

© 2020 Xinying Wang

MACHINE LEARNING APPROACH FOR HIGH SPEED LINK MODELING  
AND IBIS-AMI MODEL GENERATION

BY

XINYING WANG

DISSERTATION

Submitted in partial fulfillment of the requirements  
for the degree of Doctor of Philosophy in Electrical and Computer Engineering  
Department  
in the Graduate College of the  
University of Illinois at Urbana-Champaign, 2020

Urbana, Illinois

Doctoral Committee:

Professor Jose E. Schutt-Aine, Chair  
Professor Andreas Cangellaris  
Professor Deming Chen  
Professor Elyse Rosenbaum

# Abstract

High speed link system is one of the major components in the networking infrastructure. Developing a high performance behavioral model for such system is crucial but challenging, especially when taking non-linearity into account. This work reports modeling high speed link (HSL) system using machine learning and implementing the model into IBIS-AMI, an industrial standard for SerDes simulation and verification. We started with developing a Volterra series model by extracting the Volterra kernels using feed forward neural network. We proposed a monomial power series neural network (MP-SNN) which can extract Volterra kernels that relate to non-linearity up to the third order. We developed an analytical mapping from neural network weights to Volterra kernels. The analytical mapping allows accurate time domain signal reconstruction with extracted Volterra kernels. We applied the MPSNN to model pulse amplitude modulation 4 level (PAM-4) and non-return-to-zero (NRZ) system. Volterra kernels up to the third order can be accurately identified.

The curse of dimensionality associated with Volterra series impedes the practical applications of the Volterra series. The number of Volterra kernels increase exponentially with the increase in memory length and the non-linearity order. The large number of Volterra kernels consume vast amount of computational power during signal reconstruction. To address this challenge, we proposed a Laguerre-Volterra feed forward neural network (LVFFN). The input time-series signal is orthogonalized, in other words, Laguerre-expanded, before it is feed to the neural network. The dimension of the input signal is significantly reduced, which results much less number of neurons in hidden layer. We modeled the PAM-4 and NRZ system with LVFFN. The resulted model has the number of parameters that are up to 6 order of magnitudes less than Volterra series. We could also model just the receiver instead of the

whole system to add more flexibility to the model in practical applications.

The LVFFN model greatly addressed the curse of dimensionality associated with Volterra series. Then the next question is how are we going to use it. Since the machine learning based model is not a standardized model, it is difficult to be co-simulated with models generated by other approaches. To circumvent the challenges in model transportability and interoperability, we implemented the LVFFN into IBIS-AMI, an industrial standard model that is compatible with most of the circuit simulators. We could simulate the LVFFN IBIS-AMI model in Keysight ADS and conduct the eye-diagram analysis.

IBIS-AMI model generation is not trivial. It requires cross-disciplinary knowledge in signal integrity, HSL circuit, and software engineering. To facilitate the process of IBIS-AMI model generation, we developed a software, ezAMI, that can generate the IBIS-AMI model by clicks. The software is developed using Qt/C++ and is an open-source software. The software architecture and tutorial are introduced in this thesis as well.

*To my wife, my son, and my parents, for their love and support.*

# Acknowledgments

First of all, I would like to thank my wife, Xiaoli, for her sacrifice and support along this long challenging process. Although I had so many challenges, difficulties, hard times, I was able to overcome them with her love and inspiration.

I would like to express my sincere gratitude to my advisor Prof. Jose E. Schutt-Aine for him bringing me into this exciting field. I thank him for the support of my Ph.D study and related research. His guidance helped me all through my research and this thesis.

Besides my advisor, I would like to thank the rest of my thesis committee: Prof. Rosenbaum, Prof. Cangelaris, and Prof. Chen, for their insightful comments and encouragement, as well as for the hard questions which drive me to widen my research from various perspectives.

My sincere thanks also go to Professor Pavan Hanumolu, Professor Zuofu Cheng, and Dr Chandrasekhar Radhakrishnan. I benefited a lot either from the classes they taught or being a TA to their classes.

Special thanks to my fellow group mate Thong Nugyen. I thank him for the help he gave me on my research, the constructive discussions, and the PAM-2 and PAM-4 data he provided.

I would like to thank my current and former fellow group mates, Dr. Xiao Ma, Dr. Da Wei, Yixuan (Nancy) Zhao, Gene Shiue, Bobi Shi, Juhitha Konduru for the help I received.

Special thanks to Karen L. Kuhns for her help on the office affairs and for her jokes.

I would like to thank James Hutchinson for his great editing work towards my paper draft and my thesis.

My thanks also go to the ECE department staff Jen, Pascal, and Laurie for helping me continue my Ph.D program.

Last but not the least, I would like to thank my parents for supporting me unconditionally throughout my Ph.D study and my life in general.

# Table of Contents

List of Figures . . . . .	ix
Chapter 1 Introduction and Background . . . . .	1
Chapter 2 Mathematical Formulation of MPSNN and LVFFN . . . . .	6
2.1 Volterra theory . . . . .	6
2.2 Monomial Power Series Neural Network . . . . .	8
2.3 Laguerre-Volterra Feed Forward Neural Network . . . . .	13
Chapter 3 Volterra Kernel Extraction Through MPSNN for Behavior Modeling of High Speed Link System . . . . .	18
3.1 Introduction . . . . .	18
3.2 Model Validation with Analytical Wiener System . . . . .	19
3.3 Modeling PAM-4 HSL System . . . . .	24
3.4 Modeling NRZ HSL System . . . . .	28
3.5 Conclusion and Future work . . . . .	32
Chapter 4 Approach of Laguerre-Volterra Feed Forward Neural Network for Modeling High Speed Links . . . . .	33
4.1 Introduction . . . . .	33
4.2 Modeling PAM-4 System with LVFFN . . . . .	37
4.3 Modeling NRZ System with LVFFN . . . . .	47
4.4 Modeling CMOS Inverter with LVFFN . . . . .	54
4.5 Conclusion . . . . .	55
Chapter 5 IBIS-AMI Model Generation and Simulation Environment . . . . .	57
5.1 IBIS-AMI Model Basics . . . . .	57
5.2 IBIS-AMI Implementation . . . . .	59
5.3 The ezAMI software . . . . .	62
5.4 Conclusion . . . . .	78
Chapter 6 Summary and Future Work . . . . .	80
6.1 Summary . . . . .	80
6.2 Future work . . . . .	83



Appendix A	ezAMI Development Environment Setup . . . . .	86
A.1	Installing Visual Studio 2019 . . . . .	86
A.2	Installing QT IDE 5.12 . . . . .	87
Appendix B	Tutorial for LVFFN example in ezAMI . . . . .	90
B.1	Install ezAMI Software . . . . .	90
B.2	LVFFN example . . . . .	91
Refernces	. . . . .	96

# List of Figures

1.1	HSL technology illustration . . . . .	1
2.1	FFN structure with one hidden layer . . . . .	9
2.2	LVFFN structure with one hidden layer . . . . .	17
3.1	Impulse response used in Wiener system example . . . . .	20
3.2	Training signals for the second order Wiener system example . . . . .	21
3.3	Extracted kernels for $2^{nd}$ order Wiener system example . . . . .	22
3.4	$2^{nd}$ order kernel of (a) Analytical system, (b)Extracted Using MPSNN, and (c)Extracted using traditional FFN . . . . .	23
3.5	A high-speed channel with I/O buffer . . . . .	24
3.6	Training signals in PAM4 buffer example. . . . .	24
3.7	Extracted kernels for PAM4 buffer example . . . . .	25
3.8	Output response of PAM4 channel under unseen PRBS excitation using extracted VK kernels. . . . .	27
3.9	Output response comparison between reference signal and VK model for PAM4 example . . . . .	28
3.10	Output response of PAM2 channel under unseen PRBS excitation using extracted VK kernels. . . . .	30
3.11	Extracted kernels for PAM2 buffer example . . . . .	31
4.1	Bit pattern for (a)NRZ and (b)PAM4 . . . . .	36
4.2	PAM-4 high speed link system block diagram . . . . .	38
4.3	Plot of PAM-4 input output data . . . . .	39
4.4	Plot of first five Laguerre functions . . . . .	40
4.5	Plots of MSE vs (a) $\alpha$ , and (b) number of Laguerre functions . . . . .	42
4.6	Model output comparison of LVFFN, RNN, FFN, and the reference signal . . . . .	43
4.7	Comparison of model size for Volterra series, RNN, FFN, and LVFFN . . . . .	44
4.8	Comparison of number of multiplications for calculation of one output sample for Volterra series, RNN, FFN, and LVFFN . . . . .	45
4.9	First order VK extracted with FFN and LVFFN . . . . .	46

4.10	NRZ waveform . . . . .	48
4.11	PAM-2 system LVFFN model prediction and reference waveform . . . . .	49
4.12	LVFFN NRZ system model size comparison for Volterra series, FFN, and LVFFN . . . . .	51
4.13	Number of multiplication comparison for Volterra series, FFN, and LVFFN . . . . .	52
4.14	PAM-2 receiver LVFFN model prediction and reference waveform . . . . .	53
4.15	Schematic of inverter being modeled . . . . .	54
4.16	Output signal of inverter output (blue) and prediction from LVFFN (orange) . . . . .	55
5.1	IBIS-AMI model diagram . . . . .	58
5.2	Snapshot of simulation environment in ADS for LVFFN IBIS-AMI model . . . . .	60
5.3	Flow diagram of Rx DLL execution . . . . .	61
5.4	Eye-diagram plot for (a)PAM-4 LVFFN model and (b)reference model . . . . .	62
5.5	ezAMI software architecture diagram . . . . .	64
5.6	Tree model diagram . . . . .	68
5.7	Code region . . . . .	70
5.8	ezAMI main interface . . . . .	73
5.9	Menu actions . . . . .	76
5.10	Excitation generation dialog . . . . .	78
6.1	RoadMap of current research . . . . .	81
6.2	Illustration of determination of $\alpha$ . . . . .	85
A.1	VS 2019 installation snapshot . . . . .	86
A.2	QT IDE download . . . . .	87
A.3	QT IDE installation login . . . . .	88
A.4	QT IDE installation setup . . . . .	88
A.5	Launch QT Creator . . . . .	89
A.6	QT Creator interface . . . . .	89
B.1	ezAMI installation . . . . .	90
B.2	Add schematic . . . . .	91
B.3	Launch LVFFN example . . . . .	91
B.4	Select folder to save LVFFN project . . . . .	92
B.5	Software main interface after LVFFN example is loaded . . . . .	92
B.6	Software main interface after LVFFN example is loaded . . . . .	93
B.7	Link dll file for simulation . . . . .	93
B.8	Excitation generation . . . . .	94
B.9	Result display . . . . .	94

B.10 Generate AMI model . . . . .	95
B.11 Generated AMI model . . . . .	95

# Chapter 1

## Introduction and Background

This chapter discusses advancements in high speed link (HSL) technology, behavioral modeling and signal integrity, neural networks, and IBIS-AMI. Challenges associated with HSL modeling and verification are also reviewed. The structure of dissertation is outlined at the end of chapter.

The growth of new technologies such as big data, cloud computing, the Internet-of-Things, 5G, and artificial intelligence, has driven demand for instant data exchange around the world. High speed link(HSL) technology's limitations in data transmission is one of the bottle-neck threatens the entire infrastructure. Figure 4.1 demonstrates the general architecture of a HSL I/O buffer.

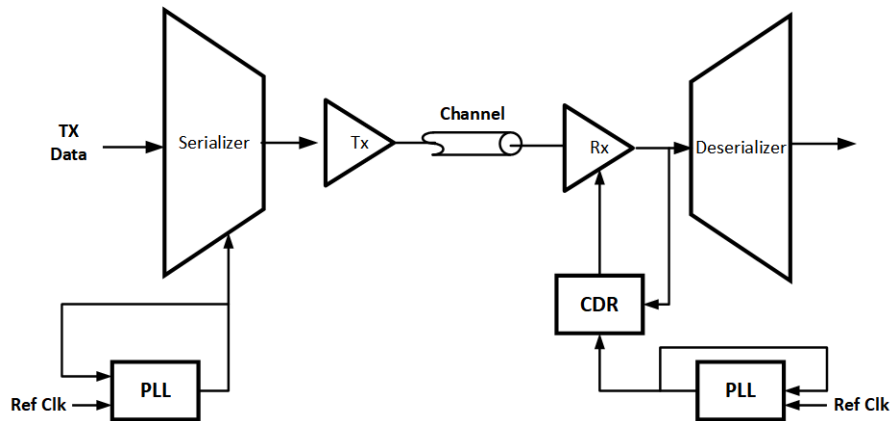


Figure 1.1: HSL technology illustration

HSL technology can be divided into two categories: Non-Return-to-Zero(NRZ), also known as pulse amplitude modulation 2-level(PAM-2) and pulse amplitude modulation 4-level(PAM-4). Over the past five decades, NRZ technology has been evolving from 10Gb/s to current 100Gb/s and it will continue

to scale up to next generation 400Gb/s HSL technology. PAM-4 is an emerging technology targeting 400Gb/s (8lanes, 56Gb/s/lane). Each technology presents advantages and challenges. NRZ technology has two levels, -1 and 1. Each level represents one bit. Therefore NRZ has only one eye in its eye diagram plot. The advantage of NRZ technologies is maturity: it has evolved over decades. There are many design references and considerable documentations available for designers/engineers. However there are also challenges when linearly scaling up from 100Gb/s to 400Gb/s. At 400Gb/s, these include totally closed eye, shorter unit interval(UI), and tighter jitter requirements. The eye closing issue may be addressed by applying more aggressive equalization technologies such as improved Decision-Feedback Equalizer(DFE) and Continuous Time Linear Equalizer (CTLE). The shorter UI, only 17ps at 400Gb/s, is more challenging because of the resulting tighter jitter budget which may be below the intrinsic jitter budget of the testing or support equipments.

PAM-4, on the other hand, will alleviate some of challenges of NRZ technology such as the tighter jitter budget and worse channel loss etc. PAM-4 technology has four digital amplitude levels(-3, -1, 1 and 3). Each symbol carries 2-bit information. This means that at the same throughput as NRZ, PAM-4 only requires half of the bandwidth. This addresses the shorter jitter budget problem that NRZ technology faces. However it alters the design, testing, and verification of the system due to multi-level signaling. Unlike NRZ, for which there is decades of experience on which to back new developments, there is limited information on PAM-4 to which designers and testers can refer. Therefore, instead of designing and building PAM-4 testing and measurement equipment, simulating and validating PAM-4 system is crucial.

Further more, behavioral modeling is needed to validate the performance of a newly designed serial link. In order to conduct simulation and analysis, a system developer has to extract a generic model from either a SPICE model or system measurements. Another option directly conducts simulation on the original SPICE model. This second option would impose challenges on HSL IP vendors since most are unwilling to disclose their designs. In addition, a specialized simulation environment would have to be provided together with their SPICE models. Therefore, in most of cases, the first option, supplying

a generic model (i.e., a behavioral model), is a viable solution. A behavioral model, as its name indicates, should capture all the behaviors precisely with pre-defined input while hiding design details. Typically, the model should have interoperability in the sense that models supplied by different vendors can be simulated together without a specialized environment. Therefore, developing a versatile behavioral model for HSL is desirable.

At present, HSL behavioral modeling is dominated by IBIS-AMI model[1, 2]. The IBIS advanced technology modeling committee sketched up a specification for HSL modeling. The specification is called the Algorithmic Modeling Interface (AMI). This model is specifically for SerDes modeling and performance verification. It delivers executables in the form of Dynamically Link Library (DLL) together with IBIS electrical specifications. The advantages of IBIS-AMI include the following:

- Fast simulation—The model can simulate one million bits in under 5 minutes, especially for LTI systems.
- Excellent IP protection—The vendor can decide what details to reveal to users.
- Interoperability and transportability—The models from different vendors can simulate together and the same model can be simulated in different simulation environments.
- Accuracy—The accuracy depends on which model vendors enclose in the standard. It can be very accurate.

With the aforementioned advantages, IBIS-AMI has gained popularity and has dominated in HSL behavioral modeling, especially for the emerging PAM-4 technology. However, generating an IBIS-AMI model for HSL systems is not trivial. It could take several month or even a year to close the development loop. To address this issue, simulator vendors have provided tools for HSL IP vendors to generate their own model. IBIS-AMI model generated with these tools are typically derived from pulse responses and the simulation outputs are the superposition of the responses. However, when the LTI rule is violated, the prediction accuracy diminishes.

Developing a behavioral model for HSL system that incorporates the nonlinear effects and time-varying properties is attracting increasing attention. Research on the Volterra series, feed forward neural network (FFN), recurrent neural network (RNN), and system identification have been reported. The Volterra series is a versatile mathematical model for nonlinear time-varying systems. Considerable research has been reported on applying Volterra series to model various nonlinear systems, e.g. design of mechanical systems[3, 4] and control system[5]. Telescu et.al. have reported modeling a IC buffer with the Volterra series. In their work, the Volterra model is more accurate than traditional approaches[6].

System identification is another popular approach for HSL behavior modeling. This approach uses measured input/output pairs to produce a map for a set of explanatory and predicted variables[7]. There are two type of system identification models: linear and nonlinear. Linear system models include AutoRegressive eXternal input (ARX) model and an AutoRegressive Moving Average External input (ARMAX) model. Nonlinear system identification models are structured as ARX and ARMAX, but the internal structure is remodeled into a FFN structure. Nonlinear system identification models are more popular than the linear ones. Li et.al. have reported modeling SerDes link using linear and nonlinear system identification models. The performance is better than that of the RNN models[8]. Choi et.al. have also reported using system identification approach to model HSL[9]. Their results show that model generated from system identification can significantly reduce simulation time.

Growing interests on applying machine learning approaches to model HSL buffers and other electronic systems are indicated by the increasing number of new publications on this topic. Neural networks are not a new topic. They have long been used to model electronic systems. However, due to computational resources limit and the lack of an efficient optimization/training algorithm, the method has been growing slowly. With recent advancements in back-propagation [10] training algorithms, deep neuron network has progressed in a variety of fields. Recent publications on modeling HSL system and other electronic circuit/system is an evidence that deep neural network is thriving in the field of electronic system and IC design[11, 12, 13, 14, 15].



In this dissertation, we report modeling HSL buffer with LVFFN and implementing the model in IBIS-AMI, an industrial standard for SerDes simulation and verification. We aim to develop a compact behavioral model for HSL buffer to conduct rapid and accurate signal integrity verification. This model will be compatible with main stream EDA softwares to enable simulation and analysis under the universal industrial production environment. The structure of this dissertation is structured as follows: Chapter 2 discusses the mathematical formulation for MPSNN and LVFFN. Chapter 3 models the PAM-4 and NRZ system using MPSNN. Volterra kernel identification via MPSNN is discussed as well. Chapter 4 discusses modeling PAM-4 and NRZ HSL system and a COMS inverter with LVFFN. Dimension reduction with the proposed LVFFN is discussed as well. In chapter 5, the LVFFN PAM-4 model is implemented in IBIS-AMI and is simulated in commercial circuit simulator. IBIS-AMI model generation software, ezAMI, is also presented in this chapter. The last chapter, chapter 6, summarizes the research progress and proposes the future work.

# Chapter 2

## Mathematical Formulation of MPSNN and LVFFN

### 2.1 Volterra theory

Volterra series is a versatile mathematical model describing nonlinear system with memory. Successful applications in modeling power amplifiers [16, 17], analog integrated circuits [18], image processing [19], coupled devices and circuits [20], nonlinear equalizers [21], and filters [22] have been reported.

A non-linear time-invariant (NLTI) system is considered weakly non-linear if it processes fading memory property. That is, the present output does not depend on the infinitely long past [23]. For a weakly non-linear time-invariant (NLTI) system, memory effects can be well approximated by an N-term truncated Volterra series [24], the output response  $y(t)$  of the system under the input excitation  $x(t)$  is

$$y(t) = \sum_{n=1}^N y_n(t) \quad (2.1)$$

$$y_n(t) = \int_{[-\infty, t]^n} h_n(\tau_1, \tau_2, \dots, \tau_n) \prod_{i=1}^n x(t - \tau_i) d\tau_i \quad (2.2)$$

where  $h_n(\tau_1, \tau_2, \dots, \tau_n)$  is the  $n^{\text{th}}$  order Volterra kernel (VK). A frequency-domain VK, or generalized frequency response function (GFRF) [25], can be obtained by multi-dimensional Fourier transform.

$$H_n(\omega_1, \omega_2, \dots, \omega_n) = \int_{\mathbb{R}^n} h_n(\tau_1, \tau_2, \dots, \tau_n) \times \exp\left(-j \sum_{k=1}^n \omega_k \tau_k\right) \prod_{i=1}^n d\tau_i \quad (2.3)$$

Previous works have used GFRF for distortion and inter-modulation anal-

ysis [26, 27]. Inversely, time-domain kernels can be recovered by multi-dimensional inverse Fourier transform

$$h_n(t_1, t_2, \dots, t_n) = \int_{\mathbb{R}^n} H_n(\omega_1, \omega_2, \dots, \omega_n) \times \exp\left(j \sum_{k=1}^n \omega_k t_k\right) \prod_{i=1}^n d\omega_i \quad (2.4)$$

It is worth noting that Volterra kernels representing a system are, in general, not unique [28, 24]. However, it can be seen in Equation (2.2) that any permutation of  $\tau$ 's will leave the output  $y(t)$  unchanged. Thus, symmetrical kernels are unique, mathematically, it is found by taking the average over all  $n!$  possible permutations  $\pi$  of  $\tau$ 's.

$$h_{n_{sym}}(\tau_1, \tau_2, \dots, \tau_n) = \frac{1}{n!} \sum_{\pi} h_{n_{asym}}(\tau_{\pi(1)}, \tau_{\pi(2)}, \dots, \tau_{\pi(n)}) \quad (2.5)$$

For simplicity, all VK notations used in this paper refer to the symmetric kernel though the subscript ‘‘sym’’ is omitted. In discrete time, the contribution from  $n^{th}$  order Volterra kernel response is given by the  $n^{th}$ -dimensional discrete convolution

$$\begin{aligned} y(n) = & h_0 + \sum_{m=0}^M h_1(m)u(n-m) \\ & + \sum_{m_1=0}^M \sum_{m_2=0}^M h_2(m_1, m_2)u(n-m_1)u(n-m_2) \\ & + \sum_{m_1=0}^M \sum_{m_2=0}^M \sum_{m_3=0}^M h_3(m_1, m_2, m_3)u(n-m_1) \\ & \times u(n-m_2)u(n-m_3) \\ & + \dots \end{aligned} \quad (2.6)$$

where  $n$  denotes the time step,  $u(n)$  the input data sequence,  $y(n)$  the output data sequence,  $M$  the system memory length and  $h_i$  the  $i^{th}$ - order Volterra kernels (VKs). Equation (2.6) can be rewritten in the form

$$\mathbf{Y} = \mathbf{U}\mathbf{h} \quad (2.7)$$

where  $U$  denote the input Matrix with memory  $M$  and order  $N$ .  $Y$  denotes the output vector:

$$\begin{aligned}
\mathbf{Y} &= [y_1, y_2, \dots, y_n]^T \\
\mathbf{U} &= [\mathbf{u}_1, \mathbf{u}_2, \dots, \mathbf{u}_n]^T \\
\mathbf{u}_n &= [u_n(0), \dots, u_n(m), u_n(0)u_n(0), u_n(0)u_n(1), \dots] \\
\mathbf{h} &= [h_0, h_1(0), h_1(1), \dots, h_1(m), h_2(0, 0), h_2(0, 1), \dots]
\end{aligned} \tag{2.8}$$

The VKs can be estimated with the least mean square (LMS) method:

$$\mathbf{h} = (\mathbf{U}'\mathbf{U})^{-1}\mathbf{U}'\mathbf{Y} \tag{2.9}$$

The number of kernel values needed to be estimated is a function of the memory length and the system order. The curse of dimensionality for Volterra series has greatly limited its applications. For example, to model a second order system with memory length of 50, Volterra series would need total  $50 \times 50 + 50 = 2,550$  kernel values, and a fourth order system with the same memory length requires 6,377,550 kernel values. The number of kernel values increases exponentially with the system order and memory length. The total number of kernel values for a  $N$ th order system with memory length  $M$  can be expressed as below:

$$\frac{(N + M)(N + M - 1)\dots(M + 1)}{N!} \tag{2.10}$$

At the certain point, the number of kernel values exceeds the number of data available for kernel extraction, which makes the problem intractable due to the vast number of parameters. This limitation leads to computational latency and over-fitting.

## 2.2 Monomial Power Series Neural Network

For discretized input-output data, the Volterra series of causal, stable, non-linear, time-invariant system is given by

$$\begin{aligned}
y(n) = & h_0 + \sum_{m=0}^M h_1(m)x(n-m) \\
& + \sum_{m_1=0}^M \sum_{m_2=0}^M h_2(m_1, m_2)x(n-m_1)x(n-m_2) \\
& + \sum_{m_1=0}^M \sum_{m_2=0}^M \sum_{m_3=0}^M h_3(m_1, m_2, m_3)x(n-m_1) \\
& \times x(n-m_2)x(n-m_3) \\
& + \dots
\end{aligned} \tag{2.11}$$

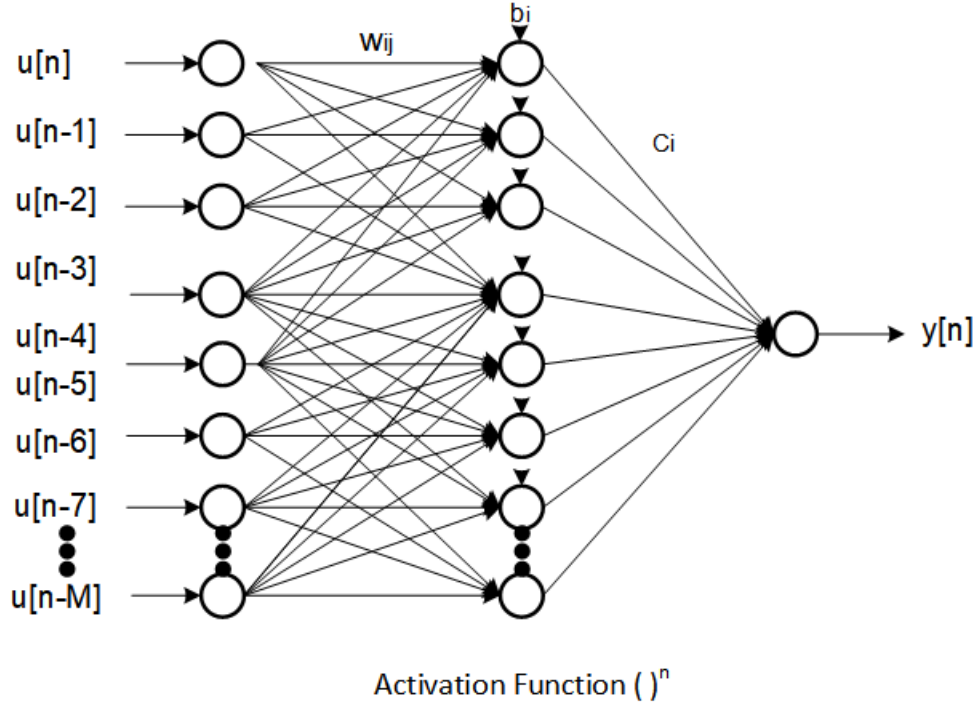


Figure 2.1: FFN structure with one hidden layer

where  $n$  denotes the time step,  $x(n)$  the input data sequence,  $y(n)$  the output data sequence,  $M$  the system memory length and  $h_i$  are the  $i^{th}$ - order Volterra kernels to be identified. Identification of VKs can be achieved by a variety of methods including the method extraction of kernels from a trained Feed-forward neural network(FFN). A detailed discussion can be found in [29, 30, 31, 32]. Such method employs a FFN with one input layer, one

hidden layer, and one output layer with single output node. The number of neurons in hidden layer is the same as the system memory length. Figure 2.1 shows the FFN structure. The input-output mapping through FFN is given by

$$y(t) = \sum_{i=1}^M c_i \sigma_i \left( b_i + \sum_{j=0}^M w_{ij} u(t-j) \right) \quad (2.12)$$

Where  $y(t)$  is the output,  $u(t)$  the input,  $c_i$  the weight from hidden unit  $i$  to the output,  $w_{ij}$  the weight from input  $i$  to hidden unit  $j$ , and  $\sigma_i$  is the activation function. The activation function, if infinitely differentiable, can be represented by an equivalent truncated polynomial function  $g_i$  given by

$$g_i(x) = \sum_{j=0}^M a_{ji} x^j \quad (2.13)$$

where  $x^j$  is given by

$$x^j = u(t)^{v_0} u(t-1)^{v_1} \dots u(t-M)^{v_M} \Big|_{v_0+v_1+\dots+v_M=j} \quad (2.14)$$

Combining Equations (2.13) and (2.14) and plugging into (2.12), we can represent the output  $y(t)$  in the form

$$\begin{aligned} y(t) = & \sum_{i=1}^M c_i a_{0i} + \sum_{i=1}^M c_i a_{1i} w_{ij} u(t-j) + \dots \\ & + \sum_{i=1}^M c_i a_{ni} w_{v_1 i} w_{v_2 i} \dots w_{v_n i} \\ & \times u(t-v_{1i}) u(t-v_{2i}) \dots u(t-v_{ni}). \end{aligned} \quad (2.15)$$

Comparing (2.15) to (2.11), the Volterra kernel can be represented by

$$k_0 = \sum_{i=1}^M c_i a_{0i} \quad (2.16)$$

$$k_1(j) = \sum_{i=1}^M c_i a_{1i} w_{ji} \quad (2.17)$$

$$k_2(j, k) = \sum_{i=1}^M c_i a_{2i} w_{ji} w_{ki} \quad (2.18)$$

$$k_n(v_1, v_2, \dots, v_n) = \sum_{i=1}^M c_i a_{ni} w_{v_1 i} w_{v_2 i} \dots w_{v_n i} \quad (2.19)$$

where  $k_n$  is the  $n^{\text{th}}$  Volterra kernel,  $c_i$  and  $w_{ni}$ . The weights obtained from the FFN (see Figure 2.1), and  $a_{ni}$  are the parameters in equation (2.13). Obtaining the correct  $a_{ni}$  is crucial for accurately mapping the FFN weights to the Volterra kernels. The existing method uses an hyperbolic tangent activation function,  $\sigma(x) = \tanh(x)$ , in FFN.  $a_{ni}$  can be obtained from the Taylor expansion of  $\tanh(x)$ , which is given by

$$\tanh(x) = x - \frac{1}{3}x^3 + \frac{2}{15}x^5 - \frac{17}{315}x^7 + \dots \quad (2.20)$$

Plug (2.14) into (2.20), the nodal output function is given below

$$\begin{aligned} \tanh(x) &= \left( b_i + \sum_{j=0}^M w_{ij} u(t-j) \right) \\ &\quad - \frac{1}{3} \left( b_i + \sum_{j=0}^M w_{ij} u(t-j) \right)^3 \\ &\quad + \frac{2}{15} \left( b_i + \sum_{j=0}^M w_{ij} u(t-j) \right)^5 + \dots \end{aligned} \quad (2.21)$$

The coefficients  $a_{ni}$  can be obtained by expansion of (2.21)

$$a_{ni} = \sum_{k=n}^{\infty} C_n^k d_k b_i^{k-n} \quad (2.22)$$

where  $C_n^k$  is the combination given by  $\frac{k!}{(k-n)!n!}$ ,  $d_k$  denotes the coefficient of the  $k$ th power in the Taylor expansion of Hyperbolic Tangent activation function, and  $b_i$  is the bias value for the  $j^{\text{th}}$  hidden node. The critical factor on the convergence of the series is the bias value  $b_i$ , which has to be in the range of  $[-\pi/2, \pi/2]$ . Although, the value  $b_i$  is usually within this range, this cannot be guaranteed which can result in significant deviation of  $a_{ni}$  from the correct value.

