FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Automatic Characterization of Differential High-Speed Digital Interfaces

Luís Cruz

Master in Electrical and Computer Engineering

Supervisor: José Carlos Alves (Prof.)

July, 2011

Resumo

Neste trabalho foi desenvolvida uma ferramenta de análise de sinais. Esta ferramenta pode complementar o ambiente de simulação de circuitos integrados, possibilitando ainda a geração de diagramas de olho e análises de *jitter*.

Tradicionalmente o software utilizado para a analise de sinal é muito genérico, suportando diferentes tipos de análises, resultando num ambiente mais complexo.

Este trabalho apresenta os diferentes tipos de *jitter*, condensando num só ficheiro as definições importantes acerca do mesmo.

A fim de se ter uma melhor percepção da influência do jitter num canal, este trabalho apresenta uma possível técnica de decomposição do mesmo. O método *Q-scale* foi usado na extracção das componentes aleatória e determinística do *jitter*, tendo por base os diagramas de intervalo temporal de erro. Este procedimento de extracção de jitter foi validado através do uso de histogramas de referência, possibilitando a geração de gráficos do tipo *bathtub* e a obtenção do valor total de jitter para uma determinada taxa de erros.

Os parâmetros de caracterização de uma interface podem ser estimados. Neste trabalho foi apresentada uma técnica de extrapolação do digrama de olho. Esta técnica tem por base análises estatísticas, seguindo um mecanismo similar ao da técnica *Stateye*.

A precisão da operação de transferência do domínio da frequência para o domínio dos tempos é afectada pela máxima frequência disponível. Neste trabalho um novo método de extrapolação da resposta em frequência, baseado nas relações da transformada de Hilbert, foi apresentado aumentando desta forma a precisão da conversão para o domínio dos tempos.

Engenheiros de sistema podem usar a ferramenta apresentada nas definições de arquitectura, possibilitando a observação dos efeitos de cabos, *PCBs* e circuitos de recuperação de relógio em termos de *jitter* e diagrama de olho.

Abstract

In this work a free signal analysis tool was developed. This tool can complement the simulation environment of an interconnection integrated circuit, giving the ability to perform jitter analysis and eye diagram generations.

The traditional software for signal analysis is very generic, supporting different types of analysis, resulting in a more complex environment.

This work describes the different types of jitter, concatenating on a single document important jitter definitions.

To understand the influence of jitter in a channel, this work presents a possible jitter decomposition technique. The Q-scale method was used to proceed with the extraction of random and deterministic jitter components, based in the time-interval error histogram. This decomposition procedure was validated through golden time-interval error histograms. The jitter decomposition procedure is able to produce bathtub plots and a total jitter value for a targeted BER.

Interface characterization parameters can be estimated. In this work an eye diagram extrapolation technique was presented. Eye diagram extrapolation technique is based in statistical analysis following an approach similar to Stateye.

Frequency response to time domain response accuracy is affected by the maximum available frequency information. In this work a new frequency response extrapolation method, based on Hilbert transform relations, was presented increasing in this way the accuracy of the domain conversion.

System level engineers can use the presented tool to perform architecture definitions, allowing them to observe the influence of cables, PCB's and clock recovery units in jitter and eye diagrams.

Agradecimentos

Quase todo o meu tempo disponível foi dedicado a este projecto. Todos os dias, depois de 9 horas de trabalho intenso, lá estava eu de volta, para mais umas 5 horas de dedicação. Passei a dormir entre 4 a 5 horas por dia, a fim de poder chegar a este momento com um bom trabalho e com novos conhecimentos.

Alguns dos efeitos estudados neste trabalho não são alvo de estudo nesta Universidade. Espero sinceramente que este documento possa ser usado por futuros alunos, como base para novos estudos.

As dificuldades sentidas foram colmatadas pelo incentivo familiar, dos colegas de trabalho e dos grandes amigos. Sempre que precisava de discutir alguma coisa lá estavam eles, prontos a ouvir e ajudar.

Aproveito esta parte do documento para fazer um agradecimento público aos colegas de trabalho, que sempre me apoiaram. Gostaria também de fazer um agradecimento especial aos caros amigos: António Rocha, João Marcos e Mara Carvalho, pela preciosa ajuda que me deram nestes momentos finais.

Por último queria agradecer à minha família, por ter tolerado a minha falta de comparência nas reuniões familiares, sobretudo à minha esposa que me apoiou e ajudou desde o primeiro dia.

A todos o meu sincero MUITO OBRIGADO.

O Autor

Contents

List of Figures

List of Tables

Abbreviations and Symbols

BER	Bit Error Ratio (Number of errors divided by the number of transmitted bits)
BERT	Bit Error Ratio Tester
CAD	Computer-Aided Design
CASE	Computer-Aided Software Engineering
CDF	Cumulative Density Function
CDR	Clock and Data Recovery
CRU	Clock Recovery Unit
CRU-TF	Clock Recovery Unit Transfer Function
CSV	Comma Separated Values
DDJ	Data Dependent Jitter
DFT	Discrete Fourier Transform
DJ	Deterministic Jitter
EDF	Empirical Distribution Function
FT	Fourier Transform
LFSR	Linear Feedback Shift Register
IC	Integrated Circuits
IDFT	Inverse Discrete Fourier Transform
ISI	Inter-symbol interference
PCS	Phase Changing Speed
PDF	Probability Density Function
PLL	Phased-looked Loop
PRBS	Pseudo Random Binary Sequence
RMS	Root Mean Square
RJ	Random Jitter
SER/DES	Serializer/Deserializer
TDR	Time Domain Reflectometer
TIE	Time Error Interval
TF	Transfer Function
TJ	Total Jitter

Chapter 1

Introduction

Signal analysis is very important on today's developments. The idea of copying a file from a PC internal disk to an USB external memory should not take more than a blink of the eye. Current societies live in a complex environment where productivity, multi-tasking, security and quality are the cultural foundations.

Current consumer products face a tremendous challenge: how to survive on a very competitive market where the today's best product could be the tomorrow's old fashion gadget.

Consumer electronics companies have two solutions to prevail: intensive marketing campaigns or overall quality. Marketing campaigns push the differentiating factors to the limit, for example: a memory vender can state that their product can deliver more Gbps than the competition. Alternatively the same example but now with a quality mindset: a memory vender can state that their product delivers the same Gbps as the average competition, but with 100 times less errors.

Quality and marketing are not the only factors pushing the boundaries of consumer electronics (indirectly are...). Current philosophy is to add value. Consumer electronics have to be pretty, with a big display, high transfer data rates, fast... among a few other things.

Consumer electronics products are built from smaller components where each component needs to interact with other components and can be provided from different suppliers. It's then important to ensure the interconnection. Different products can have different interfaces, making even more difficult the interconnection.

A key aspect on current products is interoperability; no one wants to have a gadget which is not able to communicate with other devices. Interconnection standards play an important role on this new industry environment, ensuring the interoperability between them.

The increasing number of supported functions on current consumer electronics demands for high speed interconnection standards. High speed data transfer allows the users to see internet videos on portable devices, to communicate with video information instead of only audio, to support high resolution displays (with/out 3D), between others.

1.1 Extent

The demand for new devices, filled with interactive applications, requires high speed data transfers. New communication standards like HDMI1.4, USB3.0, PCI-E3.0, SATA3.0, 10G XAUI were developed for different applications but with the same purpose: provide an interface capable of supporting the current consumer electronics demands.

The data transferred across different devices needs to be effective. Once the number of bytes shared across different devices increases, the number of errors allowed reduces, to have an effective data transfer rate.

The BER metric is defined as the number of bits transmitted with errors divided by the total number of transmitted bits, the typical value for current standards is 10^{-12} (1 error per $1e^{12}$ transmitted bits). BER metric is not shared with the consumer, but products with low BER will never reach the market since they will not pass a compliance test.

The BER values are difficult to obtain in simulation since the designer needs to transmit billions of bytes to ensure that the design is in accordance with the specifications. To achieve the targeted BER in simulation the designer will spend weeks or even months simulating the device, which is not reasonable. It is necessary to use new methods to achieve this target.

The major goal for each platform/device is to transmit a certain number of bits in the shortest time and with the shortest number of errors.

High speed interconnection standards have more signal characteristics defined than just BER, like: Eye Diagram, Signal Amplitude, Rise/Fall times, High and Low voltage values, Jitter values, Resistor Termination values.

Interconnection standards also include protocol layers, interface pin description and pin functionalities. Such functionalities can be easily verified through fast digital simulations. The major difficulties are on the signal requirements since those physical interface signals are dependent on the protocol layer. This dependence moves the simulation to a mixed-signal environment (digital and analog) so the idea of verifying the analog and digital parts of an IC (Integrated Circuit - device) as separated blocks is no longer valid.

The IC development stage starts with a high level specification, performed by system engineers, intended to provide the specifications to the internal sub-blocks. Different interconnection standards will demand different architectures. During the specification stage, system engineers have to select the most appropriate architecture to be implemented in the integrated circuit.

Consumer electronics devices are created from multiple IC's which leads to constant pressure over the IC development teams. The device's perceptible quality can be defined by a small IC component bringing new challenges to the IC providers.

1.2 Actual verification process and difficulties

Recent Integrated Circuits can be divided in tree major groups: Analog, Digital and Mixed-Signal. Each group is associated with a group of specific simulation tools.

