COMPREHENSIVE ALGORITHM FOR EYE DIAGRAM CROSSING POINT DETECTION AND EYE PARAMETER EXTRACTION IN TRANSIENT SIMULATION

BY

HALIM PARK

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Electrical and Computer Engineering
in the Graduate College of the
University of Illinois Urbana-Champaign, 2024

Urbana, Illinois

Adviser:

Professor José E. Schutt-Ainé

ABSTRACT

This thesis provides a comprehensive and robust approach to decipher an algorithm that is only partially accurate in and not fully disclosed by the existing industry tools - crossing point detection in an eye diagram, a fundamental starting point of eye parameter extraction. Edge cases where the existing industry tools fail to deliver accurate analysis are identified, and this paper's algorithm is introduced and proved with various cases. Finally, eye parameter extraction is followed and compared with existing tools for verification.

To Yunseo Gongju and my family, for their love, guidance, and support.

ACKNOWLEDGMENTS

In 2006, my family moved from our home country, South Korea, to Beijing, China. Having lived 10 years in Beijing, I slowly became used to overcoming the challenges of facing a new environment. When I first arrived at UIUC in 2016, it was also my first time ever in the United States. It was then that I started my journey in ECE, and a 2-year gap for my military service between 2018 and 2020 was long enough to adjust myself into pursuing a career in wireless communication. This passion extended me into graduate school, which I found arduous but absolutely rewarding. This thesis is a significant milestone in my academic journey at the University of Illinois.

First of all, I would like to express my gratitude to my advisor, Professor José E. Schutt-Ainé. He welcomed me as an incoming graduate student without second thought, providing necessary resources and connecting me to other research group members for inspirations. He has always couraged me during research meetings, and the courses he has lectured such as ECE453, ECE451, and ECE546 better prepared me as a RF engineer.

I am grateful to Dr. Thong Nguyen for his passion in research. Upon joining the group, he helped me understand what each member has been working on, as well as the why of their projects. He is excellent in understanding of various topics, and has provided me useful guidance in research. Whenever I was stuck, he provided me helpful examples that either validate or invalidate my proposed approach, inspiring me to attempt different approaches.

My appreciation goes to Dr. Bobi Shi and PhD candicate Yi Zhou. My research is heavily related to theirs, and they helped me understand working with eye diagrams and how the simulations are run. I would often approach Bobi with new ideas, and he had always been patient to understand them and provide useful information to gain better understanding. Bobi helped me narrow down my topic for the thesis, and helped me understand how solving this issue can help with his research as well. Yi was alawys in the

office, helping me clear out some of my confusion throughout my research.

I express my appreciation to rest of the EM Lab members in the office, including Dr. Nancy Zhao, PhD candicate Juhitha Konduru, Haofeng Sun, Tahsin Shameem, Zohreh Salehi, and Rui Gong. My conversations with them were enlightening and couraging, helping me go though challenging times during my research.

Furthermore, I would like to thank Yunseo Kim who patiently walked together during my journey at UIUC, providing emotional support and courage to never be let down.

Last but not least, I thank my family for supporting me throughout my 8 years of study at UIUC. They have always supporting my decisions and wishing the best of me, even though I have been on the other side of the world. My 8.5-year long chapter at UIUC ends soon, and I am confident that they will always show their love and support wherever I am.

TABLE OF CONTENTS

LIST OF ABBREVIATIONS

SPICE	Simulation Program with Integrated Circuit Emphasis
LIM	Latency Insertion Method
ISI	Inter Symbol Interference
PAM2	Pulse Amplitude Modulation 2-Level
PAM4	Pulse Amplitude Modulation 2-Level
PRBS	Pseudo Random Binary Sequence
UI	Unit Interval
NRZ	Non Return to Zero
LFSR	Linear Feedback Shift Register
EDA	Electronic Design Automation
XOR	Exclusive Or
LSB	Least Significant Bit
ADS	Advanced Design System
PDF	Probability Distribution Function
BER	Bit Error Rate

CHAPTER 1

INTRODUCTION

Since the invention of computers, the need for the speed of data transfer has increased exponentially. Such need has been further driven by increasing number of electronic devices around us, from computers and smartphones to smartwatches and vehicles. Enhanced functionality and higher definition for these devices urged engineers to push the limit for data rates. As engineers started exhibiting data rates in the Gigabit level, high speed serial link and the study of signal integrity started to play more important roles. It is estimated that the data rate will be pushed to the high-Gigahertz or Terabit level, due to the high amount of data calculations required in AI models. Characteristics of an electric channel not only introduces losses and time de-

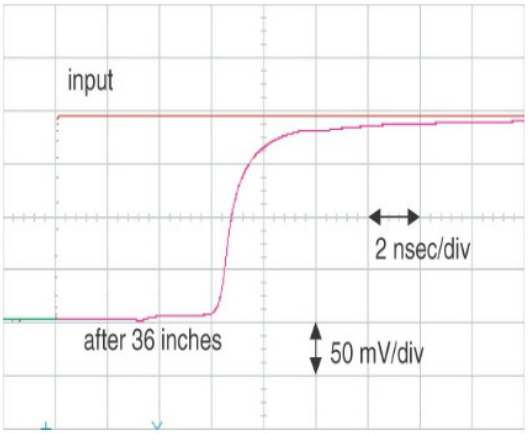


Figure 1.1: Measured input (red) and output (purple) signals through a 36-inch-long, 50Ω backplane line in FR4.

lay, but also exhibits non-ideal behaviors such as noise and jitter. Figure 1.1 shows how the output looks like in reality even if the input is a step function. Furthermore, due to the need of placing channels at very small distances, signals from one channel may leak into the other channel, a phenomenon known as crosstalk. There is a plethora of tools and techniques that are used

to assess the characteristics and the performance of digital communication systems. One of the most widely used tool is the "Eye Diagram".

The eye diagram is a widely used visualization tool that can help assess the quality of the signal transitions in electrical communication signals and channel compliance. Its advantage lies in the fact that it not only visually illustrates the quality of bit transitions, hence providing an intuitive view to engineers, but also outputs quantitative measures (eye parameters) for engineers to directly compare and quantify the degree of non-idealities. Imperfections or the characteristics in the channel cause the received signal to be distorted, even if the transmitter transmits a perfect square wave. The degree of such distortions – intersymbol interference (ISI), noise, jitter, etc. – can be assessed using an eye diagram. Eye diagram functionality is available in oscilloscopes, as well as software EDA simulation tools such as ADS, Ansys, and Virtuoso.

Even though the eye diagram is simple to construct, extracting the eye parameters involves an extra step – locating the crossing point. As introduced in Chapter 2, there exist mathematical definitions for eye parameters, most of which requires one to locate the crossing point at the 25% of the 2UI window. By doing so, the most optimum sampling time can be decided to be the midpoint of the two eye crossing point since it is the most "open" part of the eye. Reflecting such consideration, eye diagrams that we see are set with some intentional timing delay. In order to do so, we need to know exactly at which time instant should the slicing happen. Consider different slicing timings for a series of time domain voltage waveform:

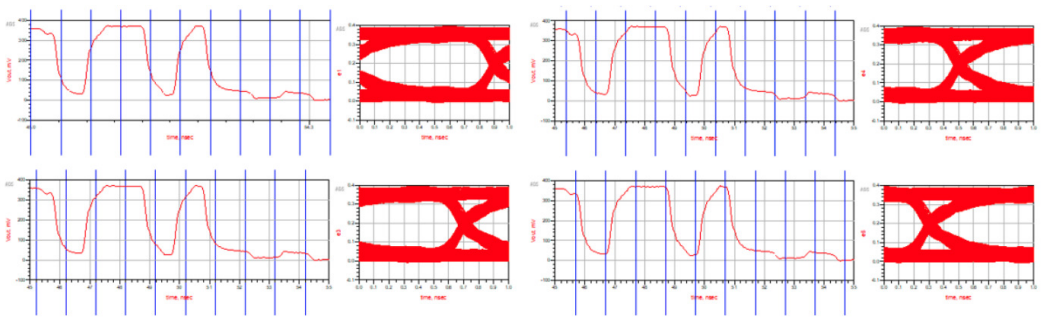


Figure 1.2: Different eye diagrams (column 2 and 4) formed by choosing different slicing timings for an identical waveform.

Figure 1.2 shows different eye patterns with different slicing timings, while the time domain voltage waveform is identical. We see circular-shifted ver-

sions of each other, depending on the time instant chosen for slicing. Since this is within a 1-UI window, we want to place the crossing point at the center, which would be equivalent to 25% point of the 2-UI window. By locating where the eye crossing point is, we can effectively decide where the slicing instant should be. The top right of Figure 1.2 would be our desired output since the crossing point is located at 50% point of the 1-UI window (same as 25% of 2-UI) in the time axis.

In real implementation, clock data recovery (CDR) is often used to preserve the sampling timing on the receiver side. However, engineers run circuit simulations before the implementation using EDA tools and evaluate the signal integrity at different nodes. For example, ADS has as the "Eye Probe" functionality where a user can probe to different nodes of the circuit to assess signal integrity using eye diagrams. Electrical delay or any form of distortion can be different, and timing recovery methods such as CDR would not be applicable. Therefore, simulation tools require a separate algorithm to generate the eye diagram with the ability to locate the crossing point in order to quantify the degree of signal integrity.

We as humans can simply point to where an approximate crossing point would be, but it requires a much more complex method to find the exact crossing point with various forms of eye shapes. Furthermore, a mathematical definition of a crossing point needs to be established.

Aforementioned existing tools do have the ability of doing so, but the exact method remains proprietary, with some inaccuracies in some edge cases. This thesis aims to illustrate a robust algorithm for eye crossing point detection, a required step for computing eye parameters. This paper involves not only a verification of the working cases by validating and comparing eye parameters extracted by existing tools, but also provide a robust solution that existing tools fail to compute.

Run time consideration is another important aspect. The goal is to keep the crossing point detection algorithm comparably faster than transient simulation run time, which scales with the size of the netlist. While a commonly used simulator, known as simulation program with integrated circuit emphasis (SPICE), has a non-linear computational complexity with respect to circuit size due to its nature of methodology, there also has been newly proposed simulation method such as latency insertion method (LIM) [1] that aims to linearize the computational complexity with larger scale circuits. Therefore,

it is projected that the transient simulation run time will become shorter while achieving same level of accuracy, also alluding to the importance of optimizing the run time for this algorithm.

1.1 Outline

The thesis is organized as follows:

Chapter 2 of this thesis introduces necessary background, including the construction of the eye diagram, mathematical definitions of eye parameters, and non-linearities.

Chapter 3 visits some of the existing patents and technologies to finding the eye crossing point, and some drawbacks to these existing approach. Also, some of the previous attempts of my own made for the solution are presented, with counter examples illustrating the lack of comprehensiveness.

Chapter 4 dives into the specific steps for the final algorithm to find the eye crossing point.

Chapter 5 presents the result of the algorithm using various test cases.

Chapter 6 discusses the run time optimization of the algorithm.

Chapter 7 concludes the thesis.

CHAPTER 2

BACKGROUND

2.1 Construction of the eye diagram

The eye diagram is constructed by slicing the time domain signal, usually the output of a channel, in 2 unit intervals (UI) and overlaying the rising and falling edges of the transitions. The time domain signal is sliced by a 2-UI window that shifts by 1-UI, and the sliced portions get superposed into a single 2-UI window. The resulting graph resembles a human eye, hence the name "eye diagram". This 2-UI window represents two transitions, hence representing 3 bits, from 000, 001, 010... to 111.

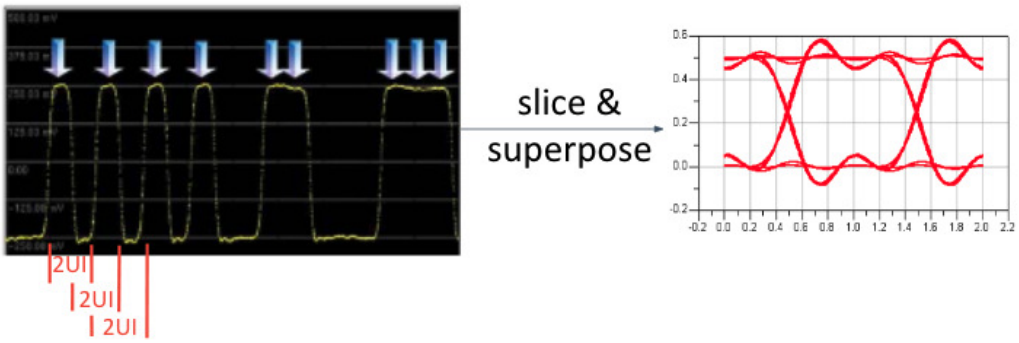


Figure 2.1: Visual illustration of how time domain signal is sliced to construct the eye diagram.

Figure 2.1 uses a Pulse Amplitude Modulation 2-level (PAM-2, also known as non-return-to-zero (NRZ)) signal modulation, where the low voltage represents a 0, and the high represents a 1. There also exists different signal modulation schemes for faster data transfer, among with the most commonly used is PAM-4. PAM-4 uses four voltage levels to represents two bits simultaneously. The data bits are logically represented in combinations of 00, 01, 10, or 11. It allows twice of the data speed compared to PAM-2, at the cost

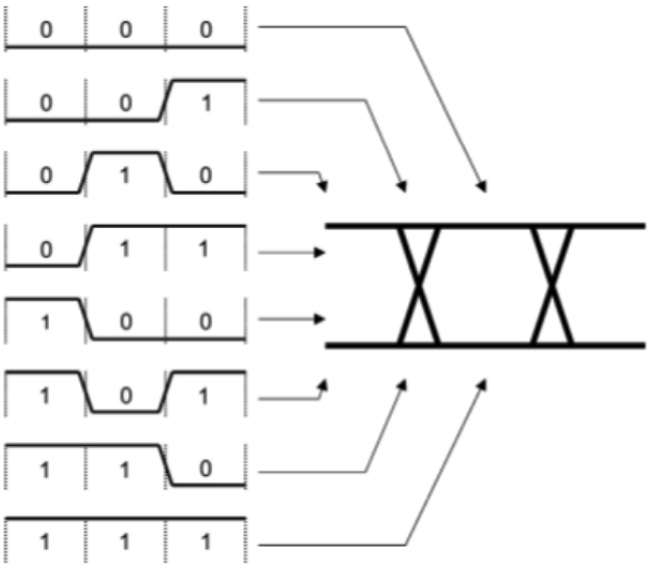


Figure 2.2: Visual illustration of how 3 bit transitions are combined to construct the eye diagram.

of higher power consumption caused by equalization to minimize BER and high susceptibility to noise due to smaller voltage difference between adjacent logic voltage levels. PAM-4 is widely used in optical modules such as 50G and 500G to empower various forms of carrier networks.

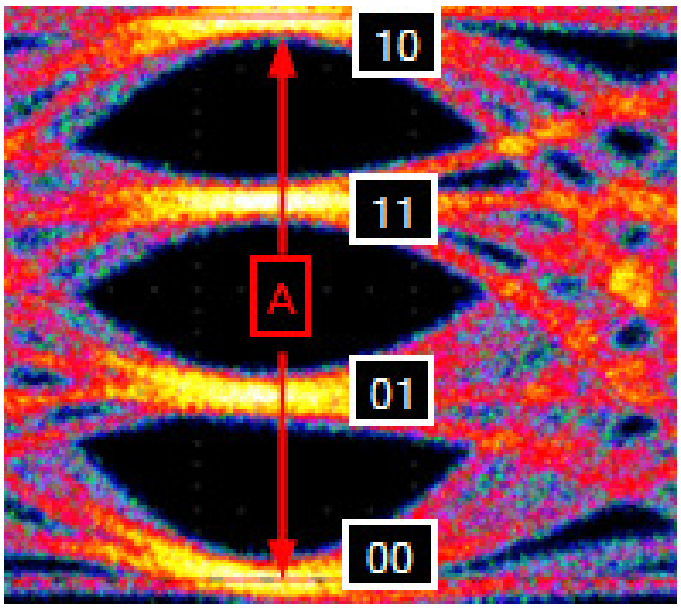


Figure 2.3: A PAM-4 eye diagram, with 4 distinct voltage levels representing 2-bit combinations – 00, 01, 11, 10.

The waveform that we wish to convert into eye diagrams is generated using transient simulation with simulation program with integrated circuit emphasis (SPICE), which takes a netlist that includes information about the circuit components and their connections and solves for the desired output. Slicing occurs after SPICE has solved for the output in order to generate the eye diagram.

Another way that we can generate an eye diagram is by knowing the model of the channel, for example, through a known S-parameter. If this is known, the output generation is simplified into taking the convolution between the inverse FFT of the channel channel and the input bit stream waveform using pseudo-random binary sequence (PRBS). The slicing methodology is the same as above. This is illustrated in Figure 2.4 [2].

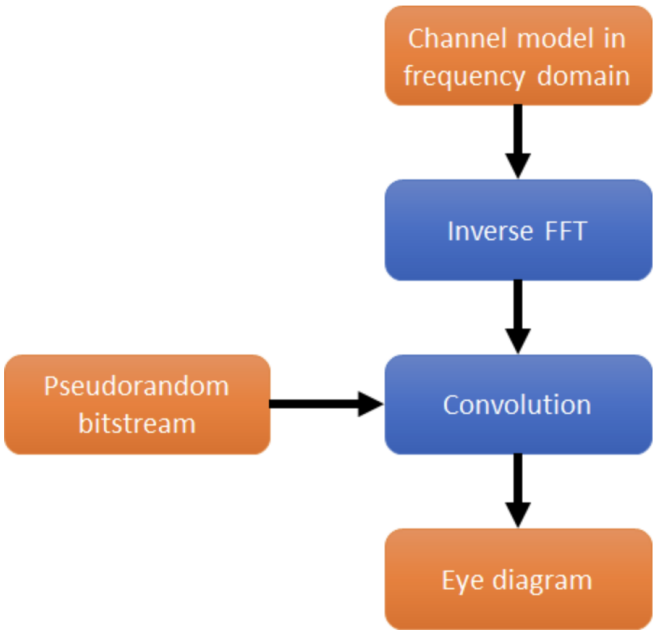


Figure 2.4: Eye diagram simulation process when the channel model is known.

2.2 Heatmap-based Eye Density Plot

While we generate the eye diagram through overlaying the sliced waveform, engineers are often interested in a heatmap plot showing the density. While it is almost impossible to have two sliced waveform that are perfectly identi-

cal, for instance, up to 10th decimal, EDA tools employ histogram method to bin certain nearby waveform together. Then, bins with higher counts are displayed with higher temperature color to illustrate the density, as illustrated on the right of Figure 2.5.

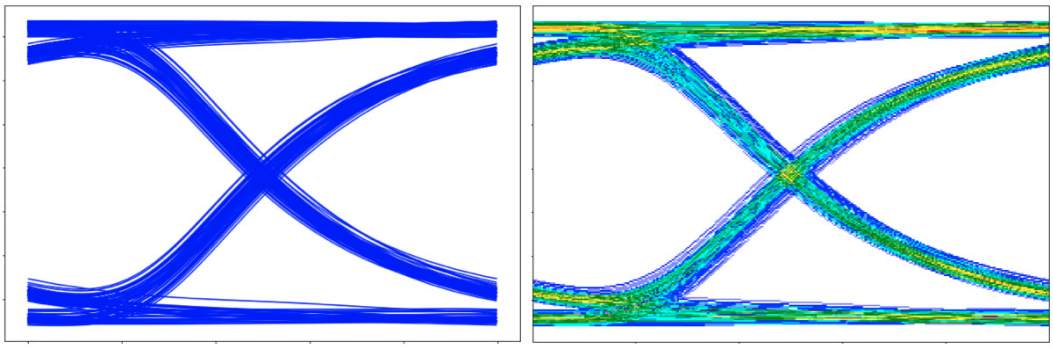


Figure 2.5: Eye diagram (left) and its density heatmap plot (right).