An alternative calculation of  $a_{ni}$  is also proposed in [29]. It is given by

$$a_{ni} = \frac{1}{n} \tanh^{(n)}(b_i) \quad (2.23)$$

Where  $\tanh^{(n)}(x)$  is the  $n^{\text{th}}$ - derivative of  $\tanh(x)$ ,  $b_i$  is the bias value of the  $i$ th hidden node. Although, this method greatly alleviates the complexity in calculating  $a_{ni}$ , it introduces a truncation error when applied to a Taylor series around  $b_i$  to approximate  $\tanh(x)$ .

The method proposed in this paper to obtain the coefficient  $a_{ni}$  lies on the direct expansion of activation function which is monomial power series activation function. The new activation function is given by

$$\sigma(x) = x^n \quad (2.24)$$

the nodal output function can be obtained by replacing  $x$  with (2.14) and given by

$$\sigma(x) = \left( b_i + \sum_{j=0}^M w_{ij} u(t-j) \right)^n \quad (2.25)$$

To simplify the derivation, we can represent  $w_{ij}u(t-j)$  with  $u_j$  and  $b_i$  with  $b$ . The simplified equation (2.25) is given by

$$\sigma(x) = (b + u_0 + u_1 + \dots u_M)^n \quad (2.26)$$

Direct expansion of (2.26) is straightforward. The coefficient  $a_{ni}$  can be obtained from the  $n^{\text{th}}$ - power term in the expansion, which is given by

$$a_{ni} = b_i^{k_b} \binom{n}{k_0, k_1, k_2, \dots, k_M}_{k_b + k_0 + \dots + k_M = n} \quad (2.27)$$

Where  $n$  is the order of the kernel,  $k_b$  the order of  $b$  in the expansion of (2.26), and  $k_0, k_1, \dots, k_M$  are the order of  $u_0, u_1, \dots, u_M$ , respectively. The sum of all parameter  $k$  has to be equal to the order  $n$

The proposed method employs the same FFN structure in which only one hidden layer is used. Promisingly, it can be adapted to allow multiple hidden layer to achieve higher order Volterra kernel identification. The proposed method has two advantages

- i it allows analytical mapping from learned weights to Volterra kernels.
- ii it significantly reduces the number parameters to be identified by virtue



of the fact that the order of the kernel is not aligned with the system memory length.

Since there is no truncation needed in the activation function expansion, this proposed method produces more accurate kernels. The number of parameters will be regulated by the order of the activation function, thus the total number of parameters needed to be identified is reduced significantly. For instance, the system requires memory length of 100 for FFN to be well trained. Then, with the existing method, the order of the Volterra kernel to be identified has to be up to 100th. The total parameters to be identified can be calculated by 2.10. The total number of parameters is too large to be practical for real applications. With our method, if a third order activation function is employed, the total number of parameters needed for identification can be reduced from 100 to 3 which is more practical for real applications.

## 2.3 Laguerre-Volterra Feed Forward Neural Network

To reduce the curse of dimensionality associated with Volterra series, one of the solution is projecting VKs on an set of orthonormal basis functions [33]. Laguerre functions are among the most popular ones. Laguerre functions are the solutions of Laguerre differential function. The general form of discrete Laguerre functions are shown below [34]:

$$\phi_r(\tau) = \alpha^{\frac{\tau-r}{2}} (1-\alpha)^{\frac{1}{2}} \sum_{k=0}^r (-1)^k \binom{\tau}{k} \binom{r}{k} \alpha^{r-k} (1-\alpha)^k \quad (2.28)$$

The VKs are expandable using Laguerre functions up to  $R^{th}$  order

$$\begin{aligned} h_0 &= \theta_0 \\ h_1(\tau) &= \sum_{r=1}^{r=R} \theta_r \phi_r(\tau) \\ h_2(\tau_1, \tau_2) &= \sum_{r_1=1}^{r_1=R} \sum_{r_2=1}^{r_2=R} \theta_{r_1, r_2} \phi_{r_1}(\tau_1) \phi_{r_2}(\tau_2) \\ h_n(\tau_1, \dots, \tau_n) &= \sum_{r_1=1}^{r_1=R} \dots \sum_{r_n=1}^{r_n=R} \theta_{r_1, \dots, r_n} \prod_{l=1}^n \phi_l(\tau_l) \end{aligned} \quad (2.29)$$

where  $\phi_r(\tau)$  denotes the discrete  $r^{\text{th}}$  orthonormal basis function. Plugging equation (2.29) into equation (2.6):

$$\begin{aligned}
y(n) &= \theta_0 + \sum_{r=1}^R \theta_r \sum_{\tau=0}^M \phi_r(\tau) u(n - \tau) \\
&+ \sum_{r_1=1}^{r_1=R} \sum_{r_2=1}^{r_2=R} \theta_{r_1, r_2} \sum_{\tau_1=0}^M \phi_{r_1}(\tau_1) u(n - \tau_1) \sum_{\tau_2=0}^M \phi_{r_2}(\tau_2) u(n - \tau_2) \\
&+ \dots \\
&+ \sum_{r_1=1}^{r_1=R} \dots \sum_{r_n=1}^{r_n=R} \theta_{r_1, \dots, r_n} \sum_{\tau_1=0}^M \phi_{r_1}(\tau_1) u(n - \tau_1) \dots \\
&\dots \sum_{\tau_n=0}^M \phi_{r_n}(\tau_n) u(n - \tau_n)
\end{aligned} \tag{2.30}$$

In equation (2.30), the term  $\sum_{\tau=0}^M \phi_r(\tau) u(n - \tau)$  is nothing but the convolved output of the  $r^{\text{th}}$  order Laguerre function and the time series input signal. Use symbol  $\ell$ :

$$\ell_r = \sum_{\tau=0}^M \phi_r(\tau) u(n - \tau) \tag{2.31}$$

Plugging equation (2.31) into equation (2.30):

$$\begin{aligned}
y(n) &= \theta_0 + \sum_{r=1}^R \theta_r \ell_r + \sum_{r_1=1}^{r_1=R} \sum_{r_2=1}^{r_2=R} \theta_{r_1, r_2} \ell_{r_1} \ell_{r_2} \\
&+ \dots + \sum_{r_1=1}^{r_1=R} \dots \sum_{r_n=1}^{r_n=R} \theta_{r_1, \dots, r_n} \prod_{i=1}^n \ell_{r_i}
\end{aligned} \tag{2.32}$$

Equation (2.32) has the same form as equation (2.6). Parameter identification shifts from VKs  $h_n$  in equation (2.6) to Laguerre parameters  $\theta_r$  in equation (2.32). The number of parameters to be identified has been reduced dramatically. Typically the number of Laguerre functions  $R$  employed for expansion is much smaller than the memory length  $M$  which determines the number of kernel values. 2.1 and 2.2 illustrate the dimension reduction between Volterra series and Laguerre-Volterra expansion.

Table 2.1: Number of parameters required to model system with order up to 4<sup>th</sup> and memory length up to 40 using Volterra Series

	1 <sup>st</sup> order	2 <sup>nd</sup> order	3 <sup>rd</sup> order	4 <sup>th</sup> order
M = 10	10	110	1,110	11,110
M = 20	20	420	8,420	168,420
M = 30	30	930	27,930	837,930
M = 40	40	1,640	641,640	3,201,640

Table 2.2: Number of parameters required to model system with order up to 4<sup>th</sup> and memory length up to 40 using Laguerre expansion

	1 <sup>st</sup> order	2 <sup>nd</sup> order	3 <sup>rd</sup> order	4 <sup>th</sup> order
R = 2	2	6	14	30
R = 3	3	12	39	120
R = 4	4	20	84	340
R = 5	5	30	155	780

Laguerre parameter identification is crucial. LMS has been widely used to obtain the Laguerre parameters. However, this method relies on matrix inversion and consequently becomes unpractical when the parameter dimension becomes large. The other method is through feed-forward neural network. Geng et al. proposed a recurrent Laguerre-Volterra network which can identify the Laguerre parameters and even the decay factor  $\alpha$  associated with Laguerre functions [35]. Such method has been successfully applied to model biological systems such as brain neuron cells. This method is specifically proposed for biological signals which have dynamics in the scale of several tens of Hz. However, it encounters difficulty when modeling an HSL system which usually runs at several tens of GHz or even hundreds of GHz. The latency depends largely on the channel length. Therefore, a novel parameter identification method is desirable for HSL systems.

In our previous works, we proposed a MPSNN to identify VKs for PAM-2 and PAM-4 HSL system [13][15]. The advantages of the proposed method are that it (1) allows the analytical mapping of trained weights to VKs, which produce higher accuracy and (2) reduces the number of parameters to be identified and enable reconstruction of the signal through extracted kernels. This method can be adapted to identify all the Laguerre parameters  $\theta_r$  in equation (2.30).

The proposed method employs the same feed-forward neural network structure in which only one hidden layer is used. Promisingly, it can be adapted to allow multiple hidden layers to achieve higher order Volterra kernel identification. Since there is no truncation needed in the activation function expansion, our method produces more accurate kernels. The number of parameters will be regulated by the order of the activation function, so the total number of parameters to be identified is reduced significantly.

Identification of Laguerre parameters is the same as Volterra kernel identification except that an extra layer is added between the input layer and the hidden layer to compute the convolution of the input with the Laguerre functions. The neural network used for identification of Laguerre parameters is illustrated in 2.2. The added layer which is labeled in the red frame does not change the derivation above for parameter identification. The Laguerre parameter  $\theta_r$  takes the general form:

$$\theta_{r_1, r_2, \dots, r_n} = \sum_{i=1}^R c_i a_{ni} w_{r_1 i} w_{r_2 i} \dots w_{r_n i} \quad (2.33)$$

where  $a_{ni}$  is calculated through equation (2.27);  $R$  denotes the number of Laguerre functions employed for expansion. In this work, we choose the third order monomial power series,  $\sigma = x^3$ , as activation function. Laguerre parameters up to the third order can be calculated through equation (2.34) to (2.37):

$$\theta_0 = \sum_{i=1}^R c_i b_i^3 \quad (2.34)$$

$$\theta_{r_1} = \sum_{i=1}^R c_i b_i^2 w_{r_1 i} \quad (2.35)$$

$$\theta_{r_1, r_2} = \sum_{i=1}^R c_i b_i w_{r_1 i} w_{r_2 i} \quad (2.36)$$

$$\theta_{r_1, r_2, r_3} = \sum_{i=1}^R c_i w_{r_1 i} w_{r_2 i} w_{r_3 i} \quad (2.37)$$

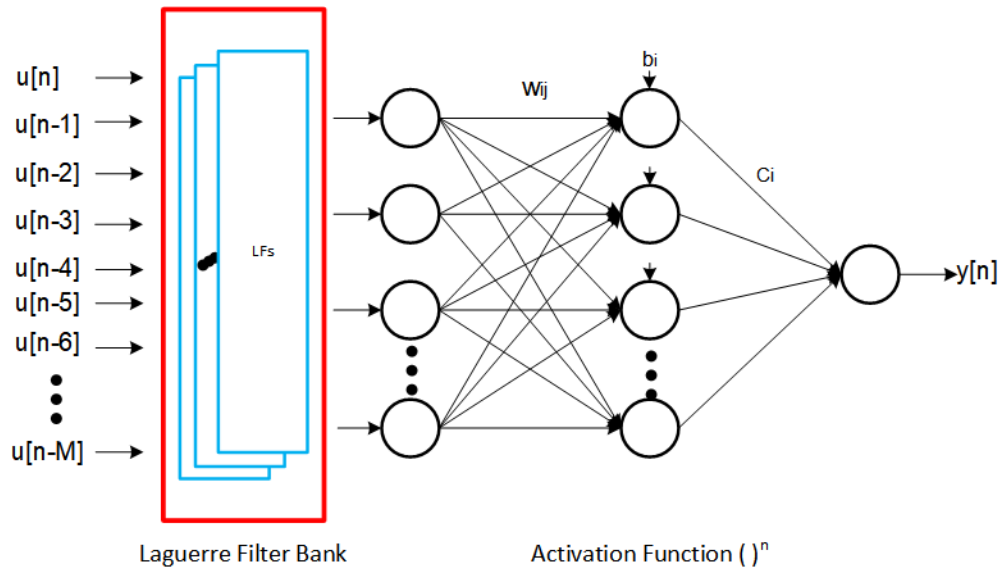


Figure 2.2: LVFFN structure with one hidden layer

## Chapter 3

# Volterra Kernel Extraction Through MPSNN for Behavior Modeling of High Speed Link System

### 3.1 Introduction

Signal integrity (SI) and Electromagnetic compatibility (EMC) verification for integrated circuit (IC) design has become increasingly important in the design of high-speed systems. The numerical model that represents the high speed circuit or channel plays a key role in the assessment of SI/EMC for early stage error screening. The modeling approach must be both accurate and efficient and possess the ability to protect the intellectual property (IP). Behavioral modeling satisfy the above requirements. Such models are usually based on simplified equivalent circuits model and the Input/output Buffer Information Specifications (IBIS)[36]. To generate these models, either measurement or translation of other simulation model is needed. Volterra series is one of the mathematical models that can be used to represent high speed circuit buffer to generate IBIS model.

Volterra series is a versatile mathematical model describing nonlinear system with memory. Successful applications in modeling power amplifier [37, 38], analog integrated circuit[39], image processing [40], coupled device and circuit [41], nonlinear equalizer [42], and filters [3] have been reported. One of the challenges in Volterra series is VK identification, particularly for higher order kernels. There has been a variety of identification methods proposed. Identification of VKs using artificial neural network, proposed by [29], provides a efficient way to identify VKs up to an order equal to the memory length.

Applications of such method in power amplifier [30], nonlinear circuit component [31], and nonlinear wireless communication device [32] behavioral modeling have been reported. However in such methods, extraction of kernels relies on approximation of hyperbolic tangent (HBT) activation function

by Taylor expansion. Such approximation imposes significant amount of uncertainty which results in less accurate kernel extraction. In addition, the order of the kernel in the extraction is determined by the memory length, which is typically in the range of several tens, even hundreds, for extended memory nonlinear system. The large number parameters identified in such high order Volterra system impedes the practical use of such system in real applications.

Neural network has been used to model input/output mapping for a long while. In the past, there were attempts to use neural network to extract Volterra kernels using input/output data [29, 30, 32]. However, due to limitation of computational resources and the fact that training a neural network was not efficient, the method does not gain much popularity. With recent advancement in deep learning, training a neural network with back-propagation [10] is more efficient and faster compared to the old method such as the Levenberg-Marquardt algorithm. In this paper, we propose to leverage modern development of deep learning to extract Volterra kernels for I/O buffers. Once the Volterra kernels are extracted, time domain simulation of high-speed channels can be performed accurately.

### 3.2 Model Validation with Analytical Wiener System

The proposed method is now tested using an analytical system describes by

$$y(t) = \left( \int_{-\infty}^t h(t - \tau) x(\tau) d\tau \right)^2 \quad (3.1)$$

where  $h(t) = h_0 e^{-kt} \sin(\omega t)$ . The response for  $h_0 = 4$ ,  $\omega = 0.5$  and  $k = 1$  is shown in 3.1. Notice that this system is a  $2^{nd}$  order Wiener system. It can be shown that for a Wiener system, the  $n^{th}$ -order VK can be derived from the LTI response. In particular,

$$h_n(\tau_1, \tau_2, \dots, \tau_n) = \prod_{i=1}^n h(\tau_i) \quad (3.2)$$

The MPSFNN is trained with a white-noise signal input with magnitude of zero mean and 1.0 deviation  $\{\chi \sim \mathcal{N}(0, 1.0)\}$ . The input and its correspond-

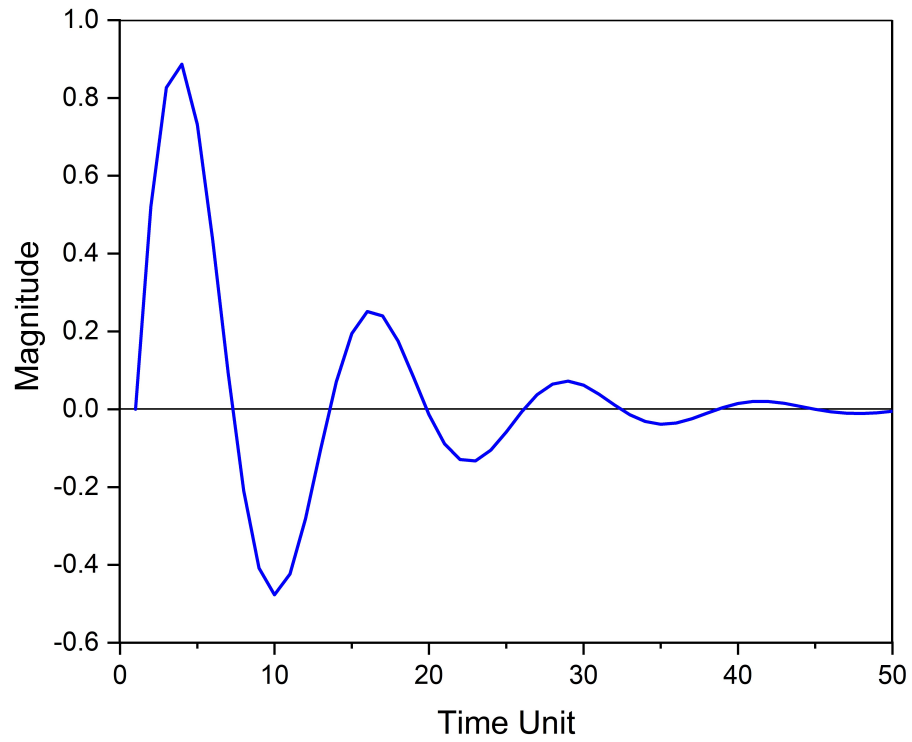


Figure 3.1: Impulse response used in Wiener system example

ing output as shown in 3.2.



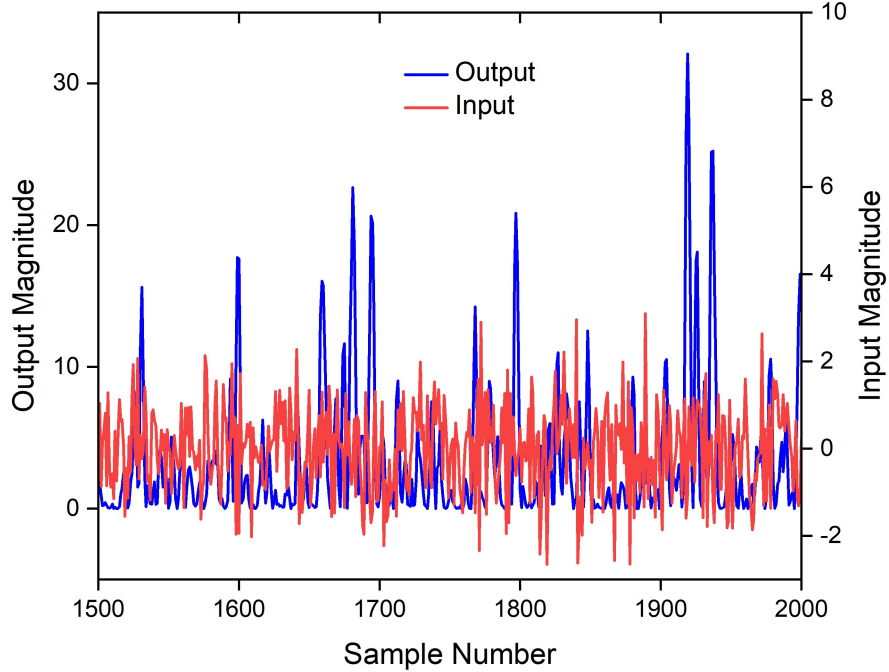
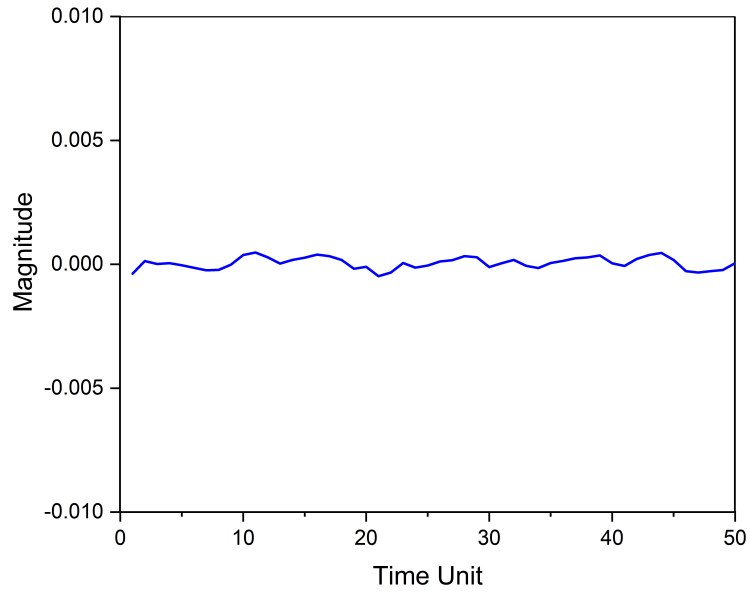


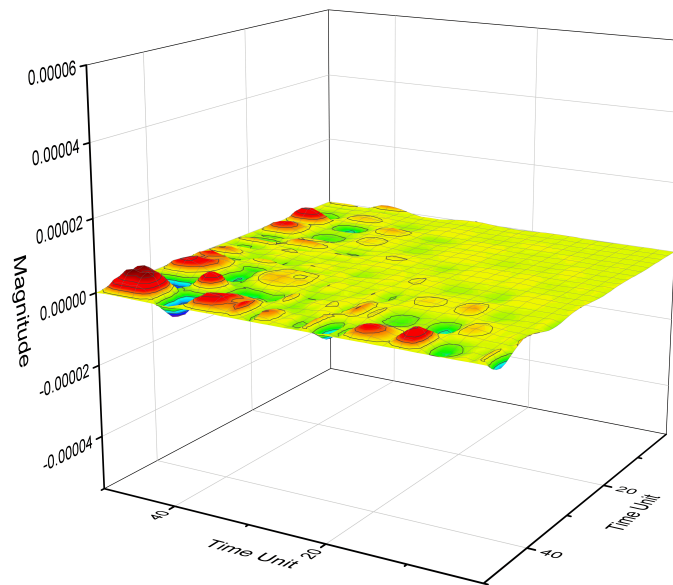
Figure 3.2: Training signals for the second order Wiener system example

The activation function for MPSNN is chosen to be  $\sigma = ()^3$ , the Volterra kernel can be identified with such network is higher than the non-linear system under modeling. Thereby, the first and third order kernels are expected to be zero. As expected, the extracted first and third Vks are shown in Figure3.3 and they are statistically close to zero.

Figure3.4 shows comparison between the analytical kernel (Figure3.4(a)), extracted kernel from MPSNN (Figure3.4(b)), and extracted kernel from traditional FFN(Figure3.4(c)) As shown, the method is able to capture the exact nonlinear order of the system. Only the second order kernel is significant and matches the analytical kernel. However the kernel extracted with the traditional FFN appears discrepancy from the correct one.

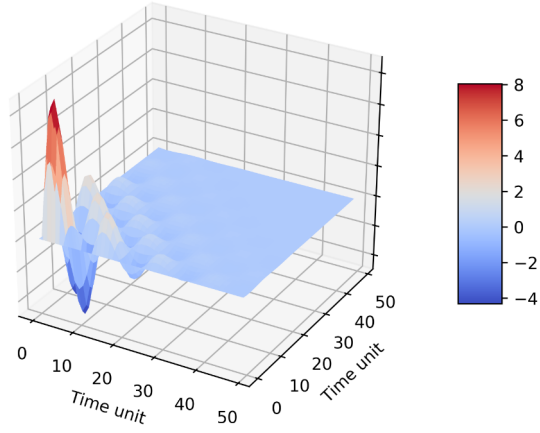


(a) Extracted 1<sup>st</sup> order kernel

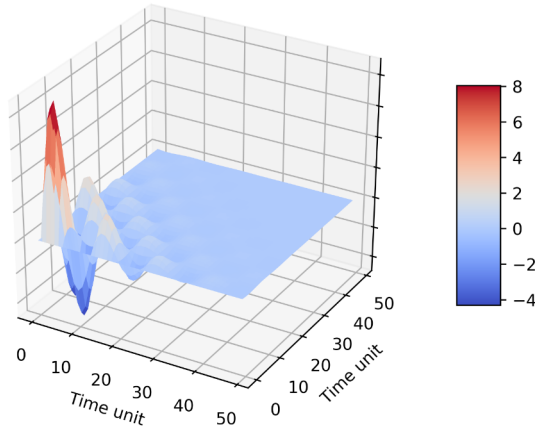


(b) Extracted 3<sup>rd</sup> order kernel (averaged over the 3<sup>rd</sup> time dimension)

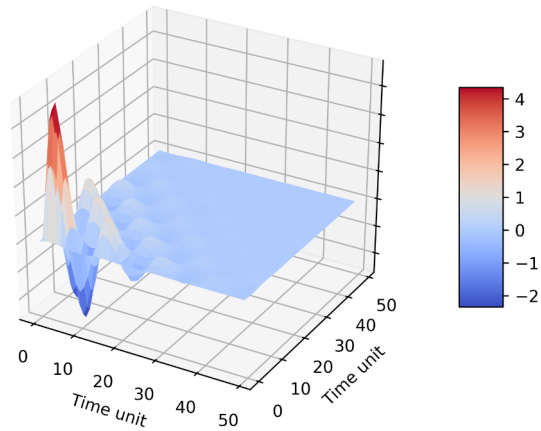
Figure 3.3: Extracted kernels for 2<sup>nd</sup> order Wiener system example



(a) Analytical  $2^{nd}$  kernel



(b) Extracted  $2^{nd}$  order kernel using MPSNN



(c) Extracted  $2^{nd}$  order kernel using traditional FFN

Figure 3.4:  $2^{nd}$  order kernel of (a) Analytical system, (b) Extracted Using MPSNN, and (c) Extracted using traditional FFN

### 3.3 Modeling PAM-4 HSL System

MPSNN model is validated on a high-speed channel PAM-4 buffer circuit transmitting data at 28Gbp/s rate. A typical schematic of a high-speed channel is shown in 3.5. The input voltage to the transmitter TX is denoted as  $V_{T0}$ , input and output voltage to the channel is denoted as  $V_{TX}$  and  $V_{RX}$  respectively.  $V_{RO}$  is the output the receiver RX.



Figure 3.5: A high-speed channel with I/O buffer

Differentiating from the previous example, as mentioned, the training signal for this buffer circuit is a PRBS similar to that exists in the working condition of the channel. A portion of the training signals is shown in 3.6. Input signal is scaled down to be an half of original signal so that all signals can be plotted in the same graph.