Digital circuits are oriented to events; the simulator creates a list of events leading to a variable time step. The languages associated with Digital circuits are very powerful, a designer can create a complete automatic verification environment, reducing the human interactions and speeding up the verification tasks. Usually the simulation results are written to a logfile. At the end of the simulation a PASS or FAIL indication is provided to the designer.

Digital simulation environments are constantly improved during a product development process. Testbench coverage plays an important role during the product development since the designer can check what parts of the digital circuit are not being stimulated by the testbench. Therefore the designer can add new functionalities to the testbench, in order to increase the coverage.

The major advantage of the digital environment is the number of tests that can be performed automatically; therefore the risk associated is reduced, leading to better first time right ratios than the ones on the pure analog developments.

Analog circuits are oriented to time events. The simulator uses a minimum time step increment which can be defined by the designer. Each step has associated a voltage and current calculation for each circuit node. This nature leads to long simulations, for example: let's consider a 1GHz PLL design. Such PLL will require 100us to achieve lock (reference and feedback clocks aligned in phase and frequency), if it was used a Verilog model for such block, the lock condition would be achieved in less than one minute. When an analog simulation environment is used, the same simulation can easily take 3/4 days on current machines. Therefore, the number of simulations performed on analog environments is less than in digital environments, reducing the first time right ratio.

Analog simulation environments are user dependent although some automatic tasks can be created. In the majority of cases, such tasks are dependent of the verification tool and of the circuit implementation specific behavior. The same circuit can have different behaviors, depending on the technology node used.

Analog simulations are time/user consuming: BER measures and accurate eye diagrams generations are almost impractical to obtain in simulation. The only way to solve this lack of verification is to verify the worst case conditions and, based on those values, create a regression curve that allows the designer to check if the design is meeting the interconnection standard specifications.

Mixed-signal integrated devices are made of analog and digital circuits which will work together in order to perform a specific task. The majority of these circuits are on the interface stack, like USB, HDMI, PCI-E. They are responsible for the connection between pure digital circuits and the messy real analog world. In terms of simulation environment, such circuits share the same pros and cons of the digital and analog circuits.

Over the last few years, tool vendors have produced mixed mode simulation environments, allowing the designers to simulate the digital and analog parts of a Mixed-signal circuit with their specific simulation tools. Therefore such circuits can achieve high simulation coverage within a reasonable time frame. This new environment is usually called Co-sim (Co simulation - digital and analog circuits working together). BER and eye diagram results are still being obtained using the same technique as the analog circuits (definition of worst case conditions followed by extrapolation).

One of the biggest problems relates with the definition of the worst case conditions which can vary from product to product. The emerging of statistical eye diagram generation can be helpful with this major task.

The dependence of the user on the pure analog verifications is constantly revised. Designers try to create a couple of tasks for a specific product that will reduce the verification time, but each new product demands new tasks. There are verification tasks implemented using specific waveform analyzers routines. Such routines can change from tool version to tool version, therefore it is necessary to spend extra time to update the verification environment.

Mixed-signal environments have an extra verification loop: digital and analog blocks can be evaluated separately, but they should work correctly when connected. This loop requires top level simulations that are normally performed using Co-sim simulation environments. Protocol routines and interactions between digital and analog circuits need to be evaluated at this level.

Digital designers play a big role on Co-sim environment since in most of the cases the digital testbench is the one used at this level. Digital and analog tools deal with the digital to analog and analog to digital conversions, allowing the designer to be focused on the stimuli generation.

The use of the digital testbench allows the designer to create self-checking tasks, reducing the human interface. The lack on this environment is that some analog parameters cannot be evaluated (they were previously converted to digital '1s' and '0s'). Such parameters, like rise and fall times, need to be checked by hand. There are analog parameters that are not affected by the digital circuit; those ones are verified on the analog simulation environment.

1.3 Motivation and objectives

Product Development teams have to deal with a restricted number of software licenses (software licenses are very expensive), determining the maximum number of parallel simulations that can be performed at the same time.

Each new product usually demands a new verification environment which, when combined with a lack of licenses, has a big impact on the product development plan.

Certification of a new product is done using a set of tools including digital oscilloscopes. New oscilloscopes come with very good software for signal processing but this software can only be executed under the oscilloscope. It would be good if such software were available for use within Linux/Windows PC environment, helping this way the verification tasks.

The scope of this work is to bring some of the signal processing tools to the verification environment. There are products, like USB2.0/3.0, which have available for download software tools capable of analyzing the simulation waveforms and evaluate if the circuit respects the standard specifications for interface signals.

The main idea of this thesis will be to create a similar software tool, without product limitations, capable of processing simulation saved files and evaluate if the previously defined parameters are met just like the signal processing software available inside the new digital oscilloscopes. It is also necessary to introduce new techniques for signal evaluation in order to reduce the simulation time. On this work, statistical eye diagram technique will be added to the software verification tool.

The proposed software will add new features and will be based on open source tools, allowing the users to improve the existent libraries and to add new ones. The main characteristics are:

- Proposed software will allow the end user to characterize and estimate the full SER/DES architecture prior to the block implementation. This will reduce the number of simulation steps and the number of silicon re-spins
- Silicon characterization without using third-party software will be allowed. User's will need to use an oscilloscope to capture data. The data processing steps will be done on the proposed software. The need for third-party characterization software will decrease
- Proposed software will provide to the final user a set of functions/libraries to help on jitter characterization/decomposition (Random Jitter, Periodic Jitter, Data Dependent Jitter)
- Based on short but accurate transistor level simulations, silicon behavior will be estimated
- The proposed software will allow system engineers to perform system level simulations reducing in this way the complexity of the architecture definition task.

Proposed software will give to the end user the possibility to characterize all the major analog characteristics with only one tool (free and open source). It will also allow the user to estimate and predict the product characteristics before silicon.

The jitter/decomposition can be performed without using third-party software. The use of python will allow the use of the same script across different operating systems (Windows, Linux, MacOs).

Jitter budget definition is always a hard task; with this software it will be easier. The end user will have the possibility to specify the jitter budget for each major block. Then it is possible to evaluate if the overall target is achieved. It will be also possible to update the model with preliminary results obtained from simulations, to check if all blocks are under the internal specification. The addition of accurate cable models will allow the end user to better understand the overall effect in TX and RX sides. The must important functions are:

- Clock recovery unit
- Eye diagram generator
- Statistical eye diagram generator
- Analog to digital and digital to analog conversion

- Measurement units

  Rise and Fall times measurement

  Minimum and Maximum values of the analog signal for each digital representation

- Data generation

  Internally generated

  Extracted from .csv files

- Random noise and duty cycle distortion generation
- Cable emulator
- Equalization
- Channel characterization

1.4 Original contributions

This thesis addresses the signal characterization problems providing a free and easy to use tool, allowing engineers to perform channel characterization tasks without deep knowledge of signal analysis.

A linearization technique for Q-scale jitter decomposition method was proposed. Based on jitter measurements it is possible to decompose jitter in deterministic and random events. This technique allows the estimation of total jitter values for a given BER.

Maximum frequency information available in frequency response parameters, affects the accuracy of the frequency domain to time domain transformation. In this work a new method for frequency response extrapolation was presented.

Clock recovery algorithms affect the signal analysis results; the proposed algorithms allow the user to verify the influence of different transfer functions in terms of eye diagram and jitter analysis.

Eye diagram extrapolation method was presented; allowing the end user to have an idea of what will be the eye diagram at the characterization phase. Based on statistical analysis it is possible to estimate the eye diagram shape in terms of jitter from simulation results with a very limited number of acquired bits (when compared to the number of bits acquired in characterization phase).

Signal analysis tool brings to the PC environment functions available in oscilloscopes. The major difference is related with cost since the presented tool is free. In terms of accuracy it was proven that the differences are very small, making this tool a good start point for system architecture definitions.

1.5 Thesis organization

Chapter 2 presents a review of the jitter components and signal analysis tools. The boundary between deterministic jitter and random jitter is presented. The dual-Dirac decomposition method is presented with the associated implications explained. A tail fitting algorithm is presented based on Q-scale method. Signal analysis tools pros and cons are described for the different environments. The use of python for the software tool development is also explained.

Chapter 3 presents the developed jitter algorithms. An improved Q-scale method is presented with implications over total jitter values calculations for a given BER.

Details of the developed signal analysis tool are presented in chapter 4. This chapter describes in detail the signal analysis python functions and their usage.

Chapter 5 contains the description of the implemented python classes.

In chapter 6, different simulation environment results are presented. CRU-TF analyses are performed over clock shared and plesiochronous systems. TX automatic characterization is demonstrated. An example of channel characterization environment for 10Gbit/s is demonstrated.

In chapter 7, the main conclusions drawn from this work are presented together with suggestions regarding further work to enhance the application.

Chapter 2

State of the Art

2.1 Jitter definition

The increasing demand of high speed interface standards raised considerable signal integrity issues. Designers have to ensure the correct functionality and signal integrity of multi Gigabit per second (Gbps) communication channels. At those frequencies, the low pass characteristic of the medium changes the shape of the data signals, making their waveform very different then from square shape. Digital signals at that speed will have voltage and timing variations, which can lead to wrong signal recovery. Timing variations are called as jitter (wander for frequencies below 10Hz).