The heatmap density plot on the right better visually illustrates the distributions of the waveform. For instance, the jitter probability distribution at the crossing point can be easily visualized by looking at the density heatmap plot on the right of Figure 2.5, while the left cannot. EDA tools, except Cadence Virtuoso, have such implementation. It saves computational time by grouping waveform into bins before plotting. This can be easily achieved by defining a set of colors with respect to bin counts in the $plt.hist2d$ function, and is implemented into the eye plotting solution provided in this paper as well.

2.3 Crossing Point Definition

Definition of a crossing point must be established in order to define our scope of interest. While the output, the crossing point, is a single pair of coordinates containing the timing and voltage value, there are multiple line segments that contribute to the crossing point. It is obvious that the rising and falling edges contribute to the crossing point, but N rising edges and M falling edges would create NM crossing points, while we need one point being the output. Furthermore, some edges are close enough such that they are counted in identical bins in the histogram. We hence need to account for such weight when calculating a suitable crossing point. Therefore, we can

define the crossing point as **the arithmetic mean of all the NM crossing points generated by N rising edges and M falling edges**.

For computational simplicity, we do not necessary need to calculate every NM crossing point, but rather calculate a single "average contour" for each of the N rising edges and M falling edges, then calculate the crossing point between the two average contours. For example, consider a set of linear lines, with N rising lines defined as $y = ax + b_n, n \in \mathbb{Z}$ and M falling lines defined with $y = c_n - dx, n \in \mathbb{Z}$. We would do crossing point-by-crossing point calculation by solving:

$$ax + b_n = c_m - dx \tag{2.1}$$

The resulting crossing point is calculated as:

$$x_{nm} = \frac{c_m - b_n}{a + d},\ y_{nm} = a\frac{c_m - b_n}{a + d} + b_n \tag{2.2}$$

The arithmetic mean of these NM crossing points can be calculated as:

$$x_{xing} = (\frac{1}{n})[x_{11} + x_{22} + x_{33}...] = (\frac{1}{n})\frac{c_1 - b_1 + c_2 + b_2 + c_3 + b_3 + ...}{a + d} \tag{2.3}$$

$$y_{xing} = (\frac{1}{n})[y_{11} + y_{22} + y_{33}...] = (\frac{a}{n})[\frac{c_1 - b_1 + c_2 - b_2 + ...}{a + d}] + (\frac{b_1 + b_2 + ...}{n}) \tag{2.4}$$

On the other hand, if we calculate the average contour first, we would get:

$$avgcontour_{rising} = ax + \frac{b_1 + b_2 + ...}{n},\ avgcontour_{falling} = \frac{c_1 + c_2 + ...}{n} - dx \tag{2.5}$$

Equating the two linear equations in 2.5, we would get

$$x_{xing} = \frac{(c_1 + c_2 + ...) - (b_1 + b_2 + ...)}{n(a + d)} \tag{2.6}$$

$$y_{xing} = a\frac{(c_1 + c_2 + ...) - (b_1 + b_2 + ...)}{n(a + d)} + \frac{b_1 + b_2 + ...}{n} \tag{2.7}$$

We see that Equations 2.3, 2.6 and 2.4, 2.7 are equal. Figure 2.6 illustrates a rather more general case. Top row figures are the heatmap density plots of the bottom row figures. The middle column figure has uniform distribution of lines, while the other figures have non-uniform distributions – meaning that

some lines are duplicates and coincide with one another, therefore shown in deeper colors (red – falling edges, blue – rising edges). While the black dots show all of the crossing points between every single rising edges, the red dot is the arithmetic mean of those crossing points. Furthermore, the dashed black lines are the average contours for each of the rising and falling edges. We can see that the intersection of the two average contours (black dashed line) occurs exactly at the red dot.

We can see that while the lines occupy the same Euclidian space, different line distributions result in different arithmetic mean crossing point. Therefore, considering the weights created by coinciding curves is important in defining the crossing point.

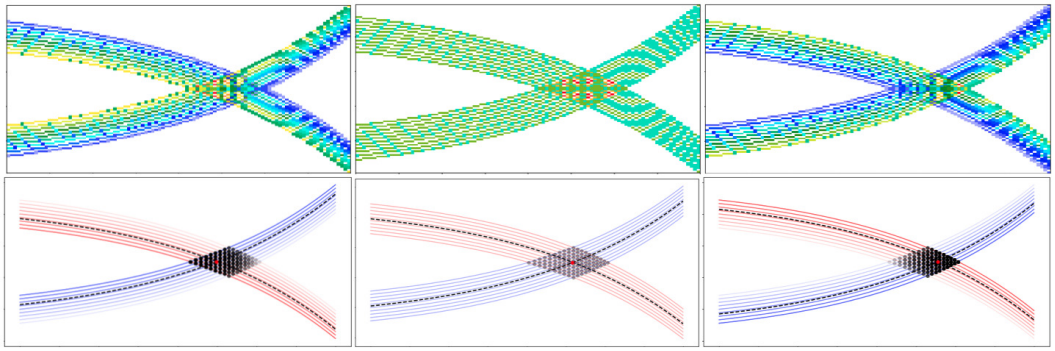


Figure 2.6: Crossing point (red dot) with different line distributions in the same Euclidian space, with their heatmap density plot on the top row.

This definition can be also extended into a multi-crossing point situation. Figure 2.7 illustrates such case, having two distinct groups of rising edges and two groups of falling edges, thereby creating 4 noticable crossing points. In this case, the final crossing point that we want to calculate is identical as the previous approach. The arithmetic mean of these 4 crossing points is identical to a single crossing point generated by average contour of rising and falling edges.

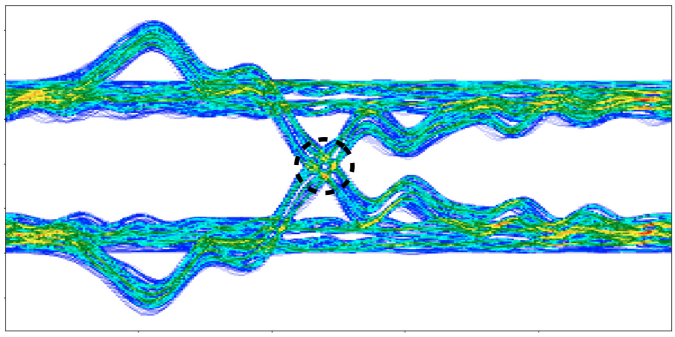


Figure 2.7: Eye diagram with 4 crossing points in the black dashed circle.

2.4 Channel Distortions

This section explains some of the terminology used to describe both qualitative and quantitative degree of distortions of a channel, which deforms the shape of the eye diagram, hence posing a challenge in finding the crossing point.

2.4.1 Inter-Symbol Interference (ISI)

Inter-Symbol Interference (ISI) is a generic term describing the spread of energy of a symbol over into adjacent symbols, causing interference [3]. Consider Figure 2.8 showing input and output waveform of a transmission line. Cursors indicate the best positions to sample the waveform, and they are spaced 1-UI between adjacent ones since the input bit stream would be separated by 1-UI in the time domain as well. For an ideal channel, we would

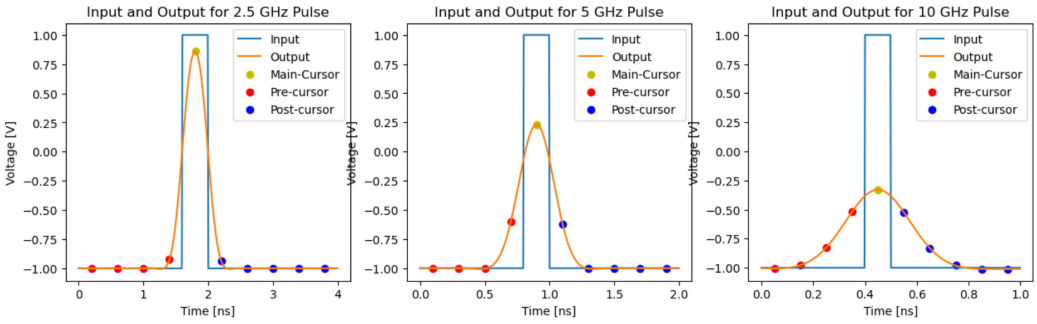


Figure 2.8: Input and output waveform of a transmission line at different bit rates, with cursors showing ISI

want all of the pre and post cursors to be 0. However, in reality the output

waveform is affected by many factors such as delay, jitter, noise, impedance mismatch, etc, thereby causing nonzero pre and post cursor values. Though only a single pulse is illustrated in Figure 2.8, the output waveform looks worse when multiple bits (1s and 0s) are transmitted through the channel, as the pre and post cursors constructively or destructively interfere with adjacent main cursors. This can lead to inaccurate determination of the received bit, leading to a high BER. In general, pre and post cursors have higher values with higher bit rates as shown in the Figure, which can worsen the eye diagram from a signal integrity perspective.

2.4.2 Jitter

While we commonly use noise to describe uncertainties in voltage, jitter is used to describe uncertainties in the time domain. It is commonly observed in digital transmission systems and is often the most important aspect to consider in signal integrity. Similar to ISI, system loss, crosstalk, interference, reflections can be sources of jitter. Jitter can be classified into two main types – Random Jitter (RJ) and Deterministic Jitter (DJ). As the name implies, DJ is further divided into many categories as illustrated in Figure 2.9 [4].

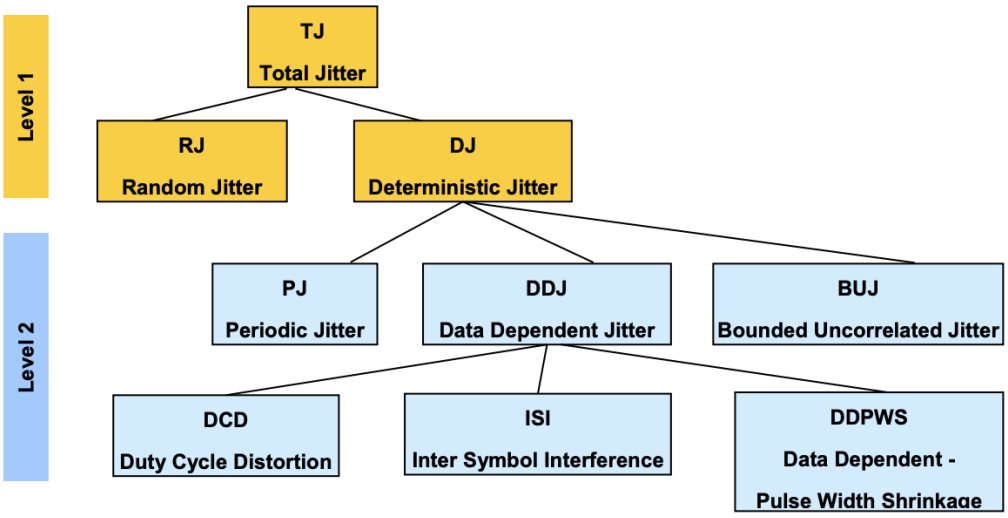


Figure 2.9: Jitter classification and analysis level

For the sake of this paper, we focus on the types of jitter that influences eye shapes near the crossing point – Random Jitter (RJ) and Random Jitter (PJ).

Random jitter is a type of jitter that can be modelled by a Gaussian distribution with the following probability density function:

$$PDF_{RJ}(\Delta t) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(\Delta t-\mu)^2}{2\sigma^2}} \tag{2.8}$$

where x is the independent variable, corresponding to the amount of time deviation, σ is the RMS value, and μ is the mean of the distribution.

Periodic jitter is the jitter that occurs at a fixed frequency. There are different models for PJ. For instance, the PDF of a single sinusoidal PJ can be modelled as:

$$f_{PJ}(\Delta t) = \frac{1}{\pi\sqrt{1-(\Delta t/A)^2}}, -A \leq \Delta t \leq A \tag{2.9}$$

When we combine all types of jitter, each jitter components are added together. In the statistical domain, the total jitter PDF would be equal to the convolution of each jitter component PDFs. For instance, if jitter components contain only Gaussian RJ and sinusoidal PJ, the total jitter PDF would be:

$$PDF_{TJ}(\Delta t) = [\frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(\Delta t-\mu)^2}{2\sigma^2}}] * [\frac{1}{\pi\sqrt{1-(\Delta t/A)^2}}], -A \leq \Delta t \leq A \tag{2.10}$$

Equation 2.4.2 is visualized in Figure 2.10. In eye diagrams, dual-modal

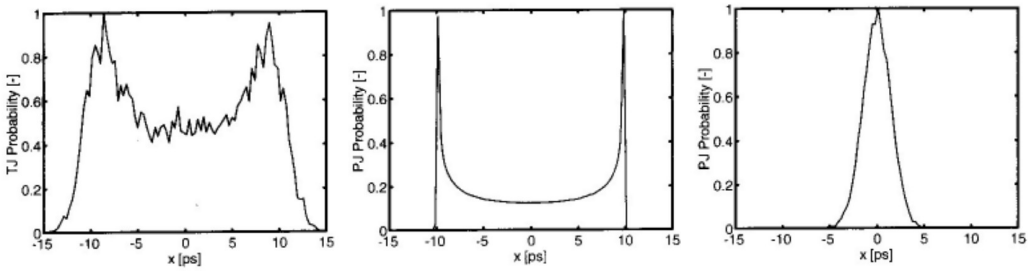


Figure 2.10: PDF_{TJ} (left), PDF_{PJ} (middle), PDF_{RJ} (right)

PDF_{TJ} corresponds to dual-modal rising and falling edges, which distinguishes the crossing point calculation compared to a regular eye with a single peak jitter PDF. The model of PJ can be modified such that multi-modal eye diagrams can be generated, as illustrated in Figure 2.11. It is important to account for these kinds of eye shapes by considering different combinations

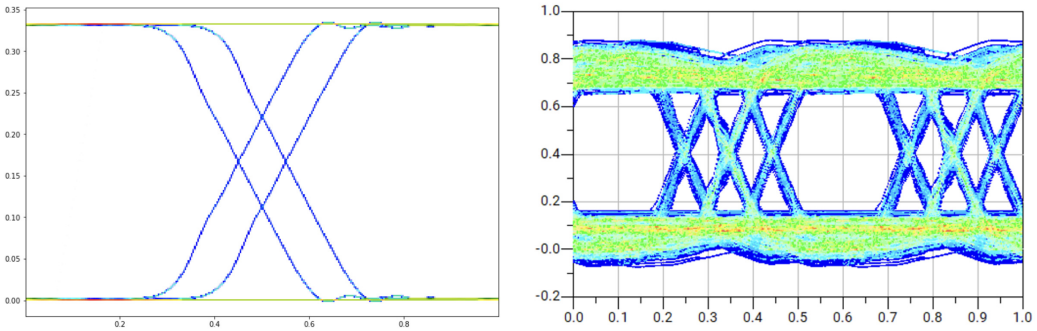


Figure 2.11: dual-modal PDF_{TJ} (left), tri-modal PDF_{TJ} (right)

of jitter components.

2.5 Eye Parameter Definitions

While the eye diagram serves as a visualization tool to assess the quality of the signal, there exist several "eye parameters" that quantitatively evaluate such quality, supported by their mathematical definitions [5]. The majority of them require the location of eye crossing points, hence the need for exploration in this paper. This section covers eye parameters for PAM-2 as well as PAM-4.

2.5.1 PAM-2 - Horizontal Parameters

Eye Width (EW)

$$EW = (\mu_{t2} - 3 * \sigma_{t2}) - (\mu_{t1} + 3 * \sigma_{t1}). \tag{2.11}$$

Where μ and σ are the mean and the variance of a thin horizontal strip near the two crossing points. EW hence requires that one specify the vertical value of the crossing point.

Jitter - Root Mean Square (RMS) and Peak-to-Peak (p-p) Jitter refers to the time deviation from the ideal timing of the data bit. This is one of the most important aspect in determining the quality of the digital data signals, as it directly contributes to the BER. To quantify jitter, a "thin"

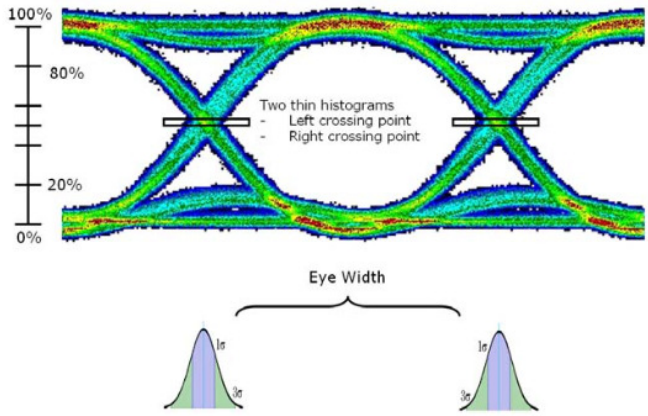


Figure 2.12: Eye Width is measured between the 3σ inner points.

horizontal strip window is applied to form a histogram. The p-p jitter is defined as the full width of the histogram, representing the maximum margin, and the RMS jitter is defined as the standard deviation of the histogram.

$$Jitter_{\text{p-p}} = t_{\text{max of histogram}} - t_{\text{min of histogram}} \tag{2.12}$$

$$Jitter_{\text{RMS}} = \sigma_{\text{histogram}} \tag{2.13}$$

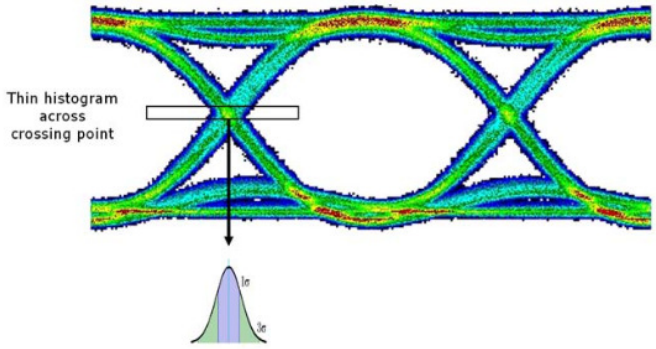


Figure 2.13: Jitter measurements are obtained from the histogram data near the eye crossing point.

It is important to note that it is required to locate where the eye crossing point is in order to obtain these parameters, and $Jitter_{\text{p-p}}$ can change depending on how "thin" the horizontal strip window is chosen. EDA simulation tools and oscilloscopes do not have a specific method disclosed to how "thin" the strip should be.

Rise Time ($\mathbf{t_{rise}}$) Rise Time refers to the mean transition time of the data on the upward slope ($20 \sim 80\%$) of an eye.

$$t_{rise} = \mu_{80\% \text{ level}} - \mu_{20\% \text{ level}}. \tag{2.14}$$

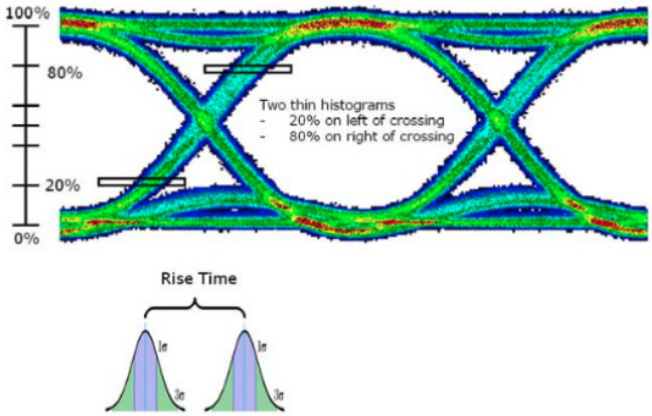


Figure 2.14: Visual representation of t_{rise}.