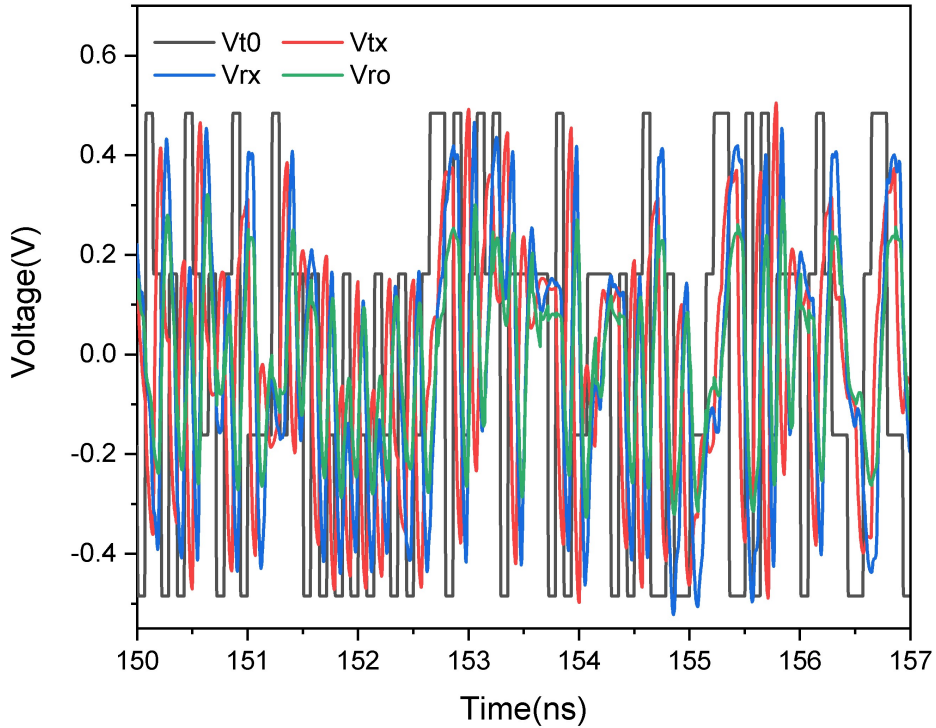
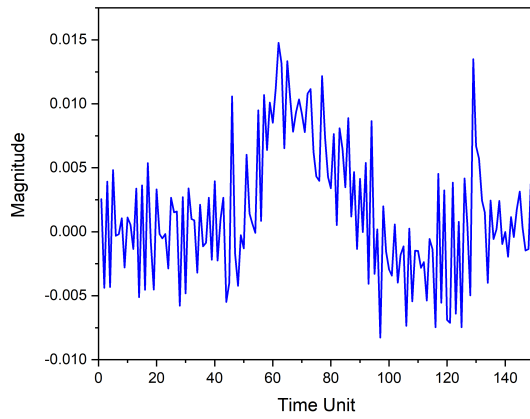
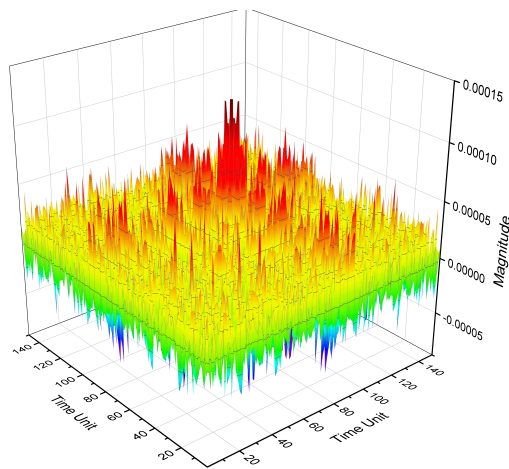


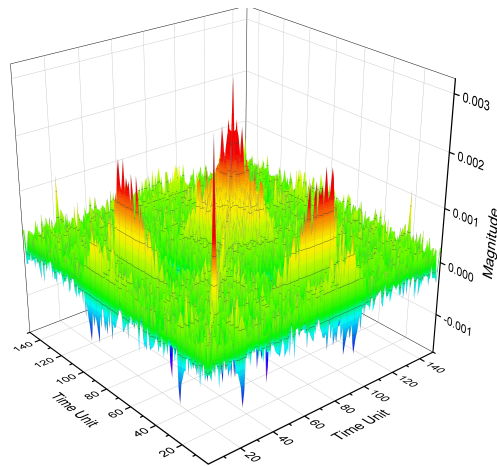
Figure 3.6: Training signals in PAM4 buffer example.



(a) Extracted 1<sup>st</sup> order kernel



(b) Extracted 2<sup>nd</sup> order kernel



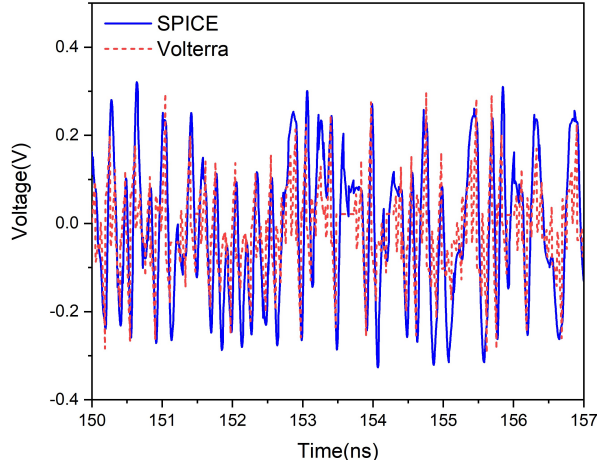
(c) Extracted 3<sup>rd</sup> order kernel (averaged over the 3<sup>rd</sup> time dimension)

Figure 3.7: Extracted kernels for PAM4 buffer example

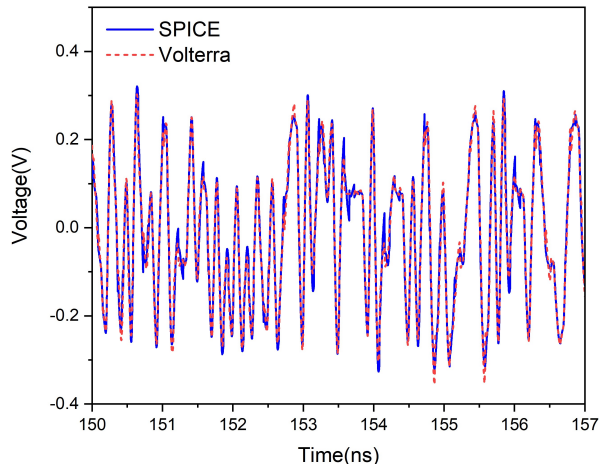
Due to many flat regions in the signals, especially the input signal, the memory length  $M$  has to be increased up to 150 in order for the training to converge. Thanks to the proposed method, the number of kernels remains 3, the order of chosen activation function, instead of  $M$  as in the existing method in the literature [31, 32]. In addition, Adam [43] optimizer is found to outperform other stochastic optimization methods and gives the best convergence for the training in this example.

Extracted kernels are shown in figure 3.7. Once the kernels are extracted, an unseen PRBS is used to obtain the output response of the PAM4 channel and verify it against simulation result from SPICE-based simulators. Up to 3-dimensional direct convolution was used to produce the response. Different results obtained from different sets of VKs trained using various memory lengths are reported in 3.8 to compare the effect of memory length  $M$  to the accuracy of the output response. As shown, when  $M = 150$ , the information contained in 3 extracted kernels is sufficient so that the output obtained from them matches the SPICE level simulation very well.

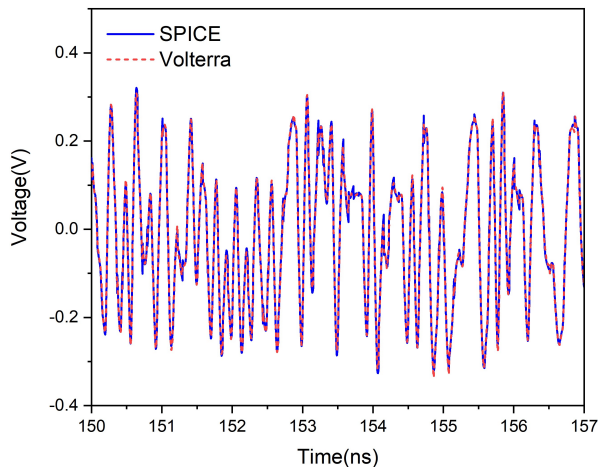
Figure 3.9 shows a comparison of the output voltage after equalization in PAM4 channel obtained from an SPICE model and the above Volterra series model. They were both trained with memory length  $M = 150$ , Adam with initial learning rate of 0.001 were used to train the model. The prediction accuracy is 96.5% based on MSE error.



(a) When  $M = 50$



(b) When  $M = 100$



(c) When  $M = 150$

Figure 3.8: Output response of PAM4 channel under unseen PRBS excitation using extracted VK kernels.

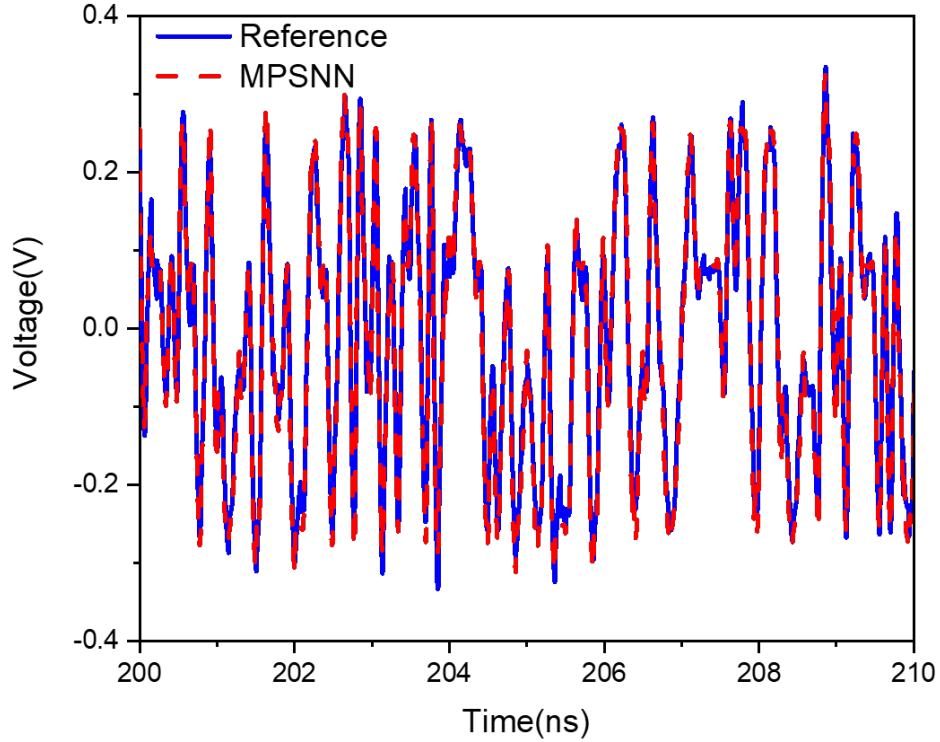


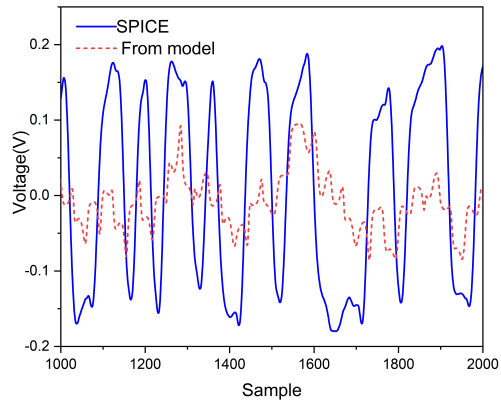
Figure 3.9: Output response comparison between reference signal and VK model for PAM4 example

### 3.4 Modeling NRZ HSL System

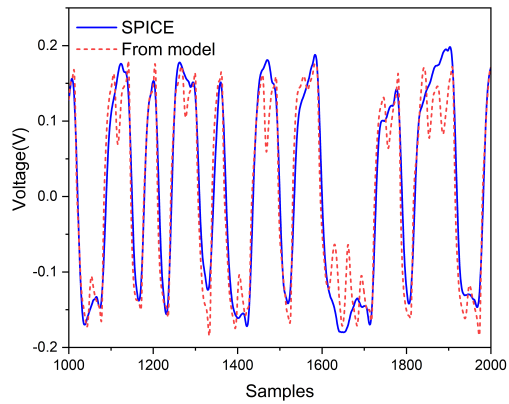
We have shown that MPSNN model can model PAM-4 system and VKs up to the third order can be extracted accurately. In this section, modeling NRZ system is validated with MPSNN method as well. As mentioned in Chapter1, the signal in NRZ system has only two levels (-1,1). The system under modeling is running at 28Gb/s. The activation function order is set at 3. All other settings for training and validation is the same as for PAM-4 system. As expected, varying memory length will effecting the prediction accuracy. The optimal memory length is 300 for NRZ system under modeling. With the memory length longer than 300, the accuracy improvement is getting less obvious. However, the memory length shorter than 300 will result in compromised accuracy. The result is shown in figure 3.10. The extracted VKs up to 3<sup>rd</sup> order are shown in figure 3.11. As seen from the figure, the



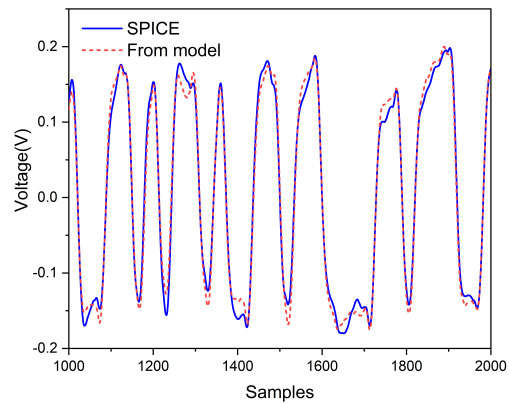
majority of the energy is in the first order kernel. The system under modeling is essentially a first order system.



(a) When  $M = 100$

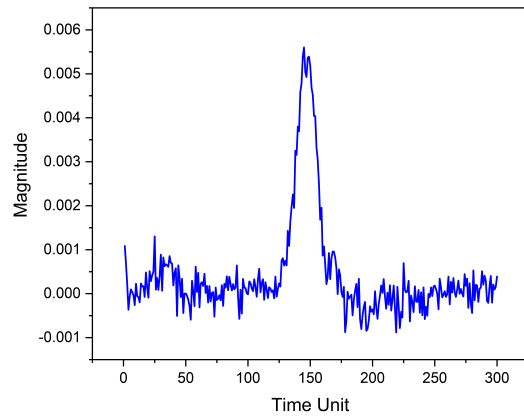


(b) When  $M = 200$

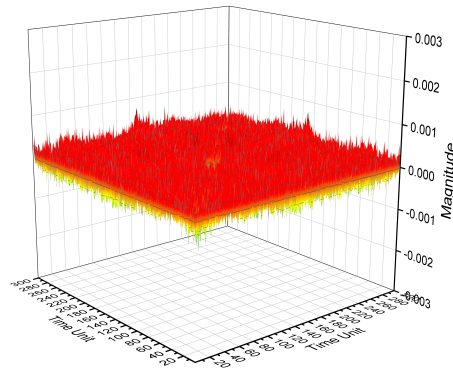


(c) When  $M = 300$

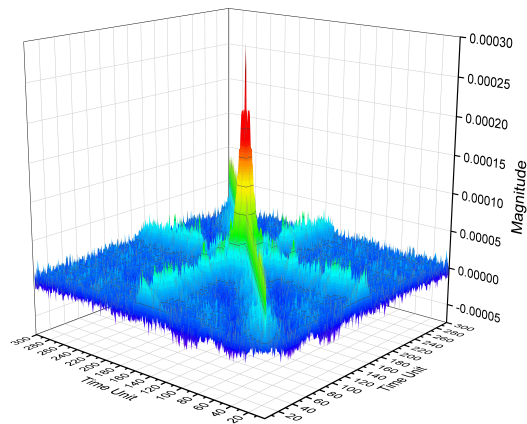
Figure 3.10: Output response of PAM2 channel under unseen PRBS excitation using extracted VK kernels.



(a) Extracted 1<sup>st</sup> order kernel



(b) Extracted 2<sup>nd</sup> order kernel



(c) Extracted 3<sup>rd</sup> order kernel (averaged over the 3<sup>rd</sup> time dimension)

Figure 3.11: Extracted kernels for PAM2 buffer example

### 3.5 Conclusion and Future work

The connection between the weights of a trained neural network and VK is established in this paper. A closed form expression is available to extract the VK from the feed-forward neural network. It has been pointed out that using tanh activation function and training the neural network with random signal as prior works would lead to unstable extraction process and high numerical cost when using the kernels for transient simulation afterward. The robustness of the proposed method was demonstrated with two different examples, one of which is a 28 Gbps PAM4 high-speed link circuit. Due to the long delay of the channel, a memory length of at least  $M = 100$  was needed to obtain an acceptable accuracy using VK compared with SPICE simulation. In contrast with previous works, which will require up to the 100<sup>th</sup>-order kernel, the kernel extraction process with power series neural network could be limited the number of needed kernels to 3 only as shown above.

In the future work, a different representation of VK, namely the Volterra Laguerre model, and the extraction process of them will be studied to reduce the number of parameters needed to represent the VK. This would reduce a substantial amount of numerical effort for use of VK models in simulation, hence speed up the simulation process.

# Chapter 4

## Approach of Laguerre-Volterra Feed Forward Neural Network for Modeling High Speed Links

### 4.1 Introduction

The growth of new technologies such as big data, cloud computing, the Internet-of-Things, 5G, and artificial intelligence, has driven demand for instant data exchange around the world. Global IP traffic will reach 4.8 zettabytes per year by 2022, a three-fold increase since 2017[44]. Boost in the data transmission rate to 400Gb/s and above has been called in the next generation high-speed link (HSL) technologies. Traditional non-return-to-zero (NRZ), also known as pulse-amplitude modulation 2-Level (PAM2), has been evolving over the past five decades from 10Gb/s to 100Gb/s (4 lanes, 25-28Gb/s/lane). Scaling up NRZ technology to 400Gb/s challenges the transceiver design for closed eye at receiver output. At 400Gb/s, NRZ experiences enhanced receiver sensitivity, tighter jitter budgets, and deteriorated signal-to-noise ratio. To open the eye, advanced correction techniques must be implemented at both transmitter and receiver side [45]. For instance, enhanced receiver equalization schemes such as continuous-time-linear equalization (CTLE) and decision-feedback-equalization (DFE) are needed to correct the channel loss and reflections.

To alleviate the challenges of linear data rate scale-up with NRZ technology, pulse amplitude modulation 4-level (PAM-4) is introduced in the IEEE P802.3bs standard [46]. Unlike NRZ, which has two levels (-1,1) encoding one bit (0, 1) for each level, PAM-4 has 4 amplitude levels, (-3, -1, 1,3) encoding 2 bits (00, 01, 10, 11) for each level. Therefore, one symbol in PAM-4 carries twice as much information as in NRZ, which can be interpreted as meaning that PAM-4 doubles the throughput for the same baud rate as NRZ. From the frequency domain perspective, PAM-4 requires only half of the band-

width of NRZ for the same throughput. In light of these advantages, PAM-4 has gained growing attention since its introduction.

Although PAM-4 offers higher spectral efficiency, lower channel loss, and less stringent timing requirement, challenges lie in the design, test, and validation of PAM-4 transceivers [47]. 4.1 illustrates the signal and eye diagram difference between NRZ and PAM-4. As seen, the time domain waveform for NRZ has two levels and only one eye is observed (4.1(a)) while that of PAM-4 has four levels and three eyes (4.1(b)). Thus, multi-level signal modulation in PAM-4 is the game-changer in transceiver design, test, and validation. New chip designs to support PAM-4 multi-level modulation will have to address the area and power increase to accommodate more transistors. The size of the integrated circuits (ICs) for PAM-4 has increased nearly 30%, while the power consumption increased up to 35% [47]. PAM-4 transceivers of 32Gb/s [48], 45Gb/s [49], and 56Gb/s [45] [50] fabricated with 90 nm, 60 nm, and 14 nm technologies have been reported. However, few papers are published on testing and validation. The reason is that the testing and validation strategy for the PAM-4 system does not share the same basis as that for the NRZ system due to multi-level signal modulation. Consequently, more sophisticated equipment is needed for PAM-4 system testing and validation. Moreover, due to the much higher data rate, the jitter budget for the PAM-4 system could be lower than the intrinsic jitter tolerance of the testing and validation equipment. Therefore, design and construction of such equipment are expensive and time-consuming.

To alleviate the challenges of linear data rate growth on NRZ technology, pulse amplitude modulation 4-level (PAM-4) is introduced in IEEE P802.3bs standard[46]. Unlike NRZ, which has two levels (-1,1) encoding one bit (0, 1) for each level, PAM-4 has 4 amplitude level (-3, -1, 1,3) encoding 2 bits (00, 01, 10, 11) for each level. Therefore one symbol in PAM-4 carries twice as much information as in NRZ, which can be interpreted as PAM-4 doubles throughput at the same baud rate as NRZ. From frequency domain perspective, PAM-4 requires only half of the bandwidth as NRZ for the same throughput. With that being said, PAM-4 has gained great attentions since its introduction. A few leading HSL companies has announced their PAM-4 products and without doubt, PAM-4 is leading the next generation HSL

technology.

PAM-4 system simulation turns out to be a viable solution to alleviate the challenges off the PAM-4 system testing and validation. Compared to testing and validation of a PAM-4 system in a real production environment, verification of such a system through computer simulation can be fast and low cost. However, the success of this method heavily relies on the accuracy and efficiency of the PAM-4 behavioral model that is used in simulation. The behavioral model has two major advantages: (1) it is fast and computationally efficient, and (2) it is good for IP-protection. There are a variety of excellent behavioral models such as state-space [51], NARX [52], Verilog-A [11], Volterra series [37] [38], M/spl $\pi$ /log [53]. Such models have achieved success in a various fields such digital IC, analog/mixed-signal module, RF amplifier/mixer, communication channels. However limited simulation work has been reported on PAM-4 behavior modeling.

In recent years, there has been growing interest in using machine learning methods for behavior modeling of high-speed links and integrated circuits. An increasing number of studies have demonstrated that multi-layer artificial neural networks and recurrent neural networks can be trained to accurately represent non-linear electrical circuits and systems, such as zero-in zero-out (ZIZO) circuits [13][11], PAM-4 system [15][14], and high-speed links [54]. In these models, a time-series input/output signal pair collected from either measurement of the real circuits or a SPICE model are used for machine learning model training. Once the models are trained, they can be used to model the real circuit/system for prediction. They are typically computationally faster than traditional models. Moreover, they are inherently suitable for IP protection. However, there are two main limitations associated with such models: they lack interoperability, and are computationally inefficient when more layers and more neurons are required. For instance, such models are not compatible with mainstream EDA software, which limits their application in industry. Therefore further improvement in such models is essential to promote their use in behavioral modeling of high-speed circuit/systems.

This chapter presents the work modeling HSL systems include PAM-4 and NRZ using LVFFN. The proposed LVFFN can drastically reduce the com-

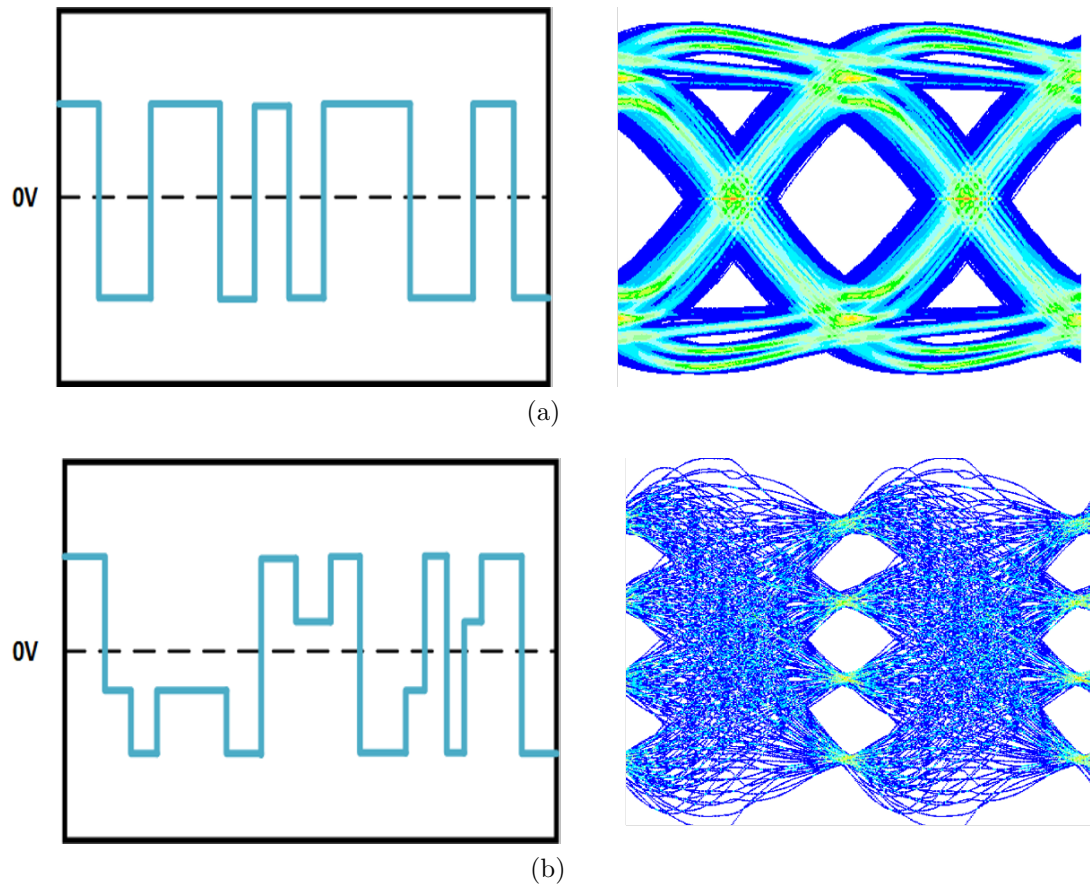


Figure 4.1: Bit pattern for (a)NRZ and (b)PAM4



plexity of FFN otherwise employed. In addition, the LVFFN model can be implemented into IBIS-AMI, an industry standard, in order to enhance model interoperability and transportability. The IBIS-AMI model implementation will be discussed in next chapter. This chapter is organized as follows. Modeling PAM-4 system with LVFFN is introduced in Section II. Modeling NRZ system (PAM-2) and just NRZ system receiver using LVFFN is presented in Section III. To demonstrate the versatility of the LVFFN model, a heavily distorted CMOS inverter is modeling. Some preliminary results are presented. Section IV concludes this chapter.

## 4.2 Modeling PAM-4 System with LVFFN

In this section, a PAM-4 high-speed link system is modeled with LVFFN. A block diagram of a typical high speed link system is illustrated in 4.2. The system consists of three portions: transmitter, channel, and receiver. To compensate for the signal loss and distortion during transmission, correction blocks such as equalizers, feed-forward equalizer (FFE), and decision-feedback equalizer (DFE) are employed. Multi-level signaling presented in the PAM-4 system severely complicates the model development. To demonstrate the effectiveness of LVFFN on modeling the PAM-4 system, we model the system with both a regular FFN and an LVFFN proposed in this chapter.

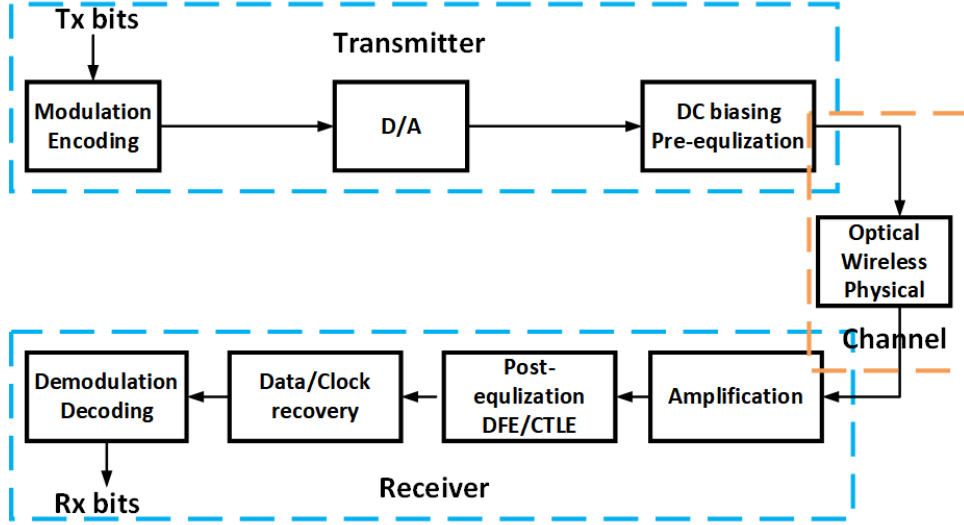


Figure 4.2: PAM-4 high speed link system block diagram

4.3 shows the input and output signals that are used for training the models. The input/output signals are generated from an industrial PAM-4 IBIS-AMI model. The input signal is a 4-level pseudo-random binary sequence (PRBS) with amplitude between -1.0 V and 1.0 V. The data rate is 28Gb/s. Total 16,651 time series samples are collected for training and testing (70/30%). The regular FFN model employed in this work has three layers: one input, hidden, and output. A third-order monomial power series is used for the activation function. Determining the right memory length for the system under modeling is important. In our previous work, we have demonstrated that longer memory length would result in better accuracy. There is an elbow point for the memory length after which increasing length will not obviously benefit accuracy. We found that for PAM-4 system under modeling, memory length of 150 produces the optimal result. In this work, we use the memory length of 150 for both FFN model and LVFFN model. Back-propagation algorithm with an Adam optimizer is selected for minimizing the Mean Square Error (MSE). Dropout is not used during training. The learning rate typically takes value in between  $10^{-3}$  to  $10^{-6}$ . Bigger learning rate leads to faster converge but worse training/testing accuracy. Typically the training accuracy get above 90% in less than 25 epochs or less than 2 minutes CPU time using a desktop equipped with an Intel i7 quad core CPU. The best training/testing accuracy of 98%/97% can be achieved by allowing a longer training time.