Jitter is defined as the deviation of the digital timing event from it is ideal position. Such deviation can be represented as a probability density function (PDF) histogram, since jitter, by definition, is a time measurement. Jitter histogram can be obtained by the quantization of the measured deviations, allowing the user to identify specific types of jitter.

There are different jitter measurements, each with its own distinctive, numerical value. The most common are:

- **Cycle to cycle jitter -** Applied to periodic signals; measures the difference between the current period and the previous one. It can also be applied to non-periodic signals (or with a long repetition sequence) with the necessary improvements. This measurement is not used on current designs because a low value does not mean that the system is working properly.

- **Period jitter -** Applied to clock signals; it represents the maximum deviation of the real clock transition point to the ideal one.

- **Long term jitter -** Applied to clock signals; it represents the difference between the ideal clock transition and the real one. This measure is performed over a large number of clock cycles and can be represented on a histogram form.

- **Time interval error jitter -** Can be applied to data and clock signals; it represents the difference between the event of the signal being measured and the ideal recovered event location. This measurement is performed over a large number of bits and can be represented on a histogram form.

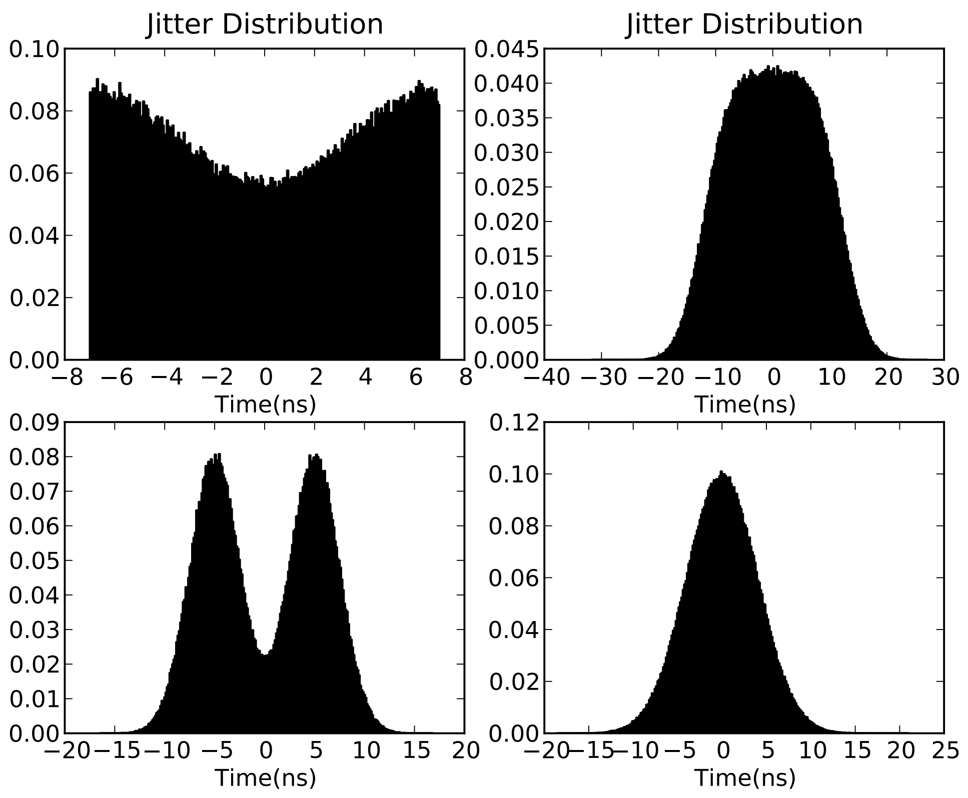


Figure 2.1: Jitter distribution examples

Figure 2.1 shows different types of jitter distributions, each one obtained from a different jitter source. Through observation, it was possible to conclude that the total jitter is data dependent, increases with the number of captured bits and is pattern dependent. Such observations allowed engineers to conclude that jitter is not bounded. In fact, jitter contributions come from two major groups: deterministic and random.

During many years, the effect of random jitter (RJ) on the overall total jitter (TJ) was negligible, due to the low data rates involved. In those cases, deterministic factors like periodic jitter dictate the total jitter value. For this reason, as of today, total jitter is defined either as RMS or peak-to-peak, although such definition is not accurate, due to the total jitter not being bounded, as it suggests. To correctly define the total jitter new methods must be applied.

New models combine the effects of deterministic and random jitter allowing the definition of a total jitter value for a given bit error ratio. $TJ = DJ + N \times RJ$ where N = number of standard deviations corresponding to the required BER.

Total jitter can be divided into the following components [2](see Figure 2.2):

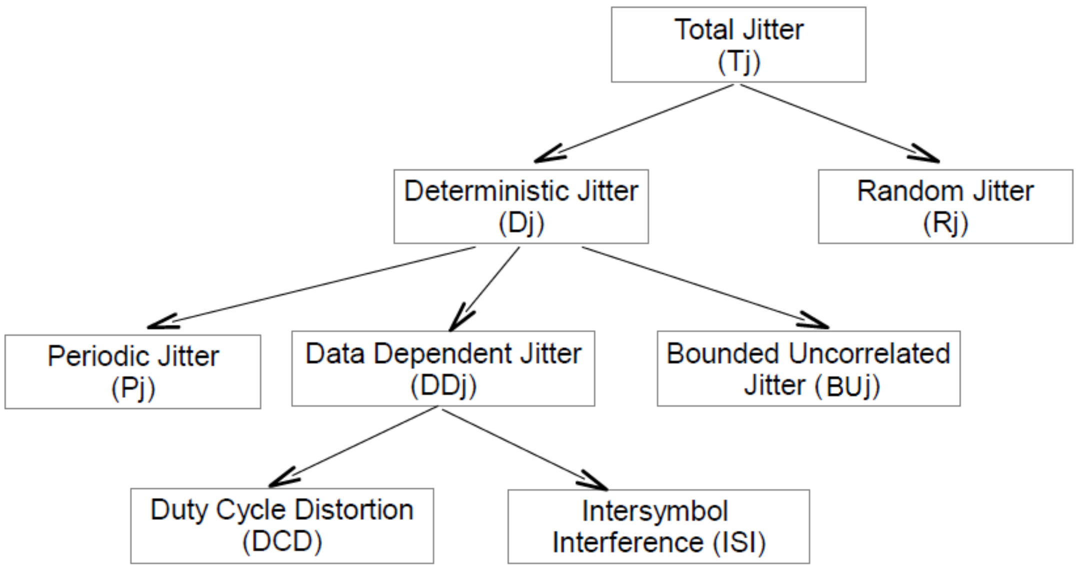

Total Jitter (Tj)
Deterministic Jitter (Dj)
Random Jitter (Rj)
Periodic Jitter (Pj)
Data Dependent Jitter (DDj)
Bounded Uncorrelated Jitter (BUj)
Duty Cycle Distortion (DCD)
Intersymbol Interference (ISI)

Figure 2.2: Jitter subcomponents

- **Deterministic jitter -** Jitter with non-Gaussian probability density function. It is always bounded in amplitude. Possible causes are imperfections of devices, EMI, grounding problems, etc;
  - **Periodic jitter -** Refers to periodic variations of signal edge positions over time. Possible causes of PJ are electromagnetic interference sources such as power supplies;
  - **Data dependent jitter -**Corresponds to a variable jitter that depends on the symbol pattern transmitted on the link under test;
    - **Duty cycle distortion -**Refers to the bit period variation of consecutive alternated patterns (101010). Possible cause of DDJ is the difference between rise and fall times on the driver buffer stage.
    - **Inter-symbol interference -**Refers to the bit time relation with the previous transmitted pattern. Possible cause of ISI is the low pass frequency response of the link under test.
  - **Bounded uncorrelated jitter -**Refers to the bit time influence from adjacent links, the value is bounded but not correlated with the transmitted bits. Possible cause of BUJ is crosstalk.
- **Random jitter -** The main source is Gaussian (white) noise within system components. It interacts with the slew rate of signals and produces timing errors at the switching points.

Decomposition of total jitter into subgroups allows a more accurate jitter budget definition, helping to control the different jitter sources. Today's SER/DES (Serializer/Deserializer) interfaces can be represented as a PLL and Driver in SER devices; PLL, Receiver and CDR (clock and data recovery) in DES devices. Each one of these blocks will be affected by different jitter components. The specification of a jitter budget on sub-circuits allows the designers to accurately design each SER/DES block.