Fall Time ($\mathbf{t_{fall}}$) Fall Time refers to the mean transition time of the data on the downward slope ($80 \sim 20\%$) of an eye.

$$t_{rise} = \mu_{20\% \text{ level}} - \mu_{80\% \text{ level}}. \tag{2.15}$$

2.5.2 PAM-2 - Vertical Parameters

One Level (Level1) One Level is defined as the average value of the upper half of $40 \sim 60\%$ of the eye. This value represents the analog voltage value that represents a digital "1".

$$Level1 = \mu_{\text{upper half of } 40 \sim 60\% \text{ of the eye}}. \tag{2.16}$$

Where $40 \sim 60\%$ segment is with respect to 2-UIs. For example, if UI = 2ns, we would take all the "1" (upper) values between 0.8ns to 1.2ns.

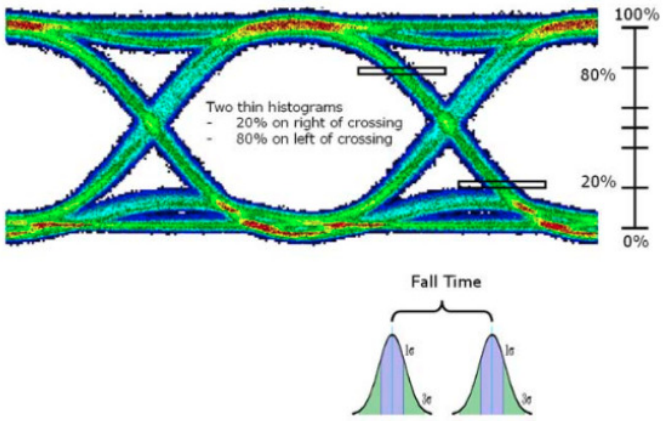


Figure 2.15: Visual representation of t_{fall}.

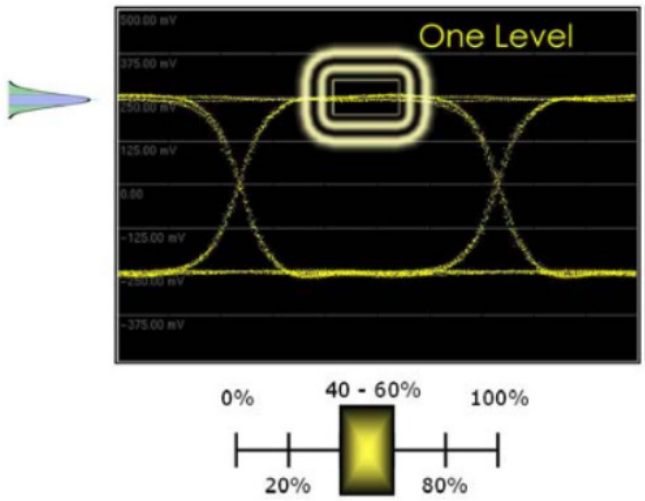


Figure 2.16: Visual representation of One Level. Values in the rectangle are averaged.

Zero Level (Level0) Zero Level is defined as the average value of the lower half of 40 $\sim$ 60% of the eye. This value represents the analog voltage value that represents a digital "0".

$$Level0 = \mu_{\text{lower half of 40 } \sim \text{ 60\% of the eye}}. \tag{2.17}$$

Where 40 $\sim$ 60% segment is with respect to 2-UIs. For example, if UI = 2ns, we would take all the "0" (lower) values between 0.8ns to 1.2ns. We notice the need of mathematically defining the threshold that differentiates upper and lower values, of which the vertical value of the eye crossing point can be an accurate value.

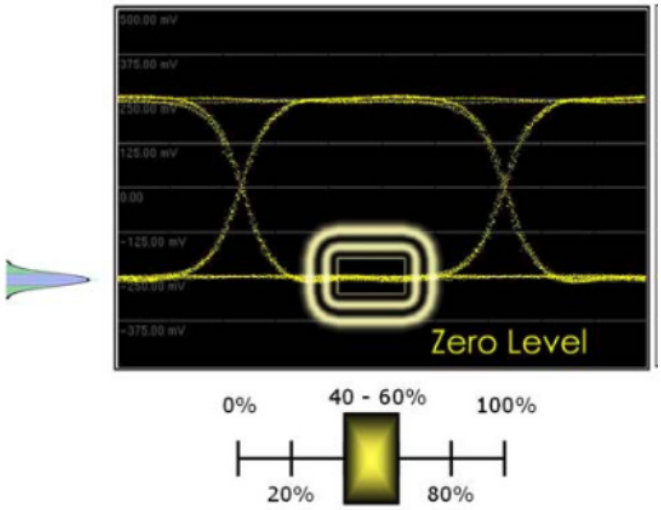


Figure 2.17: Visual representation of Zero Level. Values in the rectangle are averaged.

Eye Amplitude (EyeAmp)

$$EyeAmp = Level1 - Level0 \tag{2.18}$$

Eye Amplitude is the difference between One Level and Zero Level. It tells the degree of separation between received "0"s and "1"s.

Eye Height (EH) Eye Height is the difference between the inner 3σ points between the one and zero levels.

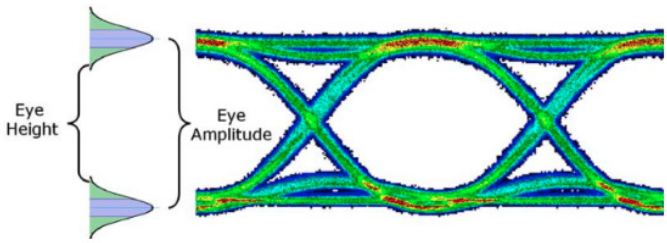


Figure 2.18: Visual representation of Eye Height and Eye Amplitude.

$$EH = (Level1 - 3 * \sigma_{\text{Level1}}) - (Level0 + 3 * \sigma_{\text{Level0}}) \tag{2.19}$$

Where σ refers to the variance of the vertical values between $40 \sim 60\%$ of the eye.

Signal-to-Noise Ratio (SNR) Signal-to-Ratio (SNR) is a ratio of the desired signal level to the level of the background noise plus any distortion. High SNR values are desired in real world applications. In eye diagram analysis, SNR is defined as:

$$SNR = \frac{Level1 - Level0}{\sigma_{\text{Level1}} + \sigma_{\text{Level0}}} \tag{2.20}$$

Crossing Percentage (Crossing%) Crossing Percentage tells us at what % with respect to the eye amplitude is the eye crossing point is located.

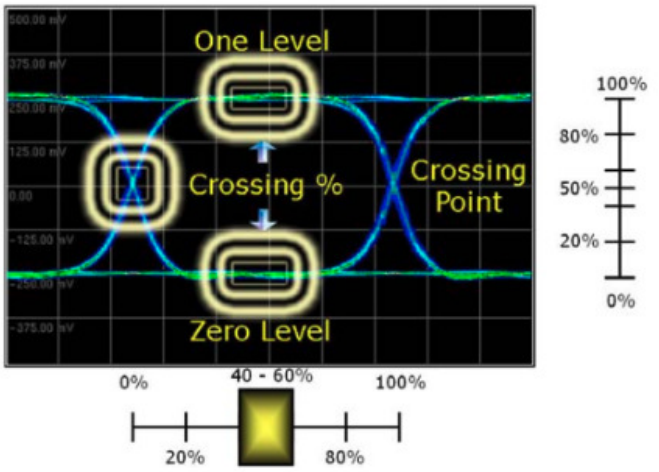


Figure 2.19: Visual representation of crossing percentage.

$$Crossing\% = 100\% * \frac{(y_{\text{crossing point}} - Level0)}{EyeAmp} \tag{2.21}$$

Crossing percentage is usually 50%, as one would often set rise time and fall time to be identical values. However, in cases where non-linearity prevails in the channel, the received signal's eye diagram can have crossing percentage way above or below 50%. Furthermore, current industry EDA tools allow different rise and fall time, which makes the eye crossing percentage away from 50% even if the channel has high linearity.

2.5.3 Other Parameters

2.5.4 PAM-4 Eye Parameters

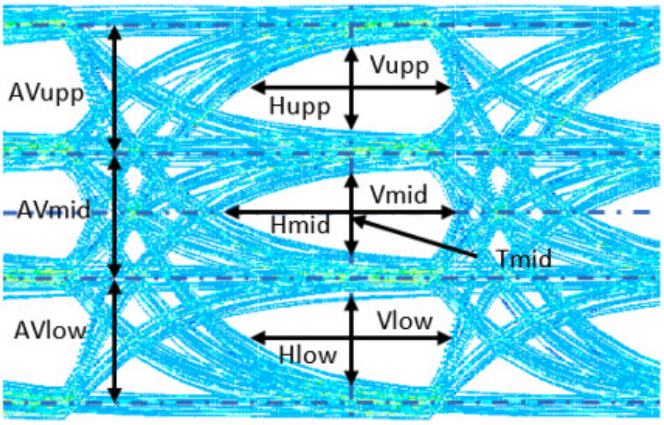


Figure 2.20: A PAM-4 Eye Diagram with eye parameters labelled

While there are various types of eye parameters for PAM-2, PAM-4 comes with a new set of parameters similar to those of PAM-2. PAM-4 eye parameters are listed in Table 2.1.

Table 2.1: PAM-4 Eye Parameters

	Definitions
T_{mid}	Midpoint of the maximum horizontal eye opening of the middle eye
AV_{mid}	Difference of the mean levels of the $+\frac{1}{3}$ and $-\frac{1}{3}$ level voltage in a $\pm$0.025UI time window centered on T_{mid}
AV_{upp}	Difference of the mean levels of the +1 and $-\frac{1}{3}$ level voltage in a $\pm$0.025UI time window centered on T_{mid}
AV_{low}	Difference of the mean levels of the $-\frac{1}{3}$ and -1 level voltage in a $\pm$0.025UI time window centered on T_{mid}
H_{mid}	10^{-6} inner eye width calculated at $\frac{V_{\text{mid}}}{2}$
H_{upp}	10^{-6} inner eye width calculated at $\frac{V_{\text{upp}}}{2}$
H_{low}	10^{-6} inner eye width calculated at $\frac{V_{\text{low}}}{2}$
V_{mid}	10^{-6} inner eye height calculated in a $\pm$0.025UI time window centered on T_{mid}
V_{upp}	10^{-6} inner eye height calculated in a $\pm$0.025UI time window centered on T_{upp}
V_{low}	10^{-6} inner eye height calculated in a $\pm$0.025UI time window centered on T_{low}

As opposed to PAM-2, PAM-4 parameters do not require one to locate the crossing point, but rather find T_{mid} to be placed at the 0.5UI point. This is because there are many crossing points occur for different level transitions. For example, crossing point created by 00→10 and 10→00 is not the same as the crossing created by 00→01 and 01→10 transitions.

2.6 Pseudo-Random Binary Sequence (PRBS)

In signal integrity, we can use a single pulse response to quantify the degree of intersymbol interference (ISI) present in a channel that we wish to evaluate. However, a single pulse does not give us the whole picture. In practice, a very large number of bits are transmitted through the channel and previous bits can affect waveform shape of later bits.

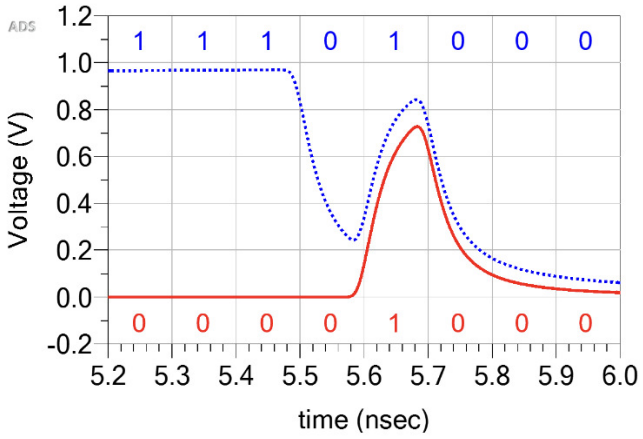


Figure 2.21: Two single pulse responses with different preceding bits.

Figure 2.21 shows two single pulse responses between 5.5 5.8ns, with different preceding starting bits. Therefore, in order to evaluate the signal integrity of a channel, it is important to consider a larger number of bit pattern combinations. The concept of Pseudo-Random Binary Sequence (PRBS) is hence introduced to mimic different levels of ISI in a channel. PRBS generates binary square wave patterns which can be convolved with the pulse response directly to obtain the resulting output waveform. The PRBS can be generated in a specific pattern using linear-feedback shift register (LFSR), which ensures there are as many "1"s as there are "0"s. An L-bit LFSR generates PRBS of length $2^L - 1$ with 2^{L-1} "1"s, $2^{L-1} - 1$ "0"s, and 2^{L-1} edges (rising and falling). L-bit LFSR takes some bits as input to an XOR gate and its output is the next bit to be bit-shifted into the register. The least significant bit (LSB) then becomes the output of the LFSR. The bit locations can be described as a polynomial. As an example, Figure 2.22 illustrates a 4-bit LFSR, which uses 3rd and 4th bit as inputs to the XOR gate. It generates 15 bits of PRBS, with 8 "1"s, 7"0"s, and 8 edges [6].

In Figure 2.21, we observed that a single pulse (0→1→0) response can

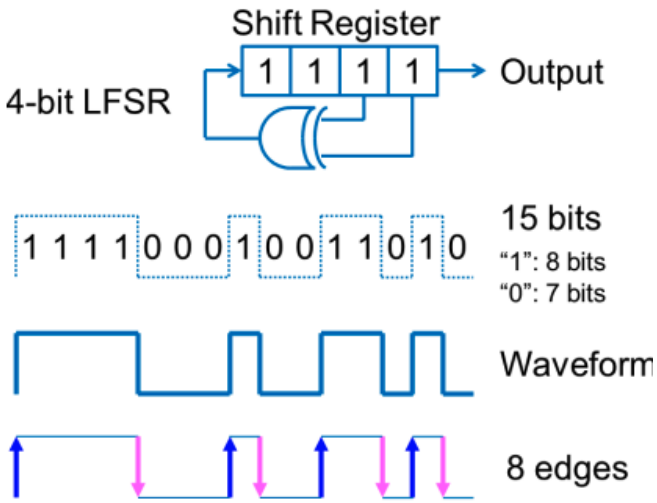


Figure 2.22: 4-bit LFSR visual illustration.

be different if we consider different bit combinations preceding the pulse. Taking 4-bit LFSR as an example, Figure 2.23 illustrates one period of LFSR

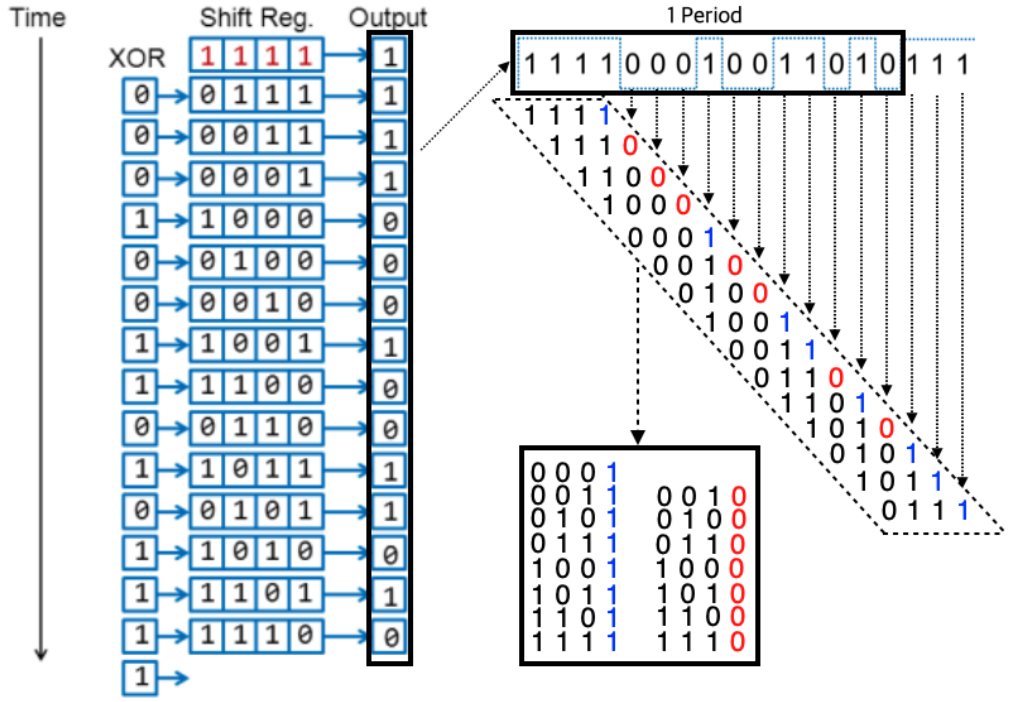


Figure 2.23: Illustration of 4-bit LFSR outputs in the time domain.

outputs, as well as each stage's shift register values. We can see that for every output (i.e. the LSB of the shift register) 1 and 0, we have all possible 3-bit

combinations that comes before the current output, thereby considering all preceding bit combinations before the pulse. Note that "0000" doesn't exist since feeding it would generate only 0's forever. The placement of the XOR gate with specific bit locations help generate this pattern. This logic gate structure can be mathematically described as a polynomial of $x^4 + x^3 + 1$. Table 2.2 shows LFSRs with different bit lengths:

Table 2.2: L-bit LSFR feedback polynomial and their period

L	Feedback Polynomial	Period $(2^L - 1)$
4	$x^4 + x^3 + 1$	15
5	$x^5 + x^3 + 1$	31
6	$x^6 + x^5 + 1$	63
7	$x^7 + x^6 + 1$	127
8	$x^8 + x^6 + x^5 + x^4 + 1$	255
9	$x^9 + x^5 + 1$	511
10	$x^{10} + x^7 + 1$	1023
11	$x^{11} + x^9 + 1$	2047

It can be implied that bigger L-bit LFSR would generate more sophisticated preceding bit patterns before the pulse. Commercial EDA tools such as ADS employs 8-bit LFSR as the default mode, though it can be user-defined as well.

2.7 K-means Clustering Algorithm

During the steps of locating the crossing point, it is crucial to differentiate transitions that do and do not contribute to the crossing point. For a PAM-2 eye as an example, we would have 1→0 and 0→1 transitions that do contribute to the crossing point, and 1→1 and 0→0 that do not contribute to the crossing point. Before this classification, we need to find approximate voltage values for 1→1 and 0→0 transitions. To do so, a vector quantization method called **K-Means Clustering** is an effective solution. While Chapter 4.4 illustrates the usage of it in more detail, this section explains the methodology behind this algorithm.

First proposed by Stuart Lloyd of Bell Labs in 1957 as part of a pulse-code modulation (PCM) technique, the "k-means" algorithm partitions a set of

data into k sets that minimizes the sum of squares within the cluster [7]. In mathematical terms, it aims to find:

$$\arg\min_{\mathbf{S}} \sum_{i=1}^{k} \sum_{\mathbf{x} \in S_i} \|\mathbf{x} - \boldsymbol{\mu}_i\|^2 \tag{2.22}$$

where $\boldsymbol{\mu}_i$ is the mean of points in $\boldsymbol{S}_i$. $\boldsymbol{\mu}_i$ is also called a centroid. The algorithm first begins by randomly assigning k mean points. Then, k clusters are created by grouping every data with the nearest mean. For each cluster, a centroid using Equation 2.22 is determined, which becomes the new "mean". Finally, the above process is repeated until all centroids converge.

This can be a computationally heavy algorithm if the input data is multi-dimensional, but thankfully our scope narrows down to a single dimension – voltage. The application of this algorithm is illustrated in more detail in chapter 4.4

CHAPTER 3

EXISTING METHODS AND THEIR DRAWBACKS

Eye diagram functionality is implemented in oscilloscopes, as well as software EDA simulation tools such as ADS, Ansys, and Virtuoso. While the mathematical definitions are already established, the crossing point detection algorithm remains proprietary as part of the companies' intellectual properties. There also had been also a patent that aimed to produce this algorithm. In this chapter, some inaccuracies of these attempts are identified for each tool approach, as well as some initial attempts that were proposed on my own with their drawbacks.