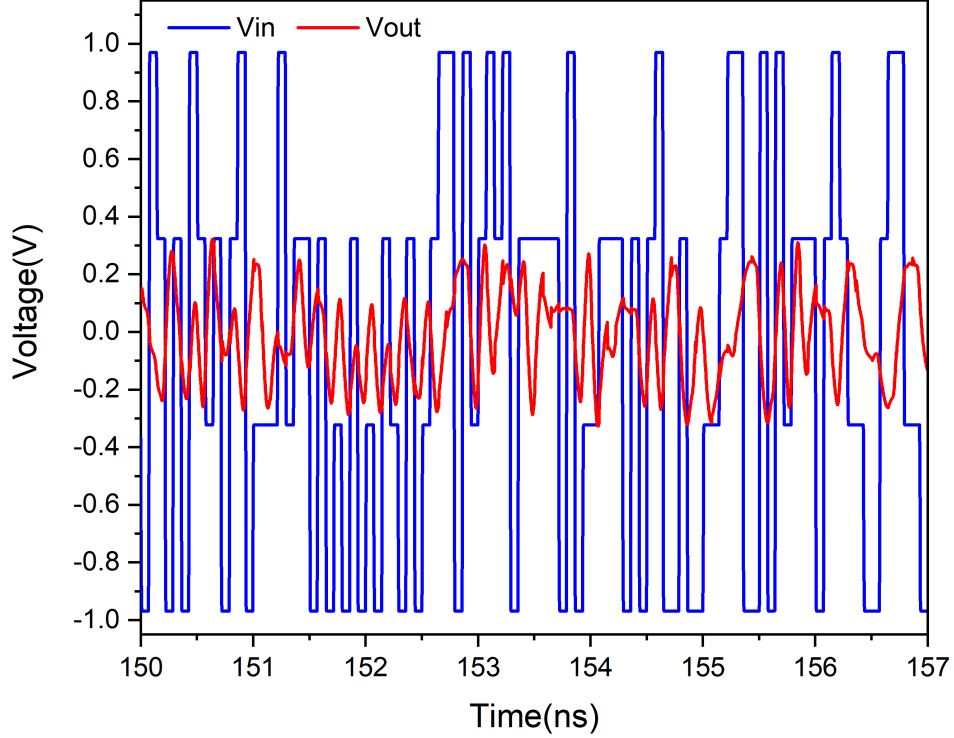


Figure 4.3: Plot of PAM-4 input output data

#### 4.2.1 Determining Decay factor $\alpha$ and the Order Laguerre Function

In LVFFN, the input signal is first convolved with the Laguerre functions. 4.4 plots the first five Laguerre functions for  $\alpha = 0.2$  and  $\alpha = 0.5$ . As we can see, the decay factor controls how fast the Laguerre function approaches zero. Finding the right decay factor  $\alpha$  is crucial for modeling. There have been works reporting on how to obtain optimal  $\alpha$  for a system. Campello et al. reported that in  $Z$  domain, the corresponding  $r^{th}$  Laguerre functions are given by [55]

$$\Phi_r(z) = \frac{z\sqrt{1-P_r^2}}{z-P_r} \left( \frac{1-P_r z}{z-P_r} \right)^n \quad (4.1)$$

$P_r$  is also called a real number Laguerre pole. Then the optimal Laguerre pole  $P_r$  can be obtained by solving the following optimization problem:

$$\min_{-1 < p_r < 1} J_r = \sum_{i_1=1}^{\infty} \dots \sum_{i_r=1}^{\infty} (i_1 + \dots + i_r) \alpha_{i_1, \dots, i_r}^n \quad (4.2)$$

The optimal pole can be obtained using equation (4.2). However the process could be computationally prohibitive. Another approach to obtain the right  $\alpha$  takes advantage of neural network training.  $\alpha$  is treated as one of the parameters in the neural network and then converges to the optimal value through back-propagation.

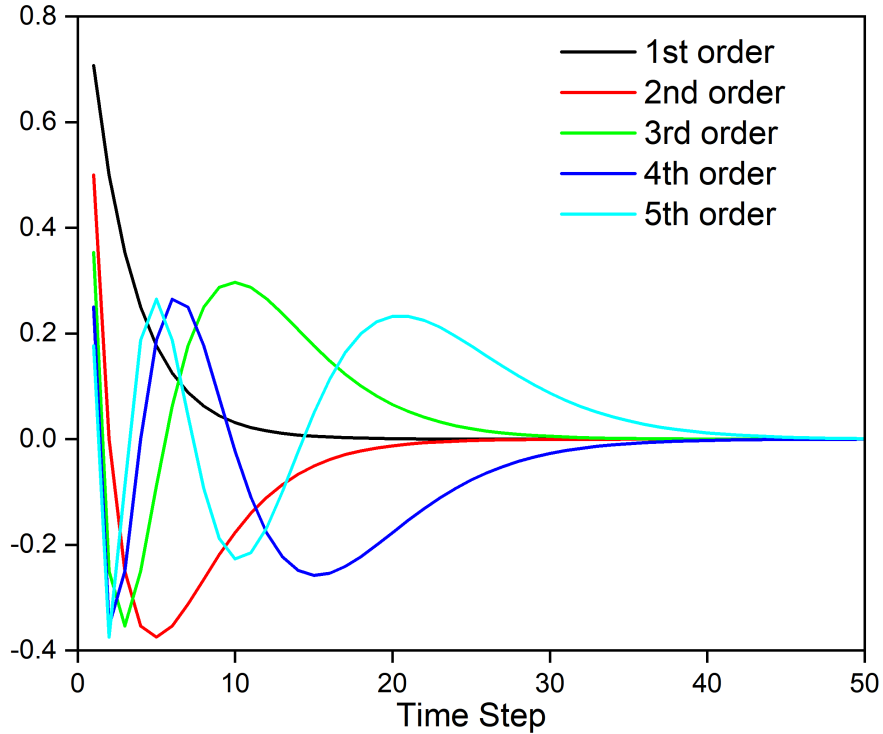
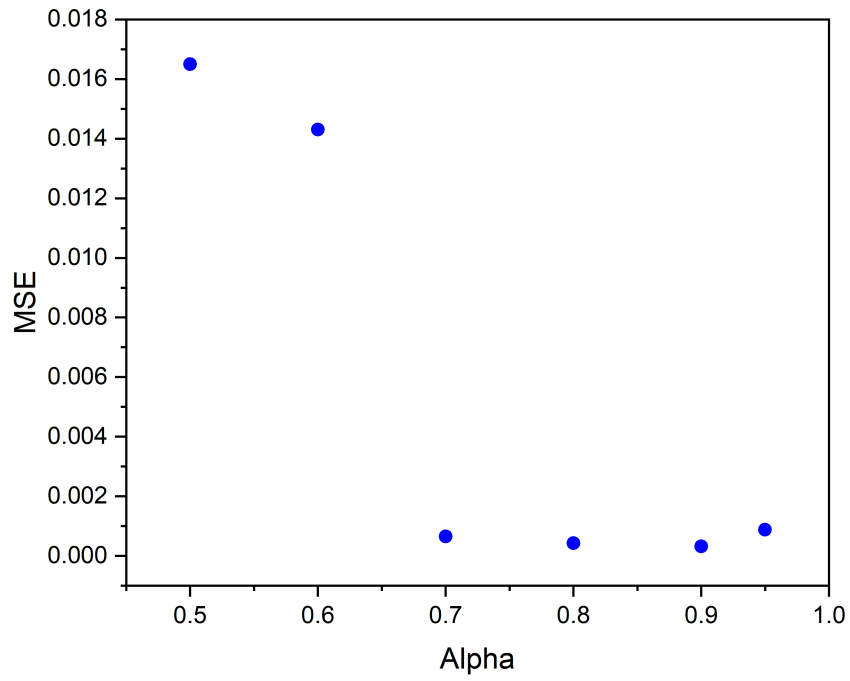


Figure 4.4: Plot of first five Laguerre functions

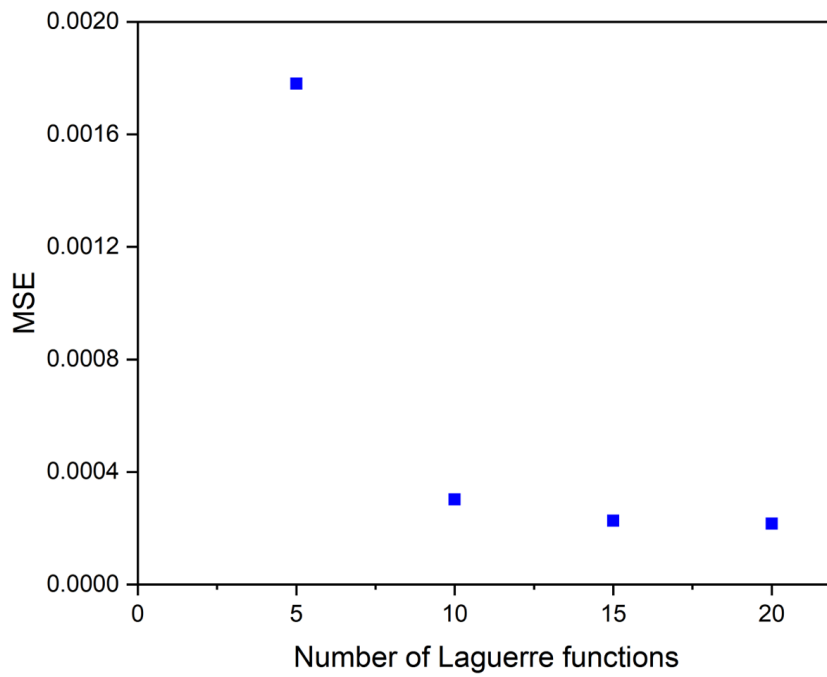
In the expansion, the number of Laguerre functions is truncated to a certain quantity. Determining the minimum number of functions that are enough to achieve the desired model accuracy is another challenge. Typically, more functions are desired to achieve better prediction accuracy. However, more

functions will deteriorate the benefit of Laguerre expansion for dimension reduction. Kang and Marmarelis et al. reported a method of principal dynamic mode analysis to obtain the least number of Laguerre functions. This method has also been applied to model EEG data for diagnosis of Alzheimer's disease [56]. In this method, the first and second order VKs have to be identified using a traditional algorithm such as LMS. Then singular value decomposition (SVD) is applied to obtain the most significant elements on the first and second order VK matrix. The number of significant VK elements relates to the number of Laguerre functions is used in the expansion.

In this work, a simple brute force algorithm is used to obtain the optimal  $\alpha$  and the least number of Laguerre functions. Figure 4.5 shows the plot for  $\alpha$  value ( 4.5 (a)) and number of Laguerre functions (4.5 (b)) versus MSE value. As seen in the figure, MSE magnitude approaches equilibrium as both  $\alpha$  value and the number of functions increase. Based on the plot, we can determine the optimal value of  $\alpha$  and the number of Laguerre functions for the PAM-4 system. The optimal value of  $\alpha$  is determined to be 0.91, and the number of Laguerre functions is selected to be 10.



(a)



(b)

Figure 4.5: Plots of MSE vs (a)  $\alpha$ , and (b) number of Laguerre functions

## 4.2.2 Modeling Result

The PAM-4 system is modeled with LVFFN, regular FFN, and recurrent neural network (RNN). In the RNN model, 6 stacked layers with 20 neurons for each layer and memory length of 100 are employed [14]. 4.6 plots the model output of LVFFN, FFN, RNN, and reference signal. As seen, all the models demonstrate great accuracy. The FFN model needs 1 hidden layer and 150 neurons, while LVFFN only needs 1 hidden layer and 10 neurons.

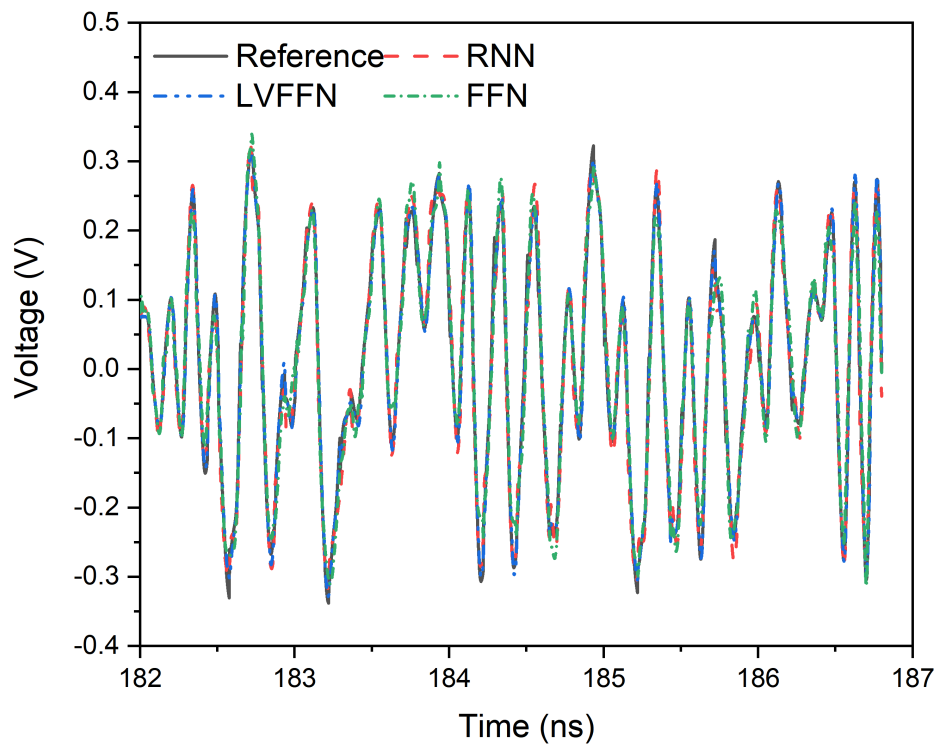


Figure 4.6: Model output comparison of LVFFN, RNN, FFN, and the reference signal

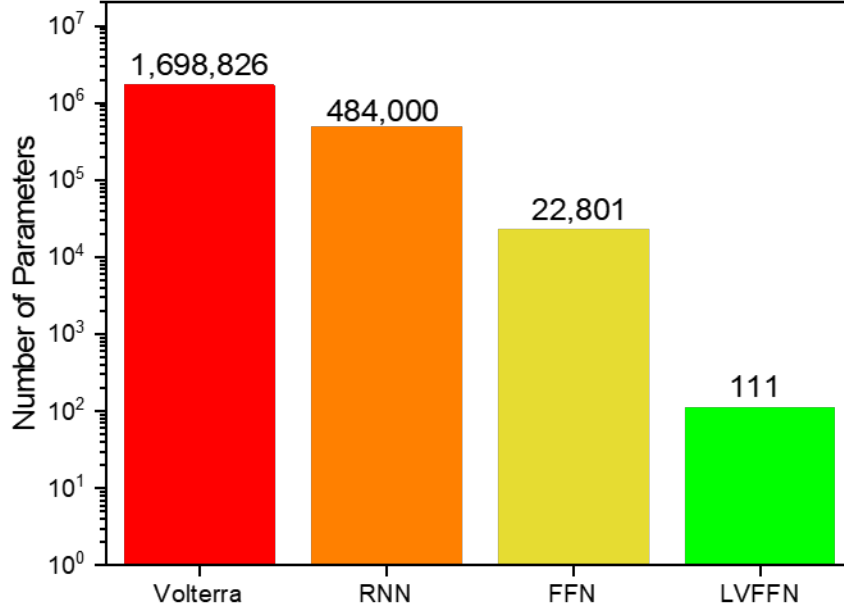


Figure 4.7: Comparison of model size for Volterra series, RNN, FFN, and LVFFN

Model size comparison for Volterra series, RNN, FFN, and LVFFN is presented in 4.7. From the regular FFN network, VKs up to the  $3^{rd}$  order can be obtained through equation (2.16)-(2.19). With the identified VKs, the PAM-4 system can be completely described. However, according to equation (2.10), we can calculate the total number of kernel value for the system of  $3^{rd}$  order non-linearity and memory length of 150. The number of kernel value is 3,397,651. VKs have the attribute of symmetry, e.g. the  $h(a, b)$  is identical to  $h(b, a)$ . Taking this attribute into account, the total number of kernel value is reduced to 1,698,826, which is still an exceptionally large model. The model size that comes in the second place is the RNN model which has 485,200 parameters. Following RNN, the regular FFN requires 22,801 parameters. The LVFFN only needs 111 parameters which is substantially smaller for similar prediction accuracy. This is a significant model size reduction compared to other regular machine learning models and Volterra series. The model size reduction will improve the model transportability and computation efficiency.



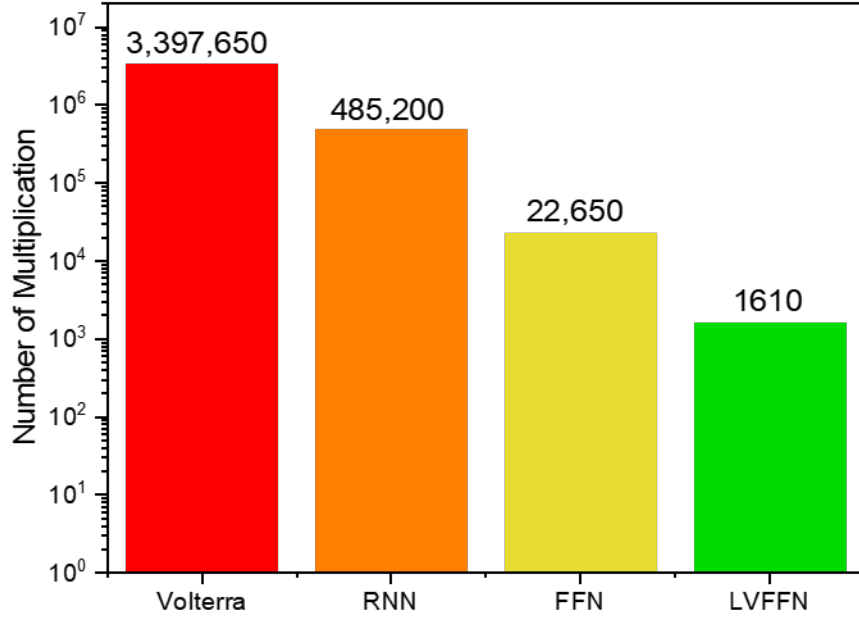


Figure 4.8: Comparison of number of multiplications for calculation of one output sample for Volterra series, RNN, FFN, and LVFFN

4.8 shows the computational efficiency comparison for computing one output sample. The number of multiplications is used here as the criterion to characterize the model performance. As seen, LVFFN only requires 1.61K multiplications to compute one output sample. The total number of multiplications for computing one output sample includes the computation of the neural network, which is only 110, and the convolution of the input signal with the 10 Laguerre functions, which is 1500. As we can see, the improvement in computation efficiency is significant for the proposed LVFFN model. In LVFFN, the majority of the computational power is used by the convolution of input signal with each Laguerre function. This computation can be further reduced by calculating the filtered output recursively using [35]:

$$l_r(n) = \sqrt{\alpha}l_r(n-1) + \sqrt{\alpha}l_{r-1}(n) - l_{r-1}(n-1) \quad (4.3)$$

Initialization is carried out by

$$l_0(n) = \sqrt{\alpha}l_0(n-1) + \sqrt{1-\alpha}x(n) \quad (4.4)$$

System identification is popular in modeling non-linear systems. Typically, VKs up to the desired order are identified with time series input signal. In our previous work, we have reported MPSNN to extract VKs up to the third order [13] for PAM-4 HSL system. In this work, we are able to extract the VKs up to the third order using LVFFN for the same HSL system. In LVFFN, we can identify Laguerre parameters. Then the Volterra kernels up to desired order can be obtained through extracted Laguerre parameters using equation (2.30)

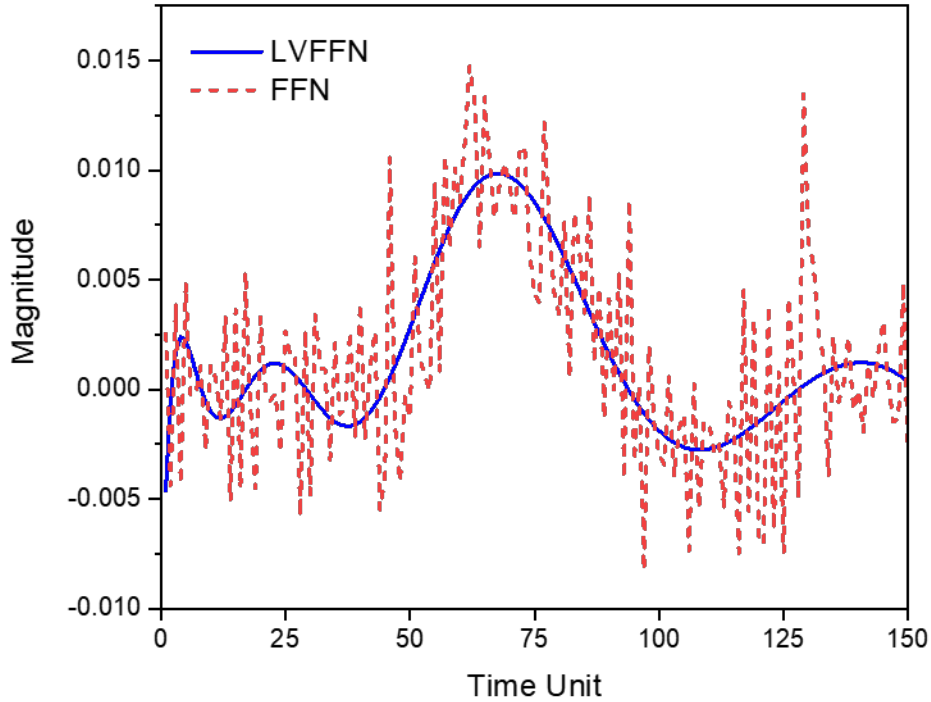


Figure 4.9: First order VK extracted with FFN and LVFFN

4.9 shows the first order VKs extracted for PAM-4 system using FFN and LVFFN. As we can see, the VKs extracted from both methods match fairly well. The kernel identified with LVFFN is smoother than the ones with FFN. This is expected as the Laguerre expansion of VK is equivalent

to orthonormal decomposition of  $VK$ , which removes redundant information during the expansion process.

### 4.3 Modeling NRZ System with LVFFN

In the last section, we demonstrated that the LVFFN can model PAM-4 system. The LVFFN PAM-4 model is much more concise and efficient compared to traditional Volterra model and other machine learning models. To demonstrate the versatility of the proposed LVFFN model, in this section, we model a NRZ system (PAM-2) using LVFFN. Figure 4.10 shows the waveform at different node. Waveform at node A is the excitation waveform. It has a magnitude of  $\pm 0.1V$  and a data rate of 28 Gbps. The waveform, as shown, is a perfect square wave. After the excitation passes through a channel, the signal is deteriorated due to the channel loss and inter symbol interference (ISI). The signal waveform after channel at node B is shown in waveform B

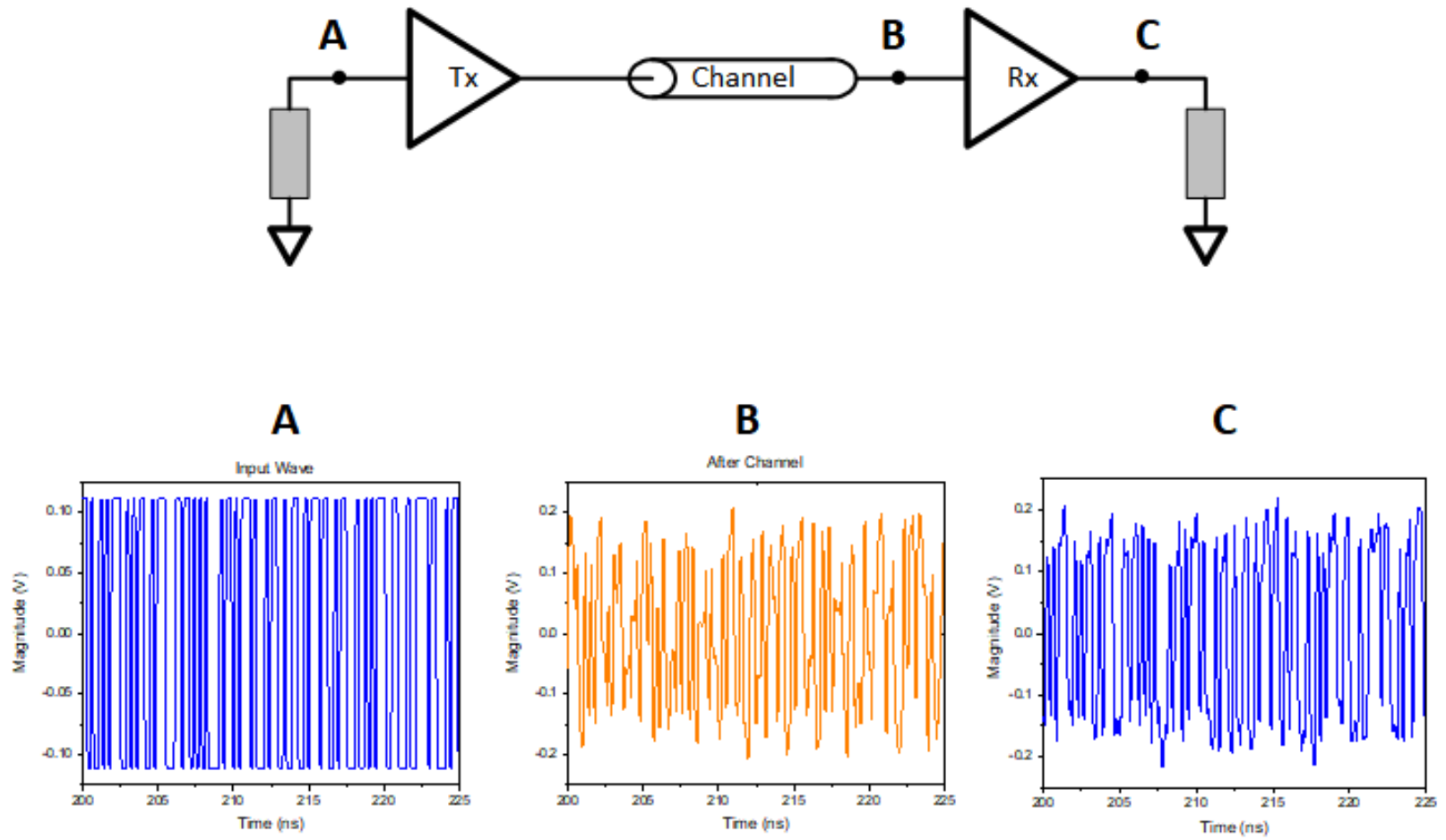


Figure 4.10: NRZ waveform

with orange color. The waveform C is the signal at node C. Distorted signal at node B is partially restored by DEF/FFE implemented in the receiver. The LVFFN models the whole system from node A to node C.

The number of Laguerre functions and the decay factor  $\alpha$  is obtained through brute force searching. there is an elbow point after which MSE improvement gets slow (number of Laguerre functions) or even worse ( $\alpha$ ). According to the plots, we can determine that the best  $\alpha$  value is 0.83 and the best number-of-Laguerre-functions is 25. The memory length employed in the application is 300 which has been approved to be efficient in chapter 3.

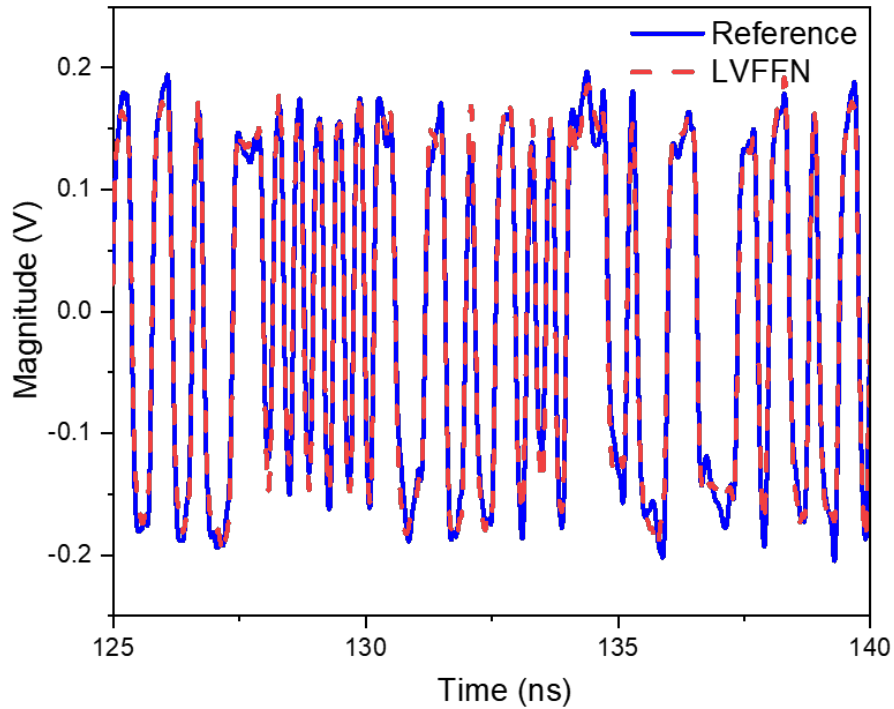


Figure 4.11: PAM-2 system LVFFN model prediction and reference waveform

Figure 4.11 plots the model prediction and the reference signal. The average accuracy in terms of MSE is 96.5%. Comparison of model size for Volterra series, FFN, and LVFFN are plotted in figure 4.12. Although, modeling NRZ

system requires more Laguerre functions, 25 functions, than modeling PAM-4 system, which only need 10 functions. The model size for LVFFN is only 650 parameters, while for FFN 90,300 and Volterra series 13,500,000. This is a significant dimensionality reduction comparing to regular FFN models and traditional Volterra series.