2.2 Jitter components

2.2.1 Periodic jitter

A fundamental limitation in high-speed digital communication systems is the intrinsic jitter of phase-locked loops (PLL) [3]. Typically, such jitter is on the form of periodic jitter. Periodic jitter added by PLLs must be tolerated by the clock and data recovery circuit (current standards define a sinusoidal jitter tolerance mask). A clock signal is a square wave with a fundamental frequency of f_s, which can be decomposed by a Fourier analysis into a sum of sine harmonics of frequencies f_s, $3f_s$, $5f_s$, etc. When this signal is passed through a band pass filter (driver, channel) with a center frequency at f_s , the components outside the fundamental ($3f_s$, $5f_s$, $7f_s$, etc.) can be discarded. The square wave can be written as (fundamental sinusoidal component with amplitude A and frequency f_s):

$$Acos(j(t)) = Acos(\overbrace{\underbrace{2\cdot\pi\cdot f_s}_{\omega s}\cdot t + \emptyset_0 + \Delta\emptyset(t)}^{\text{j(t)}}) \tag{2.1}$$

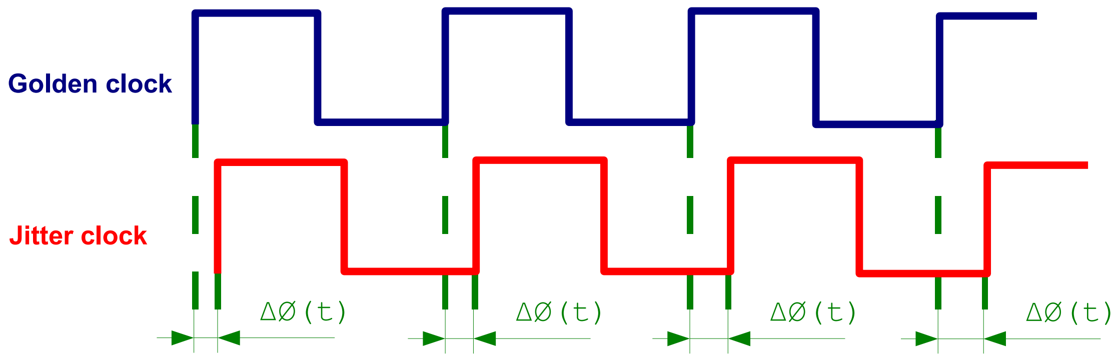


Figure 2.3: Sinusoidal timing jitter

The phase modulation component $\Delta\emptyset(t)$ (see figure 2.3), which is the timing jitter, can be defined as: $\triangle\emptyset(t) = \Delta\omega\cdot sin(\omega_j\cdot t)$. Assuming $A_j\cdot Tbit\ \ (s)$ as the maximum phase difference between the golden clock and jitter clock, $\triangle\emptyset(t)$ can be written as:

$$\triangle\emptyset(t) = A_j\cdot 2\cdot\pi\cdot sin(\omega_j\cdot t)\ \ (rad) \tag{2.2}$$

Please note that to ensure correct correlation between equations 2.1 and 2.2, the maximum phase jitter was converted from seconds to radians, since Tbit corresponds to $2\cdot\pi$ radians, $A_j\cdot Tbit$ in seconds will correspond to $A_j\cdot 2\cdot\pi$ in radians. The jitter frequency ($\omega(t)$) can be obtained by differentiating $j(t)$ [4] in relation to time as

$$\omega(t) = \omega_s + \underbrace{A_j \cdot 2 \cdot \pi \cdot \omega_j \cdot cos(\omega_j \cdot t)}_{\frac{d}{dt}\triangle\emptyset(t)} \ (rad) \tag{2.3}$$

The maximum and minimum period can be defined ($T = 2 \cdot \pi / \omega$) as:
$T = \frac{2\pi}{\omega s \pm A_j \cdot 2 \cdot \pi \cdot \omega_j} \ (s)$
$T = \frac{2\pi}{2 \cdot \pi \cdot f_s \pm A_j \cdot 2 \cdot \pi \cdot 2 \cdot \pi \cdot f_j} \ (s)$

$$T = \frac{1}{f_s(1 \pm A_j \cdot 2 \cdot \pi \cdot \frac{f_j}{f_s})} \ (s) \tag{2.4}$$

Maximum and minimum frequency can be defined as: $f = f_s(1 \pm A_j \cdot 2 \cdot \pi \cdot \frac{f_j}{f_s}) \ (Hz)$.

Another important measure is the phase changing speed (PCS). This parameter defines the minimum phase tracking on the CDR side. PCS can be obtained by differentiating $\Delta\emptyset(t)$

$$PCS = A_j \cdot 2 \cdot \pi \cdot \omega_j \cdot cos(\omega_j \cdot t) \ (rad/s) \tag{2.5}$$

Equation (2.5) can be converted to the form of UI/UI (maximum phase change in terms of bit time per bit time). To convert seconds to UI (unit interval), it is necessary to multiply seconds by f_s, because $1 \ (UI) = 1/f_s \ (s)$. The conversion from rad to UI can be done by dividing the radian value by $2 \cdot \pi$. The maximum phase changing speed (MPCS), which must be tracked by the CDR, can be defined as:

$$MPCS = A_j \cdot 2 \cdot \pi \cdot \frac{f_j}{f_s} \ (UI/UI) \tag{2.6}$$

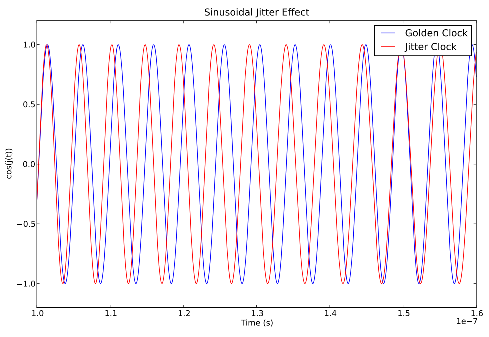
Figure 2.4 was obtained through simulation. Two clocks were created following the equations defined above. The golden clock was created using a pure cosine wave with a frequency of $f_s = 207MHz$, while the jitter clock was created assuming a sinusoidal jitter with $f_s = 207$MHz, $f_j = 10MHz$, $A_j = 0.3Tbit$. The obtained experimental results are correlated with the previous equations (see table 2.1).

Measurement	Formula	Expected	Obtained
Max Frequency	$f_s(1 + A_j \cdot 2 \cdot \pi \cdot \frac{f_j}{f_s})$	$225.850e6$	$226.030e6$
Min Frequency	$f_s(1 - A_j \cdot 2 \cdot \pi \cdot \frac{f_j}{f_s})$	$188.150e6$	$188.330e6$
Max TIE	$0.3 \cdot Tbit$	$1.449e-9$	$1.456E-9$
Min TIE	$-0.3 \cdot Tbit$	$-1.449e-9$	$-1.456E-9$

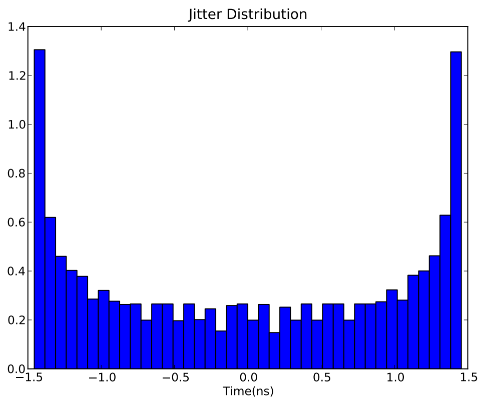
Table 2.1: Sinusoidal jitter simulation

2.2.2 Data dependent jitter

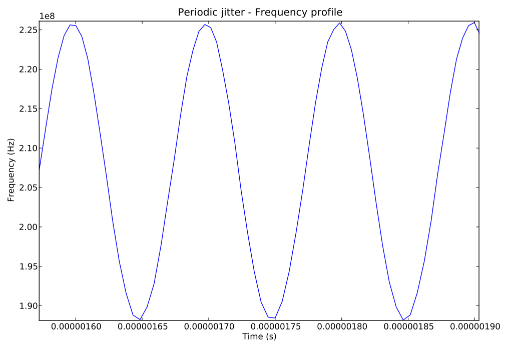
Data dependent jitter is the deviation of each data edge from the ideal point due to the memory of the system.



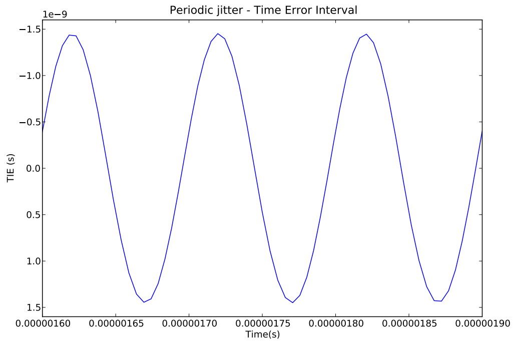
(a) Golden clock and jitter clock



(b) TIE histogram



(c) Frequency variation of jitter clock



(d) Difference between jitter clock and golden clock edge locations

Figure 2.4: Sinusoidal clock jitter example ($f_s = 207e6, f_j = 10e6$, $A_j = 0.3 \cdot Tbit$)

Duty cycle distortion and Inter-symbol interference are good examples of system memory, meaning that the current bit waveform will depend on the previous bits.

Duty cycle distortion refers to a deterministic and bounded value characterized by the period difference of alternate consecutive bits.

Inter-symbol interference refers to the band limited characteristic of the system (cable, driver, connectors). This effect is proportional to the number of equal bits transmitted consecutively.

Duty cycle distortion phenomenon is related with the unbalanced generation of the logical ones and zeros. Today's drivers are half rate, meaning that the high speed serial clock is, in terms of frequency, half of the bit rate, being the selection between bits based on the clock level ("1" or "0").

Duty cycle of the high speed clock will be directly translated into the overall duty cycle distortion.

At the driver side, there are two other factors responsible for duty cycle distortion: different rise and fall times and mismatch between positive and negative networks. The last two in conjunction with the clock duty cycle distortion can produce a clean signal (cancelation), but in the majority of the cases, the three contribute to the duty cycle distortion, leading to $DCD = ck\_dc + rt\_ft + n\_p \ \ (s)$, where ck_dc represents the clock duty cycle distortion, rt_ft represents the difference between rise and fall times and n_p represents the difference between positive and negative networks (all parameters are specified in seconds).