3.1 Current EDA Tools and Patents

3.1.1 Advanced Design System (ADS)

Advanced Design System (ADS) from Keysight is a powerful electronic design automation (EDA) software platform that provides design solution for RF, microwave, and high-speed circuits. The "Eye Probe" tool provides eye diagram analysis solution. In terms of its crossing point detection capability, it had been observed that its accuracy largely depends on the eye crossing percentage as defined in Equation 2.5.2. It provides accurate placement of the eye at 25% of 2-UI window when the crossing percentage is near 50%, but becomes inaccurate as the percentage deviates from 50%. In fact, in the help page of the "Eye Probe" tool, it explicitly states that this tool "uses automatic algorithms to detect eye crossing thresholds. If the eye is closed or highly distorted, these automated algorithms may fail, resulting in an all-zero output to the data display".

Figure 3.1 depicts cases where the error occurs, even though a crossing point clearly exists. The two figures in the left column are eye diagrams with

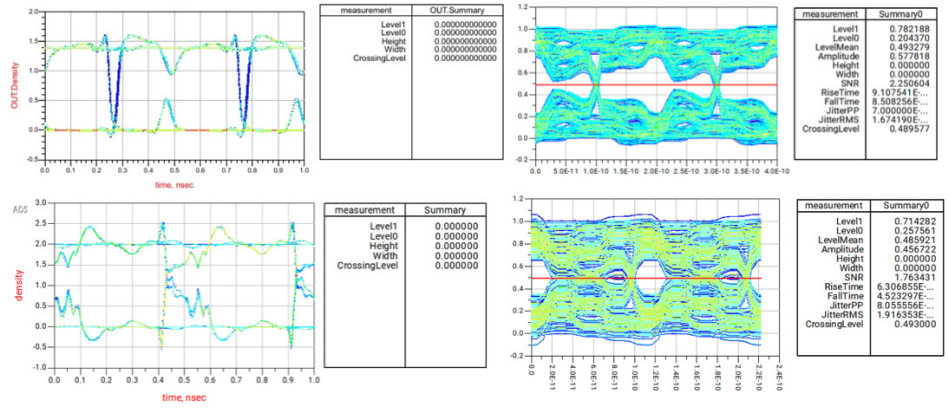


Figure 3.1: Different eye diagrams with eye parameter summary table on the right – Non-identifiable eye parameters are zeroed-out.

a relatively low bit rate of 2GHz. ADS outputs all zeros in the parameters output, showing that it is incapable of locating the crossing point. The two figures in the right colum are eye diagrams with higher bit rate higher than 9GHz. The top right figure at first sight looks like it has located the vertical and horizontal eye crossing point correctly, with each parameter produced. However, it is unable to calculate the eye height and eye width. The bottom right has a similar issue, but with an additional problem that the eye is not shifted correctly.

More examples with various channels and high bit rates are tested to come to a conclusion that ADS's algorithm attempts to find the crossing point by firstly assuming crossing point occurs near 50% of one and zero level, then finds mean timing value around that voltage level. This explains why in top right of Figure 3.1 was able to place the crossing point at 25% of the 2-UI window by considering a thin horizontal strip near 0.5V, but the bottom right of the same figure placed the crossing point incorrectly since the preceding ISI before the crossing point coincided with the 0.5V threshold, therefore counting not only the crossing point but also the intersections between the 0.5V threshold as well as some of the ISI components. Furthermore, ADS comes with auto scale issues where some values are not subtracted correctly such that they are higher than the 2UI value. This is illustrated in the bottom right of Figure 3.1.

3.1.2 Ansys Circuits

Ansys is another electronic design software platform, famously known for is 3D simulation high-frequency structure simuilator (HFSS). Ansys is also capable of providing eye diagram in "circuits" simulator. Figure 3.2 shows the option tab for eye plotting. While it does have buttons for automatic eye delay and crossing amplitude, a user can also uncheck the button to manually input any value of delay as well as any value of crossing amplitude, which effectively changes all the eye parameters [8].

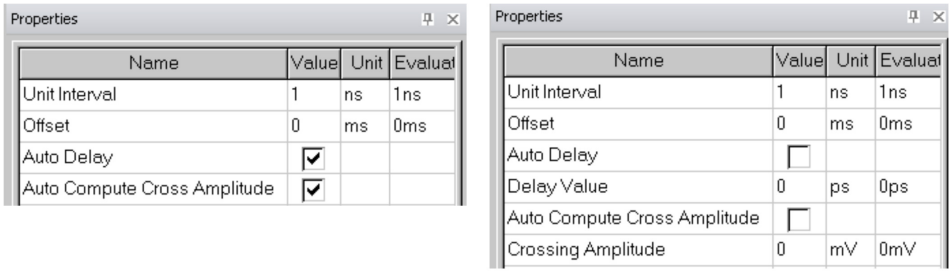


Figure 3.2: Ansys Circuits eye probe settings tab.

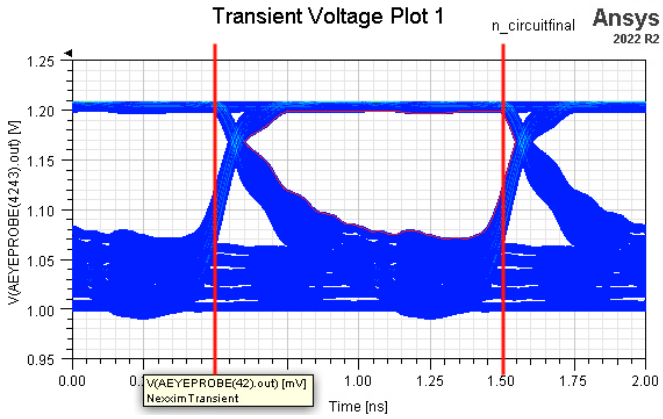


Figure 3.3: Inaccurate eye diagram produced by Ansys Circuit for a non-linear eye.

Ansys' help documentation somewhat explains the logic behind their functionalities, which explains what the buttons in Figure 3.2 do:

> *"...crossing time is calculated by creating a horizontal histogram across a small, narrow strip at the middle of the eye. The middle of the eye is computed as the* ***middle of the vertical extremities*** *of the eye i.e. the midpoint of max & min voltage values across the complete eye diagram."*

> *"...with Auto Mode "on, the eye width is calculated as the average of the two peaks of the vertical histogram. This is shown as the "Eye Crossing Amplitude" in the figure above. Both statistical and minimum eye widths will be calculated at this amplitude."*

This helps explain why Ansys isn't able to place the eye correctly in Figure 3.3, since it seems to assume crossing percentage of 50%. Also, the eye width calculation method is slightly different from the definition in Equation 2.18. This can be accurate if the variance within the definition window (40% - 60% of 2UI) is small, but can become inaccurate if either the variance is high, or if there is significant ISI such that the peaks of the vertical histogram do not represent voltage values of digital "0" and "1".

3.1.3 Cadence Virtuoso

Virtuoso, developed by Cadence, is a widely used simulation EDA software for electrical engineers in RF, IC, and mixed-signal design [9]. While it also supports eye diagram analysis, it has two critical drawbacks. First, as seen in Figure 3.4, it requires the user to input a threshold voltage value before plotting the eye and calculating eye parameters. This value corresponds to the voltage value of the crossing point.

Secondly, as seen in Figure 3.5, Virtuoso only overlays the waveform in a single color, unable to show a density plot, which is more visually straightforward for a user.

3.1.4 Existing Patent

An expired patent filed by Tektronix called "Algorithm for Finding the Eye Crossing Level of a Multilevel Signal" gives us an insight into how its oscilloscopes employ crossing point search [10].

Figure 3.6 shows part of Tektronix's patent. It also uses histogram approach to solve this problem. It places a thin horizontal box at multiple voltage levels and looks for the minimum standard deviation. Then, the timing location at which the standard deviation of the horizontal histogram box is minimum would be the crossing point. For example, top right of Figure 3.6 places a thin horizontal box (box 26) at an arbitrary location, and its

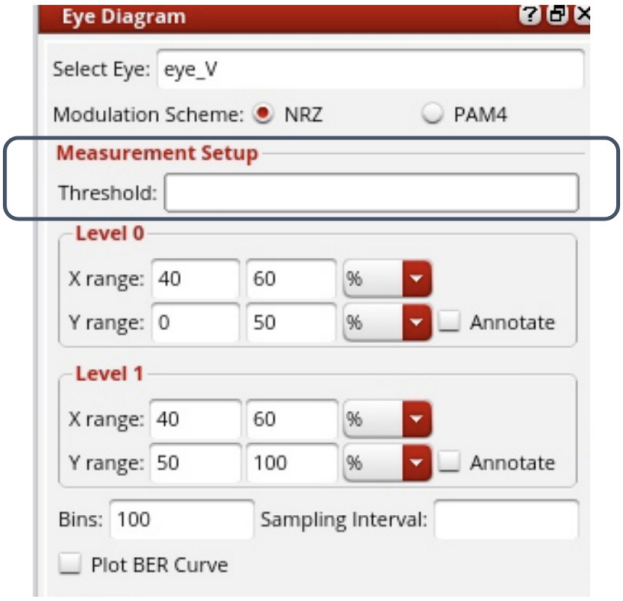


Figure 3.4: Inaccurate eye diagram produced by Ansys Circuit for a non-linear eye.

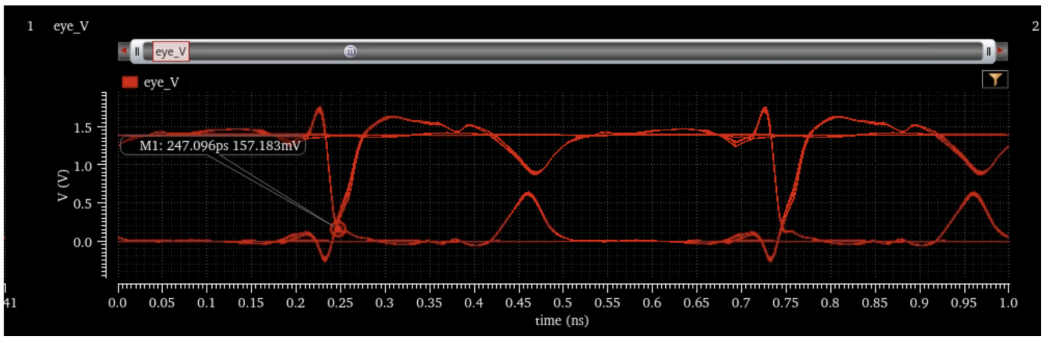


Figure 3.5: Inaccurate eye diagram produced by Ansys Circuit for a non-linear eye.

standard deviation is calculated with plot 28 showing its probability distribution function. This is repeated until a minimum is found, which would correspond to bottom right of Figure 3.6.

Though the numbers are not specifically stated, the width and height of the thin horitonzal strip as well as its horizontal location seems to be the most important considerations. It can be surmised that this algorithm can be very accurate for an eye with minimal ISI and jitter, even being able to account for non-linear or rise-fall mismatch scenarios leading to crossing percentage deviating from 50% locations. However, crossing point detection

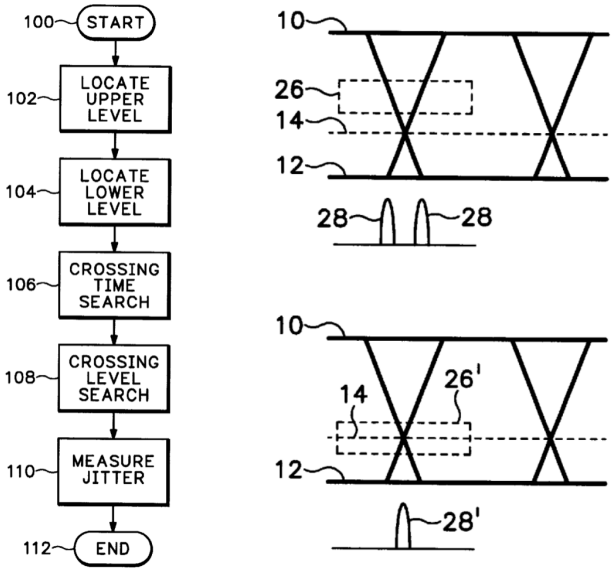


Figure 3.6: Flowchart of the algorithm (left) and visual illustration (right).

can become inaccurate for eyes with high ISI and jitter.

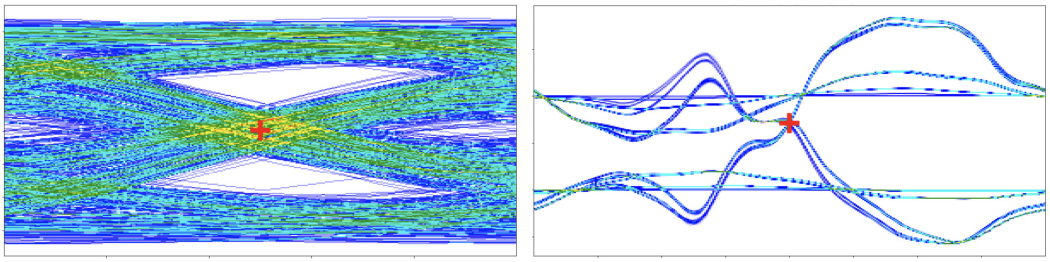


Figure 3.7: Eye Diagrams with high jitter (left) and high ISI (right), with their crossing point marked with red cross.

Consider the two eye diagrams in Figure 3.7. For the case of left figure with high jitter, the "thin horizontal box" approach can become unstable since the PDF can vary depending on the box's height, width, and location. Furthermore, for the case of right figure with high ISI, the PDF would look different when the width horizontal box is set to different lengths. From these scenarios, we can see that minimum σ point is not always equal to the eye crossing point.

3.2 Self-developed Initial attempts

Before presenting the final algorithm in Chapter 3, some of the initial attempts of finding an optimal solution is presented in this section, with counter examples that invalidate their comprehensiveness.

First idea was to construct a 2D histogram to find the bin with the highest count, since the crossing point would contain both rising and falling curves. However, this fails for an eye with bimodal jitter such as one in Figure 3.8. First of all, the binning method in histogram can result in different count matrix for different bin size, hence making the output not consistent with the waveform. Second, it is favorable that we define the crossing point to be the average of the crossing points when there are multiple crossings, but the counting method would not be applied correctly for this example as our desired output would be an empty space (average of the two crossing points).

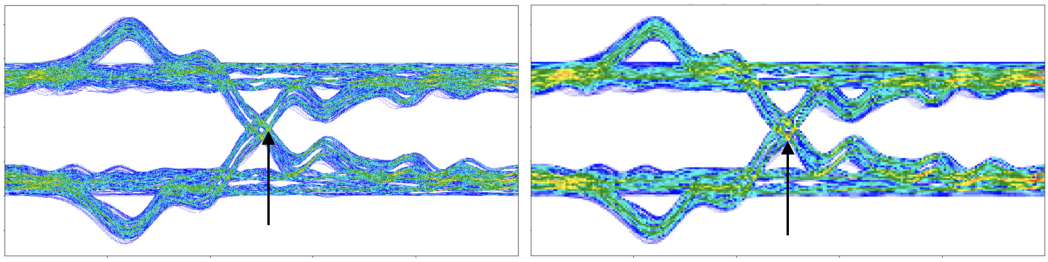


Figure 3.8: 2D density histogram of identical eye waveform, with bin size of 1000x500 (left) and 500x125 (right).

In order to resolve the second issue, another idea was to keep using the histogram, but find a horizontal strip with longest consecutive empty count bins. Left of Figure 3.9 shows a successful calculation of the crossing point, with red arrow indicating the horizontal strip with longest empty bins. By finding such strip from both left and right side of the eye diagram, multiple crossing scenarios can be accounted for. However, as shown in the right of Figure **??**, it can fail when the ISI is significant in a way that results in undershoots in 1 level and overshoots in 0 level, thereby impeding the calculation accuracy.

More ideas other than the two presented above were suggested and were faced with counter examples that revealed their comprehensiveness. Our goal is not only to validate the existing algorithm which work very well for "normally" shaped eye, but also account for non-ideal channels that other

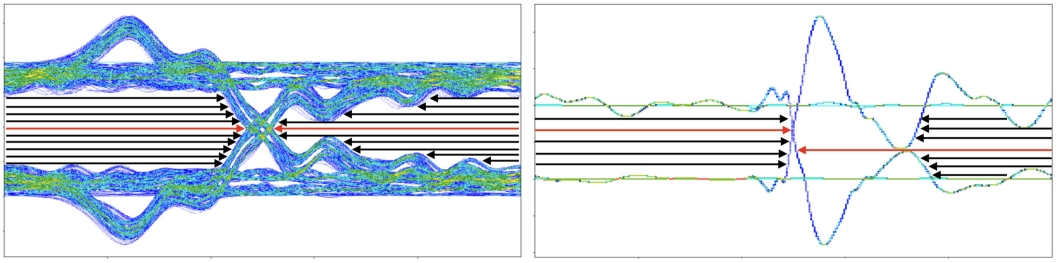


Figure 3.9: Brief illustration of the thin horizontal empty strip method. Red arrows indicate the finally obtained crossing point boundary, with a working case (left) and failing case (right).

existing methods are not able to achieve the crossing point detection. Broader comprehensiveness would allow us to quantify, if so, how "bad" the channel is without the user having to eyeball the approximate delay they need to apply in order to proceed with eye parameter analysis.

3.3 Summary Table of Existing Methods

By comparing identical waveforms with different tools, Table 3.1 is established as a summary. The desired goal of this paper is to present an algorithm that can overcome some of the disadvantages or failing cases.

Table 3.1: Summary Table of Eye Diagram Tool Functionality

◯ = Yes △ = Sometimes × = Fails

	ADS	**Ansys Circuits**	**Cadence Virtuoso**	**Proposed Algorithm**
Automatic Eye Centering for Eye with 50% Crossing	△	◯	◯	◯
Automatic Eye Centering for Eye with non-50% Crossing	△	△	△	◯
Automatic Eye Parameter Calculation	◯	◯	×, needs to define threshold	◯
Uses Histogram for Eye Parameter Calculation	◯	◯	◯	◯
Default Histogram Bin Size	456 × 321	unknown	500 × 500	user-defined
Heatmap Density Plot	◯	◯	×	◯

CHAPTER 4

ALGORITHM FOR PAM-2 EYE CROSSING POINT DETECTION

As introduced in Chapter 2, quantitative eye diagram analysis requires placement of the crossing point at the 0.5 UI location. This chapter dives into the developed algorithm of finding the eye crossing point for PAM-2 eye diagram simulation. Figure 4.1 shows the overall process of it, with the left column representing an overall flowchart and the right with a visual example of it. The following sections in this chapter goes through the specifics for each of the blocks in the flowchart and explain the reasoning behind such blocks and code snippets to illustrate how they are realized.

It is important to consider the runtime of this algorithm, and there had been a lot of trial-and-errors and comparison of the runtime with different algorithms or libraries being used. While the initial attempt at constructing the algorithm involved numerous external libraries, runtime optimization over the course of research resulted in a major use of the Numpy library. Another major decision was to only utilize the first UI of the eye diagram. Even though a typical eye diagram would span across 2 UI, the algorithm is implemented with 1 UI. As illustrated in Figure 2.1, a 2-UI window is shifted in 1-UI intervals, implying that considering the left half of the 2-UI eye diagram is sufficient. Detailed comparisons and runtime considerations are explained in more detail in Chapter 5.