Figure 4.13 shows the computation efficiency comparison for each model to compute one output sample. The number of multiplications is used as the criteria to characterize the model performance. As seen, LVFFN only needs 8,150 multiplications to compute one output sample, while FFN needs 90,300, and Volterra series needs the most, 27,090,300 multiplications. With LVFFN, the computation performance improvement is obvious, a several order of magnitude enhancement. The majority of computation power in LVFFN model attributes to the convolution of input signal with each Laguerre filter in the filter bank. Such computation can be further improved by calculation the filtered output recursively using the formula (4.3) and (4.4).

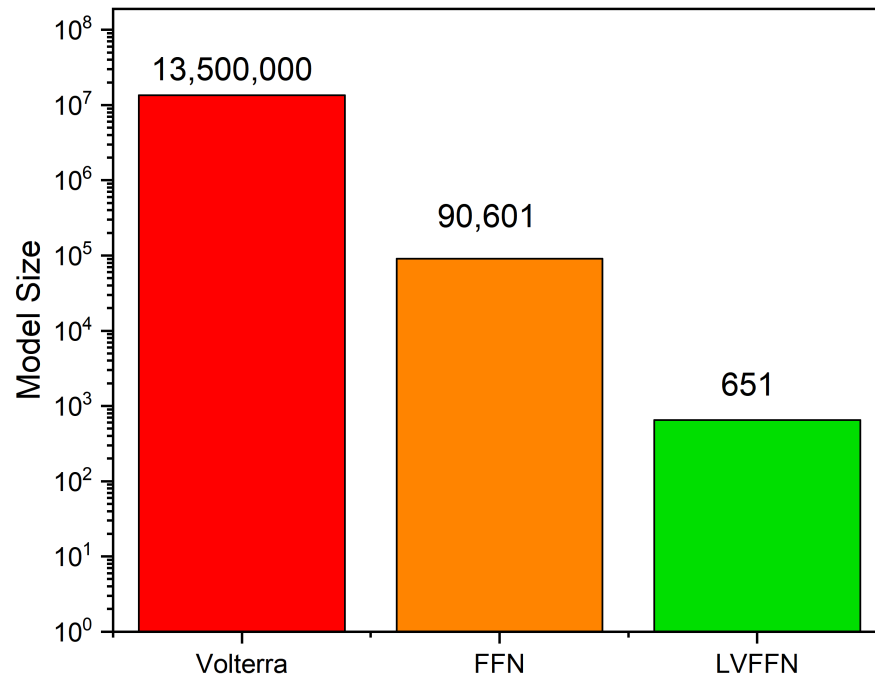


Figure 4.12: LVFFN NRZ system model size comparison for Volterra series, FFN, and LVFFN

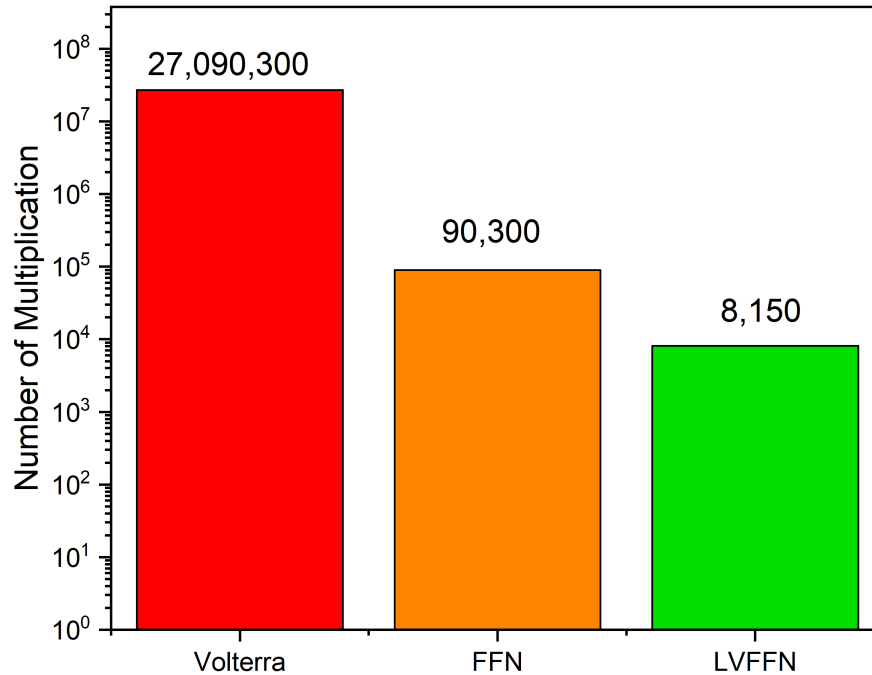


Figure 4.13: Number of multiplication comparison for Volterra series, FFN, and LVFFN

### 4.3.1 Modeling NRZ system receiver

The whole NRZ and PAM-4 system can be modeled with LVFFN. However, sometimes it is desired to model just the transmitter or receiver, or even the FFE/DFE. The idea here is to separate channel out so that user can use the model with different channels. In this part, we demonstrate that the LVFFN can model individual component, e.g. NRZ receiver. In figure 4.10, the part being modeled is between B and C. The waveform at B and C are used as input and output for training.

The number of Laguerre functions used for modeling the NRZ receiver is 35 which is higher than the number functions for modeling the whole system. The  $\alpha$  is identified to be 0.83, the same as modeling the whole NRZ system. Figure 4.14 shows the prediction waveform and the reference waveform. The



prediction accuracy is 92.1% in terms of MSE.

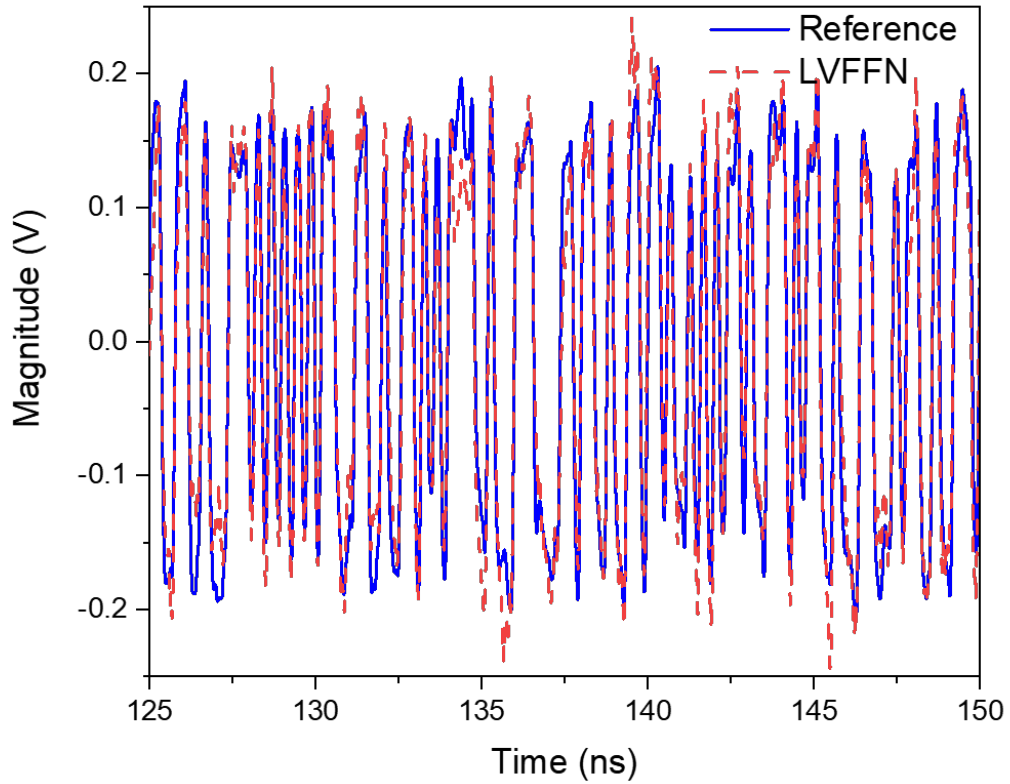


Figure 4.14: PAM-2 receiver LVFFN model prediction and reference waveform

Modeling NRZ system receiver with LVFFN requires more laguerre functions than modeling the whole system. And the prediction accuracy is worse. This is a proof-of-concept work demonstrating that the LVFFN can model individual component in the HSL system. However more work on this research is required to improve the model accuracy. This would be a future work to extend the research reported here.

## 4.4 Modeling CMOS Inverter with LVFFN

To demonstrate that LVFNN can not only model HSL system, a weakly nonlinear system, but also can model a COMS inverter, a strongly nonlinear system. The CMOS model is shown in figure 4.15. The inverter is poorly biased. The input signal is a sinusoidal wave with amplitude of 6V centering at 0V, while the resulted output is distorted square wave with amplitude from 8.5V to 12V. LVFFN employed to model such inverter has a memory length of 40 and the number of Laguerre functions is 20. The prediction accuracy of 97% can be achieved in terms of the MSE error. The output waveform of reference signal and predicted signal is shown in figure 4.16.

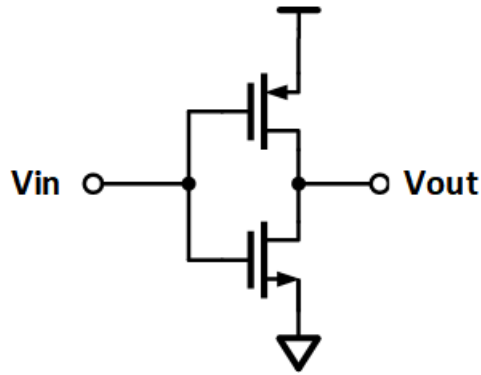


Figure 4.15: Schematic of inverter being modeled

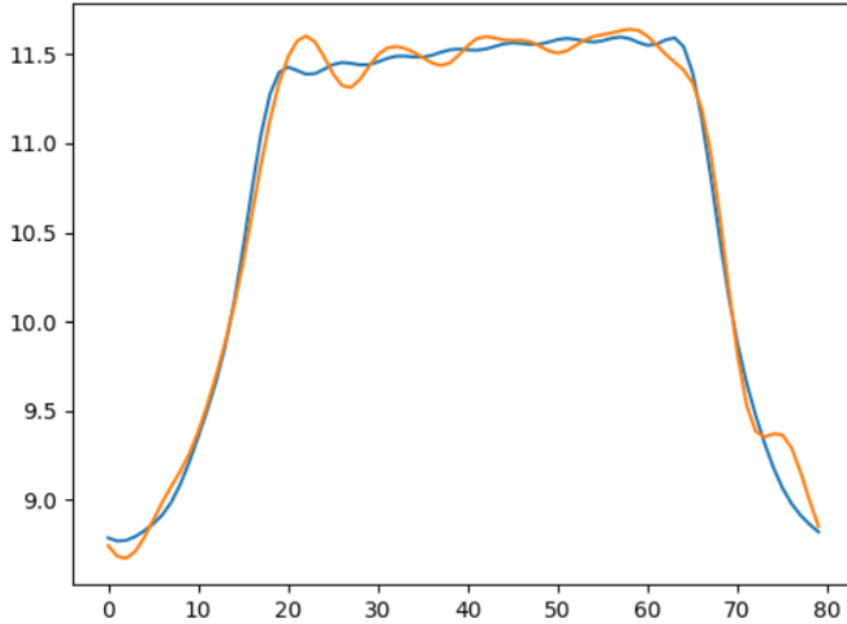


Figure 4.16: Output signal of inverter output (blue) and prediction from LVFFN (orange)

## 4.5 Conclusion

In this chapter, we proposed an LVFFN approach to model the PAM-4 and NRZ (PAM-2) HSL systems. In our approach, the time series input is pre-processed by convolving the input with a certain number of orthonormal Laguerre functions. The convolved output is then fed to the FFN for training and prediction. The LVFFN model with only one hidden layer and 10 neurons can be used to model the whole PAM-4 system. The model size is dramatically reduced due to the signal preprocessing. Compared to other models such as FFN, RNN, and Volterra series, the model size for LVFFN is reduced by 5 orders of magnitude for the same prediction accuracy. The improvement in computation efficiency for LVFFN is significant as well. To compute one sample at output, the LVFFN only requires 1.61K multiplications, while regular FFN, RNN, and Volterra series would need 22.65K, 485.2K, and 3,397.65K multiplications, respectively.

In addition, LVFFN can be used to conduct non-linear system identifica-

tion. The first three order VKs are identified with the LVFFN. The highest order VKs are controlled by the activation function. We compared the first order VK identified with LVFFN to the one identified with regular FFN. They match very well and VK identified with LVFFN is more accurate.

Modeling NRZ system requires more Laguerre functions than PAM-4 system to achieve similar prediction accuracy. This could be resulted from the channel in NRZ system. The channel could be longer or more lossy than the channel in PAM-4 system, which makes the system more difficult to model. Since the waveform used in training the model is from a black box circuit provided by the HSL industry. We were not exposed the details about the circuit. Therefore, more research needs to be done to understand the relationship between number of Laguerre functions,  $\alpha$ , and system being modeled.

Moreover, a proof-of-concept work is conducted on modeling HSL receiver. The whole HSL system can be modeled with LVFFN. However such model lacks of flexibility for co-simulation. For example, once you have the model for the whole system, you cannot replace any individual component such as transmitter, channel, or receiver. Therefore, modeling individual component would make the model more practical for applications. The result shows that the NRZ system receiver in which DFE/FFE, CDR, etc. can be modeled with LVFFN. Although the accuracy is not as high as modeling the whole system, more research is allocated to further improve the accuracy.

At last, we modeled a poorly biased COMS inverter to demonstrate the versatility of the LVFFN model. With only 20 Laguerre functions, we can get 97% prediction accuracy.

# Chapter 5

## IBIS-AMI Model Generation and Simulation Environment

### 5.1 IBIS-AMI Model Basics

IBIS-AMI standard specifies the interface between SerDes behavior model and simulator. The model created under such specification is ensured to work with the simulator that fulfills the same standard. A typical IBIS-AMI model contains analog and algorithmic portions. The analog part includes the IBIS specifications and channel S-parameter files. The algorithmic part contains executable files and .ami files. The executable files are Dynamic Link Library (DLL) files typically written in C++. The .ami files are ASCII text files specifying all the parameters that are used by the executables and the simulator.

Figure 5.1 presents a high-level diagram of IBIS-AMI model. The analog part is composed of a list of S-parameter files which are used by the simulator to generate impulse responses. The S-parameter files are frequency response of the chip packages and the channels. Typically, the algorithmic model has two parts, Tx part and Rx part which are implemented in DLLs and placed at the both ends. In Tx part, it either includes a FFE which modifies the impulse response or receives and processes a bit stream coming from simulator and gives them back to the simulator. In the Rx portion, the Rx DLL implements a pre-processing block, the DFE, and a clock-data recovery (CDR) block. The output of Rx DLL is either the recovered data wave form and clock data, or the modified impulse response which is then convolved with the data stream generated by the Tx DLL.

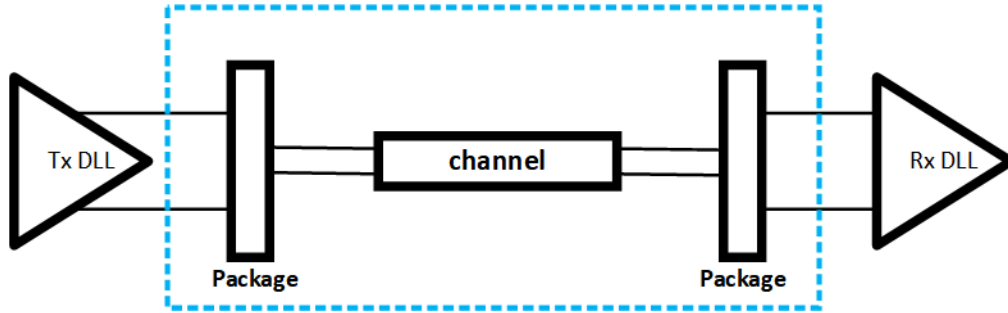


Figure 5.1: IBIS-AMI model diagram

IBIS-AMI specification defines the application program interface (API) functions which provides a platform for user to implement customized algorithmic model. The model is specifically used for HSL simulation and analysis. Such API includes three main functions: `AMI_Init()`, `AMI_GetWave()`, and `AMI_Close()`. `AMI_Init()` is mainly doing initialization which includes allocating/initializing dynamic memory, parsing parameters from `.ami` file, and conducting impulse response filtering. `AMI_GetWave()`, on the other hand, conducts bit-to-bit simulation on the in—ut waveform for nonlinear and time-variant system. `AMI_GetWave()` is optional function which can be explicitly disabled in the `.ami` file. If this function is disabled, the simulator will use the impulse response returned by `AMI_Init()` for simulation. Otherwise, the simulator will ignore the impulse response returned by `AMI_Init()`. It will then take the waveform returned by `AMI_GetWave()` for further simulation. The last function, `AMI_Close()`, typically conducts post-simulation clean-up, which includes shutting down simulation engine, releasing memory allocated, informing simulator that the simulation has been completed successfully or with errors.

The whole IBIS-AMI simulation process starts from generating PRBS excitation bit sequence. Then the simulator pre-processes the `.ami` file and computes the impulse response from the analog S-parameter files. In the following, it presents all the information to the Tx AMI model. After that, the simulator takes the impulse-response output from the `AMI_Init()` and convolves with initial excitation bit sequence, if the `AMI_GetWave()` is disabled. Otherwise, it takes the waveform from `AMI_GetWave()` and convolves it with analog impulse response. Next, the resulting waveform is presented at the Rx AMI model input. The process is repeated in Rx DLL. The resulted out-

put is either impulse response from Rx `AMI_Init()` or the waveform from Rx `AMI_GetWave()`. The simulator will take the output from Rx DLL to conduct the final eye diagram analysis.

IBIS-AMI model is an industry standard which defines the interface between model and simulator. Generating IBIS-AMI model is not trivial because it involves circuit design, signal processing, and software development. There are tools available to help IBIS-AMI model generation. However such tools require licenses, are not easy to learn. In this work, we report our approach of implementing the PAM-4 behavior model generated from LVFFN into IBIS-AMI model. This model can be simulated in Keysight ADS, a well-known circuit simulator.

## 5.2 IBIS-AMI Implementation

The trained LVFFN is implemented in IBIS-AMI DLLs. Figure 5.2 presents the simulation environment in ADS for LVFFN IBIS-AMI model. The Tx DLL is a come-and-pass model which passes the bits received from simulator to Rx DLL. The whole PAM-4 LVFFN model is implemented in Rx DLL. There is no channel in between Tx DLL and Rx DLL since the LVFFN model incorporates the whole PAM-4 system including channel and packages. The output of Rx DLL is feed to the eye diagram generator and analyzer for signal integrity analysis.

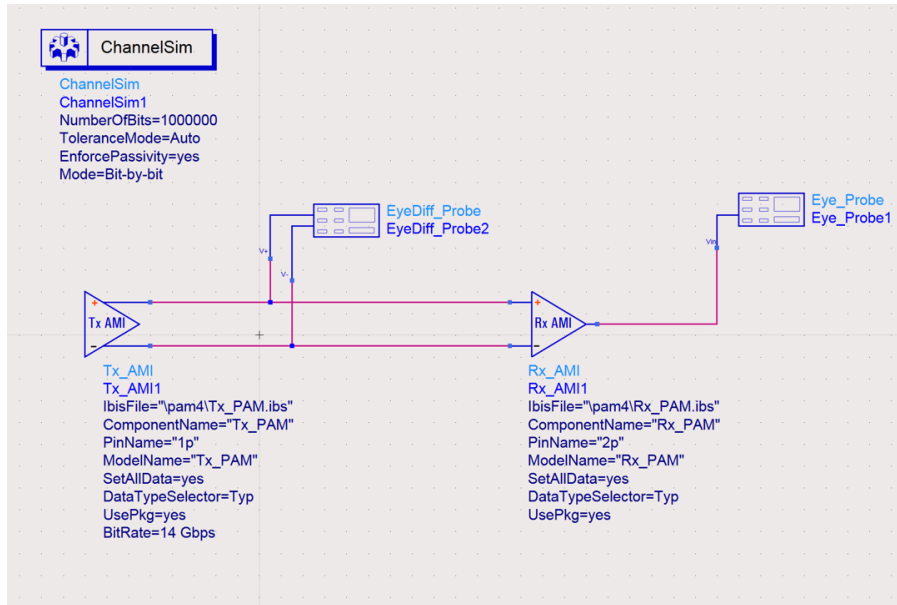


Figure 5.2: Snapshot of simulation environment in ADS for LVFFN IBIS-AMI model

In Rx DLL there are three functions implemented: `AMI_init()`, `AMI_getWave()`, and `AMI_close()`. The execution flow is illustrated in figure 5.3. Once the bit signal is ready at input of Rx DLL, the simulator calls `AMI_init()` function. This function allocates necessary memory and loads Laguerre functions. The trained FFN parameters are loaded as well. At the end of `AMI_init()` function call, messages contains information such as execution status, debug information, intermediate values, are printed at the simulator interface. After `AMI_init()` returned, simulator calls `AMI_getWave()` function multiple times to conduct the bit-to-bit simulation. The LVFFN model is implemented in this function. The bit information is conveyed to `AMI_getWave()` and then is returned to the simulator after processing completed. Once all the bits are processed, simulator calls `AMI_close()` function to clean up the memory allocated and to inform the simulator that the whole simulation has been completed successfully or with errors. In the following, the simulator will launch eye diagram analysis routine to conduct eye-diagram analysis.



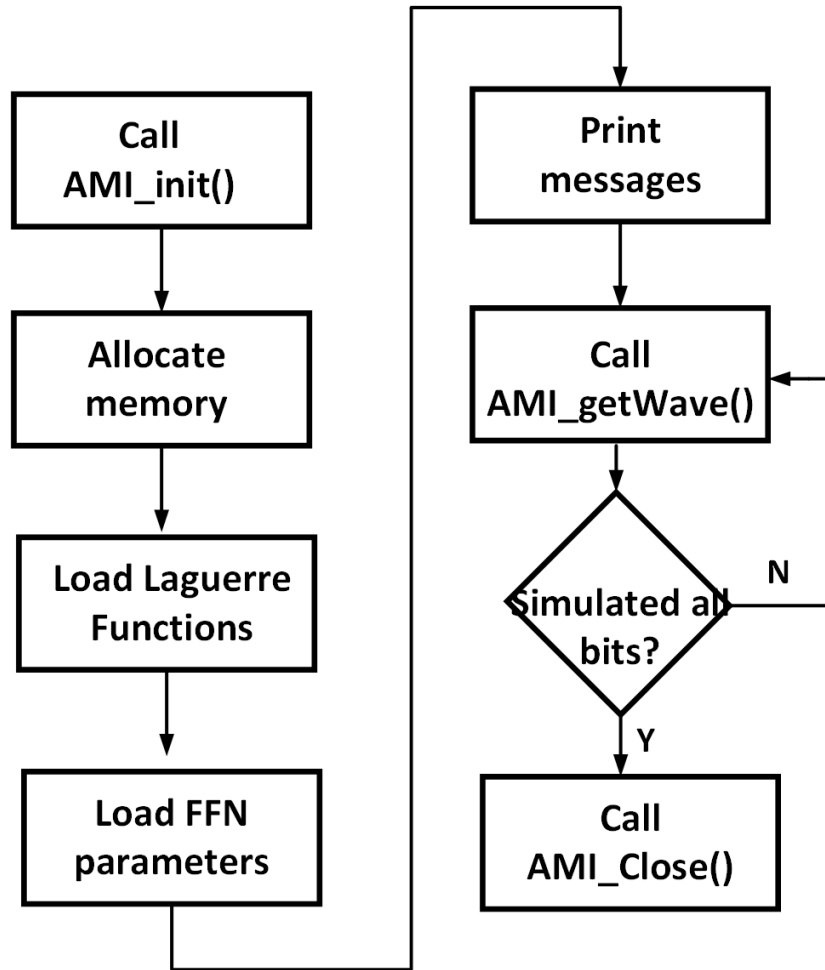
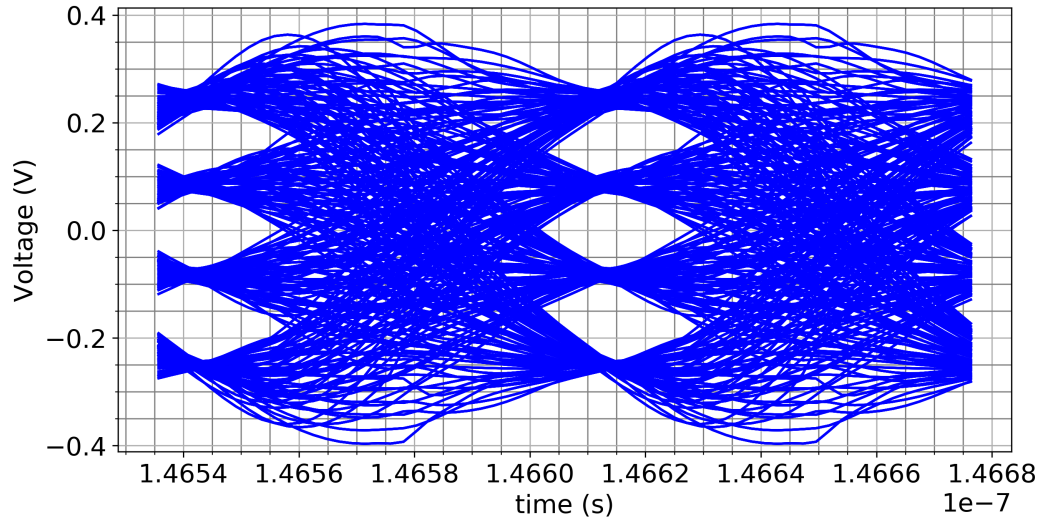
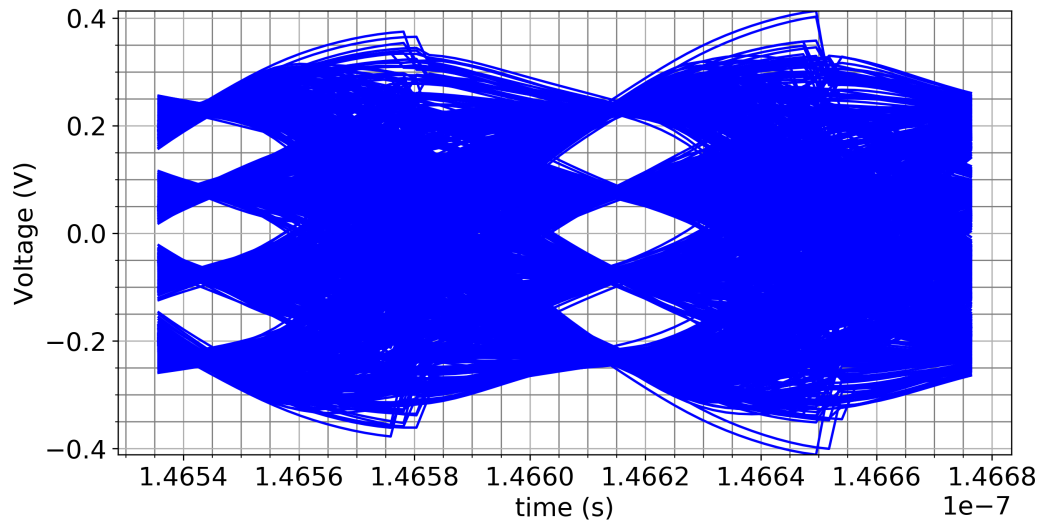


Figure 5.3: Flow diagram of Rx DLL execution

Figure 5.4 shows eye-diagram plotting for PAM-4 LVFFN model and the reference signal. The bits for LVFFN model are generated from ADS and the reference signal are generated from the PAM-4 IBIS-AMI model provided by industry. For PAM-4 model, there are three eyes for eye-diagram plot. As seen from the figure 5.4, eyes in both plots look very similar. Simulation of 1 million bits in ADS with the LVFFN model takes 142s. The majority computation power goes to convolution of input signal with Laguerre functions. Speed-up can be achieved further by replacing the convolution by a recursive computation using equation 4.3 and 4.4



(a)



(b)

Figure 5.4: Eye-diagram plot for (a)PAM-4 LVFFN model and (b)reference model

### 5.3 THE ezAMI software

IBIS-AMI provides a great interface for HSL modeling. However, generation of IBIS-AMI model is not trivial. It requires cross-disciplinary expertise such as HSL circuit design, signal integrity, and C/C++ programming across different operating systems and platforms. There are commercial software for

IBIS-AMI model generation. They are Matlab SerDes Toolbox<sup>TM</sup> in conjunction with Quantum Channel Designer from SiSoft, SystemVue from Keysight, and AMI Builder from Cadence. These tools are excellent tools in generating industrial standard IBIS-AMI model for high speed link transmitter/receiver systems such as PCIe4, DDR5, and USB 3.1. However all these software are graphic and wizard based in creating the model. The details behind generating IBIS-AMI model is hidden from users. Hence, the flexibility is limited in customizing the model when using commercial IBIS-AMI generation tools.

ezAMI is an open source software for IBIS-AMI model generation. The purpose of this software is to grant users more flexibility in IBIS-AMI model generation. It allows user to either generate industrial standard SerDes model, or implement any customized model, i.e. neural network model, into IBIS-AMI standard. As the software is an open source software, users are able to modify the software source code to add more flexibility to their IBIS-AMI model generation. Moreover, this software supports model pre-generation verification. Users can conduct high-fidelity simulation in ezAMI before generating the final IBIS-AMI model.