Inter-symbol interference phenomenon is related with the low pass characteristic of the link. Different edge patterns have different frequency components. Fast-changing patterns behave as high-frequency signals; slow-changing patterns behave as low-frequency signals. Because of the conductors filtering effects, different patterns propagate at different speeds through the conductors. This difference in propagation speeds causes bits to smear into adjacent bits, resulting in ISI [2].

The extraction of the data dependent jitter of each sub-circuit based on the entire system value is unpractical. The solution is to measure each block independently, $DDJ_t = \sum_{i=0}^{N} DDJ_{block}(i)$ where DDJ_t represents the total data dependent jitter of the system (theorical), N represents the total number of blocks and $DDJ_{block}(i)$ represents the data dependent jitter of each block. As stated before, the observed Total DDJ can be lower than DDJ_t. To better understand this type of effects let's take a look at a real system:

- Two independent data sources LFSR7 and LFSR15. Linear feedback shift register module 7 (LFSR7) ensures a maximum run length of 127 then the sequence is repeated. The maximum number of consecutive equal bits is 7. LFSR15 has a bigger run length: 32767. The maximum number of consecutive equal bits is 15. Each data source generates 3.4Gbps ($tbit \simeq 294ps$).

- Two duty cycle distortion jitter sources per each data source, with 3.5ps and 6.5ps of deterministic jitter respectively.

- Cable with a low pass characteristic per each data source. The frequency domain response is described in figure 2.5.

The characterization of the system was done using a Spice simulator. The cable was modulated as a series of inductors, resistors and capacitors and its frequency response obtained doing an AC analysis. Spice simulator was used to simulate the entire system, reason why the data sources and jitter source were described in spice like format. During the simulation, 34000 bits were sent and the final results were post-processed to generate the jitter distributions.

Figure 2.6 contains both waveforms, of the LFSR7 and LFSR15 systems. The differences on the waveforms are not clear, but it is possible to observe that LFSR15 signal seems to be "slower" than the one on LFSR7 system, since it takes more time to reach the final value.

In figure 2.7 the effect of the data pattern can be identified through the respective probability density function.

In LSFR7 system it is possible to separate the DCD from ISI. As expected, the DCD "diracs" are placed on 0ps, 3.5ps, 6.5ps and 10ps (sum of the two DCD sources). ISI effect is seen on the spread effect, resulting on $DDJ_{LFSR7} = 12ps$.

ISI has a bigger effect on LFSR15 system, the only difference between LFSR7 and LFSR15 is the data pattern, meaning that ISI is directly related with the data pattern. In LFSR15 the four DCD effects cannot be separated, this is due to the ISI bigger factor and as expected the total DDJ is bigger, $DDJ_{LFSR15} = 16ps$.

The previous example shows the importance of a correct data codification on today's communication standards, because the increase of speed requires lower jitter values, to allow the transmission during long time periods without a single error.

2.2.3 Bounded uncorrelated jitter

Crosstalk is the more common representation of BUJ. This phenomenon can be observed when two pcb tracks (lineA and lineB), separated by a thin space transmit different patterns. LineB will induce, through its intrinsic capacitors/inductors, a signal on lineA and vice-versa. The signal at the end of the lines, in some cases will have a higher amplitude, while in other cases, a lower amplitude, resulting on different crossing points (timing jitter).

Crosstalk effect can be reduced by separating the lines, increasing the shielding or using differential signals. No further analysis will be done over this topic, since the use of differential signals reduces this effect.

2.2.4 Random jitter

Random Jitter is caused by a huge number of very small effects like: thermal oscillations, flicker, and shot noise, variations in characteristic impedance of the pcb tracks, fluctuations of conductivity in a conductor caused by impurities, variations in resistance caused by random fluctuations in the local voltage that individual electrons feel and, literally an infinite number of other tiny effects combined in a way that result in levels of jitter that cannot be predicted. Thankfully, the fundamental theorem of Statistical Physics - The Central Limit Theorem [5] - makes the whole thing very simple.

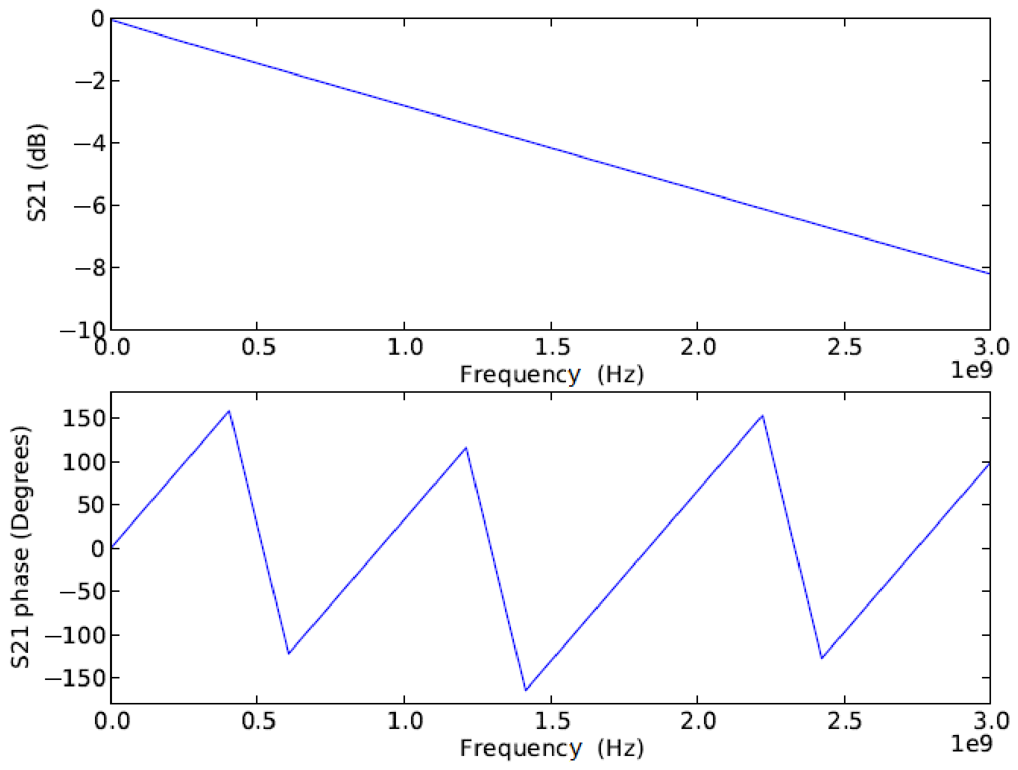
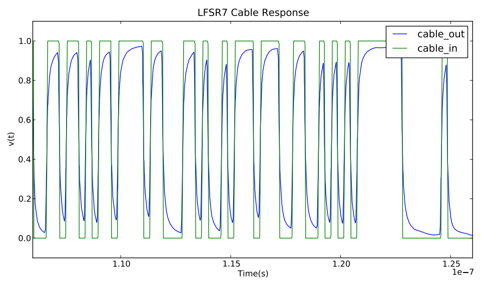
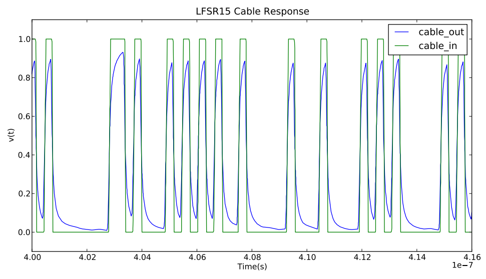


Figure 2.5: Cable frequency characteristic

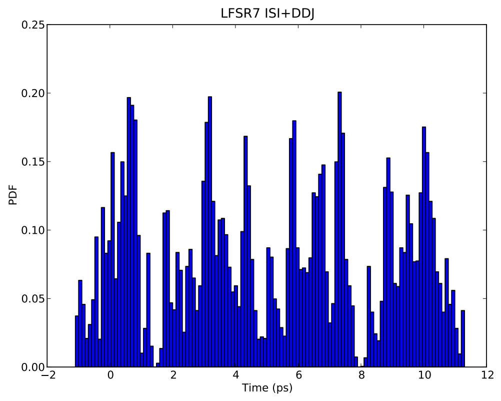


(a) LFSR7 waveform

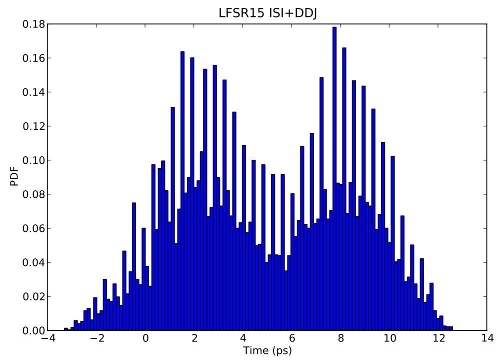


(b) LFSR15 waveform

Figure 2.6: Data patterns LFSR15 and LFSR15 waveforms at the input and output of the cable



(a) LFSR7 PDF



(b) LFSR15 PDF

Figure 2.7: DDJ probability density function for LFSR7 and LFSR15 systems

The central limit theorem says that if S_n is the sum of n mutually independent random variables, then the distribution function of S_n is well-approximated by a certain type of continuous function known as a normal density function (Gaussian distribution for jitter).