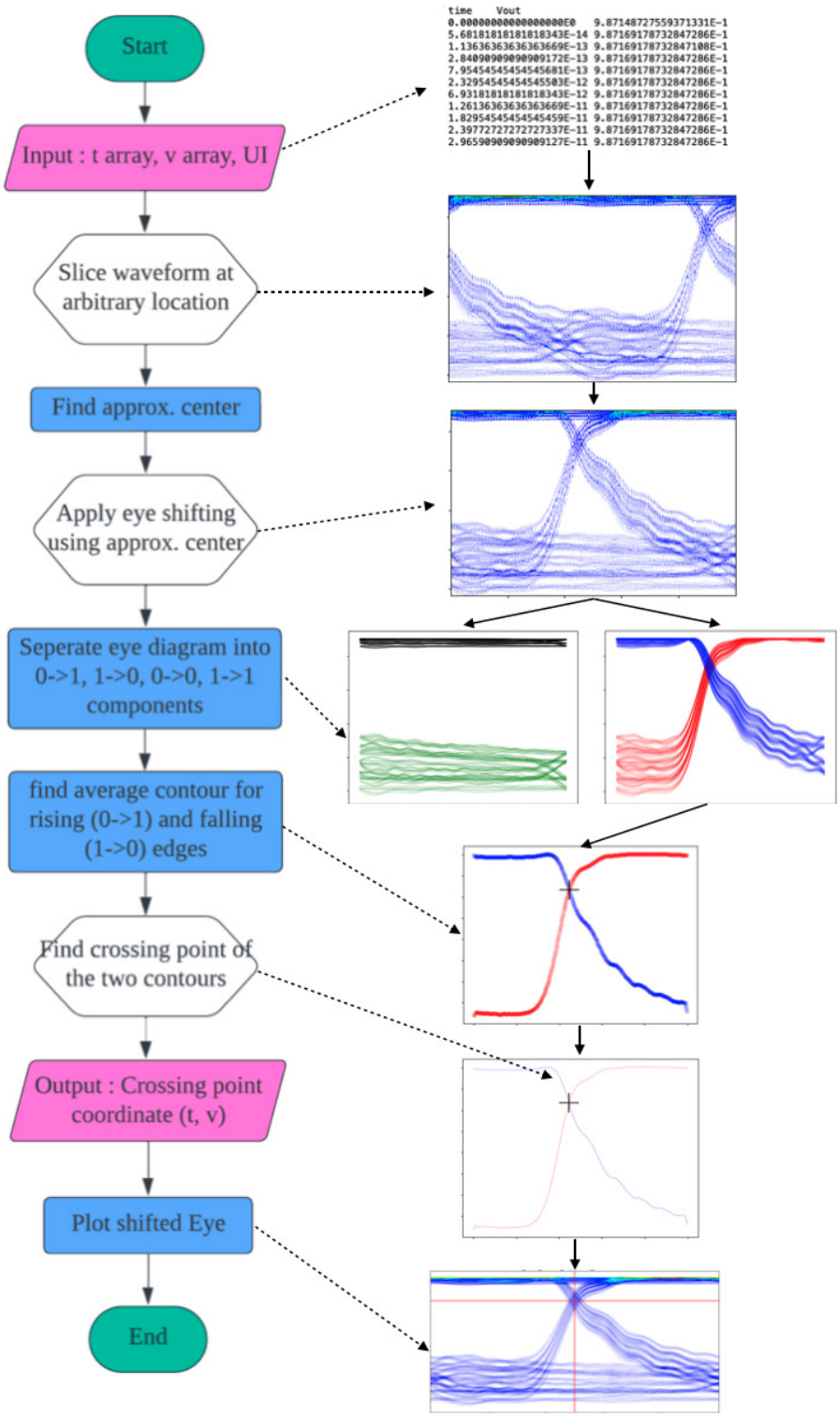
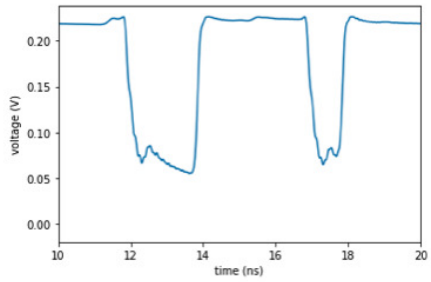


Figure 4.1: General flowchart overview of the algorithm.

4.1 Input and Output Consideration



time	Vout
0.0000000000000000E0	9.87148727559371331E-1
5.68181818181818343E-14	9.87169178732847286E-1
1.13636363636363669E-13	9.87169178732847108E-1
2.84090909090909172E-13	9.87169178732847286E-1
7.95454545454545681E-13	9.87169178732847286E-1
2.32954545454545503E-12	9.87169178732847286E-1
6.93181818181818343E-12	9.87169178732847286E-1
1.26136363636363669E-11	9.87169178732847286E-1
1.82954545454545459E-11	9.87169178732847286E-1
2.39772727272727337E-11	9.87169178732847286E-1
2.96590909090909127E-11	9.87169178732847286E-1

Figure 4.2: Plot of a transient simulation output (left) with actual data that stores the plot information (right).

The eye diagram is generated from a voltage waveform in the time domain for bit-by-bit and transient simulation mode. As illustrated in Figure 4.2, this can be stored as (t, V) coordinate pairs which we can import into two arrays – one for time and one for corresponding voltage. Additionally, since the frequency is defined in PRBS generator, UI can be obtained by taking the inverse of the frequency. These three are the only inputs required for this algorithm.

```
#start reading txt file that contains (t, V) pairs
data = pd.read_csv(file,sep ='\s+')
voltage = data["Waveform"].to_numpy()
time = data["time"].to_numpy()/1E-9
#converted to ns for easier analysis
```

While we only need the time horizontal coordinate of the crossing point for accurate shifting, we would also need the vertical coordinate for some eye parameter analysis such as BER and jitter. Therefore, our desired output would be a single pair of coordinates (t_{xing}, v_{xing}).

4.2 Slice Waveform at Arbitrary Location

If we know where the exact crossing point is, we can simply choose $[t_{xing} - 0.5\text{UI}, t_{xing} + 0.5\text{UI}, t_{xing} + 1.5\text{UI}, ...]$, then the crossing point would be located at exactly 50% of the UI. However, that is the desired output of this algorithm. Therefore, as the first step of the entire algorithm, we need to decide

an arbitrary location for slicing location. the eye by slicing the waveform an arbitrary location. In other words, we construct the 1-UI eye diagram by choosing a random time point $t_{arbitrary}$ and slice the original waveform at:

$$\text{slicing points} = t_{arbitrary} + \text{UI} * k, \text{where } k \in \mathbb{Z}^+ \tag{4.1}$$

Since such slicing time point can be totally arbitrary at this stage, it is the best to use non-negative integer intervals as slicing points – [0, UI, 2UI, 3UI...]. This can be simply achieved by the code below:

```
#input = t_array, output = eye_t
eye_t = [x-int(x/UI)*UI for x in t_array]
```

The following table illustrates an example with UI=2s. Comparing the input and the output, we see that the output is subtracted by integer multiples of UIs such that all elements in the output are bounded by [0, UI]. "int" function acts as a mathematical floor function in python.

Table 4.1: A Sample case study with UI=2s

t_array index i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
input = t_array[i]	0	0.33	0.67	1	1.33	1.67	1.99	2.01	2.33	2.67	3	3.33	3.67	4	4.33
int(input/UI)	0	0	0	0	0	0	0	1	1	1	1	1	1	2	2
int(input/UI)*UI	0	0	0	0	0	0	0	2	2	2	2	2	2	4	4
output eye_t	0	0.33	0.67	1	1.33	1.67	1.99	0.01	0.33	0.67	1	1.33	1.67	0	0.33

There are several types of eye diagrams, such as bit-by-bit, statistical, and transient-generated. The first two methods can provide a fast and statistical estimation of the eye diagram [11]. However, they can only be applied to linear channels. Transient simulated eye diagrams can yield more accurate results and can be applied to both linear and nonlinear channels at the cost of longer simulation times.

In this chapter, we present the LIM method for pulse amplitude modulation 2-level (PAM-2) eye diagram simulation. Methods for both without and with the effect of crosstalk are introduced. Since LIM has linear computational complexity, it can be faster than MNA methods which are utilized for transient-generated eye diagram simulation. We have compared LIM to both transient and channel simulation in Keysight ADS, in order to test its accuracy and speed.

4.3 Finding Approximate Horizontal Eye Center

After a temporary eye is created, we find an approximate horizontal eye center. For the purpose of this section, only the $t_{xing,approx}$ is obtained, and the voltage crossing point can be neglected. This section is necessary in order to categorize and store different transitions (00, 01, 10, 11) which is explained in the next section.

The overall approach is: 1. Extract a voltage level assuming a crossing percentage (Equation 2.11) of 50%, and draw a horizontal line at that level. 2. Find time values of the intersection between the eye and the horizontal line. 3. Take the average value of the intersection time values.

Such method would help us locate an approximate location for the eye crossing by realizing that the PRBS generator generates roughly equal number of rising and falling edges that form the eye crossing, and taking the average of these points would give an approximate time value for the crossing point. This seemingly easy task actually involves consideration of all kinds of eye cases, as well as how the eye is formed in the previous section. The following considerations are important for non-linear eyes where the crossing percentage can deviate away from the assumed 50%.

4.3.1 Removing small $\frac{dV}{dt}$ data points

The first consideration is removing small slope data points and preserving sharper slope data points which generally correspond to rising and falling edges. The motivation of this approach is the realization of 1-to-1 and 0-to-0 transitions do not contribute to the intersection. This implementation cannot perfectly remove all the same level transitions for all eye shapes, such as an eye with sharp overshoot or undershoot during same level transitions. Nevertheless, this reduces the total array length to operate with, which directly impacts the overall runtime. This is achieved by simply obtaining a slope array using *np.gradient* function, and setting a threshold to filter out indexes needed to be removed from the t and V arrays. The following code block achieves this:

```
1 #t = time array, V = voltage array
2 #compute slope value given t and V arrays
3 dvdt = np.absolute(np.gradient(V,t)
```

```
#find average of all slope values
avg_dvdt = sum(dvdt)/len(dvdt)
idx_to_remove = set([i for i in range(len(dvdt)) if dvdt[i] < avg_dvdt])
# print(indexes_to_remove)
V = [i for j, i in enumerate(V) if j not in indexes_to_remove]
t = [i for j, i in enumerate(t) if j not in indexes_to_remove]
```

4.3.2 Savitzky-Golay Filter for Data Smoothing

The second consideration is for eyes with large ripples that occur near the 50% crossing level when looking for intersections. Figure 4.3 is a voltage waveform with $f = 7GHz$, with each vertical grid marking every UI.

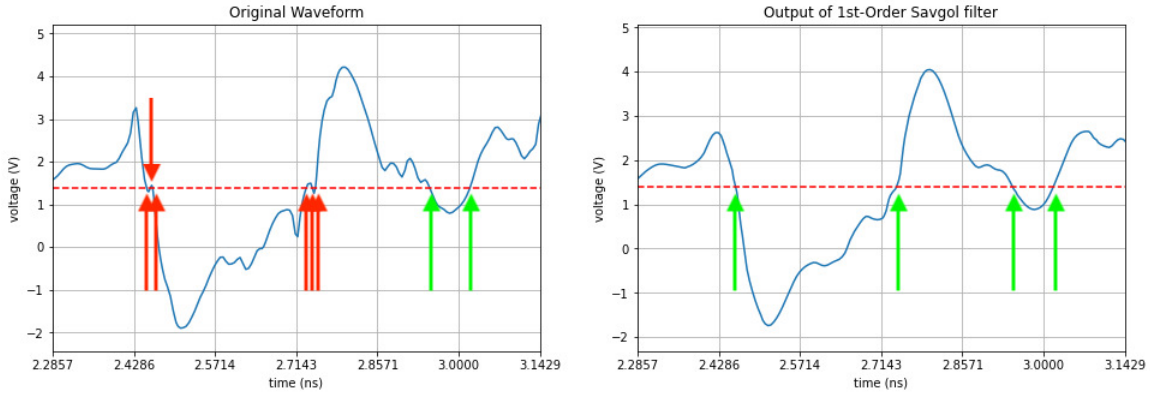


Figure 4.3: Original 7GHz voltage waveform (left) and output after applying Savitzky-Golay filter (right).

The red dashed horizontal line is the pre-determined threshold value assuming crossing percentage of 50%. Green arrows are the regular transitions with no issues, but the red ones show minor fluctuations that result in crossing the threshold line multiple times. This makes the output biased where we take the average of the intersections. Since we are looking for an approximate point only at this stage, some form of data smoothing technique needs to be applied. There are several kinds of data smoothing filter, including median filter, moving average, local regression, which all act as digital low-pass filters that filter out sudden fluctuations in data. A technique called Savitzky-Golay filter is chosen due to its capability of handling non-uniform-spaced data and the slowest runtime compared to other choices.

Developed by Abraham Savitzky and Marcel J.E. Golay, the Savitzky-

Golay filter is a digital filter that can smooth out abrupt deviations in data [12]. This is done by taking adjacent data points in an N sized window to find a polynomial of order $K, K < N$ to find the fitting line with the least squares. In the case of this paper, we are only interested in $K = 1$ since we only want to consider the linear information – whether the data is rising or falling. This filter effectively ensures one crossing point with the 50% reference line for every rising or falling edges and eliminate the effect of ripple noise in affecting the outcome of this algorithm. The required window size N is empirically determined to be 25% of array length per UI. The *Scipy* library supports this filter and can be implemented as follows:

```
1 N = int(samp_in_1UI/4) #window size
2 K = 1                  #fitting polynomial order
3 v_filtered = savgol_filter(voltage,N,K)
```

4.3.3 Looping Through Different Slicing Locations

The final consideration is the slicing location. Consider Figure 4.4, a transient voltage waveform sliced at two different timings for 1-UI eye formation. The red dash horizontal line is obtained by calculating the midpoint of $V_{\max}$ and $V_{\min}$, which we wish to find the intersections with the eye. The intersections are denoted in black boxes, and the orange star indicates the average of these intersections.

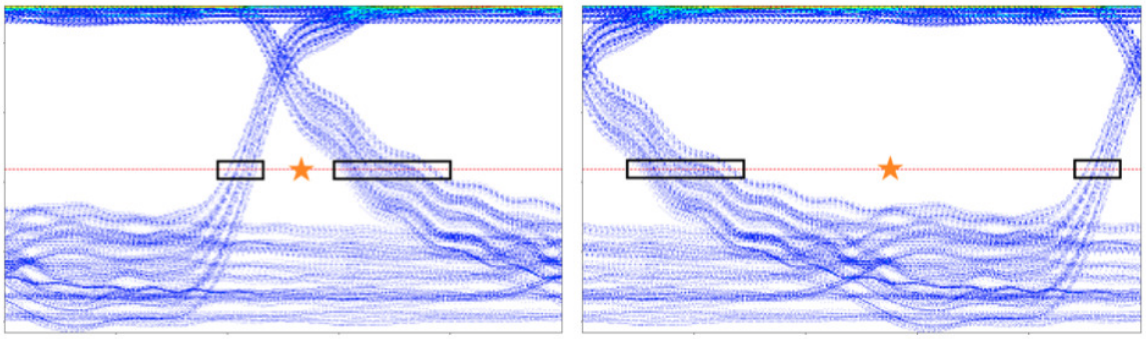


Figure 4.4: Intersections with a threshold (black box) and their average timing point (orange star) with different default eye diagram.

We can see from this example that if the rising and falling edges lie near the center of the eye, this method is valid, but not the other case where

they are separated. In order to overcome this caveat, the eye is circular shifted multiple times, with each time applying the approximate crossing point calculation. It is then subtracted by the initial shift to find the original (unshifted) crossing point and appended to an array. The mode of these stored crossing time values would be the final approximate crossing point.

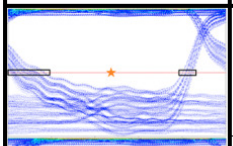
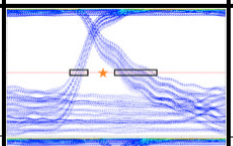
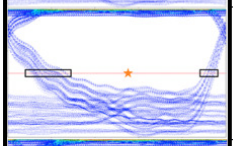
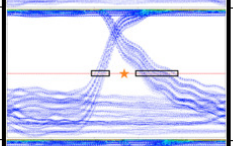
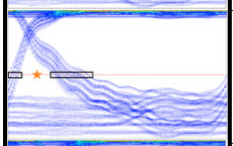
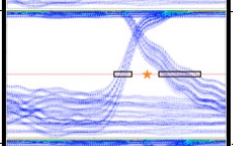
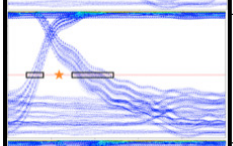
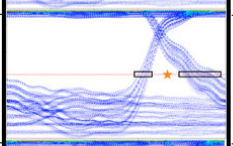
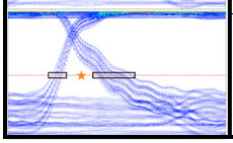
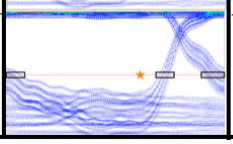
	Loop	t_shift	Xing	Final Xing		Loop	t_shift	Xing	Final Xing
	0	0 * UI	0.5UI	0.5UI		5	0.5UI	0.4UI	-0.1UI = 0.9UI
	1	0.1UI	0.6UI	0.5UI		6	0.6UI	0.5UI	-0.1UI = 0.9UI
	2	0.2UI	0.1UI	-0.1UI = 0.9UI		7	0.7UI	0.6UI	-0.1UI = 0.9UI
	3	0.3UI	0.2UI	-0.1UI = 0.9UI		8	0.7UI	0.6UI	-0.1UI = 0.9UI
	4	0.4UI	0.3UI	-0.1UI = 0.9UI		9	0.8UI	0.6UI	-0.2UI = 0.8UI

Figure 4.5: Default-sliced eye diagams with different offsets to find the approximate horizontal crossing point.

A code snippet of the loop would look something like this:

```
iterate = 11
final_approx_t = []
for i in range(iterate):
    time_shifted = time + UI/iterate*i
    #subtract UI/iterate*i to cancel out the offset applied above
    approx_t = approx_crossing(time_shifted, voltage, UI) - UI/iterate*i
    if(approx_t < 0):
        approx_t += UI
    final_approx_t.append(approx_t/UI)
final_approx_t = np.round(final_approx_t, 2)
approx_xing = mode(final_approx_t)
```

The function “approx_crossing” would look like:

```
def approx_crossing(t, v, UI):
    #vertical shift such that the red dashed line is at 0
    threshold = sum(voltage)/len(voltage)
    v_shift = v - threshold
    #find indexes of zero crossing
    xing_idx = np.where(np.sign(v_shift[:-1])!=np.sign(v_shift[1:]))[0]+1
    #get time values at those indexes
    resultarray = np.take(t, xing_idx)
    #if there are multiple crossings within 1UI, take the average
    approx_xing = sum(resultarray)/len(resultarray)
    return approx_xing
```

At the end of this block, the time array is shifted such that the calculated approximate center is placed at 0.5UI.

```
t = t - (approx_xing*UI - 0.5*UI)
```

4.4 Classifying Transitions

Now that the eye crossing point is "somewhat" centered, we can classify the eye into different transitions. Since a PAM-2 eye with 1-UI window encodes transitions between two bits, we would have $2^2 = 4$ transitions – 00, 01, 10, 11. In general, a PAM-N eye would encode 2^N transitions. Of these transitions, only 01 and 10 are of our interest. This is simply achieved by looking at each 1-UI slicing window and comparing first and last data point to determine a rising or falling edge. In fact, evaluating the sign of the subtraction between first and last data point of the voltage value is more accurate and more computationally efficient than obtaining average of the gradient.

In this process, we would require a threshold voltage level to differentiate between rising/falling eye waveforms that contribute to the crossing point and ones that do not. Consider Figure 4.6 where only the 11 and 00 transitions are plotted. The red lines indicate a "falling" edge if we only consider the difference between the first and last voltage value, and vice versa for the blue lines. However, none of these lines correspond to a real transition that contributes to the eye crossing. Hence, we require an additional condition that the lines must cross a certain threshold.