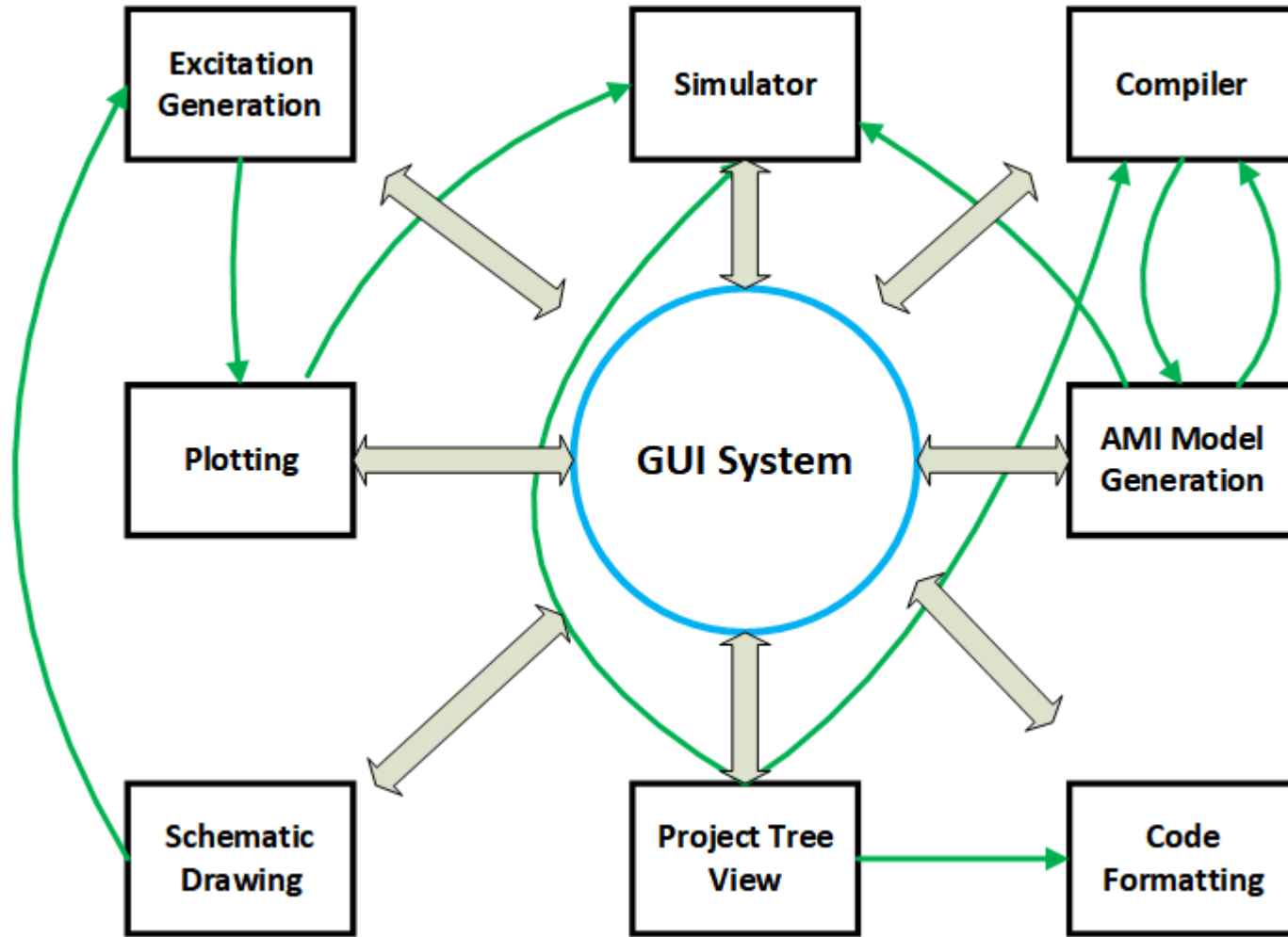


Figure 5.5: ezAMI software architecture diagram

### 5.3.1 Architecture of the Software

The software design is following the object oriented design (OOD) rule. Each function module is an object. The communication between objects is either through class inheritance or the signal-slot mechanism that is built in the Qt interface design environment (IDE). Figure 5.5 shows the flow diagram of the ezAMI architecture. Each box represents an function object. The arrow lines between those objects denote the interactions. The center blue circle represents the top graphic user interface (GUI). All other rectangle boxes are the sub systems interacting with the main GUI. The whole software has the following function blocks.

- Main interface
- Simulator
- Compiler
- Excitation generation
- Plotting
- Schematic drawing
- Project tree view
- Code formatting
- AMI model Generation

Detailed description of each function object is presented in the following.

#### Main GUI interface

The main GUI interface is implemented in `MainWindow.cpp` . It is the top level object which interacts with each sub-object as illustrated in figure 5.5. During initialization, an instance of each sub-object will be created. Table 5.1 summarizes the functionality of the subroutines.

Most of those functions are the handling functions when a menu or button is clicked in the software main interface. Communication between the objects

is also handled in some of the functions, e.g. `updateProjectTreeFromCCompiler()` which handles any signal sent by the compiler object. More detailed discussion on this topic will be presented in the following object introduction sections.

## Simulator

Simulator object is implemented in the `simulator.cpp`. According to figure 5.5, the simulator receives the excitation waveform from plotting object and then prepare for simulation input. This is done in the function `receiveInputWave()` and `prepareInputWave()`. Once the input waveform is ready, then the `run()` function is called, in which a DLL is loaded and three AMI functions: `AMI_Init()`, `AMI_GetWave()`, and `AMI_Close()` are called consecutively. At the end of compilation, the `prepareOutput()` is called and the `outputReady()` signal is emitted. The `setDllPath()` function is called whenever a DLL is selected and is linked to the model in the AMI model dialog.

## Compiler and AMI model generation

The compiler object is implemented in the `compiler.cpp`. The AMI model generation object is implemented in the `generatedlldialog.cpp`. Both objects are interacting closely with each other. Once all the development is completed, the next step is compiling. A dialog window is popped up when the build action is clicked from the software main interface.

There are three import functions: `compile()`, `generateDll()`, and `generateAmiFile()` in compiler object. The first function builds the DLL for simulation. The second function builds the IBIS-AMI DLL for simulation in circuit simulator. The last function generates the `.ami` file that is associated with the DLL when simulated in circuit simulator. There are two signals: `updateProjectArch()` and `sendBuildInfo()`. The first one is emitted when the project architecture is changed by the compilation. The main interface updates project management window accordingly. The second sends the rules

Table 5.1: Main interface functions

Functions	Comments
MainWindow()	Object constructor
~MainWindow()	Object destructor
on_actionAMI_Generation_triggered()	Generate AMI dialog
on_actionRun_2_triggered()	Start simulation
on_actionAMI_Generation_triggered()	Start AMI model generation
on_actionBuild_2_triggered()	Start compiler build
on_actionSave_All_triggered()	Save project and code
on_actionSave_triggered()	Save just the code
on_actionExcitation_triggered()	Draw excitation schematic
on_actionAMI_triggered()	Draw all schematic
on_actionPlot_triggered()	Draw plot schematic
on_doubleClicked()	Action when schematic is double clicked
isInRegion()	Check which schematic is double clicked
on_actionOpen_triggered()	Open new project
on_projectTreeView_doubleClicked()	Open file from project management window
onCustomContextMenu()	Handle right click in project management window
setupContextMenu()	Set up right click menu
setProjectInfo()	Initiate project architecture when new project created
on_CustomContextMenu_triggered()	Handle a click in right-click menu
saveProjectFile()	Save project info into file
saveCodeFile()	Save code into file
on_actionClean_2_triggered()	Clear project management window when clear project menu is clicked
on_actionLVFFN_triggered()	Setup LVFFN example in ezAMI
updateProjectTreeFromCompiler()	Update any change when compilation is done
addModelFilesInDirectory()	Parse files in the directory selected and add them to project model
updateModelByChild()	Update project model when branch change
parseAmiFunctions()	Fill three AMI template functions into code area
copyPath()	Copy whole LVFFN project file into the new directory user selected

to the AMI model generation object for AMI model and .ami file generation.

The AMI model generation object is essentially a dialog window which

allows users to select the platform on which the AMI model is going to be used and to specify parameters in the .ami file. Besides, any building errors and messages generated by the compiler will be displayed to the users. This object calls function in the compiler object to conduct the actual building and compilation.

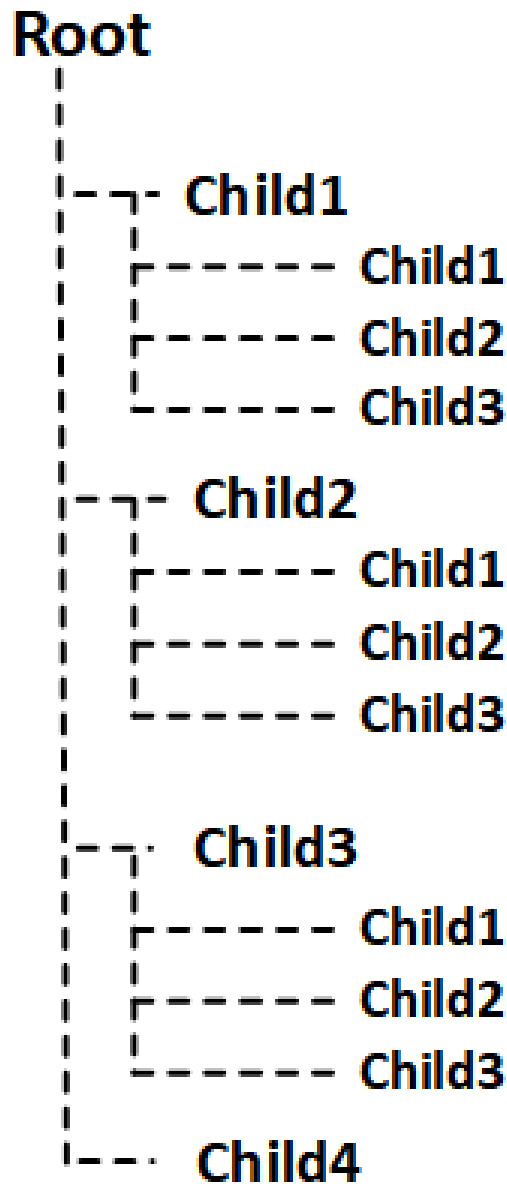


Figure 5.6: Tree model diagram



## Plotting

The plotting object is another big object which is implemented in the `plotting.cpp`. This object draws the waveform generated from the excitation object and simulator object. The `setupCor()` function creates an empty coordinate system from scratch. `XaxisSetup()` updates the X axis with respect to a variety of time scales.

The `coordinateSetup()` is a function handling communications between the plotting object and the excitation generation object. It receives information such as type, sample per bit, total number of bits, and amplitude. Then it calls the `updatePlotPoints()` and `updateCoor()`.

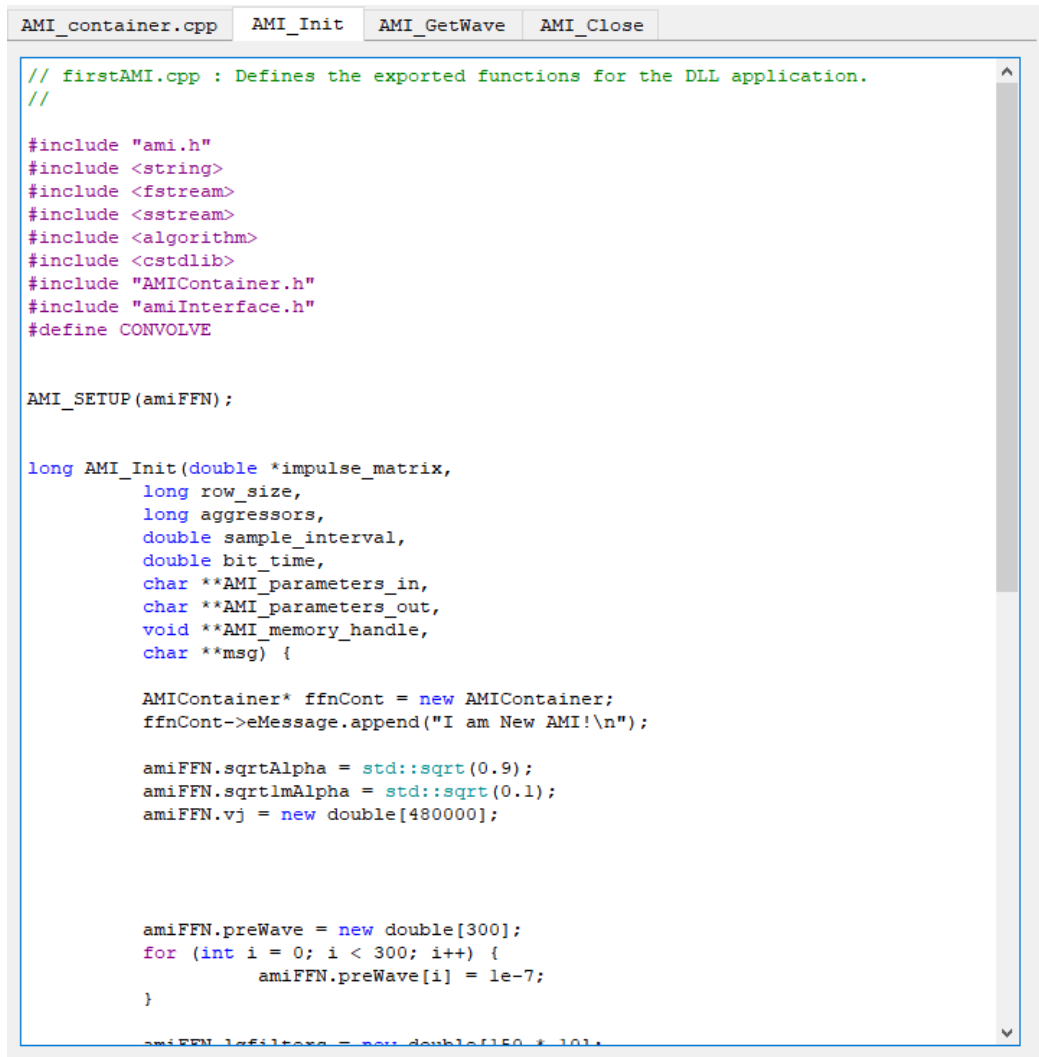
The `updateCoor()` function clears old plotting information and adds new plot into the coordinate system in terms of the updated plotting information from either the excitation generation or simulator object. The `updatePlotPoints()` actually does plotting by drawing all the individual points. The last function, `addSimulatedWave()`, receives simulated waveform from the simulator object and conducts necessary processing, and subsequently calls the `updateCoor()` function.

## Excitation generation

The excitation generation object and schematic object are implemented in the `excitationdialog.cpp`. The excitation generation object generates two level modulated and four level modulated PRBS bit sequences. The dialog window is activated by a double-click on the excitation icon. Users can specify the data rate, samples-per-bit, total number of bits, amplitude, and offset in the dialog. With the input from user, the sample waveform is plotted in real time at the bottom half of the dialog window. User specifies the excitation type which is either PAM-2 or PAM-4. The specifications in the dialog window is routed to the simulator and plotting object when the "Ok" button is clicked.

There are two important functions implemented in the `excitationdialog.cpp`. They are `updateHash()` and `getSamples()`. the user specified information

in the dialog window is stored in a hash table. The `updateHash()` function updates the hash table whenever there is a change in the dialog window. The `getSamples()` function updates the excitation waveform information for plotting.



```
AMI_container.cpp  AMI_Init  AMI_GetWave  AMI_Close

// firstAMI.cpp : Defines the exported functions for the DLL application.
//

#include "ami.h"
#include <string>
#include <fstream>
#include <sstream>
#include <algorithm>
#include <cstdlib>
#include "AMIContainer.h"
#include "amiInterface.h"
#define CONVOLVE

AMI_SETUP(amiFFN);

long AMI_Init(double *impulse_matrix,
             long row_size,
             long aggressors,
             double sample_interval,
             double bit_time,
             char **AMI_parameters_in,
             char **AMI_parameters_out,
             void **AMI_memory_handle,
             char **msg) {

    AMIContainer* ffncont = new AMIContainer;
    ffncont->eMessage.append("I am New AMI!\n");

    amiFFN.sqrtAlpha = std::sqrt(0.9);
    amiFFN.sqrtlmAlpha = std::sqrt(0.1);
    amiFFN.vj = new double[480000];

    amiFFN.preWave = new double[300];
    for (int i = 0; i < 300; i++) {
        amiFFN.preWave[i] = 1e-7;
    }

    amiFFN.lofilters = new double[150 * 10];
```

Figure 5.7: Code region

## Schematic

There is a schematic window in the main interface. It locates at the lower left corner of the main window. The schematic symbols are handled by the `svgload.cpp`, and `sceneclick.cpp`. Both classes are derived from the

`QGraphicView` and `QGraphicsScene` class. Most of the functions are overloaded functions from the parent classes. Both objects handle loading SVG icons and double-clicking events.

## Project Tree View

Project tree view object is a project hierarchy maintainer. The object is implemented in the `projecttreemodel.cpp` and `projecttreeitem.cpp`. The structure of the project is a tree-like structure as shown in figure 5.6. There is a root node under which multiple children inherit. The child could also have children. Each node preserves partial project information which is retrieved and used by other objects.

There are three level descendants for the ezAMI software. A project root node serves as the ancestor. At the second level, there are five siblings. They are the project name node, source code node, executable node, AMI model node, and resource node. The project name node contains the project name and the local directory storing the project. The source code node contains all the `.cpp` and `.h` files. The executable node saves the DLL for the simulator project. The AMI model node contains the AMI DLL and the `.ami` files after the AMI model generation is completed. The resource node saves all the `.txt` files that are used during simulation.

The `projecttreeitem.cpp` implements the `projectTreeItem` object which maintains a single tree node. It stores pointers to its parent and children. The data associated with this object is stored in a vector. User can add/remove child to the node using the function `appendChild()`, `removeChild()`, and `removeAllChild()`, respectively. Data can be retrieved using the `data()` function. Parent/child node can be accessed through the `child()` and `parentItem()` function. The node information such as the number of child, data elements, and the position of itself in its parent node can be read out with the `childCount()`, `columnCount()`, and `row()` function.

The `projecttreemodel.cpp` is an object maintains the project architecture. This is an important object to other objects in the software. For

instance, the compiler will have to retrieve the information stored in the source code node before compilation. The main interface reads the information in this object to update the graphics display in the project management window. The object is inherited from the `QAbstractItemModel`. All the functions implemented in this object are overloaded functions except for the `openModelData`, `getProjectRoot()`, `removeRow()`, and `setupModelData()` function. The documentation of the overloaded functions can be found in the documentation of the `QAbstractItemModel` object on the Qt maintenance website.

The `openModel()` function basically reads in the `.ezproj` project file, populates the contents, and creates a project tree model. The `getProjectRoot()` function retrieves the ancestor of the tree model. The `removeRow()` removes the child given the parent node and the position.

## Code formatting

ezAMI software is essentially an integrated development environment (IDE). The code formatting object is to make the coding in the code space more user friendly. This object is implemented in the `codeformathighlight.cpp`. The object specifies font color for key words in the coding region, e.g. variable type, macros, control key words etc. Figure 5.7 shows the result of this object. When the project tree model object loads source code into coding space, the source code is first parsed by the code formatting object and is rendered with respect to the keywords as shown in figure 5.7.

The `textToProcess()` function reads all the text contents as a string from the code space. Then it parse the text into a variety of categories such as macros, comments, and regular code. The `setCommentFlag()` function sets a flag specifically for comments in between `"/**"` and `**/"`. There is a special case `"/**/"` which is rare but there is possibility of existence. The `setCommentFlag()` function is designed to address such scenario. The `setTextColor()` conducts the evaluation per the input word and returns the font color for the text editor to render.

The screenshot displays the ezAMI1.0 software interface, which is divided into several sections:

- Top Menu Bar (A):** Contains 'File', 'Edit', 'Project', and 'Help' menus. Below the menus are icons for 'Save', 'Save All', 'Build', 'Run', and 'AMI Generation'.
- Project Tree (B):** A tree view showing the project structure:
  - Project: ezAMI1.0
  - LVFFN.ezproj (Path: F:/Research/ezAMI/AMI/LVFFN.ezproj)
  - Source Code:
    - ami.cpp (Path: F:/Research/ezAMI/AMI -backup/ami.cpp)
    - ami.h (Path: F:/Research/ezAMI/AMI -backup/ami.h)
    - AMI\_container.cpp (Path: F:/Research/ezAMI/AMI -backup/AMI\_container.cpp)
    - AMIContainer.h (Path: F:/Research/ezAMI/AMI -backup/AMIContainer.h)
    - amiinterface.h (Path: F:/Research/ezAMI/AMI -backup/amiinterface.h)
  - Executable: AMI Model
  - Resource:
    - lgfilters.txt (Path: F:/Research/ezAMI/AMI -backup/lgfilters.txt)
    - weight.txt (Path: F:/Research/ezAMI/AMI -backup/weight.txt)
- Block Diagram (C):** A flow diagram showing a signal source (a square wave) connected to a central block labeled 'AMI', which is then connected to a plotter displaying a filtered signal.
- Code Editor (D):** Displays the source code for 'ami.cpp'. The code includes a header file 'ami.h' and defines three functions:
 

```

//This is the ami interface file.
#include "ami.h"

long AMI_Init(double *impulse_matrix,
              long row_size,
              long aggressors,
              double sample_interval,
              double bit_time,
              char **AMI_parameters_in,
              char **AMI_parameters_out,
              void **AMI_memory_handle,
              char **msg) {
    return 1;
}

/*****/
long AMI_GetWave(double *wave,
                 long wave_size,
                 long aggressors,
                 double *clock_times,
                 char **AMI_parameters_out,
                 void *AMI_memory) {
    return 1;
}

/*****/
long AMI_Close(void *AMI_memory) {
    return 1;
}

```

Figure 5.8: ezAMI main interface

### 5.3.2 Software interface

In this section, the software main window is introduced. The main interface has four main regions: The menu action region, the project management region, the schematic region, and the coding region. Figure 5.8 shows the main window in which label A, B, C, and D denote the four regions mentioned above respectively.

#### Menu and actions

The menu actions are placed at the top of the software main interface. Figure 5.9 shows all the expanded menu. Figure 5.9(a) shows the expanded **File** menu. **New** is a sub-menu which has two actions: **Project** and **File**. When **Project** or **File** is clicked, a new project or a new file creation dialog is promoted. The **Open** action promotes the open project dialog. It allows user to open a `.ezproj` project file from a directory. **Example** is a sub-menu which includes only one action: **LVFFN**. The **LVFFN** is a built-in example project in which my PhD work on LVFFN is implemented. In Appendix B, a detailed tutorial is included for interested user to learn how to use this software.

The **Save** and **Save All** are actions saving codes in coding region and the project hierarchy in files. The **Save** action only saves code in source code files, while the **Save All** saves both code and project hierarchy. The **close** action closes the whole software. It is recommended that user should save all the information before closing the project.

In the **Edit** menu (figure 5.9(b)), there are three actions: **Copy**, **Cut**, and **Paste**. The implementation of these three actions are has not been completed in the current 1.0 release, but will be completed in the future release.

The **Project** menu (figure 5.9(c)) has six actions implemented. The **Copy Project** action copies the whole project directory into the new location. The **Close Project** will only clear up the project management region and leave others as is.

The **Build** action calls compiler to compile all the source code and gener-

ates a DLL which will be used in the simulation later on. The **Clean** action deletes all the compiler generated files and restores the directory to the initial state. The **Run** action calls the simulator to conduct the simulation using the generated DLL with the excitations generated by the excitation object. The **AMI Generation** action will generate IBIA-AMI DLL and .ami file, which can be simulated in circuit simulation software like ADS.

The **Help** menu contains all the actions related to software logistics such as software tutorial, documentation, and license information. The ezAMI 1.0 has only implemented the software introduction and license action which is the **About** action (figure 5.9(d)).

Figure 5.9(e) shows the convenient menu bar in which the actions used frequently is added. There are three icons in the red frame. They are new actions for schematic drawing. The first icon with square wave in the icon is an action for adding the excitation generation symbol and plotting symbol into the schematic region (labeled C in figure 5.8). The second diagram with "AMI" in the icon draws excitation, AMI model and plotting symbol. The third icon only places the plotting symbol into the schematic region.

#### Project management region

The project management region is labeled B in figure 5.8. Once a new project is created or an existing project is opened, the project hierarchy is displayed in this region. The diagram displayed is a tree-like structure in which one row represents one node. Each row has two columns. The left column contains the name and the right one contains the location on local computer. The project structure has a root node which is the name and the location of the project. There are four children nodes that are the source code, executable, AMI model, and resource node. These four nodes have their own child/children which is/are the specific file/files under the category. User can add/remove files through right-clicking on the node. The file contents are displayed in the code region when the node is double-clicked.

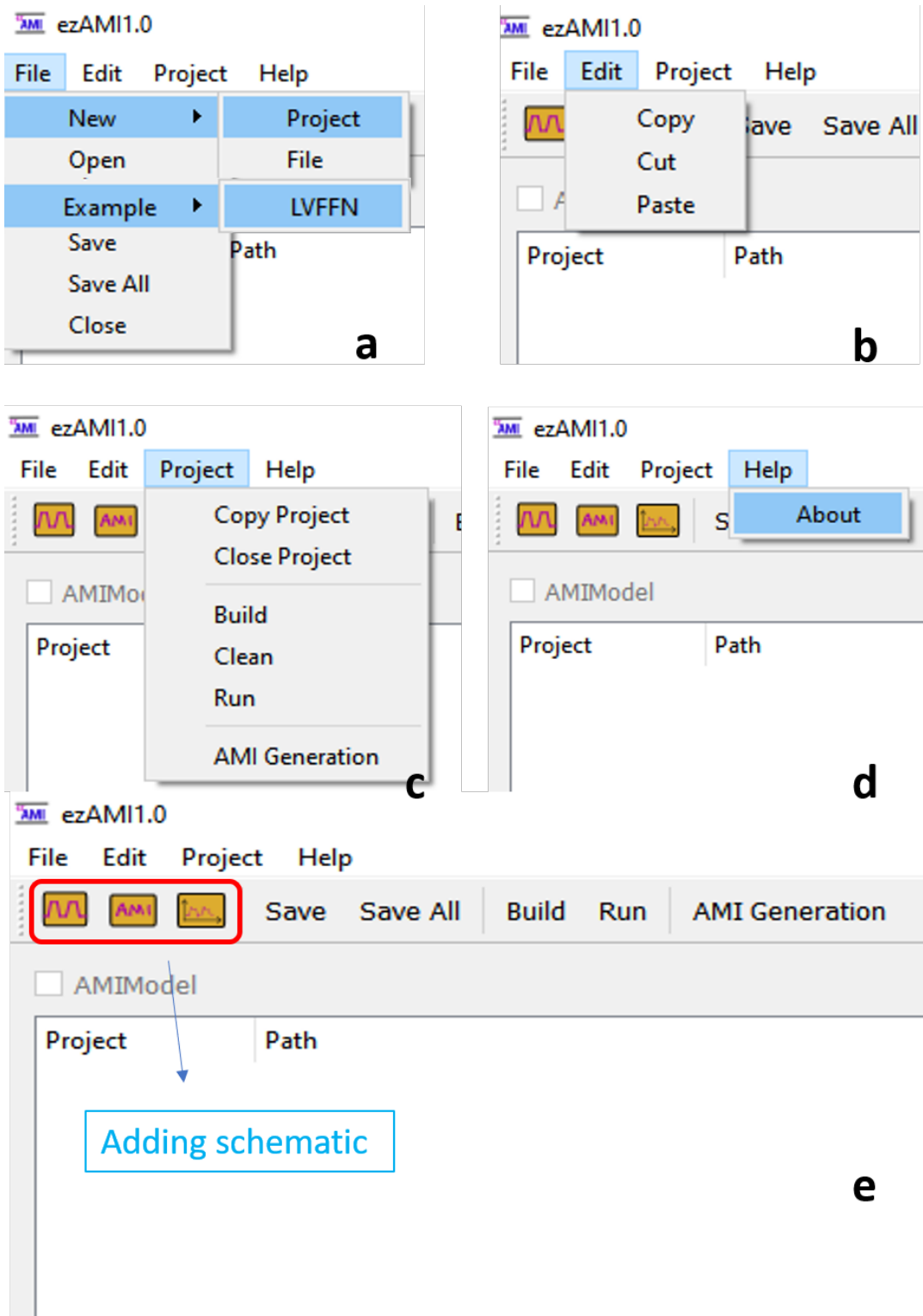


Figure 5.9: Menu actions



## Schematic region

Schematic region is labeled C in figure 5.8. As shown, there are three symbols which represents the excitation, AMI model, and plotting. The connection in between represents the data flow from the excitation to the plotting through the AMI model. The excitation is processed in the AMI model. The result is plotted in the plotting object. Figure 5.10 shows the excitation generation dialog and AMI model dialog when the specific symbol is double-clicked. The excitation generation dialog generates PRBS excitation for PAM-2 and PAM-4. Users can select which of them to be generated using the checkbox. Users can specify data rate, samples per unit interval, magnitude, offset etc. All those information will be used for excitation waveform generation. The model association dialog allows user to specify the DLL for simulation. The DLL model is selected by choosing the DLL file in a directory through browsing. Once the DLL model is selected, the model is associated with the AMI symbol in the schematic window. Once the simulation is launched, the simulator will load the DLL and call the functions in this DLL for excitation waveform processing.

## Coding region

The coding region is labeled D in figure 5.8. It is for users to write their code which will be compiled into the DLL. The coding region is a text editor with keyword rendering. The code loaded will be pre-processed to highlight the keywords such as variable declaration, control syntax, macros, and so on so forth. The coding region has four windows. They are `Your_Code`, `AMI_Init`, `AMI_GetWave`, and `AMI_Close`. `Your_Code` interface displays the code loaded by double-clicking the file in project management region. `AMI_Init`, `AMI_GetWave`, and `AMI_Close` are interfaces having the `AMI_Init()`, `AMI_GetWave()`, `AMI_Close()` template functions displayed. Users can modify them as desired .