Random jitter cannot be described by a simple peak to peak value. It will differ from acquisition to acquisition, even if the pattern sent was the same in all acquisitions. It is necessary to use a different measure to describe such variations, such as a probability function (PDF). The waveform can be described by equation 2.7 where σ represents the standard deviation and μ the mean value.

$$PDF_{RJ}(x) = \frac{1}{\sigma\sqrt{2\pi}} exp[-\frac{(x-\mu)^2}{2\sigma^2}] \tag{2.7}$$

"Maximum" peak to peak jitter values are usually referred taking into consideration a certain probability. Taking into consideration the target BER, the peak value for RJ can be known. Such definition allows de designer to take into consideration, a realistic RJ value on the total jitter definition.

Figure 2.8 describes the relation between $BER(x)$ and $PDF_{RJ}(x)$. $BER(x)$ is an even function, so the left side ($BER_l(x)$) is equal to right side ($BER_r(x)$).This means that the contribution for total jitter will be equal to the contribution of both sides. Mathematically, $BER_l(x) = \oint_x^\infty PDF_{RJ}(u)du$, resulting on a total jitter of $2 \cdot BER_l^{-1}$. Ideally, the designer would like to have an equation on the form of:

$$TJ_{RJ}(BER) = 2 \cdot N\sigma \quad (s) \tag{2.8}$$

BER equation is described using an exponential integral. The idea will be to find a linear approximation to the function described below:

$$BER_l(x) = \frac{1}{\sigma\sqrt{2\pi}} \oint_x^\infty exp[-\frac{u^2}{2\sigma^2}du] \tag{2.9}$$

Using the following variable transformation $a = (\frac{u}{\sqrt{2}\sigma})$,

$$BER_l(x) = \frac{1}{\sigma\sqrt{2\pi}} \oint_x^{\infty} exp(-a^2)da \tag{2.10}$$

Such function is closely related with the complementary error function described as $erfc(x) = \frac{2}{\sigma\sqrt{2\pi}} \oint_x^{\infty} exp(-u^2)du$, resulting

$$BER_l(x) = \frac{1}{2} erfc(a) \tag{2.11}$$

Applying a second variable transformation $u = N\sigma$ to a, results in $a = \frac{N}{\sqrt{2}}$. The multiplication value (N) that returns the desired jitter for the specified BER can be found by replacing $BER_l(x)$ by the desired bit error ratio and solving the equation in relation to N.

$$N = \sqrt{2} \cdot erfc^{-1}(2 \cdot BER) \tag{2.12}$$

From equation 2.12 and 2.8, the total jitter value for a given BER can be described as:

$$TJ_{RJ}(BER) = 2 \cdot \sqrt{2} \cdot erfc^{-1}(2 \cdot BER) \cdot \sigma \;\; (s) \tag{2.13}$$

The complementary error function is tabulated, allowing the designer to easily find the total jitter value of the random jitter, for a specific bit error ratio. Table 2.2 contains the multiplication factor for a few BER cases.

Random jitter represents a big issue on today's systems. Even a system with an amount of random jitter with a low standard deviation can result on a big total jitter, due to the multiplication factor effect of the BER.

BER	N
$1e-6$	4.75
$1e-8$	5.61
$1e-10$	6.36
$1e-11$	6.71
$1e-12$	7.03
$1e-13$	7.35
$1e-14$	7.65
$1e-15$	7.94

Table 2.2: TJ_{RJ} multiplication factor

2.3 Total jitter

Total Jitter plays an important role in the development and specification of serial data links but, while it is well defined, it is not well understood. Total jitter cannot be described as a static and stand-alone value, since one of the components is RJ that varies with BER.

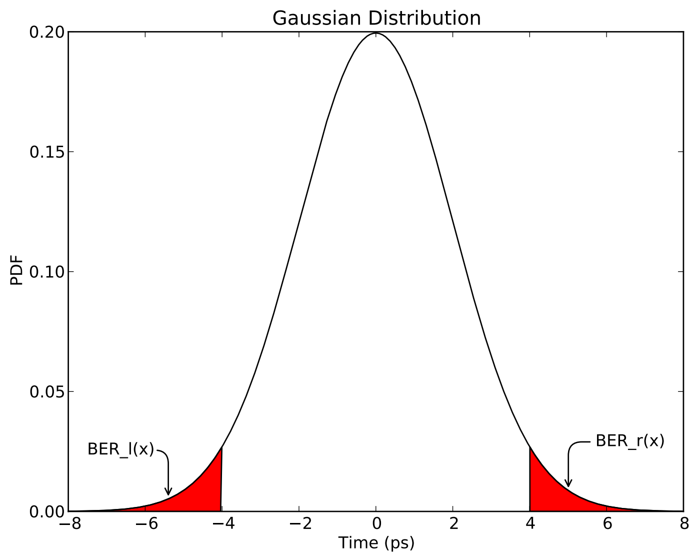


Figure 2.8: Probability density function for random jitter ($\mu = 0, \sigma = 2ps$)

Total jitter can be described by a probability density function (PDF), as the random jitter can. Such function can be obtained through time interval error (TIE) measurements, translated to a probability histogram, like on figure 2.9. Jitter elements defined previously will be combined in different ways. Jitter coming from deterministic sources will be summed, while jitter coming from random phenomenons will be combined on a single σ_{Total}.

Total jitter will be estimated based on the following equations:

$$\begin{aligned} DJ_{Total} &= DCD(p-p)+ISI(p-p)+BUJ(p-p) \quad (2.14)\\ &= ck\_dc(p-p)+rt\_ft(p-p)+n\_p(p-p)+ISI(p-p)+BUJ(p-p)\\ \sigma_{Total} &= \sqrt{\sigma_1^2+\sigma_2^2+\cdots+\sigma_n^2} \end{aligned}$$

Resulting on,

$$TJ(BER) = 2 \cdot N(BER) \cdot \sigma_{Total} + DJ_{Total} \quad (s) \tag{2.15}$$

The *N(BER)* value can be found on table 2.2. In terms of jitter PDF it is necessary to perform a convolution between all the different PDF, as described on eq. 2.16

$$PDF_{TJ} = PDF_{RJ_{Total}} \otimes PDF_{DJ_Total} \tag{2.16}$$

2.4 Jitter measurement techniques

Jitter measurements can be done using different equipments. There are two major groups: BERT Scan and Oscilloscopes.

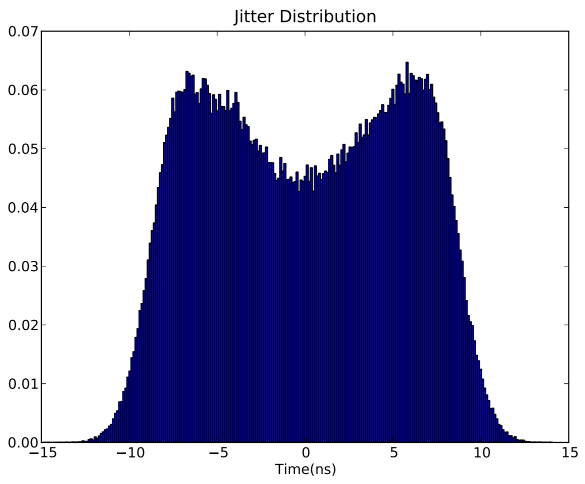


Figure 2.9: Probability density function for total jitter

Bit Error Ratio Tester (BERT) equipments can automatically measure the total jitter value. The measurement is based on the number of errors found, for a specific number of acquisitions. These equipments can accurately measure the total jitter.

Oscilloscopes work in a different way: these equipments acquire samples, therefore additional data processing is needed to estimate the total jitter value. Major limitation on these equipments is related with the maximum number of samples that can be stored. BERT equipments, on the opposite, only store the number of errors, allowing accurate total jitter measurements.

A BERT equipment is able to shift the sampling point across an entire bit time. To perform a BERT scan, the equipment generates a known pattern of data and artificially, slides the sampling point across an entire bit, counting the number of errors found in each sampling point. The number of errors acquired on each sampling point is then divided by the number of transmitted bits, resulting on a graphic that has, on the x-axis, time and on y-axis, the bit error ratio value (bathtub plot).

A BERT equipment is able to observe decoding errors, where it is assumed that such decoding errors are caused due to timing or voltage errors. BERT scans can accurately specify the total jitter boundaries, but no information related with voltage or timing margin is provided. The number of errors will be higher near to the transition bit time, since a small variation on the received signal will lead to an error. At the middle of the bit, the number of errors will be the lowest, since at this point the allowed variation on the received signal is the maximum possible. This explains the reason for sampling the bits at the middle of the bit time.

An oscilloscope equipment works on a different way. It captures the received signal, but since it is stored on a memory, the number of acquired samples is small, when compared with a BERT equipment. Oscilloscopes are very useful for determining the jitter components (voltage and timing jitter). Since it is not possible to acquire enough samples to directly specify the total jitter, post processing techniques must be used to determine that value.