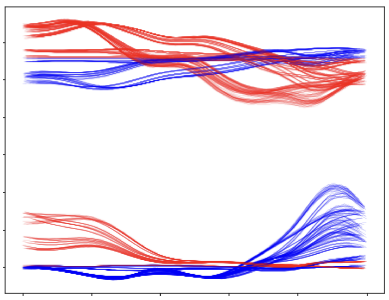


Figure 4.6: 1→1 and 0→0 transitions that are "rising" (blue) and "falling" (red) due to initial definition.

In order to determine the threshold, we need to compute an approximate "1" and "0" voltage level and calculate the average. This step is similar to Section 3.1 where an approximate center is calculated in that we only need approximate values of the two voltage levels to determine the threshold. This step considers eyes with overshoots and undershoots under presence of non-linearity, where there could be significantly higher amplitude of overshoots on the "1" level than the lower amplitude of undershoots on the "0" level, and therefore simply taking the midpoint between the maximum and the minimum voltage value would not be a holistic approach.

To find the voltage values for 1→1 and 0→0 transitions, the k-means algorithm introduced in chapter 2.7 is used. We simply let $k = 2$ and find the two centroids using this algorithm, which would each correspond to 1→1 and 0→0 transitions. This comes from the observation that identical bit transitions have smaller changes in amplitude and different bit transitions are more spread out since the voltage level needs to jump from one to another. The amplitude would be the distance that the algorithm attempts to minimize. With enough number of bits, the PRBS sequence generates enough number of data points to determine accurate approximate voltage values for logic 1 and 0.

Figure 4.7 illustrates four eye diagrams with different shapes, with its voltage histogram plotted on their right. The two red line in the histogram plot shows the computed centroid. It can be seen that regardless of how messy the eye is or whether non-linearity is present or not, the k-means algorithm is able to locate two centroids that best represent voltage values for 1→1 and 0→0 transitions. K-means clustering is well implemented in scikit-learn, an open-source machine learning library that is part of Scipy.

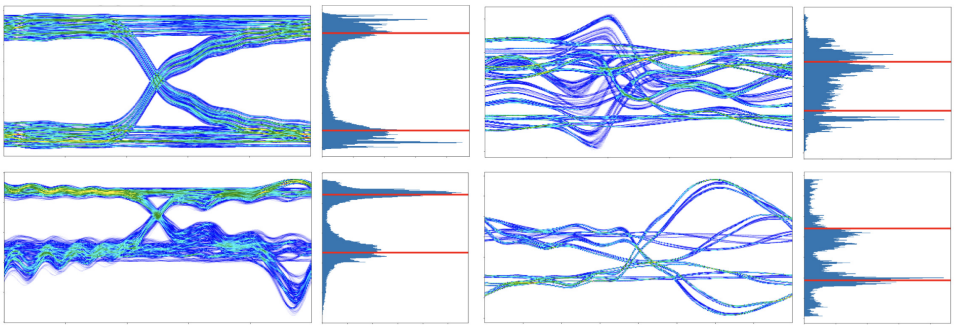


Figure 4.7: Visual illustration of applying k-means clustering. The red horizontal lines indicate the two calculated centroids, approximating voltage values for logic 1 and 0.

The following code achieves this:

```
from sklearn.cluster import KMeans
cluster = 2 #define number of clusters
model = KMeans(n_clusters=cluster, random_state=0, n_init='auto')
kmeans = model.fit(voltage.reshape(-1,1))
centroids = kmeans.cluster_centers_ #y
Vlevel = np.sort(centroids.flatten()) #low to high
threshold_1, threshold_0 = Vlevel[-1], Vlevel[0]
```

Now we can shift the focus to extracting the desired transitions – rising and falling edges that contribute to the crossing point. We first take the midpoint of the two obtained centroids, which will serve as a threshold value. Waveforms that only intersect with this threshold line will be different bit transitions, and whether the waveform is a rising or falling edge can be determined by looking at the start and end of the data within the 1-UI sliced window.

$$Threshold = \frac{\boldsymbol{\mu}_{\text{level1}} + \boldsymbol{\mu}_{\text{level0}}}{2} \tag{4.2}$$

The following code considers a 1-UI window waveform and extracts rising or falling edges if it crosses the threshold. The output is stored as a single nested array with each array element storing an array of data within 1-UI window if the condition is met.

```
def eye_risefall_only(t, v, UI, centroid_1, centroid_0):
    threshold = (centroid_1+centroid_0)/2
    n = 0
    result_t,result_v, append_t,append_v = [],[],[],[]
```

```
    for i in range(len(t)):
        if(n*UI <= t[i] <= (n+1)*UI):
            append_t.append(t[i] - (n*UI))
            append_v.append(v[i])
        else:
            #0->1 transition
            if(
            (append_v[0] < threshold and append_v[-1] > threshold)
            or
            (append_v[0] > threshold and append_v[-1] < threshold)):
                result_t.append(append_t)
                result_v.append(append_v)
            else:
                v_11and00.append(append_v)
            append_t,append_v = [],[]
            append_t.append(t[i] - (n+1)*UI)
            append_v.append(v[i])
            n = n+1
    return result_t, result_v, v_11and00
```

Note that the code doesn't yet distinguish between rising and falling edges, but rather first categorizes into different bit transitions (1→1, 0→0) and same bit transitions (1→0 and 0→1). With enough bits simulated with the PRBS generator, we can let the new crossing threshold to be the average of the same bit transition voltage array. From the different bit transitions, and with the new threshold line established, we can differentiate into rising or falling edges. The following code achieves this:

```
for i in range(len(result_t)):
    if(result_v[i][-1] - result_v[i][0] >= 0
       and result_v[i][0]<new_threshold
       and result_v[i][-1]>new_threshold):
        rising_t.append(append_t) #0->1 transitions
        rising_v.append(append_v)
    elif(result_v[i][-1] - result_v[i][0] <= 0
         and result_v[i][0]>new_threshold
         and result_v[i][-1]<new_threshold):
        falling_t.append(append_t) #1->0 transitions
        falling_v.append(append_v)
    else:
        continue
```

4.5 Finding Average Contour

Now the rising and falling waveforms are stored into two different arrays. A superficial approach at this step is to find every single crossing point between all of the rising and falling waveforms and take the average, but this entails two major flaws. First, ripples in waveforms create intersections between multiple rising (or falling) edges, which are not related to the crossing point at all. Second, given N UI of total simulation time, there would be approximately $\frac{N}{4}$ waveforms for each of the rising and falling edges, meaning that there will be at least $\frac{N^2}{16}$ crossing point calculations that need to be done, which not only is computationally expensive but also increases the runtime exponentially. To resolve such issues, the proposed algorithm computes one weighted mean waveform for each of the rising and falling edges, and the only one crossing point is calculated. In the case of multiple crossing points scenario due to multi-modal jitter PDF, it outputs the mean of the crossing points.

In order to calculate the average contour, the time domain needs to be aligned for every UI window, meaning that the waveform data should be stored in uniform time step. One way to do it is to use the conventional histogram approach by categorizing data into fixed-width bins such that the average point for every single vertical bin can be calculated. The transient simulation in many EDA tools allows users to define a maximum, minimum, or fixed time step. For fixed time step, smaller time steps have higher precision in the expense of longer simulation time, and vice versa for larger time steps. A most common approach is to set a lower or upper bound for time step size such that runtime can be minimized while maintaining a high precision. When a fixed time step is used, all data points are aligned vertically, then the average contour can be easily calculated. On the other hand, for the case of non-uniform time step, it requires an extra step for calculation.

When dealing with non-uniform time step, the conventional histogram must be used to bin the unequally-spaced data into equally-sized vertical bins. However, the problem occurs when different number of data falls into different bins. For example, it is possible for non-uniform data to have 3 data points in one vertical bin and only 1 data point in a different bin. When the average is calculated in this way, the average contour results in erratic spikes that hinders us from finding the final crossing point. Figure 4.8 depicts such

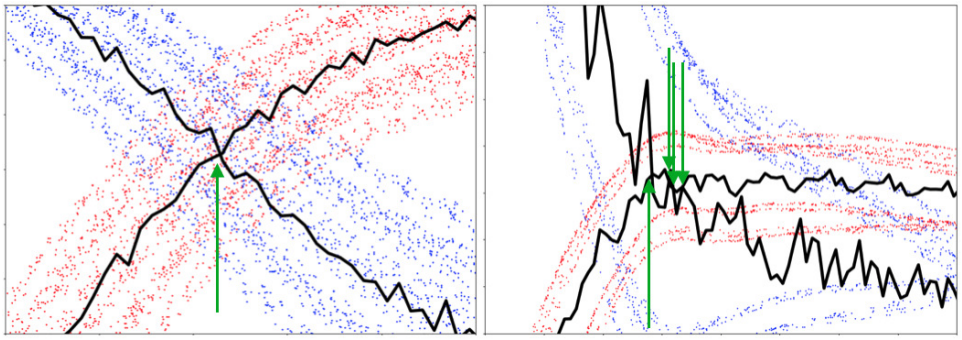


Figure 4.8: Average contour (black) calculated with non-uniform time step edges. Green arrows point to the intersection between the two black curves.

problem, where the crossing between rising and falling average contours is marked with green arrows. The left figure depicts a small distortion due to the spikes, and the right figure depicts a relatively bigger distortion where the erratic spikes make unwanted intersection between the rising and the falling edge.

This can be mitigated by re-interpolating the waveform data to have uniform spacing. Since the final purpose of the algorithm is to display the eye diagram heat map, of which its resolution can be user-defined by the vertical and horizontal bin number, we can use the horizontal bin width as the new uniform time step to be re-interpolated. In most cases where the horizontal bin width is larger than the largest step size in a transient simulation, size of arrays can be reduced and hence make the calculations less computationally expensive. Using such method, Figure 4.8 can be refined to Figure 4.9 such that now rising and falling edges are evenly spaced, hence resulting in a smooth average contour.

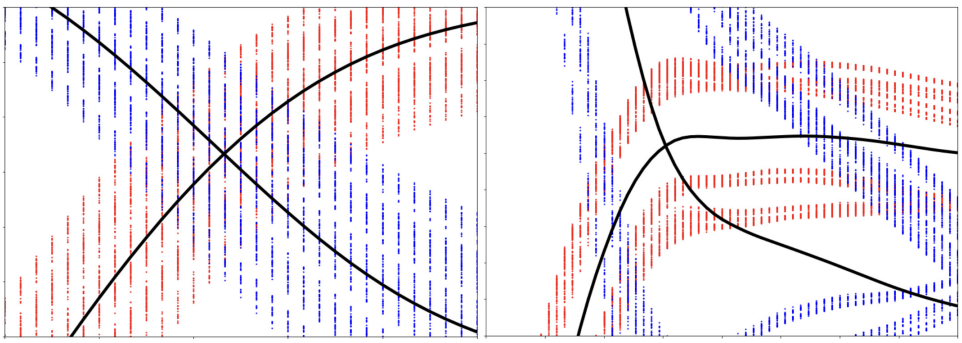


Figure 4.9: Average contour (black) calculated with uniform time step edges.

The *Scipy.interpolate* library's *interp1d* implements this using the following logic:

1. Define an equally spaced time array of size *binx*, where *binx* is the number of bins defined for eye plotting.
2. For each element t_i and v_i in the original time and voltage array, do:
   (a) Find at which equally spaced time interval $[t_{\text{low}}, t_{\text{high}}]$ t_i belongs to.
   (b) Linearly interpolate using (t_i, v_i) and $(t_{\text{i+1}}, v_{\text{i+1}})$ to find the voltage v_{new} value at $t = \frac{t_{\text{low}}+t_{\text{high}}}{2}$
3. Remove duplicate voltage data in the same time interval

The following code is implemented for interpolation:

```
from scipy.interpolate import interp1d
def interpolation_1d(t,v,UI,eyebin_x):
    evenspaced_v = []
    newt = np.linspace(0, UI, eyebin_x+1)
    for i in range(len(t)):
        f = interp1d(t[i], v[i], kind='linear', fill_value="extrapolate")
        newv = f(newt)
        evenspaced_v.append(newv)
    #find average contour
    avg_contour = np.array(evenspaced_v).sum(axis=0) / len(evenspaced_v)
    return newt, avg_contour, evenspaced_v
```

4.6 Computing Final Crossing Point

Now only two curves, the average contour for rising and falling edges, remain. The intersection would be the final crossing point that we have been looking for. Given two lines L_1, defined by two points $(x_1, y_1), (x_2, y_2)$, and L_2, defined by two points $(x_3, y_3), (x_4, y_4)$, we can compute the crossing point P_x, P_y, if L_1 and L_2 intersect, by [13]:

$$P_x = \frac{\begin{vmatrix} \begin{vmatrix} x_1 & y_1 \\ x_2 & y_2 \end{vmatrix} & \begin{vmatrix} x_1 & 1 \\ x_2 & 1 \end{vmatrix} \\ \begin{vmatrix} x_3 & y_3 \\ x_4 & y_4 \end{vmatrix} & \begin{vmatrix} x_3 & 1 \\ x_4 & 1 \end{vmatrix} \end{vmatrix}}{\begin{vmatrix} \begin{vmatrix} x_1 & 1 \\ x_2 & 1 \end{vmatrix} & \begin{vmatrix} y_1 & 1 \\ y_2 & 1 \end{vmatrix} \\ \begin{vmatrix} x_3 & 1 \\ x_4 & 1 \end{vmatrix} & \begin{vmatrix} y_3 & 1 \\ y_4 & 1 \end{vmatrix} \end{vmatrix}}, P_y = \frac{\begin{vmatrix} \begin{vmatrix} x_1 & y_1 \\ x_2 & y_2 \end{vmatrix} & \begin{vmatrix} y_1 & 1 \\ y_2 & 1 \end{vmatrix} \\ \begin{vmatrix} x_3 & y_3 \\ x_4 & y_4 \end{vmatrix} & \begin{vmatrix} y_3 & 1 \\ y_4 & 1 \end{vmatrix} \end{vmatrix}}{\begin{vmatrix} \begin{vmatrix} x_1 & 1 \\ x_2 & 1 \end{vmatrix} & \begin{vmatrix} y_1 & 1 \\ y_2 & 1 \end{vmatrix} \\ \begin{vmatrix} x_3 & 1 \\ x_4 & 1 \end{vmatrix} & \begin{vmatrix} y_3 & 1 \\ y_4 & 1 \end{vmatrix} \end{vmatrix}} \tag{4.3}$$

Expanding the determinants to obtain the equivalent expression:

$$P_x = \frac{(x_1y_2 - y_1x_2)(x_3 - x_4) - (x_1 - x_2)(x_3y_4 - y_3x_4)}{(x_1 - x_2)(y_3 - y_4) - (y_1 - y_2)(x_3 - x_4)} \tag{4.4}$$

$$P_y = \frac{(x_1y_2 - y_1x_2)(y_3 - y_4) - (y_1 - y_2)(x_3y_4 - y_3x_4)}{(x_1 - x_2)(y_3 - y_4) - (y_1 - y_2)(x_3 - x_4)} \tag{4.5}$$

Since our average contour array is evenly spaced, $x_1 = x_3$ and $x_2 = x_4$. Then Equations 4.4 and 4.5 simplifies to:

$$\Delta x = x_1 - x_2 = x_3 - x_4 \tag{4.6}$$

$$P_x = \frac{\Delta x(x_1y_2 - y_1x_2 - x_3y_4 + y_3x_4)}{\Delta x(y_3 - y_4 - y_1 + y_2)} = \frac{x_1y_2 - y_1x_2 - x_3y_4 + y_3x_4}{y_3 - y_4 - y_1 + y_2} \tag{4.7}$$

$$P_y = \frac{(x_1y_2 - y_1x_2)(y_3 - y_4) - (y_1 - y_2)(x_3y_4 - y_3x_4)}{\Delta x(y_3 - y_4 - y_1 + y_2)} \tag{4.8}$$

It is important to note that Equations 4.4 and 4.5 can be used only once while looping through every adjacent points for each of the rising and falling data since there should be only one intersection. Furthermore, we can reduce computational complexity by realizing that since the time arrays of the rising and falling average contours are already uniformly-spaced, hence Δx can be easily obtained. Therefore, it is sufficient to only compare whether or not an overlap exists between y intervals of the two adjacent data points. The calculations using Equations 4.4 and 4.5 are only triggered when an overlap exist between intervals $[y_1, y_2]$ and $[y_3, y_4]$. Following code describes the overall process:

```
def intersection(x1, y1, x2, y2):
    final_x, final_y = [], []
    for i in range(len(x1)-1):
        #account for rising and falling edge cases
        y1_min = min([y1[i], y1[i+1]])
        y1_max = max([y1[i], y1[i+1]])
        y2_min = min([y2[i], y2[i+1]])
        y2_max = max([y2[i], y2[i+1]])
        #if intervals overlap
        if((y2_min <= y1[i]   <= y2_max) or
           (y2_min <= y1[i+1] <= y2_max) or
           (y1_min <= y2[i]   <= y1_max) or
           (y1_min <= y2[i+1] <= y1_max)
           ):
            t1, t3 = x1[i], x2[i]
            t2, t4 = x1[i+1], x2[i+1]
            v1, v2 = y1[i], y1[i+1]
            v3, v4 = y2[i], y2[i+1]
            A = t1*v2 - v1*t2
            B = t3*v4 - v3*t4
            denom = (t1-t2)*(v3-v4)-(v1-v2)*(t3-t4)
            #calculate crossing points
            x_xing = (A*(t3-t4)-(t1-t2)*B)/denom
            y_xing = (A*(v3-v4)-(v1-v2)*B)/denom
        else: continue
    return final_x, final_y
```

4.7 Output and Plotting Shifted Eye

The output is a single pair of crossing point t_{xing} and v_{xing}. Noting that an approximate center was used to first shift the eye to classify waveforms into different transition categories, we need to cancel out the initial offset (horizontal only) as well in order to compute the final crossing point. The code for the crossing point algorithm is concluded by outputting $t_{\text{xing, final}}$ and $v_{\text{xing, final}}$.

```
return final_x+(approx_xing - 0.5*UI), final_y
```

The algorithm concludes at this point. $t_{\text{xing, final}}$ would tell us the amount of shift required in order to place the crossing point at 0.5UI location, and

$v_{\text{xing, final}}$ would tell us the vertical crossing value, which would be the reference point for horizontal eye parameter calculations introduced in chapter 2.5.1. After the shift is applied, we can simply apply the slicing introduced in chapter 4.2, and expand it into 2UI window to complete the eye diagram plotting.

Though irrelevant to the eye crossing point itself, we also wish to return the rising and falling edges separately, for two important eye parameters – Rise Time (t_{rise}) and Fall Time (t_{fall}) as defined in Equation 2.14 and 2.15. We can simply return the two outputs from section 4.4, which would be time and voltage arrays for rising and falling edges. This can greatly simplify eye parameter calculations at later stages.

CHAPTER 5

VERIFICATION

The validity of the algorithm is presented in two main ways. First, the eye crossing detection capability is tested. Second, eye parameters are calculated. Both of them are directly compared with EDA tools including ADS, Ansys Circuit, and Cadence Virtuoso. However, direct comparison is only possible for normal eyes (with crossing percentage = 50%). For abnormal eyes (with high ISI/jitter/different crossing percentage), only the obtained result is presented. However, calculations from the EDA tools are also presented to provide points to their inaccuracy.

5.1 Test Cases

There is an infinite set of combinations that can result in infinitely many eye shapes, hence it is impossible to fully assess the algorithm, but we can generate distinguishable cases with different eye diagram shapes to assess it holistically.

To illustrate the validity, different waveforms with distinct eye shapes are used to generate the shifted eye diagram. This is achieved by using different kinds of channel to simulate and form the eye diagram. Ideal channels are chosen to generate normal eye diagrams, and non-ideal channels are chosen to generate eye diagrams with high ISI, jitter, or any distortions that make the eye diagrams "messy" in order to account for extreme cases.