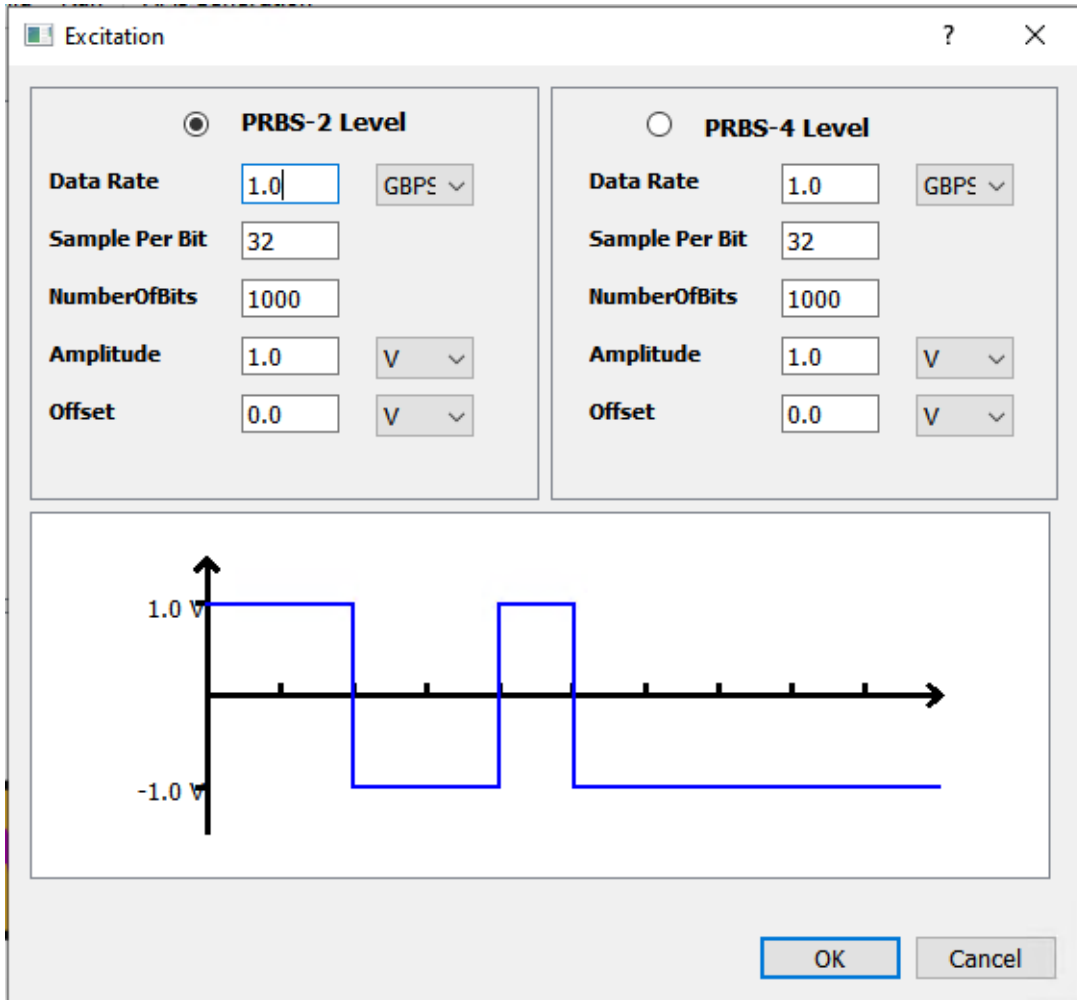


Figure 5.10: Excitation generation dialog

## 5.4 Conclusion

The behavioral model generated from machine learning cannot be directly simulated in circuit simulator software, which significantly limits its applications in practice. We successfully implemented the LVFFN model in IBIS-AMI, an industrial standard. The model was verified using ADS from Keysight. Eye diagram analysis was conducted in ADS as well. The eye diagrams from LVFFN obtained with ADS appear very similar to the one generated with the reference model.

To facilitate IBIS-AMI model generation for machine learning models, an

IBIS-AMI model generation tool software, ezAMI, is developed. The software is written in C++ and is following object-orientated design principal. The hierarchy of the software is discussed in this chapter. The main interface of the software is also introduced.

# Chapter 6

## Summary and Future Work

### 6.1 Summary

In this chapter, the completed work is first summarized. We will then go over the challenges associated with the completed work. The proposed solution is presented at the last. Figure 6.1 summarizes the current state of the research. The research started with VK extraction for HSL system behavior modeling using existing machine learning method. However the VKs extracted with existing method has the following drawbacks:

- The order of VKs extracted is the same as the length of input signal, which usually results in an astronomical number of VKs.
- It is impossible to reconstruct the signal with the vast number of VKs identified.
- Accuracy is compromised due to Taylor expansion approximation.

To tackle the challenges associated with the existing VK extraction method, we proposed a MPSNN method in which the mapping from neural network weights to VKs is analytical. The highest order of VK to be identified is detached from the select input signal length. The number of VKs extracted with MPSNN can be drastically reduced, which make the signal reconstruction possible with the extracted VKs. We validated the MPSNN with an analytical non-linear system. Then we employed such method to model PAM-4 and NRZ HLS systems. However, even the number of VKs can be reduced with our method, the curse-of-dimensionality is still exists. To overcome such challenge, we then proposed LVFFN to further reduce the number of parameters for the system under modeling. LVFFN expands VKs with a

finite number of Laguerre functions that are inter-orthonormal. Parameter extraction shifted from VKs in MPSNN to Laguerre parameters in LVFFN. The PAM-4 system is model with LVFFN in which only one layer and 10 neurons are employed.

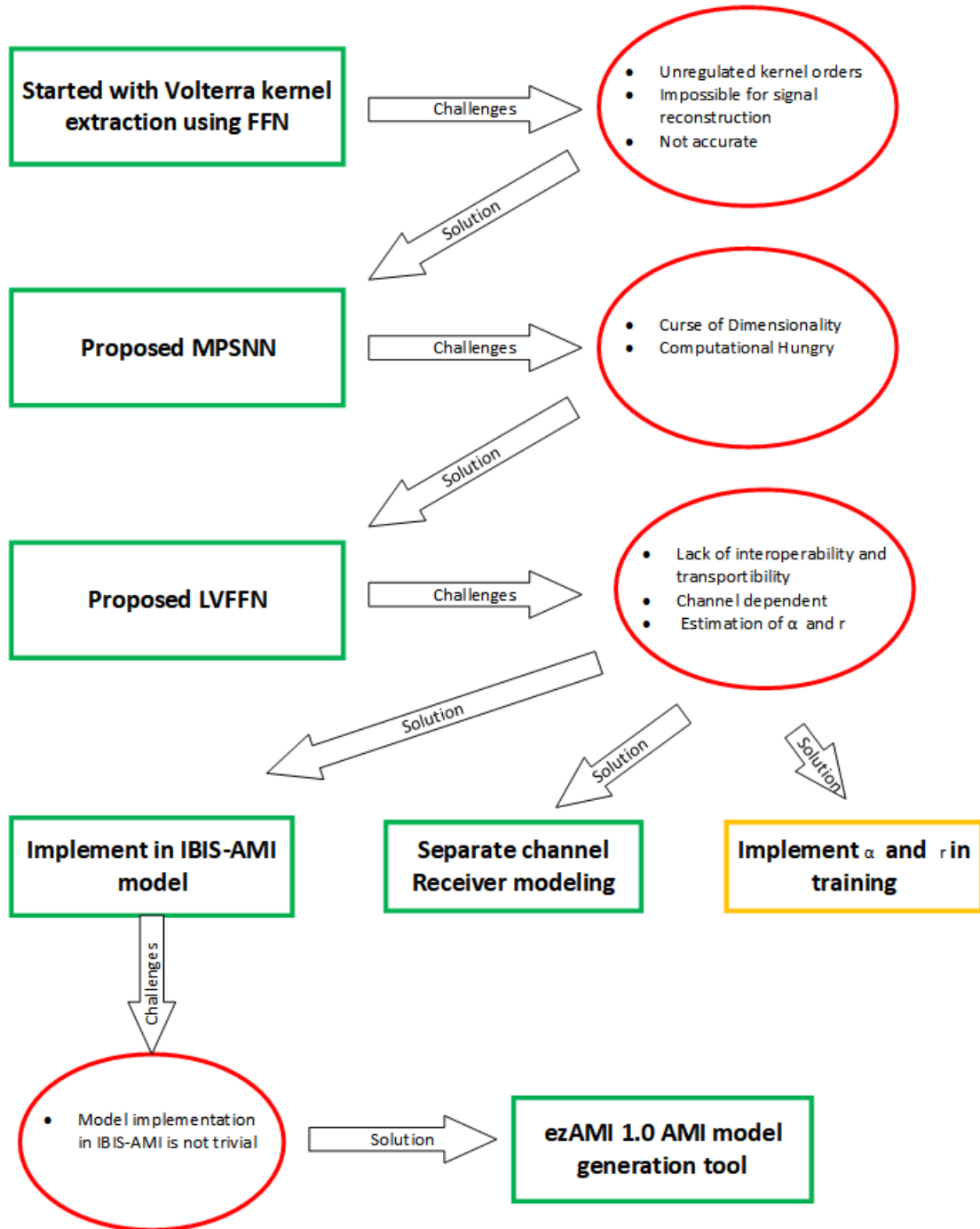


Figure 6.1: RoadMap of current research

The model size reduction is achieved up to 6 order of magnitudes using LVFFN compared to Volterra series. The computation efficiency improvement is also significant. Although, the model extracted with LVFFN is compact while maintaining the same accuracy, it has three main challenges which is listed below:

- It lacks interoperability and transportability which makes the model less useful
- It lacks flexibility because the model embrace the whole HSL system .
- The decay factor  $\alpha$  and the order of Laguerre function need to be identified separately

The interoperability and transportability challenges can be overcome by implementation of the LVFFN model into IBIS-AMI, an industrial standard for SerDes modeling. We have implemented the PAM-4 LVFFN into IBIS-AMI and conducted simulation in Keysight ADS. Conceptually, the IBIS-AMI model can be simulated and analyzed with any circuit simulator software that support IBIS-AMI standard.

The whole HSL system can be modeled with LVFFN. However such model lacks of flexibility for co-simulation. For example, once you have the model for the whole system, you cannot replace any individual component such as transmitter, channel, or receiver. Therefore, modeling individual component would make the model more practical for applications. To demonstrate the individual HSL component can be modeled with LVFFN, a NRZ system receiver in which FFE/DFE and CDR are implemented is modeled. As a proof-of-concept, the result show promising in such scenario. More research is expected to further improve the accuracy.

IBIS-AMI model generation is not trivial. It requires cross-disciplinary knowledge which involves circuit designing, signal processing, and software engineering. Typically, an AMI model generation could take 12 months to complete from scratch [57]. To overcome such challenge, an IBIS-AMI generation tool software, ezAMI, is developed. The first version, ezAMI 1.0, has been released. The software is an open source software under GPL 3.0 license. Interested readers can download the software installer and source

code from the following link: [www.ezamiuiuc.net](http://www.ezamiuiuc.net)

The ezAMI software has the following features

- In software C/C++ coding
- In software model pre-verification simulation
- NRZ and PAM-4 PRBS excitation generation
- Automated AMI model and AMI config file generation

The LVFFN model is implemented in the software as a built-in example. Interested users can use it as a tutorial to learn IBIS-AMI and this software.

## 6.2 Future work

Based on what we have accomplished, the following work are proposed for the future work.

### 6.2.1 Continue to improve the ezAMI software and add more features

The first version software has been released. However, it has a few things needed to be improved. First, the plotting coordinate in plotting window should be adapted according to the number of excitation samples. The first version has the X axis fixed. A variety of excitation has to fit into the plotting window when plotting is launched. If the plotting coordinate system can be adapted to the excitation length, the plotting will be more user friendly.

Second improvement could be done for the coding interface. The 1.0 version has a preliminary keyword color highlighting implemented. To make it more user friendly, features such as auto-indent, spell-correction, function recognition, and so on so forth should be added.

Third, the AMI generation dialog should be improved. Because of the tight schedule, only very basic functionalities are implemented. The parameter input interface should be restructured significantly.

Last but not the least, software maintenance should be made along the way using. There could be bugs, or crashes that are not identified before release. As more users are using it, those bugs/crashes could be exposed. Bug fixing, software maintenance should be done in a continuous manner.

Some new features that can be added in the next are listed below

- Add copy/paste actions
- Implement statistical simulation function
- Add draggable schematic icons
- Create Makefiles build to ease building process.

### 6.2.2 Development of channel independent IBIS-AMI HSL Model

As mentioned previously, the LVFFN PAM-4 model is channel dependent. The channel is included in the model. If there are changes in channel, the model needs to be regenerated. To overcome this challenge, we propose to develop a LVFFN model on the receiver, separating the channel out. This way, the channel can be freely replaced. We have done some proof-of-concept works on modeling HSL receiver. More research need to be done to work out a best model for modeling individual component in the HSL system to separate the channel out.

### 6.2.3 Decay Factor $\alpha$ and Laguerre Function Order $r$ Determination

The function of decay factor  $\alpha$  and the order of Laguerre function selection has been discussed in chapter 4. Determining  $\alpha$  and  $r$  is not trivial. One way



to determine  $\alpha$  is implement  $\alpha$  as one of the parameters in LVFFN. During training, the output error will be back propagated to  $\alpha$ .  $\alpha$  adjust by itself during training and will converge to the optimal value at last. The process is illustrated in Figure 6.2

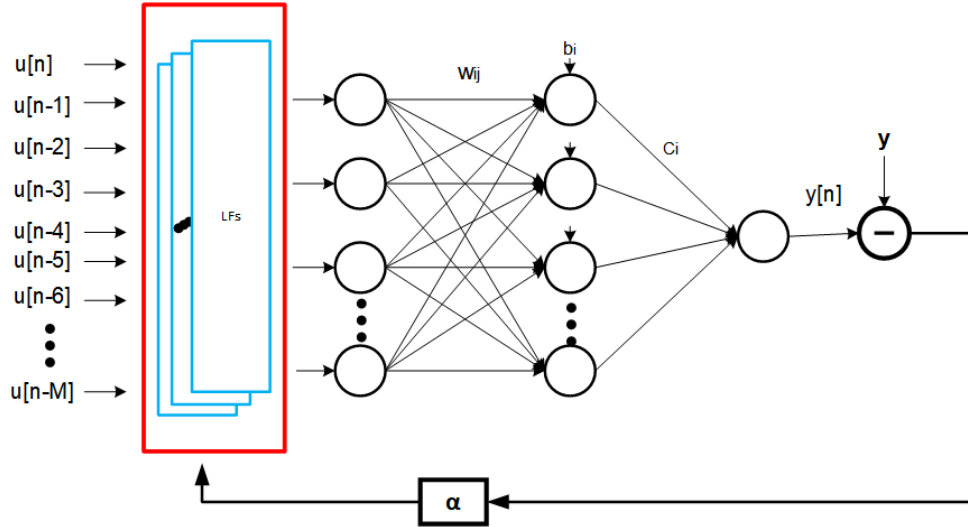


Figure 6.2: Illustration of determination of  $\alpha$

Determining the minimal order of Laguerre function  $r$  so that a most compact model can be achieved is another challenge. One way to optimize  $r$  is principal dynamic mode methods reported by Marmarelis et.al to model biological system[33]. The Procedure to identify the least  $r$  is list below:

1. Extract the 1st and 2nd VKs  $h_1$  and  $h_2$  through MPSNN method.
2. Construct the matrix  $\mathbf{M} = [h_1; s \times h_2]$ , where  $s$  is the root mean square of the input signal. This is to scale  $h_2$  so that the magnitude remain the same as  $k_1$
3. Conduct singular value decomposition (SVD) of matrix  $\mathbf{M}$ , where  $\mathbf{M} = \mathbf{U}\mathbf{\Lambda}\mathbf{V}'$ . Each column of  $\mathbf{U}$  is a orthonormal basis vector,  $\mathbf{\Lambda}$  is a diagonal matrix with corresponding singular values on the diagonal.
4. Select the top  $R$  significant basis vectors from  $\mathbf{U}$  based on the cumulative summation of singular values is greater than a threshold, which is usually chosen as 0.9[58].

# Appendix A

## ezAMI Development Environment Setup

### A.1 Installing Visual Studio 2019

ezAMI software requires MSVC C/C++ compiler. Installation of Visual Studio (VS) 2019 will automatically install the desired MSVC compiler VS2019 can be download from the link below

<https://visualstudio.microsoft.com/thank-you-downloading-visual-studio/?sku=Community&rel=16>

What you download from the link above is a VS2019 community version installer `vs_community__1176045115.1594347171.exe`. Launch the installer and follow the installation wizard to install the software on the C drive of Windows10 X64 version. If successful, the software installation location will be under `Local Disk(C:)>Program Files(x86)`. Software folder will be look like in figure A.1.

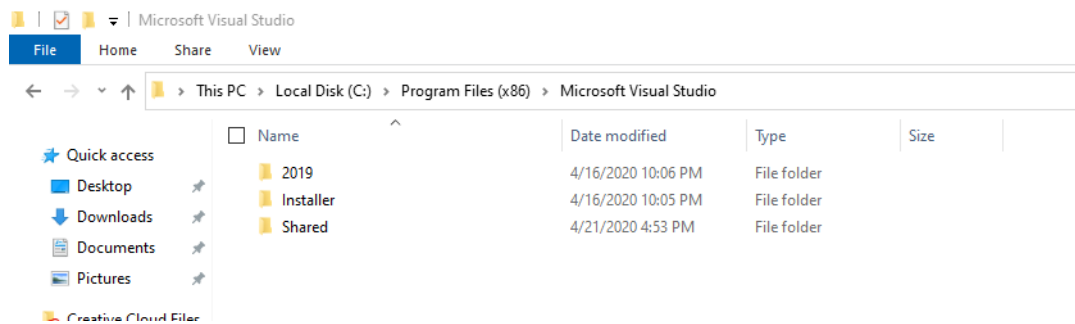


Figure A.1: VS 2019 installation snapshot

## A.2 Installing QT IDE 5.12

The ezAMI software is developed using C++ under QT IDE 5.12. The QT IDE 5.12 can be downloaded from the link below

<https://www.qt.io/offline-installers>

Download Qt 5.12.9 for Windows as shown in figure A.2

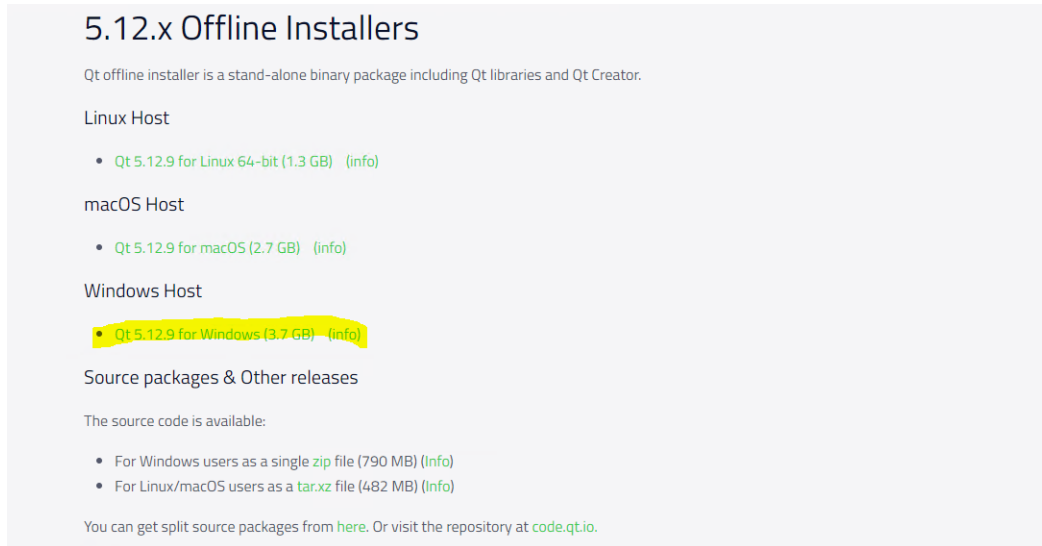


Figure A.2: QT IDE download

QT IDE installation requires a Qt account. If you have one already, you can just login to continue. Otherwise you have to sign one up (see figure A.3). Then you can follow the installation wizard till to the step **Qt 5.12.9 Setup**. Choose **Select All** as shown in figure A.4. After that, you can just follow the default setting to complete the installation. Once installation is completed, you can launch the Qt Creator 4.9 as shown in figure A.5. The main interface is shown in figure A.6.

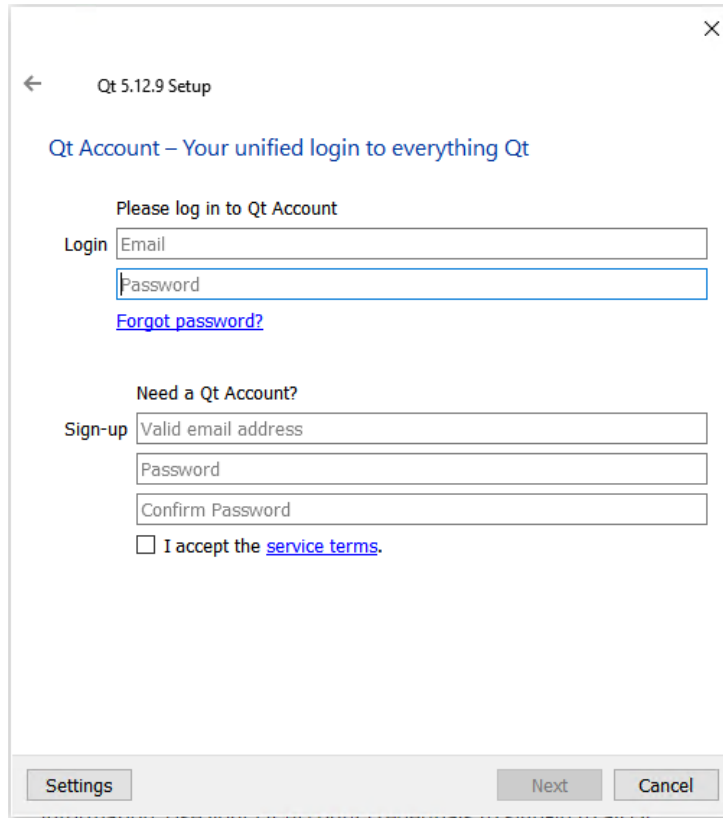


Figure A.3: QT IDE installation login

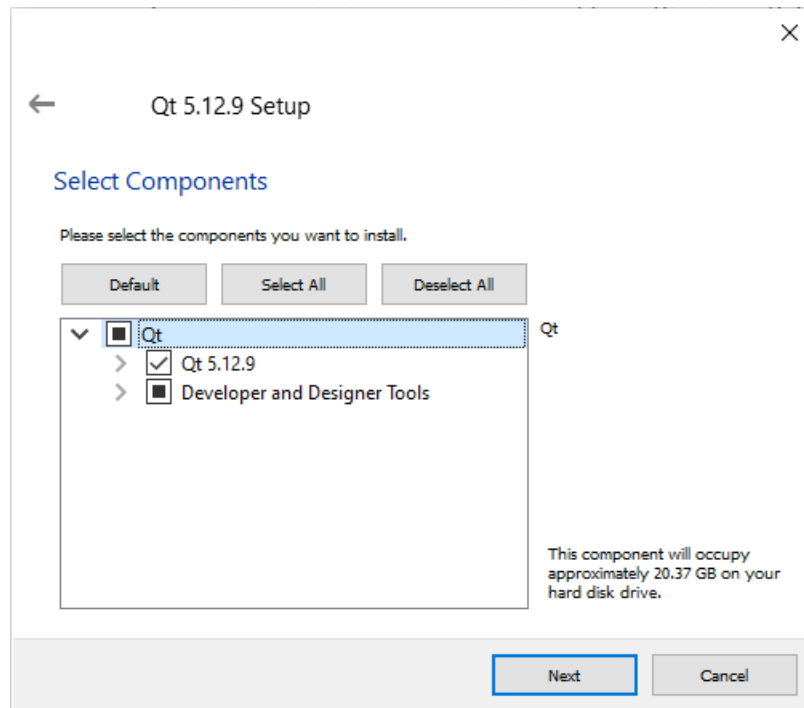


Figure A.4: QT IDE installation setup

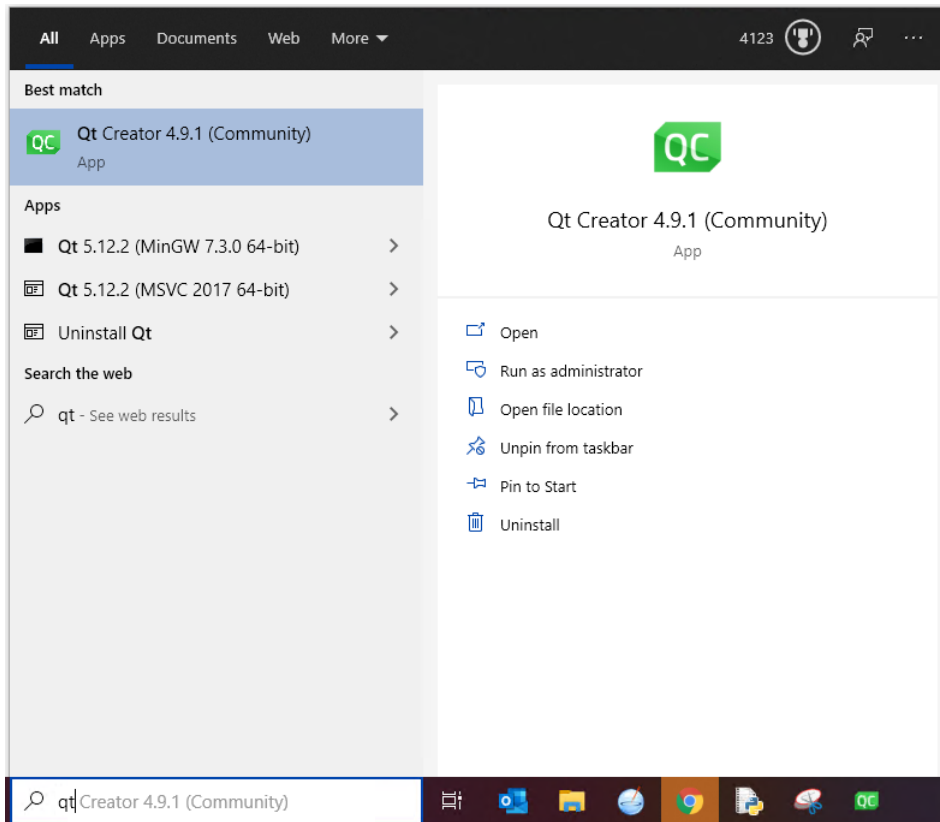


Figure A.5: Launch QT Creator

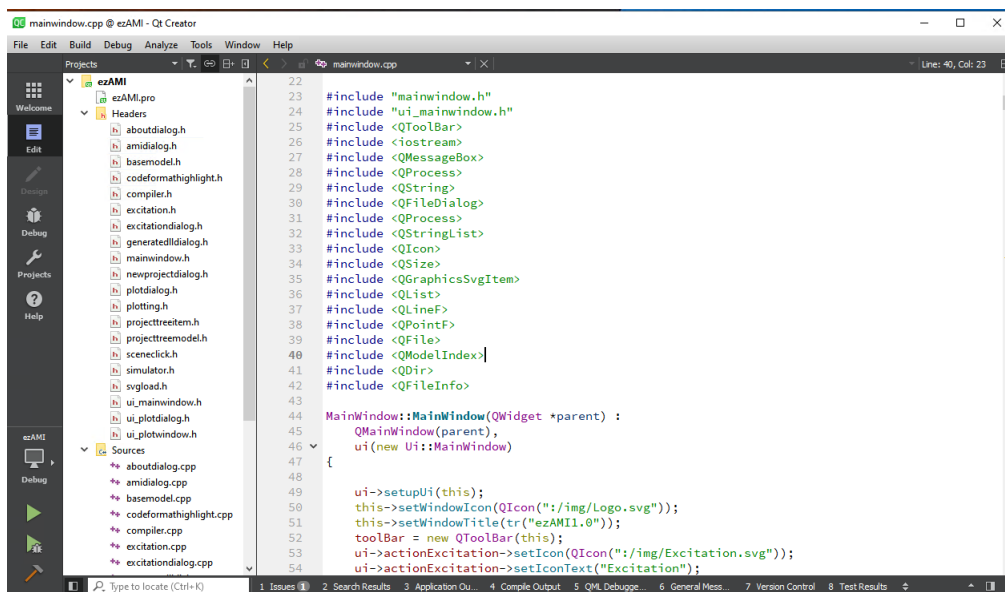


Figure A.6: QT Creator interface

# Appendix B

## Tutorial for LVFFN example in ezAMI

### B.1 Install ezAMI Software

Download ezAMI installer from the link below

[www.ezamiuiuc.net](http://www.ezamiuiuc.net)

Launch installer and follow the installation wizard to complete the installation (figure B.1).

