technol. 30, 3824-3835 (2012).

[6] J. P. Gordon and H. Kogelnik, "PMD fundamentals: Polarization mode dispersion in optical fibers," Proceedings of the National Academy of Sciences Apr 2000, 97 (9) 4541-4550; DOI: 10.1073/pnas.97.9.4541

[7] S. Savory, G. Gavioli, R. Killey, and P. Bayvel, "Electronic compensation of chromatic dispersion using a digital coherent receiver," Opt. Express 15, 2120-2126 (2007).

[8] M. Faruk and S. Savory, "Digital Signal Processing for Coherent Transceivers Employing Multilevel Formats," J. Lightwave Technol. 35, 1125-1141 (2017).

[9] D. A. Nolan, Xin Chen and Ming-Jun Li, "Fibers with low polarization-mode dispersion," in Journal of Lightwave Technology, vol. 22, no. 4, pp. 1066-1077, April 2004, doi: 10.1109/JLT.2004.825240.

[10] C. Menyuk and B. Marks, "Interaction of Polarization Mode Dispersion and Nonlinearity in Optical Fiber Transmission Systems," J. Lightwave Technol. 24, 2806- (2006).

[11] R.H. Stolen and A. Ashkin , "Optical Kerr effect in glass waveguide," Applied Physics Letters 22, 294-296 (1973) https://doi.org/10.1063/1.1654644

[12] A.Argyris, J.Bueno, and I.Fischer, "Photonic machine learning implementation for signal recovery in optical communications," Sci Rep 8, 8487 (2018). https://doi.org/10.1038/s41598-018-26927-y

[13] C. Häger, H. Pfister, R. Bütler, G. Liga, and A. Alvarado, "Model-Based Machine Learning for Joint Digital Backpropagation and PMD Compensation," in Optical Fiber Communication Conference (OFC) 2020, OSA Technical Digest (Optical Society of America, 2020), paper W3D.3.

[14] Y. Inouye and Ruey-Wen Liu, "A system-theoretic foundation for blind equalization of an FIR MIMO channel system," in IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications, vol. 49, no. 4, pp. 425-436, April 2002, doi: 10.1109/81.995657.

[15] Y.LeCun, Y.Bengio, and G.Hinton, "Deep learning," Nature 521, 436–444 (2015). https://doi.org/10.1038/nature14539

[16] P.Ramachandran, B,Zoph, and QV.Le, "Searching for Activation Functions," January 2017. Accessed June 26, 2021. https://search.ebscohost.com

[17] L.Bottou, "(2012) Stochastic Gradient Descent Tricks," In: Montavon G., Orr G.B., Müller KR. (eds) Neural Networks: Tricks of the Trade. Lecture Notes in Computer Science, vol 7700. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-35289-8-25

[18] A. Ivutin and E. Larkin, "Estimation of latency in embedded real-time systems," 2014 3rd Mediterranean Conference on Embedded Computing (MECO), 2014, pp. 236-239, doi: 10.1109/MECO.2014.6862704.

[19] www.xilinx.com/products/silicon-devices/fpga/virtex-7.html#productTable