The following is a list of various eye shape scenarios, accounting for numerous kinds of distortions such as ISI and jitter caused by various reasons such as input mismatch, channel loss, and non-linearity. Note that the shapes of eye diagrams are more important than how the distortion is introduced in the channel.

1. Overshoots / Undershoots in 1 Level / 0 Level

2. Single-modal / Multi-modal Jitter PDF

3. Different Crossing Percentage (50%, $< 50\%$, $> 50\%$)

4. Fast / Slow Rise and Fall Time

Though many more examples are tested, 21 test waveforms are listed in this paper. some of them serve as a verification that is comparable with EDA tools – meaning that EDA tools are also able to deliver accurate crossing point detection. Others serve as a verification that some or all EDA tools cannot acheive accurate crossing point detection, thereby illustrating the comprehensiveness of the proposed algorithm.

Furthermore, since we initially assume slicing at integer UI locations, it is needed account for different default slicing locations to prove that the algorithm can shift the eye correctly regardless of its default slicing location. This can be achieved by applying some offset to the waveform before starting the algorithm, which would effectively generate default (unshifted) eye diagrams as shown previously in Figure 1.2. In this section, the test bench is set up such that the offset is equal to $\frac{1}{10}$ of UI, meaning that each waveform generates 10 test cases with 0.1UI, 0.2UI, ... amount of time delays to test the algorithm. Figures below list all the test cases before shifting, with each case corresponding to 10 cases with different offsets, and shapes labeled according to the list above. The eye diagrams are formed with default slicing – at integer multiples of UI. With 21 test waveforms, it would amount to 210 test cases.

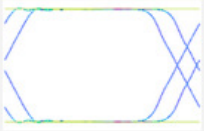
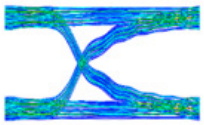
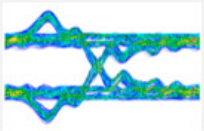
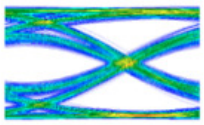
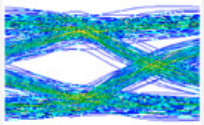
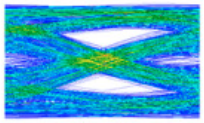
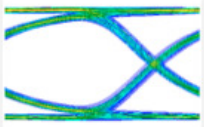
Test Case #	Eye Density Plot (before shifting)	Eye Shape Description					
		Amplitude Variations	Jitter / Jitter PDF	Crossing Percentage	Rise Time	Fall Time	Degree of ISI
1		None	Small Jitter / Single-modal	50%	Inter-mediate	Inter-mediate	None
2		None	Small Jitter / Multi-modal (2)	50%	Inter-mediate	Inter-mediate	None
3		None	Inter-mediate Jitter / Single-modal	50%	Inter-mediate	Inter-mediate	None
4		1 Level Overshoot, 0 Level Undershoot	Inter-mediate Jitter / Multi-modal (2)	50%	Inter-mediate	Inter-mediate	Small
5		None	Inter-mediate Jitter / Single-modal	50%	Slow	Slow	Small
6		None	High Jitter / Single-modal	50%	Slow	Slow	Inter-mediate
7		None	High Jitter / Single-modal	50%	Slow	Slow	Inter-mediate
8		None	Inter-mediate Jitter / Single-modal	50%	Inter-mediate	Inter-mediate	Small

Figure 5.1: Test Case Table

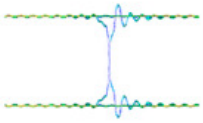
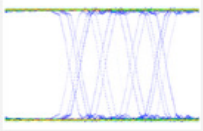
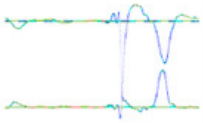
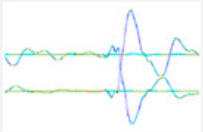
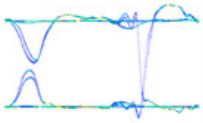
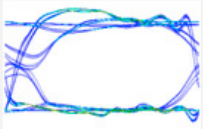
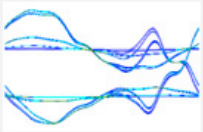
Test Case #	Eye Density Plot (before shifting)	Eye Shape Description					
		Amplitude Variations	Jitter / Jitter PDF	Crossing Percentage	Rise Time	Fall Time	Degree of ISI
9		1 Level Overshoot, 0 Level Undershoot	Small Jitter / Single-modal	50%	Fast	Fast	None
10		None	Large Periodic Jitter / Multi-modal	50%	Fast	Fast	Small
11		1 Level Undershoot, 0 Level Overshoot	Small Jitter / Single-modal	< 50%	Fast	Fast	High
12		1 & 0 Level Undershoot & Overshoot,	Small Jitter / Single-modal	50%	Fast	Fast	High
13		1 Level Undershoot, 0 Level Overshoot	Small Jitter / Single-modal	< 50%	Fast	Fast	High
14		1 Level Undershoot, 0 Level Overshoot	Small Jitter / Single-modal	< 50%	Fast	Fast	High
15		1 Level Overshoot, 0 Level Undershoot	Small Jitter / Single-modal	50%	Inter-mediate	Inter-mediate	High
16		1 & 0 Level Undershoot & Overshoot,	Small Jitter / Single-modal	> 50%	Inter-mediate	Inter-mediate	High

Figure 5.2: Test Case Table (continued)

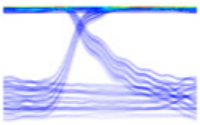
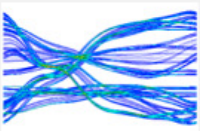
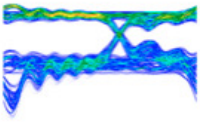
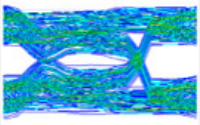
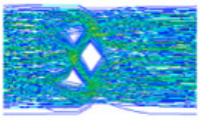
Test Case #	Eye Density Plot (before shifting)	Eye Shape Description					
		Amplitude Variations	Jitter / Jitter PDF	Crossing Percentage	Rise Time	Fall Time	Degree of ISI
17		None	Inter-mediate Jitter / Single-modal	> 50%	Inter-mediate	Inter-mediate	Inter-mediate
18		1 & 0 Level Undershoot & Overshoot,	Inter-mediate Jitter / Single-modal	50%	Slow	Slow	High
19		0 Level Undershoot	Inter-mediate Jitter / Single-modal	> 50%	Inter-mediate	Inter-mediate	Inter-mediate
20		None	High Jitter / Single-modal	50%	Fast	Fast	Inter-mediate
21		None	High Jitter / Single-modal	50%	Fast	Fast	Inter-mediate

Figure 5.3: Test Case Table (continued).

5.2 Verification of the Eye Crossing Point Detection

With the test cases established in section 5.1, eye crossing point detection algorithm is applied and shifted such that the eye crossing point is at 50% of the 1-UI window of the eye diagram. The crossing point is marked with a red cross in the table below.

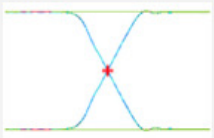
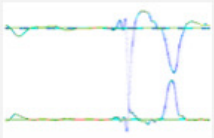
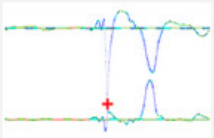
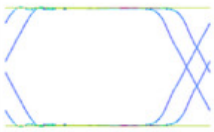
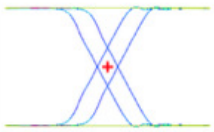
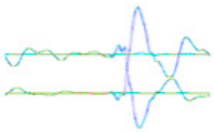
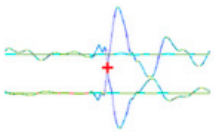
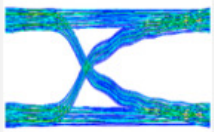
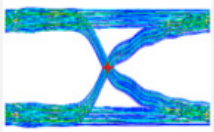
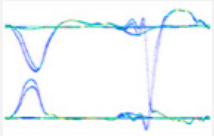
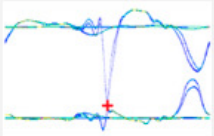
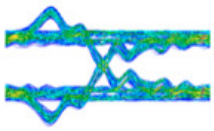
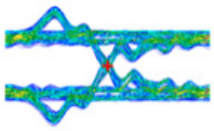
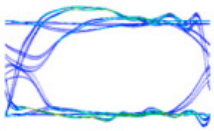
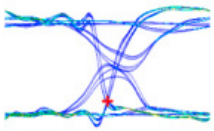
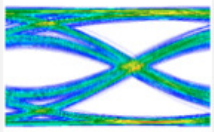
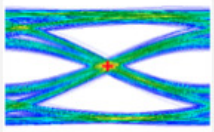
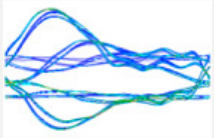
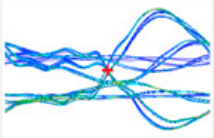
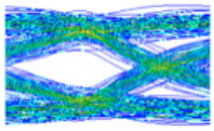
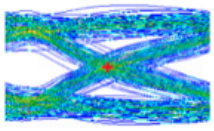
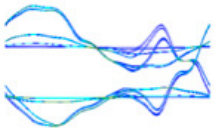
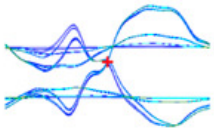
Test Case #	Default Eye	Shifted Eye	Test Case #	Default Eye	Shifted Eye
1			11		
2			12		
3			13		
4			14		
5			15		
6			16		

Figure 5.4: Eye Crossing Point Detection Verification Table.

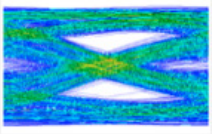
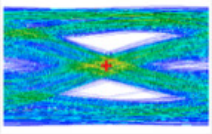
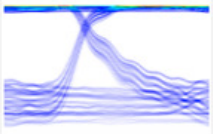
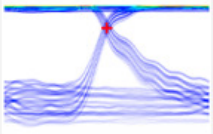
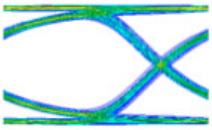
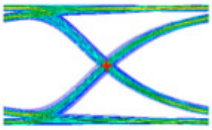
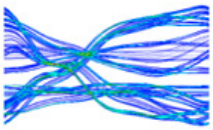
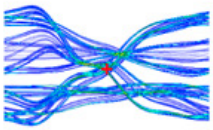
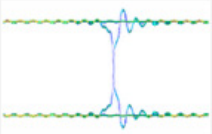
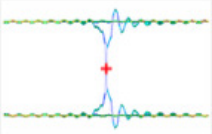
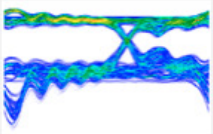
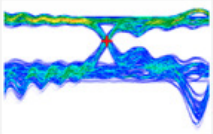
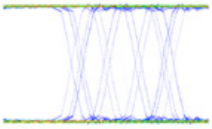
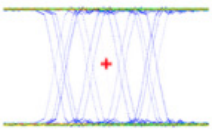
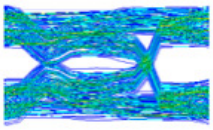
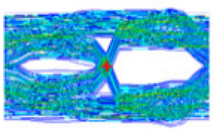
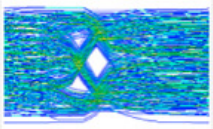
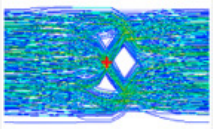
Test Case #	Default Eye	Shifted Eye	Test Case #	Default Eye	Shifted Eye
7			17		
8			18		
9			19		
10			20		
			21		

Figure 5.5: Eye Crossing Point Detection Verification Table (continued).

5.3 Verification of Eye Parameters

As eye parameters have their established mathematical definitions, we can directly compare the calculated eye parameters from the proposed algorithm with those from current EDA tools. It is achieved by inputting identical discrete data waveform into different EDA tools. This can be done by extracting the waveform that we wish to generate the eye diagram from ADS and set it as a piecewise linear voltage waveform in other tools, which then they can generate the eye diagram and eye parameters. This is illustrated in Figure 5.6.

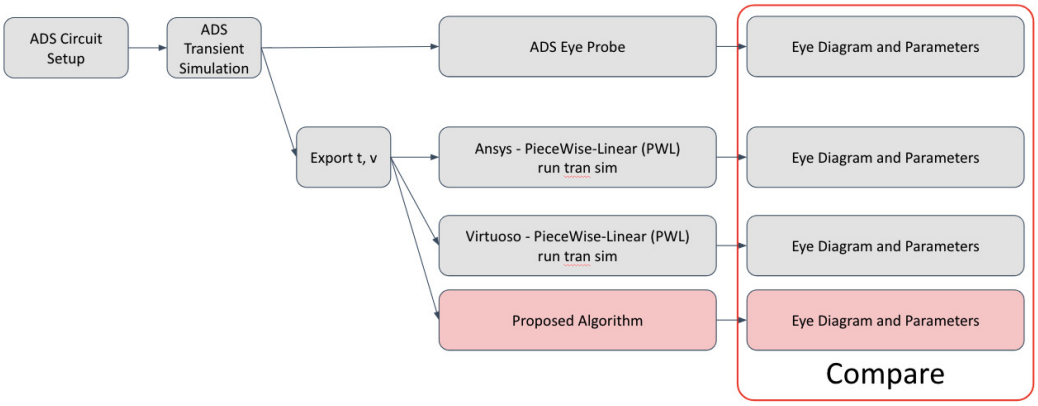


Figure 5.6: Eye Parameter Comparison Methodology.

Table 5.1 summarizes the eye crossing point detection accuracy for the proposed algorithm and the industry EDA tools, including ADS, Ansys, and Virtuoso. Ones labelled with a red $\times$ are the cases that failed to find the accurate crossing point. These ones are marked in red in Table 5.2, 5.3, 5.4, 5.5, 5.6 as well.

Table 5.1: Comparison table for correct eye shifting

Correct Eye Shifting				
test case	paper	ADS	Ansys	Virtuoso
1	◯	◯	×	◯
2	◯	◯	×	×
3	◯	◯	◯	◯
4	◯	◯	×	◯
5	◯	◯	◯	◯
6	◯	◯	◯	◯
7	◯	◯	◯	◯
8	◯	◯	◯	◯
9	◯	◯	◯	◯
10	◯	×	×	◯
11	◯	×	◯	◯
12	◯	×	◯	×
13	◯	×	×	×
14	◯	◯	×	×
15	◯	◯	×	×
16	◯	◯	×	×
17	◯	◯	×	×
18	◯	◯	×	×
19	◯	◯	×	◯
20	◯	◯	×	×
21	◯	◯	×	×

It can be seen from Table 5.2 that the numbers match very well. While Virtuoso was able to provide values for all cases, ADS and Ansys had cases where they were not able to calculate Level1 and Level0, therefore outputting 0s as an error. Some failed cases (in red) show a large discrepancy, which correspond to cases where the program was taking the wrong segment to calculate the error, implying its imperfection. For example, checking the boxed range of data Virtuoso selected for Level1 and Level0 calculation, we can see from the left of Figure 5.7 (test case #18) that the box was misplaced horizontally, and the right of it (test case #19) that the vertical threshold of the two boxes was interpreted incorrectly.

Table 5.2: Comparison table for Level 1 and Level 0

Level 1 (V) / Level 0 (V)				
test case	paper	ADS	Ansys	Virtuoso
1	0.333 / 0.001	0.332 / 0.001	0.332 / 0.001	0.332 / 0.002
2	0.333 / 0.001	0.331 / 0.001	0.331 / 0.001	0.332 / 0.002
3	0.349 / 0.039	0.344 / 0.043	0.341 / 0.046	0.337 / 0.049
4	0.344 / 0.044	0.362 / 0.024	0.373 / 0.118	0.327 / 0.061
5	0.213 / -0.215	0.212 / -0.214	0.210 / -0.202	0.186 / -0.187
6	0.861 / 0.151	0.863 / 0.148	0.858 / 0.152	0.818 / 0.193
7	0.189 / -0.174	0.188 / -0.173	0.185 / -0.159	0.160 / -0.148
8	0.290 / -0.309	0.288 / -0.306	0.286 / -0.301	0.264 / -0.281
9	0.331 / 0.002	0.331 / 0.002	0.331 / 0.002	0.333 / 0.538
10	0.331 / 0.002	0 / 0	0.331 / 0.002	0.330 / 0.004
11	1.379 / 0.008	0 / 0	1.393 / -0.003	1.420 / -0.003
12	1.956 / 0.028	0 / 0	1.974 / -0.011	1.931 / 0.002
13	1.136 / 0.016	0 / 0	1.429 / -0.008	1.364 / 0.019
14	1.497 / -0.027	1.4810 / -0.026	1.494 / -0.031	1.534 / -0.021
15	2.536 / -0.482	2.546 / -0.482	2.906 / -0.708	3.070 / -0.834
16	2.014 / -0.184	2.048 / -0.206	2.738 / -0.617	3.025 / -0.537
17	0.223 / 0.045	0.223 / 0.044	0.222 / 0.042	0.224 / 0.054
18	3.159 / -0.342	3.249 / -0.161	0 / 0	1.990 / 0.672
19	0.245 / 0.0001	0.214 / -0.019	0 / 0	0.169 / 0.009
20	0.746 / 0.242	0.828 / 0.159	0.797 / 0.189	0.817 / 0.170
21	0.725 / 0.265	0.711 / 0.246	0 / 0	0.648 / 0.331

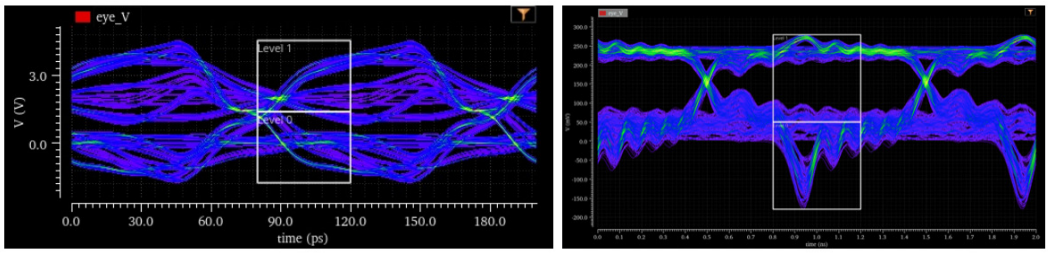


Figure 5.7: Eye diagram generated by Cadence Virtuoso, test case 18 (left) and 19 (right).

Table 5.3: Comparison table for Eye Width (EW) and Eye Height (EH)

EW (ns) / EH (V)				
test case	paper	ADS	Ansys	Virtuoso
1	0.949 / 0.328	0.500 / 0.323	0.493 / 0.321	0.987 / 0.323
2	0.693 / 0.327	0.400 / 0.317	1.177 / 0.314	0.794 / 0.218
3	0.895 / 0.192	0.045 / 0.157	0.414 / 0.126	0.924 / 0.152
4	0.899 / 0.178	0.453 / 0.127	-0.014 / -0.034	0.932 / 0.085
5	0.069 / 0.056	0.081 / 0.169	0.073 / 0.025	0 / -0.027
6	0.151 / 0.128	0.211 / 0.198	0.159 / 0.158	0 / -0.027
7	0.002 / -0.120	0.015 / 0	0.009 / -0.157	0 / -0.178
8	0.042 / 0.400	0.046 / 0.435	0.044 / 0.396	0.043 / 0.282
9	0.996 / 0.316	0 / 0.014	1 / 0.315	1 / 0.323
10	-0.071 / 0.320	0 / 0	-0.102 / 0.318	0.198 / 0.302
11	0.444 / 1.186	0 / 0	0.448 / 1.311	0.580 / 1.175
12	0.052 / 0.934	0 / 0	0.388 / 1.412	0.263 / 1.049
13	-0.069 / -0.068	0 / 0	-0.185 / 1.296	0.100 / 1.061
14	0.065 / 1.128	0.162 / 1.283	0.122 / 1.155	0.111 / 1.137
15	-0.060 / -0.070	0 / 1.368	0.133 / -0.294	0.019 / -1.069
16	0.004 / 0.732	0.120 / 1.312	0.038 / 0.518	0.085 / 0.522
17	0.898 / 0.104	0.945 / 0.123	0.695 / 0.099	0.916 / 0.087
18	-0.062 / -1.792	0.015 / 0	0 / 0	0 / -2.56
19	0.928 / 0.014	0 / 0	0 / 0	0.924 / -0.213
20	-0.108 / -0.218	0.064 / 0.206	-0.033 / -0.050	-0.013 / 0.061
21	-0.071 / -0.361	0 / 0	0 / 0	0 / -0.559

We have a good match between the proposed algorithm and other EDA tools. It can be seen that for all three EDA tools, there were cases that they could not compute the EW and EH at all. Other discrepancies could be from multiple causes :

1. While identical waveform is imported into the programs to generate the eye diagram, the transient simulator might have dealt the waveforms differently, resulting in slightly different outputs.