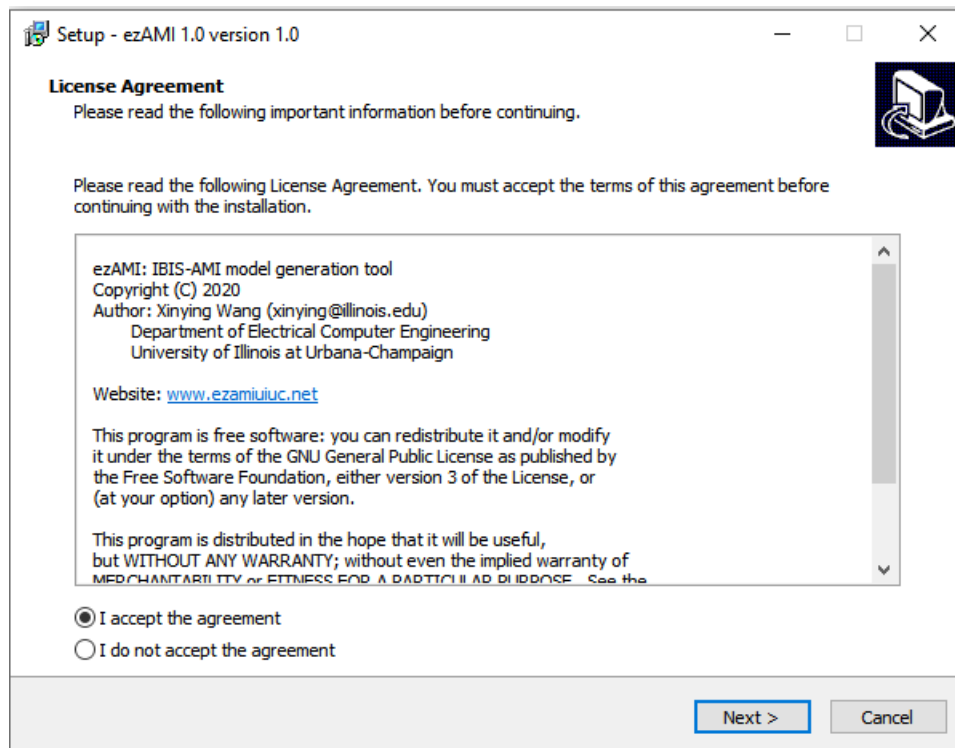


Figure B.1: ezAMI installation

## B.2 LVFFN example

Launch software and click the "AMI" icon pointed by the blue arrow to draw the schematic (figure B.2)

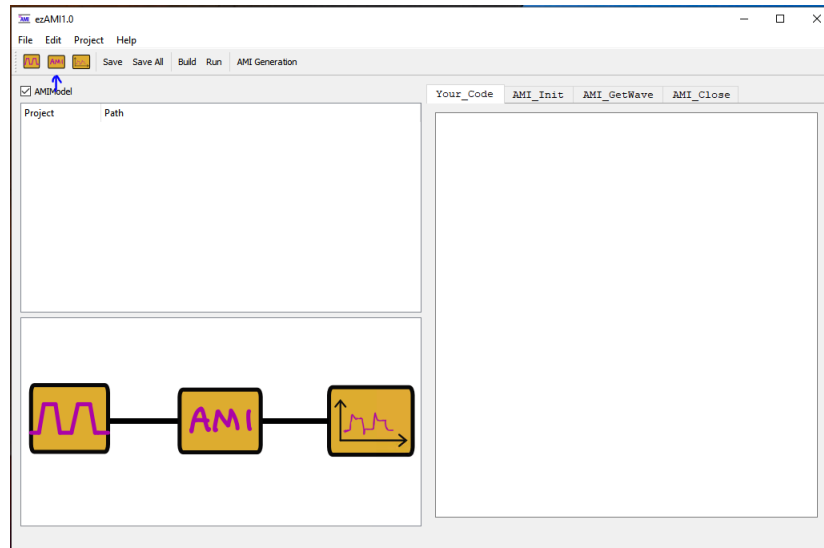


Figure B.2: Add schematic

Select "File→Example→LVFFN" as figure B.3 shows

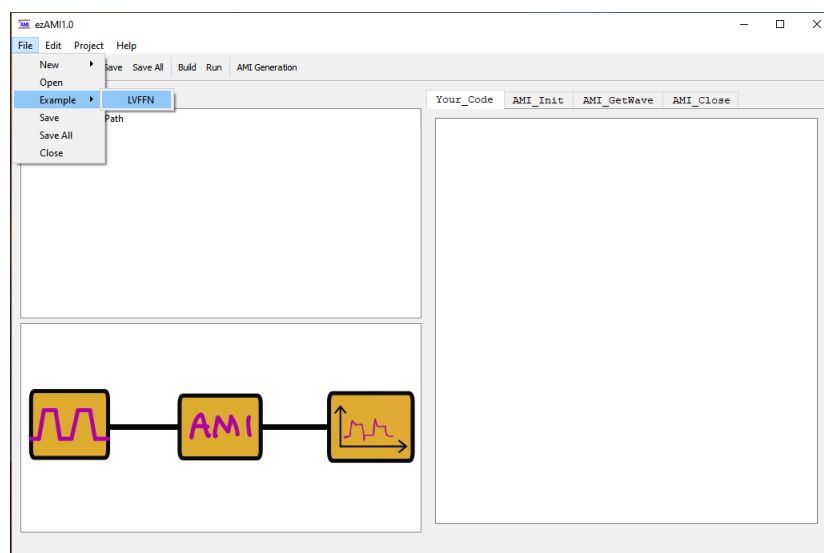


Figure B.3: Launch LVFFN example

A window pop up to let select or create a folder to store LVFFN project

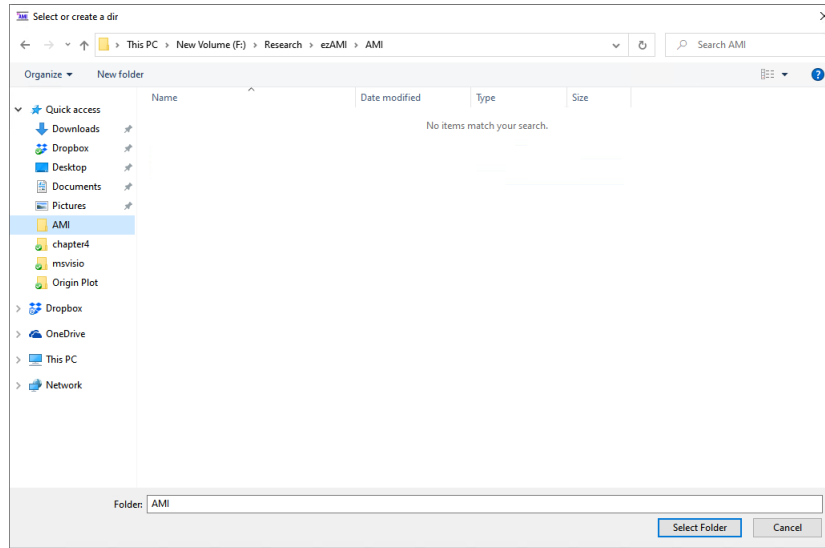


Figure B.4: Select folder to save LVFFN project

Once the LVFFN example project is loaded, the software main interface will be showing like in figure B.5

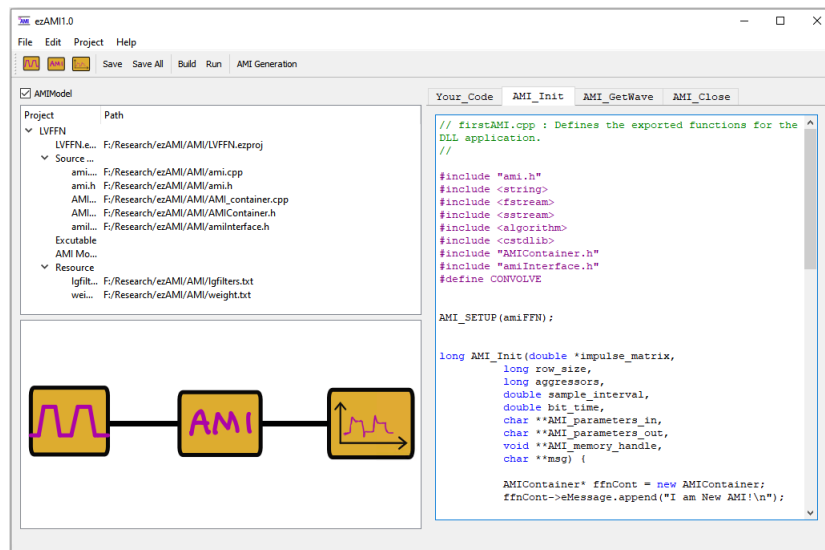


Figure B.5: Software main interface after LVFFN example is loaded



Click "Build" in menu bar or select from "Project→Build" to compile the code to generate the DLL file for simulation. If compilation is successful, it will look like in figure B.6).

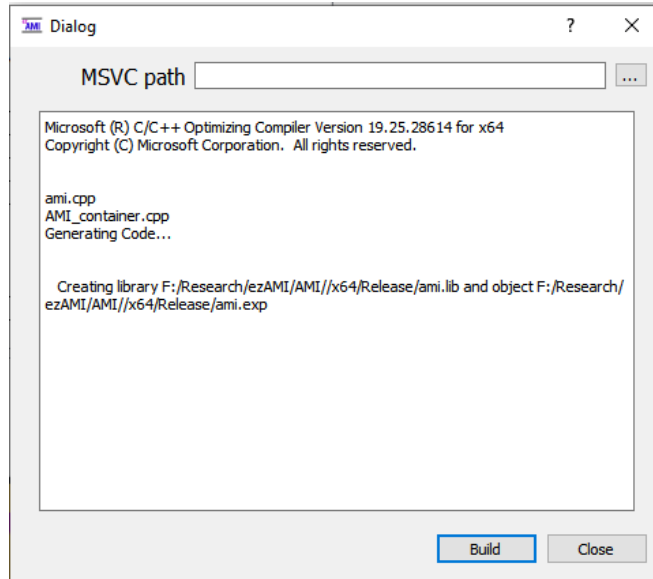


Figure B.6: Software main interface after LVFFN example is loaded

Double-click the "AMI" icon in the schematic window. A dialog window pop up to let you select the DLL file you just compiled (see figure B.7). The dll file will be in the project folder.

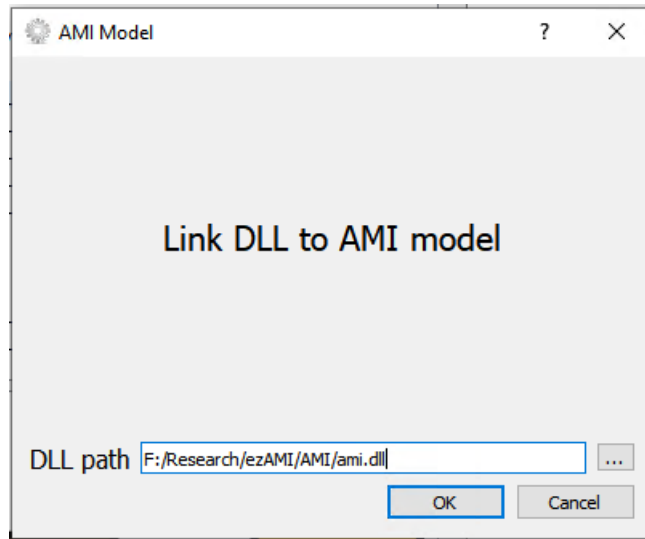


Figure B.7: Link dll file for simulation

Double-click the Excitation icon in schematic window. It is the first one from left. A dialog window is popped up to let you set the excitation setting. Choose PAM-4 and 100 bits as shown in figure B.8. Leave other settings as default, then click "Ok"

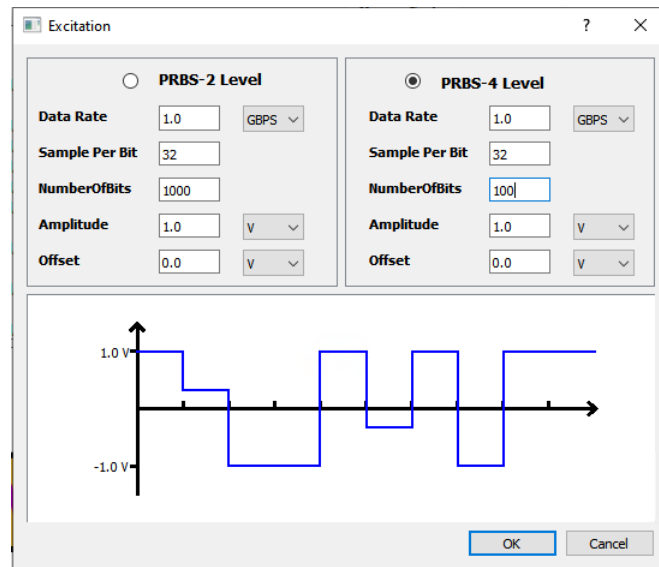


Figure B.8: Excitation generation

The next is to click "Run" in menu action bar or select "Project→Run" to launch simulation. If no errors happen, a window with plot for excitation wave and model output will be shown as figure B.9

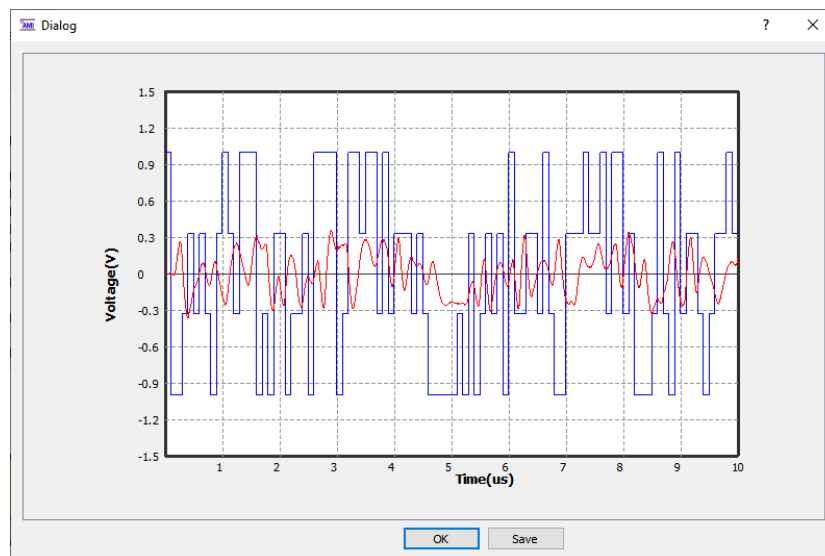


Figure B.9: Result display

The last step is to generate the AMI model for simulation in circuit simulator. Click "AMI Generation" action in menu action bar or select from "Project→AMI Generation". Select "X64". Leave ".ami Settings" as default since it is still being completed.

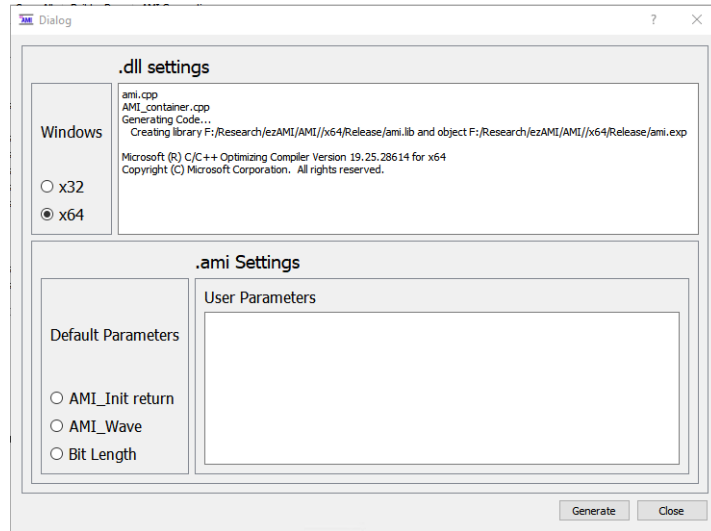


Figure B.10: Generate AMI model

If AMI model generation is successful, you will see both DLL file and .ami file appear in the project management region under the "AMI Model" node, as shown in figure B.11. Both files are the AMI model package you can use for simulation with commercial circuit simulator.

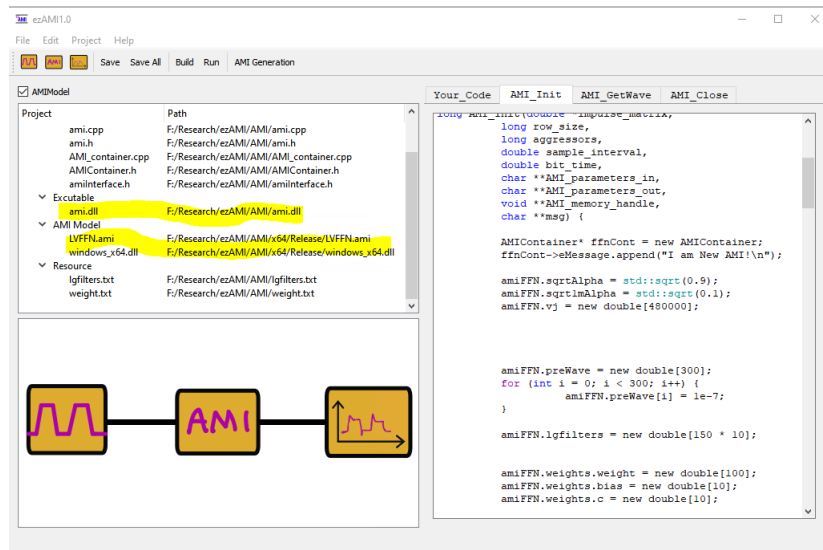


Figure B.11: Generated AMI model

## References

- [1] M. Steinberger and T. Westerhoff, “Demonstration of serdes modeling using the algorithmic model interface (ami) standard,” 2018, pp. 1–22.
- [2] T. Westerhoff, A. Hawes, M. Steinberger, K. Dramstad, W. Katz, and B. Katz, “Predicting ber with ibis-ami: experiences correlating serdes simulations and measurement,” 2010, pp. 1–24.
- [3] L. Liu, L. Li, Y. Huang, K. Cui, Q. Xiong, F. N. Hauske, C. Xie, and Y. Cai, “Intrachannel nonlinearity compensation inverse volterra series transfer function,” *Journal of Lightwave Technology*, vol. 30, no. 3, pp. 310–316, 2012.
- [4] D. Mirri, G. Iuculano, P. Traverso, G. Pasini, and F. Filicori, “Non-linear dynamic system modelling based on modified volterra series approaches,” *Measurement*, 2003.
- [5] T. L. Aliyev and E. P. Gatzke, “Prioritized constraint handling nmpc using volterra series models,” *Automation & Control Systems*, 2010.
- [6] M. Telescu, I. S. Stievano, F. G. Canavero, N. Tanguy, and N. Tanguy, “Neural networks and volterra series for modeling new wireless communication devices,” in *Signal Propagation on Interconnects (SPI)*, 2010, pp. 93–96.
- [7] A. K. Tangirala, *Principles of System Identification: Theory and Practice*. CRC Press, 2014.
- [8] B. Li, P. Franzon, Y. Choi, and C. Cheng, “Receiver behavior modeling based on system identification,” May 2018, pp. 299–301.
- [9] Y. Cho and C. Cheng, “High-speed link analysis with system identification approach,” May 2017, pp. 741–744.
- [10] Y. LeCun, Y. Bengio, and G. Hinton, “Deep learning,” *Nature*, vol. 521, pp. 436–444, May 2015, doi:10.1038/nature14539.
- [11] Z. Chen, M. Raginsky, and E. Rosenbaum, “Verilog-a compatible recurrent neural network model for transient circuit simulation,” pp. 1–3, 2017.

- [12] B. Li and P. Franzon, “Machine learning in physical design,” May 2017, pp. 147–150.
- [13] X. Wang, T. Nguyen, and J. Schutt-Aine, “Volterra kernel extraction through monomial power series feed forward neural network for behavior modeling of high speed i/o buffer,” in *EMC Sapporo & APEMC 2019*, 2019, pp. 1–4.
- [14] T. Nguyen, T. Lu, K. Wu, and J. Schutt-Aine, “Fast Transient Simulation of High-Speed Channels Using Recurrent Neural Network,” 2019. [Online]. Available: <https://arxiv.org/pdf/1902.02627.pdf>
- [15] T. Nguyen, X. Wang, X. Chen, and J. Schutt-Aine, “A deep learning approach for volterra kernel extraction for time domain simulation of weakly nonlinear circuits,” in *2019 IEEE 69th Electronic Components and Technology Conference (ECTC)*, 2019, pp. 1889–1896.
- [16] J. Dooley, B. O. Brien, and T. J. Brazil, “Behavioural modelling of rf power amplifiers using modified volterra series in the time domain,” in *High Frequency Postgraduate Student Colloquium*, Sep. 2004, pp. 169–174.
- [17] C. Crespo-Cadenas, J. Reina-Tosina, and M. J. Madero-Ayora, “Volterra behavioral model for wideband rf amplifiers,” in *IEEE Transactions on Microwave Theory and Techniques*, vol. 5, no. 3, 2007, pp. 449–457.
- [18] P. Wambacq, G. Gielen, P. Kinget, and W. Sansena, “High-frequency distortion analysis of analog integrated circuits,” in *IEEE Trans. Circuits Syst. II: Analog Digit. Signal Process*, vol. 46, no. 3, 1999, pp. 335–345.
- [19] R. Kumar, A. Banerjee, and B. Vemuri, “Trainable convolution filters and their application to face recognition,” in *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 34, no. 7, 2012, pp. 1423–1436.
- [20] Y. Hu and K. Muryam, “A modified-volterra-seriestechnique for improving the accuracy of quasistatic harmonic balance analysis in coupled device and circuit simulation,” in *IEEE 2004 Custom Integrated Circuits Conference*, Apr. 2004, pp. 125–128.
- [21] S. Im, “Adaptive equalization of nonlinear digital satellite channels using a frequency-domain volterra filter,” in *IEEE Military Communications Conference*, Apr. 1996, pp. 843–848.
- [22] L. Liu, L. Li, Y. Huang, K. Cui, Q. Xiong, F. N. Hauske, C. Xie, and Y. Cai, “Intrachannel nonlinearity compensation inverse volterra series transfer function,” *Journal of Lightwave Technology*, vol. 30, no. 3, pp. 310–316, 2012.

- [23] N. Wiener, *Nonlinear problems in random theory*. MIT Press, Cambridge, MA, 1958.
- [24] M. Schetzen, *The Volterra and Wiener Theories of Nonlinear Systems*. Melbourne, FL, USA: Krieger Publishing Co., Inc., 2006.
- [25] X. Jing and Z. Lang, *The Generalized Frequency Response Functions and Output Spectrum of Nonlinear Systems*, 12 2015, vol. 119, ch. 2, pp. 9–30.
- [26] L. Li and S. Billings, “Estimation of generalized frequency response functions for quadratically and cubically nonlinear systems,” *Journal of Sound and Vibration*, vol. 330, no. 3, pp. 461 – 470, 2011.
- [27] J. Yang and S. X.-D. Tan, “Nonlinear transient and distortion analysis via frequency domain volterra series,” *Circuits, Systems and Signal Processing*, vol. 25, no. 3, pp. 295–314, Jun 2006. [Online]. Available: <https://doi.org/10.1007/s00034-004-0819-3>
- [28] W. J. Rugh, *Nonlinear system theory: the Volterra/Wiener approach*. Johns Hopkins University Press, 1981.
- [29] J. Wray and G. G. R. Green, “Calculation of the volterra kernels of nonlinear dynamic systems using an artificial neural network,” *Biological Cybernetics*, vol. 71, no. 3, pp. 187–195, July 1994.
- [30] J. Mistic, V. Markovic, and Z. Marinkovic, “Volterra kernels extraction from neural networks for amplifier behavioral modeling,” in *International Symposium on Telecommunications*, Oct. 2014.
- [31] G. Stegmayer, “Volterra series and neural networks to model an electronic device nonlinear behavior,” in *IEEE International Joint Conference on Neural Networks*, July 2004, pp. 2907–2910.
- [32] G. Stegmayer, “Neural networks and volterra series for modeling new wireless communication devices,” in *International Joint Conference on Neural Networks*, Aug. 2007.
- [33] G. Mitsis, M. Poulin, P. Robbins, and V. Marmarelis, “Nonlinear modeling of the dynamic effects of arterial pressure and co2 variations on cerebral blood flow in healthy humans,” *IEEE TRANSACTIONS ON BIOMEDICAL ENGINEERING*, vol. 51, no. 11, pp. 1932–1943, 2004.
- [34] H. Ogura, “Estimation of wiener kernels of a nonlinear system and a fast algorithm using digital laguerre filters,” in *15th NIBB Conference*, 1985, pp. 14–62.

- [35] K. Geng and V. Z. Marmarelis, “Methodology of recurrent laguerre-volterra network for modeling nonlinear dynamic systems,” *IEEE TRANSACTIONS ON NEURAL NETWORKS AND LEARNING SYSTEMS*, vol. 28, pp. 2196 – 2208, 2016.
- [36] “IBIS (I/O Buffer Information Specification).” [Online]. Available: <https://ibis.org/about/>
- [37] J. Dooley, B. O. Brien, and T. J. Brazil, “Behavioural modelling of rf power amplifiers using modified volterra series in the time domain,” in *High Frequency Postgraduate Student Colloquium*, Sep. 2004, pp. 169–174.
- [38] C. Crespo-Cadenas, J. Reina-Tosina, and M. J. Madero-Ayora, “Volterra behavioral model for wideband rf amplifiers,” in *IEEE Transactions on Microwave Theory and Techniques*, vol. 5, no. 3, 2007, pp. 449–457.
- [39] P. Wambacq, G. Gielen, P. Kinget, and W. Sansena, “High-frequency distortion analysis of analog integrated circuits,” in *IEEE Trans. Circuits Syst. II: Analog Digit. Signal Process*, vol. 46, no. 3, 1999, pp. 335–345.
- [40] R. Kumar, A. Banerjee, and B. Vemuri, “Trainable convolution filters and their application to face recognition,” in *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 34, no. 7, 2012, pp. 1423–1436.
- [41] Y. Hu and K. Muryam, “A modified-volterra-seriestechnique for improving the accuracy of quasistatic harmonic balance analysis in coupled device and circuit simulation,” in *IEEE 2004 CUSTOM INTEGRATED CIRCUITS CONFERENCE*, Apr. 2004, pp. 125–128.
- [42] S. Im, “Adaptive equalization of nonlinear digital satellite channels using a frequency-domain volterra filter,” in *IEEE Military Communications Conference*, Apr. 1996, pp. 843–848.
- [43] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” 2015, <https://arxiv.org/abs/1412.6980>.
- [44] “Cisco Predicts More IP Traffic in the Next Five Years Than in the History of the Internet.” [Online]. Available: <https://newsroom.cisco.com/press-release-content?type=webcontent\&articleId=1955935>
- [45] J. Lee, P.-C. Chiang, P.-J. Peng, L.-Y. Chen, and C.-C. Weng, “Design of 56 gb/s nrz and pam4 serdes transceivers in cmos technologies,” *IEEE Journal of Solid-State Circuits*, vol. 50, no. 9, pp. 2061–2073, September 2015.

- [46] “Ieee P 802.3bs Baseline Summary,” 2016. [Online]. Available: <http://www.ieee802.org/3/bs/>
- [47] “PAM-4 Design Challenges and the Implications on Test .” [Online]. Available: <http://literature.cdn.keysight.com/litweb/pdf/5992-0527EN.pdf>
- [48] O. Elhadidy, A. Roshan-Zamir, H.-W. Yang, , and S. Palermo, “A 32 gb/s 0.55 mw/gbps pam4 1-fir 2-iir tap dfe receiver in 65-nm cmos,” in *2015 Symposium on VLSI Circuits Digest of Technical Papers*, 2015, pp. C224–C225.
- [49] M. Bassi, F. Radice, M. Bruccoleri, S. Erba, and A. Mazzanti, “A 45gb/s pam-4 transmitter delivering 1.3vppd output swing with 1v supply in 28nm cmos fdsoi,” in *2016 IEEE International Solid-State Circuits Conference*, 2016, pp. 66–67.
- [50] A. Roshan-Zamir, T. Iwai, Y.-H. Fan, A. Kumar, H.-W. Yang, L. Sledjeski, J. Hamilton, S. Chandramouli, A. Aude, and S. Palermo, “A 56 gb/s pam4 receiver with low-overhead threshold and edge-based dfe fir and iir-tap adaptation in 65nm cmos,” in *Custom Integrated Circuits Conference*, 2018, pp. 1–4.
- [51] I. S. Stievano, C. Siviero, F. G. Canavero, and I. A. Maio, “Behavioral modeling of digital devices via composite local linear state-space relations,” *IEEE TRANSACTIONS ON INSTRUMENTATION AND MEASUREMENT*, vol. 57, no. 8, pp. 1757–1765, 2008.
- [52] I. S. Stievano, I. A. Maio, and F. G. Canavero, “Behavioral models of i/o ports from measured transient waveforms,” *IEEE TRANSACTIONS ON INSTRUMENTATION AND MEASUREMENT*, vol. 51, no. 6, pp. 1266–1270, 2002.
- [53] I. S. Stievano, I. A. Maio, and F. G. Canavero, “M $\pi$ log macromodeling via parametric identification of logic gates,” *IEEE TRANSACTIONS ON ADVANCED PACKAGING*, vol. 27, pp. 15–23, 2004.
- [54] R. Trinchero, P. Manfredi, I. S. Stievano, and F. G. Canavero, “Machine learning for the performance assessment of high-speed links,” *IEEE Transactions on Electromagnetic Compatibility*, vol. 60, pp. 1627–1634, 2018.
- [55] R. J. Campello, G. Favier, and W. C. do Amaral, “Optimal expansions of discrete-time volterra models using laguerre functions,” *Automatica*, vol. 40, pp. 815–822, 2004.



- [56] Y. Kang, J. Escudero, D. Shin, E. Ifeachor, and V. Marmarelis, “Principal dynamic mode analysis of eeg data for assisting the diagnosis of alzheimer disease,” *Medical Imaging and Diagnostic Radiology*, vol. 3, pp. 1–10, 2015.
- [57] “Automated AMI Model Generation & Validation.” [Online]. Available: <https://ibis.org/summits/jun10/pino.pdf>
- [58] K. Geng, “Nonlinear dynamic modeling of biomedical systems with laguerre-volterra network,” Ph.D. dissertation, UNIVERSITY OF SOUTHERN CALIFORNIA, 789 East Eisenhower Parkway, P.O. Box 1346, Ann Arbor, MI 48106 - 1346, May 2017.