Today's communication standards define an eye diagram mask that must be met. Eye diagram

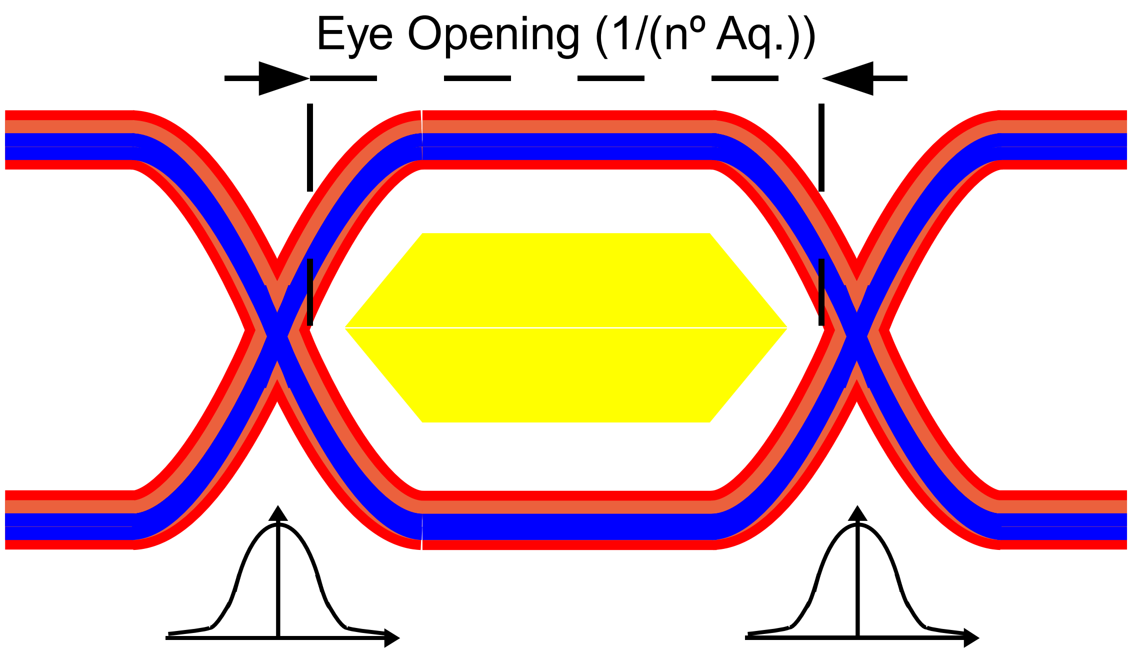


Figure 2.10: Eye diagram

measurements are easily performed on an oscilloscope, since the principle is to represent all captured bits on a single waveform, with a timing window equal to two bits (some cases one bit). At the end of each two bit cycle, the waveform pointer is reset, allowing a continuous waveform capture using a time scale of two bits (see figure 2.10).

Eye diagram is the name given to the waveform, obtained from the superposition of all captured bits in a timing window of two bits. It is then possible to measure the opening at the middle of the eye and voltage margin too.

Eye diagrams should be obtained using a clock recovery unit (CRU), sometimes called golden PLL, to correctly define the ideal bit period of each bit. This unit emulates the effect of the PLL circuit present on the receiver side (figure 2.11).

The eye diagram specification also defines the characteristic of such units. The idea of using it is to remove the low frequency noise, which is correctly tracked on the receiver side. If such circuit was not used a complete closed eye diagram will be obtained.

The combination of an oscilloscope, CRU (clock recovery unit) and data processing software

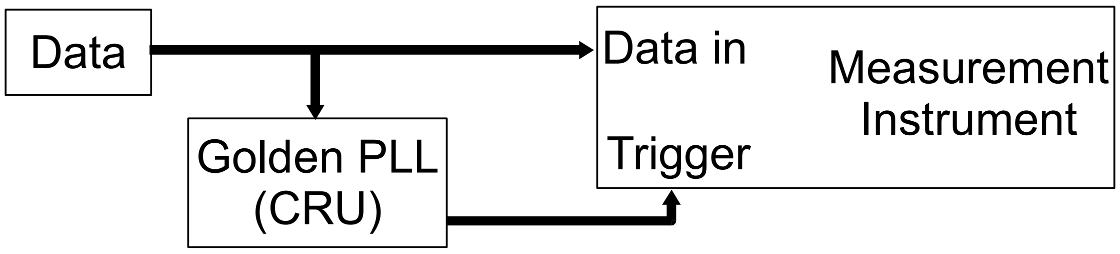
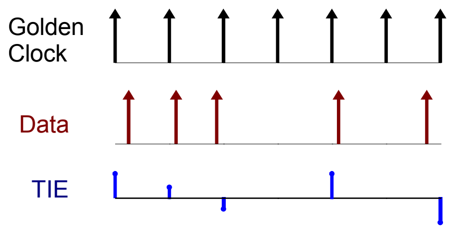
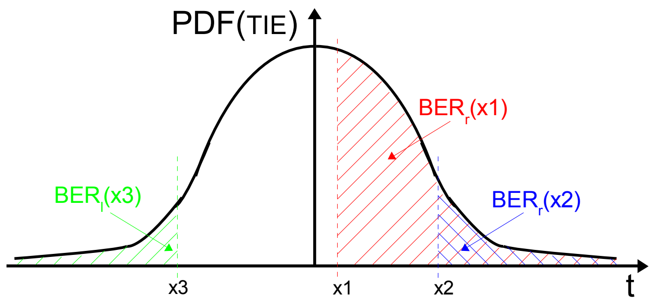


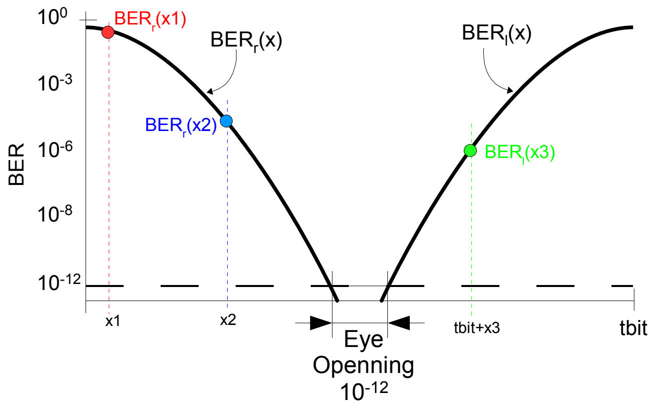
Figure 2.11: Eye diagram



(a) TIE histogram generation



(b) PDF of TIE



(c) Bathtub plot

Figure 2.12: Translation from TIE to bathtub plot

ensures a correct jitter measurement. Total jitter measurement starts with the generation of a TIE histogram. This task is done using the trigger produced by CRU as the ideal edge location, and then it is necessary to determine the difference between the ideal edge and the real edge location. Software uses an internal algorithm to accurately determine the transition points of the real signal and then it is just necessary to store the differences into a histogram. With the TIE histogram complete, bathtub plot diagrams can be directly determined by calculating the correspondent BER for each histogram point.

Total jitter for the desired BER can be obtained through a linear approximation. Figure 2.12 exemplifies that process.

2.5 Jitter decomposition techniques

The decomposition of total jitter into two major components, random and deterministic, represents a major task on today's systems design/evaluation.

The total jitter histogram only provides a maximum jitter peak-to-peak value for a specific number of acquisitions. No further extrapolations can be done for different number of acquisitions. This is very important on today's communication standards since the BER target is to have less than 1 error in one terabit. Taking the example of USB3.0 (5Gbps), to achieve 1 terabit it is necessary to acquire data during 3 minutes. For lower BER, the involved time will be even higher, making this process impracticable (3 minutes of real world can represent as much as years of simulation time).

The spread of total jitter into deterministic jitter and random jitter (following a Gaussian distribution), allows the designer to better predict the real system behavior, making the simulation time more reasonable.

2.5.1 Dual-Dirac

The dual-Dirac model [6], is universally accepted for its utility in quickly estimating total jitter, defined at a bit error ratio, TJ(BER), and for providing a mechanism for combining TJ(BER) from different network elements.

The model can be applied only if the total jitter can be separated into RJ and DJ, where RJ follows a Gaussian distribution and DJ follows a finite bounded distribution.

Dual-Dirac method represents the deterministic jitter as two Dirac functions ($DJ_{(\delta\delta)}$), and random jitter as a Gaussian distribution with standard deviation equal to $\sigma_{(\delta\delta)}$.

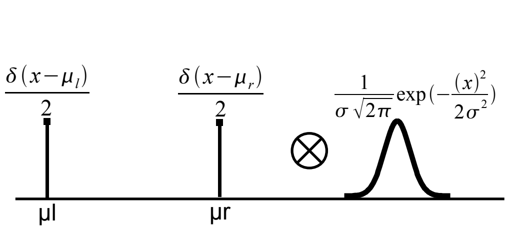
Dual-Dirac total jitter value ($TJ_{(\delta\delta)}$) assumes DJ as two Dirac functions located at μ_l ($A \cdot \delta(x-\mu_l)$) and μ_r ($B \cdot \delta(x-\mu_r)$), resulting on a total deterministic jitter defined as $DJ_{(\delta\delta)} = \mu_r - \mu_l$, RJ will be treated as a Gaussian distribution ($\frac{1}{\sigma_{(\delta\delta)}\sqrt{2\pi}} exp[-\frac{x^2}{2\sigma^2_{(\delta\delta)}}]$), total jitter is represented as the convolution of two components, resulting on equation: 2.17 (see figure 2.13).

$$\frac{A}{\sigma_{(\delta\delta)}\sqrt{2\pi}} exp[-\frac{(x-\mu_l)^2}{2\sigma^2_{(\delta\delta)}}] + \frac{B}{\sigma_{(\delta\delta)}\sqrt{2\pi}} exp[-\frac{(x-\mu_r)^2}{2\sigma^2_{(\delta\delta)}}] \tag{2.17}$$

Deterministic jitter probability density function can be very different of a dual-Dirac waveform. Once it will be bounded, it can actually be described as a dual-Dirac approach taking into consideration only the maximum deterministic jitter, let's call it as $DJ_{(pp)}$.