2. The programs use their own binning method to reduce computational complexity, and different bin sizes can affect the resulting output precision

3. The programs seem to use distribution fitting methods, such as a Gaussian distribution, to generate a curve fit to find the σ required for EW and EH calculation

4. Just like the right Figure of 5.7, even if the shifting is done correcrtly, the program might not determine the threshold between Level1 and

Table 5.4: Comparison table for SNR and Eye Amplitude

SNR / EyeAmp (V)				
test case	paper	ADS	Ansys	Virtuoso
1	48.9 / 0.332	125 / 0.331	107 / 0.330	146 / 0.330
2	45.8 / 0.332	77 / 0.330	63.2 / 0.331	9.62 / 0.317
3	17.9 / 0.310	5.85 / 0.302	5.25 / 0.295	6.33 / 0.289
4	17.3 / 0.300	3.03 / 0.338	2.74 / 0.362	4.41 / 0.266
5	10.8 / 0.427	3.40 / 0.425	3.22 / 0.413	2.80 / 0.372
6	11.3 / 0.710	3.83 / 0.715	3.86 / 0.707	8.94 / 0.720
7	7.06 / 0.362	2.22 / 0.360	2.06 / 0.343	1.90 / 0.309
8	19.1 / 0.599	8.74 / 0.593	9.23 / 0.587	6.23 / 0.545
9	37.5 / 0.330	81.8 / 0.329	71.0 / 0.329	102 / 0.332
10	40.3 / 0.329	0 / 0	91.9 / 0.329	39.6 / 0.326
11	26.9 / 1.371	0 / 0	49.2 / 1.396	17.2 / 1.423
12	15.3 / 1.928	0 / 0	10.4 / 1.985	6.58 / 1.930
13	9.03 / 1.121	0 / 0	30.6 / 1.437	14.2 / 1.345
14	21.2 / 1.524	13.9 / 1.504	12.4 / 1.525	11.2 / 1.555
15	9.34 / 3.018	2.93 / 3.028	2.78 / 3.614	2.36 / 3.904
16	13.1 / 2.198	4.51 / 2.254	3.55 / 3.355	3.52 / 3.562
17	17.1 / 0.178	7.16 / 0.179	6.73 / 0.180	6.18 / 0.170
18	5.95 / 3.502	2.12 / 3.410	0 / 0	1.02 / 1.318
19	10.1 / 0.245	1.87 / 0.233	0 / 0	1.285 / 0.159
20	6.41 / 0.503	3.41 / 0.669	2.77 / 0.608	3.32 / 0.647
21	4.51 / 0.460	1.66 / 0.465	0 / 0	1.097 / 0.318

Level0 data correctly, therefore resulting in an incorrect value for Eye Height (EH).

From Table 5.4, we see a very good match for Eye Amplitude (EyeAmp). However, we see large discrepancies in SNR. SNR is calculated from Equation 2.20 and the denominator σs have a large impact on the resulting output. As mentioned in the previous section, programs can use curve fitting method to assume a certain distribution to calculate the σs, while the proposed algorithm simply calculates the variance using its discrete form. We can see a large difference for the cases with inaccurate eye crossing point detection (marked in red). This is especially prevalent in cases where curves are more spread out, having high variance in the 40% 60% of the 2-UI window, therefore hugely impacting the SNR calculation when the shifting is not done correctly.

Table 5.5: Comparison table for Jitter (peak-to-peak and RMS)

$\text{Jitter}_{p-p}(ns)$ / $Jitter_{RMS}(ns)$				
test case	paper	ADS	Ansys	Virtuoso
1	0.030 / 0.009	0 / 0	0.005 / 0.001	not provided
2	0.454 / 0.051	0.100 / 0.050	0.168 / 0.054	not provided
3	0.096 / 0.018	0.053 / 0.014	0.058 / 0.014	not provided
4	0.066 / 0.017	0.0475 / 0.0165	0.369 / 0.086	not provided
5	0.032 / 0.005	0.020 / 0.005	0.025 / 0.005	not provided
6	0.187 / 0.030	0.122 / 0.030	0.165 / 0.029	not provided
7	0.100 / 0.016	0.096 / 0.017	0.100 / 0.015	not provided
8	0.008 / 0.001	0.004 / 0.001	0.005 / 0.001	not provided
9	0.002 / 0.001	0.005 / 0.0025	0.002 / 0.000	not provided
10	0.513 / 0.179	0 / 0	0.504 / 0.184	not provided
11	0.238 / 0.093	0 / 0	0.232 / 0.092	not provided
12	0.457 / 0.158	0 / 0	0.228 / 0.102	not provided
13	0.240 / 0.095	0 / 0	0.237 / 0.114	not provided
14	0.077 / 0.023	0.038 / 0.012	0.048 / 0.013	not provided
15	0.144 / 0.043	0.187 / 0.068	0.042 / 0.011	not provided
16	0.090 / 0.033	0.080 / 0.032	0.080 / 0.027	not provided
17	0.090 / 0.017	0.055 / 0.015	0.178 / 0.051	not provided
18	0.100 / 0.027	0.085 / 0.025	0 / 0	not provided
19	0.067 / 0.012	0 / 0.265	0 / 0	not provided
20	0.143 / 0.042	0.143 / 0.034	0.081 / 0.029	not provided
21	0.091 / 0.027	0.091 / 0.027	0 / 0	not provided

Jitter calculation is done by chossing values in a thin strip near the crossing point, but how thin it should be is not only undefined mathematically, but also different in different EDA tools. For this paper, $\pm 5\%$ values of the Eye Amplitude are chosen as the boundaries of the thin strip. The difference caused by this can increase when there is high jitter, or when there are ripples that causes the algorithm to choose different range of data with different strip thickness. Considering these uncertainties, we see a good match between the algorithm and the EDA tools. It is also important to note that Cadence Virtuoso does not provide $Jitter_{p-p}$ and $Jitter_{RMS}$ as part of its eye parameters.

Table 5.6: Comparison table for rise time and fall time

Risetime (ns) / Falltime (ns)				
test case	paper	ADS	Ansys	Virtuoso
1	0.180 / 0.180	0.18 / 0.18	0.181 / 0.181	0.182 / 0.182
2	0.180 / 0.180	0.18 / 0.18	0.181 / 0.181	0.175 / 0.175
3	0.268 / 0.264	0.302 / 0.280	0.233 / 0.232	0.219 / 0.218
4	0.245 / 0.251	0.304 / 0.302	0.490 / 0.103	0.085 / 0.074
5	0.068 / 0.067	0.070 / 0.068	0.064 / 0.062	0.052 / 0.051
6	0.186 / 0.183	0.197 / 0.194	0.189 / 0.187	0.157 / 0.151
7	0.063 / 0.059	0.063 / 0.057	0.068 / 0.063	0.043 / 0.042
8	0.020 / 0.020	0.002 / 0.002	0.002 / 0.002	0.018 / 0.018
9	0.025 / 0.025	0.003 / 0.003	0.025 / 0.025	0.026 / 0.026
10	0.062 / 0.066	0 / 0	0.070 / 0.069	0.063 / 0.063
11	0.164 / 0.184	0 / 0	0.146 / 0.150	0.040 / 0.029
12	0.297 / 0.170	0 / 0	0.281 / 0.160	0.153 / -0.632
13	0.184 / 0.169	0 / 0	0.151 / 0.170	0.039 / 0.025
14	0.019 / 0.012	0.041 / 0.048	0.019 / 0.010	0.039 / 0.024
15	0.052 / 0.077	0.085 / 0.117	0.064 / 0.077	0.032 / 0.037
16	0.063 / 0.054	0.034 / 0.075	0.067 / 0.093	0.042 / 0.077
17	0.229 / 0.325	0.164 / 0.452	0.217 / 0.298	0.095 / 0.264
18	0.029 / 0.039	0.036 / 0.039	0 / 0	0.005 / 0.009
19	0.325 / 0.461	0.304 / 0.225	0 / 0	0.109 / -0.023
20	0.084 / 0.089	0.064 / 0.088	0.065 / 0.060	0.033 / 0.043
21	0.048 / 0.039	0.045 / 0.047	0 / 0	0.027 / 0.029

We also observe a good match here. The causes of discrepancy could come from the programs unable to accurately classify rising and falling edges. For example, test case #14 has overshoots in 0 level and undershoots in 1 level that coincides with the crossing point, as illustrated in the left of Figure 5.8. The proposed algorithm can accurately distinguish rising and falling edges, while other EDA tools seem to have failing cases.

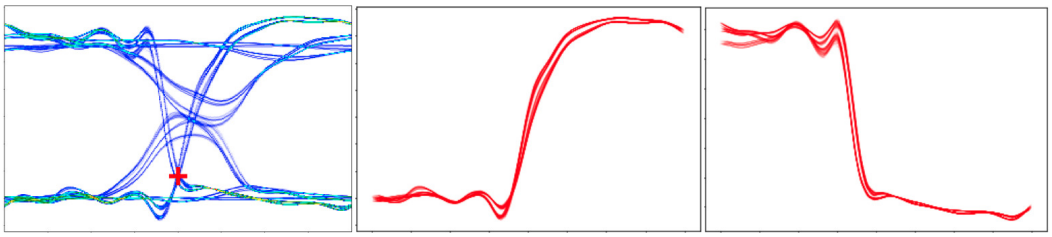


Figure 5.8: Eye diagram for test case #14, with correctly classified rising (middle) and falling (right) edges.

In conclusion, it is demonstrated that the proposed algorithm provides more accurate analysis of eye parameters, while other EDA tools have edge cases that fail to deliver them, or deliver them inaccurately.

CHAPTER 6

RUN TIME OPTIMIZATION

6.1 Run Time Summary

To test the original goal of having a very short run time compared to the transient simulation run time, identical channel with different transient simulation stop time is tested. 2000ns is often a minimum metric for statistical analysis, while on the scale of 100ns is sufficient to visualize the eye diagram and locate any signal distortions. Following tables list run time results of two cases – case #3, a 100×100 Power Distribution Network at 1GHz, and case #20. a 6-inch long meandered coupled microstrip line at 9GHz.

Table 6.1: Run time and relative percentage to transient simulation time for test case #3

Test case #3 - 100x100 Power Distribution Network (PDN)		Run Time (s) (Relative to Transient Simulation Run Time (%))			
Transient Simulation Stop Time (ns)	Input Array Length	Transient Simulation	Crossing Point Detection	Eye Diagram Plotting	Eye Parameter Calculation
100	11187	26.23	0.09 (0.34%)	0.17 (0.65%)	0.07 (0.27%)
1000	104497	96.44	0.67 (0.69%)	0.42 (0.44%)	0.33 (0.34%)
5000	217764	498.92	1.34 (0.27%)	0.89 (0.18%)	0.9 (0.18%)
10000	1057467	958.28	6.52 (0.68%)	2.29 (0.24%)	4.41 (0.46%)

Table 6.2: Run Time and relative percentage to transient simulation time for test case #20

Test case #20 - 6-inch long meandered coupled microstrip line		Run Time (s) (Relative to Transient Simulation Run Time (%))			
Transient Simulation Stop Time (ns)	Input Array Length	Transient Simulation	Crossing Point Detection	Eye Diagram Plotting	Eye Parameter Calculation
100	14406	5.83	0.28 (4.80%)	0.26 (4.46%)	0.18 (3.09%)
1000	144006	48.93	2.39 (4.88%)	1.31 (2.68%)	2.4 (4.90%)
10000	1440006	491.38	20.21 (4.11%)	16.24 (3.30%)	19.69 (4.01%)
40000	5760006	1997.67	28.19 (1.41%)	22.11 (1.11%)	23.11 (1.16%)

We can see from the above tables that the relative run time compared to the transient simulation run time is less than 5%, with test case #3 having

it less than 1%. Though not listed in this paper, all other test cases were also run and verified that the relative run time percentage stays below 5% of the transient simulation run time. Therefore, it is implied that the proposed algorithm is comparable to, if not faster than, industry EDA tools, while having higher accuracy in crossing point detection

It is also important to note that the transient simulation run time, as well as the crossing point detection algorithm run time depends on computational speed. For reference, this paper's data is tested with a Macbook Pro (14-inch, 2021) with 8-core CPU and 14-core GPU. Larger-scale computing machines would be able to run the software faster. Furthermore, it is expected that the improvements of computational capability and optimization of python libraries would reduce the run time in the near future.

Furthermore, if this is implemented along witht the transient simulator, the algorithm can be initiated during the transient simulator such that the software can run the transient simulation and append the simulated waveform into the algorithm simultaneously in order to further minimize the run time.

In terms of the algorithm itself, many approaches have been implemented and tested for same functionality, of which the fastest ones are chosen. List comprehension is generally faster than the numpy library when dealing with variable sized nested arrays. For example, it had been discovered that $sum()/len()$ is much faster than $np.average$ function, and list append is faster than $np.hstack$. When using the binning method, plt.hist2d by default plots the 2D histogram, while np.hist2d doesn't, thereby saving computational time when plotting is not needed. When implementing the Savitzky-Golay Filter, we are only interested in $K = 1$, hence the filter function in the Scipy library is custom-modified such that computational time is minimized. Even though the problem initially seemed simple, it involved great effort attempting different methodologies in order to achieve the goal of minimal run time.

CHAPTER 7

CONCLUSION AND FUTURE WORK

7.1 Conclusion

In this thesis, we introduced the algorithm for crossing point detection in eye diagrams. This algorithm provides a comprehensive approach to the crossing point detection, not only verifying its validity in comparison with current EDA tools, but also enabling accurate eye shifting such that more accurate eye parameter calculations can be achieved.

The algorithm was tested with various test cases with very different eye shapes, enabling accurate quantification of the signal quality in signal integrity, which existing EDA tools could not achieve. Furthermore, this algorithm also accounts for non-linearities, which becomes prevalent in practical applications as we shift into higher bit rates and crosstalks due to smaller transistor sizes. Inspired by Moore's law, the operation speed of electronic devices strives to increase, and new inventions will result in high-speed systems in a more compact form. Inventions such as LIM [1] is a great example of an attempt to reduce simulation runtime as it aims to achieve runtime linearity with larger circuit sizes. Having a fast simulation tool is essential for engineers, and consequently economy of scale can be achieved to realize what we dream of much earlier.

7.2 Future Work

This paper focuses on eye diagrams generated from transient simulation, which is a continuous waveform in the time domain. Channel Simulation is another method that uses convolution between a single pulse response and PRBS, which is generally faster than the transient simulation [14]. There also

exists the statistical simulation [15], which constructs the eye diagram using large scale statistical properties, which is not fully explored in this paper. A statistical eye takes a single pulse response and overlays every possible combinations (+ and -) of the ISI, which is more useful for statistical analysis such as BER or bathtub plots. Nevertheless, the illustrated algorithm in Chapter 4 can be applied to find the crossing point for statistical eye diagrams as well since the algorithm looks at each UI and classifies into rising and falling edges.

While this paper focuses on PAM-2 eye diagrams, it can be extended into PAM-4 or PAM-N as well. However, There are more than 1 crossing point for PAM-4 and above, urging the need for an agreed definition of the referenced eye crossing point. For PAM-4, It is possible to modify the algorithm such that, if the threshold voltage is set properly, all different level transitions are separated (rising/falling edges for each of 00→00, 00→01, 00→10, 00→11, 01→00, ... and so on). For PAM-8 and above schemes, there isn't a universally agreed set of definitions that quantify the quality of channel, as these schemes are employed very lately. Nevertheless, this paper's algorithm sets an important milestone for shifting adjustment algorithm needed for eye parameter calculations.

REFERENCES

[1] Z. Yi, "Latency insertion method for fast high-speed link and ic simulation," M.S. thesis, University of Illinois Urbana-Champaign, Urbana, Illinois, 2023.

[2] "What is an eye diagram — systematics," Systematics.co.il, 2024. [Online]. Available: https://www.systematics.co.il/pcb-blog/what-is-an-eye-diagram/

[3] T.Ha, Tri, *Theory and Design of Digital Communications*. Monterey, CA: Cambridge, 2011. [Online]. Available: http://www.cambridge.org/9780521761741.pdf

[4] Anritsu, "Jitter analysis - basic classification of jitter components using sampling scope," Application Note, 2012. [Online]. Available: https://dl.cdn-anritsu.com/en-au/test-measurement/files/Application-Notes/Application-Note/MP2100A_EF3100.pdf

[5] Anritsu, "Understanding eye pattern measurements," Application Note, 2010.

[6] Advantest, "Dsp-based testing fundamentals 50 prbs (pseudo random binary sequence)," Application Note, 2013. [Online]. Available: https://www3.advantest.com/documents/11348/3e95df23-22f5-441e-8598-f1d99c2382cb#:~:text=LFSR%20(Linear%20Feedback%20Shift%20Register)&text=When%20the%20shift%20register%20is,top%20of%20the%20bit%20stream.\textbf

[7] C. Piech, "K means," Stanford.edu, 2013. [Online]. Available: https://stanford.edu/~cpiech/cs221/handouts/kmeans.html

[8] "Ansoft designer 7.0 - eye measurements," Mweda.com, 2024. [Online]. Available: http://www.mweda.com/designer/ansoft-designer/reports/EyeMeasurements.htm

[9] "Virtuosity: New eye diagram measurements," Cadence.com, 02 2018. [Online]. Available: https://community.cadence.com/cadence_blogs_8/b/cic/posts/virtuosity-eye-diagram-measurements

[10] W. A. Finke, "Algorithm for finding the eye crossing level of a multilevel signal," 06 2000. [Online]. Available: https://patentimages.storage.googleapis.com/36/e3/80/6204a95e745120/US6614434.pdf

[11] B. Shi, Y. Zhou, T. Nguyen, and J. Schutt-Aine, "Statistical method for eye diagram simulation in a high-speed link nonlinear system," in *2022 IEEE Electrical Design of Advanced Packaging and Systems (EDAPS)*, 2022, pp. 1–3.

[12] A. Savitzky and M. J. E. Golay, "Smoothing and differentiation of data by simplified least squares procedures." *Analytical Chemistry*, vol. 36, pp. 1627–1639, 07 1964.

[13] Glassner, Andrew S., *The Graphic Gems Series – A Collection of Practical Techniques for the Computer Graphics Programmer*. Palo Alto, CA: Cambridge, 1992. [Online]. Available: https://theswissbay.ch/pdf/Gentoomen%20Library/Game%20Development/Programming/Graphics%20Gems%203.pdf

[14] S. Bobi, "Characterization of channel simulation method with jitter and equalization effect on eye diagram," M.S. thesis, University of Illinois Urbana-Champaign, Urbana, Illinois, 2020.

[15] S. Bobi, "Eye diagram modeling and statistical simulation in nonlinear high-speed link systems," Ph.D. dissertation, University of Illinois Urbana-Champaign, Urbana, Illinois, 2024.