Figure 2.14 tries to illustrate the basic concept of the superposition of three Dirac functions (representing the DJ), convolved with a Gaussian distribution. The general idea is that DJ can be divided into multiple Dirac functions; each function will be convolved with Gaussian distribution, originating a Total Jitter distribution created by the sum of the convolutions:

$$PDF[TJ(x)] = \sum_{i=0}^{N-1} A_i \cdot \frac{1}{\sigma\sqrt{2\pi}} exp[-\frac{(x-(\frac{\mu_r-\mu_l}{N-1} \cdot i + \mu_l))^2}{2\sigma^2}] \tag{2.18}$$


$\frac{\delta(x-\mu_l)}{2}$
$\frac{\delta(x-\mu_r)}{2}$
$\frac{1}{\sigma\sqrt{2\pi}}\exp(-\frac{(x)^2}{2\sigma^2})$
μl
μr
(a) PDF of each component

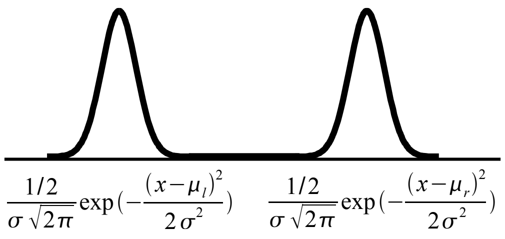

$\frac{1/2}{\sigma\sqrt{2\pi}}\exp(-\frac{(x-\mu_l)^2}{2\sigma^2})$
$\frac{1/2}{\sigma\sqrt{2\pi}}\exp(-\frac{(x-\mu_r)^2}{2\sigma^2})$
(b) $TJ_{(\delta\delta)}$ PDF

Figure 2.13: Convolution of dual-Dirac distribution with a single Gaussian

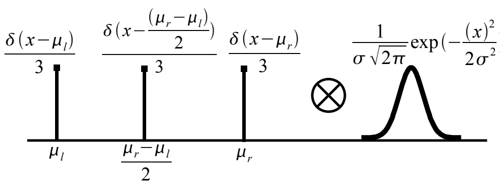
Applying the central limit theorem to eq. 2.18 and considering low probability points the resultant function can be described by a new Gaussian distribution. Equation 2.19 is the basis for dual-Dirac method, proving that it is possible to extract DJ as two Dirac functions and RJ as a Gaussian distribution. It is important to note that $\mu_{\delta\delta}$ is different of μ_{pp}.

$$\begin{aligned}\lim_{x\to-\infty} PDF[TJ(x)] &= A\cdot\frac{1}{\sigma_{l_{\delta\delta}}\sqrt{2\pi}}exp[-\frac{(x-\mu_{l_{\delta\delta}})^2}{2\sigma_{l\delta\delta}{}^2}] \\ \lim_{x\to+\infty} PDF[TJ(x)] &= B\cdot\frac{1}{\sigma_{r_{\delta\delta}}\sqrt{2\pi}}exp[-\frac{(x-\mu_{r_{\delta\delta}})^2}{2\sigma_{r\delta\delta}{}^2}]\end{aligned} \tag{2.19}$$

Dual-Dirac approach tries to fit the tails of a specific Gaussian distribution described by $\mu_{\delta\delta}$ and $\sigma_{\delta\delta}$, to the tails of the total jitter probability density function, resulting on two Gaussian distributions as exemplified on figure 2.15.

Total jitter can now be described as a function of a bounded DJ given by $DJ_{\delta\delta} = \mu_{r_{\delta\delta}} - \mu_{l_{\delta\delta}}$ and a RJ given by $\mu = 0$ and $\sigma_{\delta\delta} = \sqrt{\sigma^2_{r\delta\delta} + \sigma^2_{l\delta\delta}}$. It is important to notice that $DJ_{\delta\delta} < DJ_{pp}$, but $TJ_{\delta\delta} > TJ_{pp}$ for lower BER. This is achieved at the cost of overestimating σ.

In general dual-Dirac method underestimates deterministic jitter, but overestimates random jitter and total jitter, resulting on an accurate methodology to characterize the jitter of an entire


$\frac{\delta(x-\mu_l)}{3}$
$\frac{\delta(x-\frac{(\mu_r-\mu_l)}{2})}{3}$
$\frac{\delta(x-\mu_r)}{3}$
$\frac{1}{\sigma\sqrt{2\pi}}\exp(-\frac{(x)^2}{2\sigma^2})$
μ_l
$\frac{\mu_r-\mu_l}{2}$
μ_r
(a)

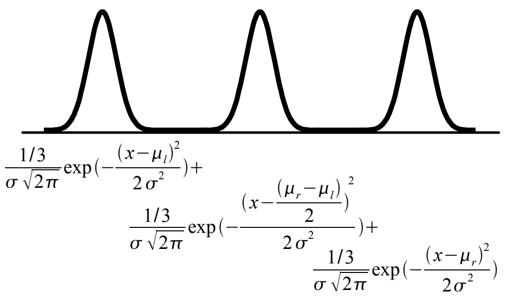

$\frac{1/3}{\sigma\sqrt{2\pi}}\exp(-\frac{(x-\mu_l)^2}{2\sigma^2})+$
$\frac{1/3}{\sigma\sqrt{2\pi}}\exp(-\frac{(x-\frac{(\mu_r-\mu_l)}{2})^2}{2\sigma^2})+$
$\frac{1/3}{\sigma\sqrt{2\pi}}\exp(-\frac{(x-\mu_r)^2}{2\sigma^2})$
(b)

Figure 2.14: Convolution of three Dirac with a single Gaussian

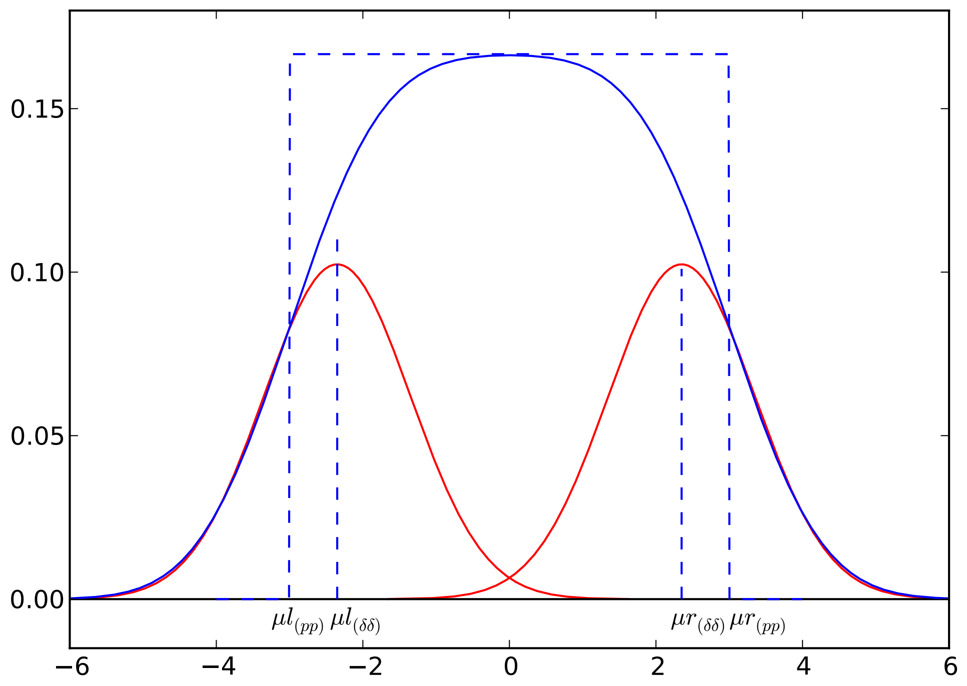


Figure 2.15: dual-Dirac tail fitting

system (see Eq. 2.20).

$$TJ_{\delta\delta} = DJ_{\delta\delta} + N(BER)_{\delta\delta} \cdot \sigma_{\delta\delta} \tag{2.20}$$

2.5.2 Q-scale

Dual-Dirac model describes total jitter as two Gaussian jitter distributions, obtained by fitting the tails of total jitter into a defined Gaussian distribution. The major difficulty resides on the tail fitting algorithm, since it is difficult to extract the exponential waveform of the total jitter. If the approximation was linear the complexity of such tasks will be minimal.

Q-scale method [6] provides a linear approximation to exponential waveform of total jitter, but can only be applied to points far from DJ distribution so that the $TJ(x)$ can be described by a Gaussian curve. Based on eq. 2.19, $BER_l(x)$ is given by

$$BER_l(x) = K_l \frac{1}{\sigma_{l_{\delta\delta}}\sqrt{2\pi}} \oint_x^{\infty} exp\left[-\frac{(x' - \mu_{l_{\delta\delta}})^2}{2\sigma_{l\delta\delta}}\right] dx' \tag{2.21}$$

Now let's apply a scale transformation (if $TJ(x)$ follows a Gaussian curve)

$$Q = \frac{x - \mu_{l_{\delta\delta}}}{\sigma_{l\delta\delta}} \tag{2.22}$$

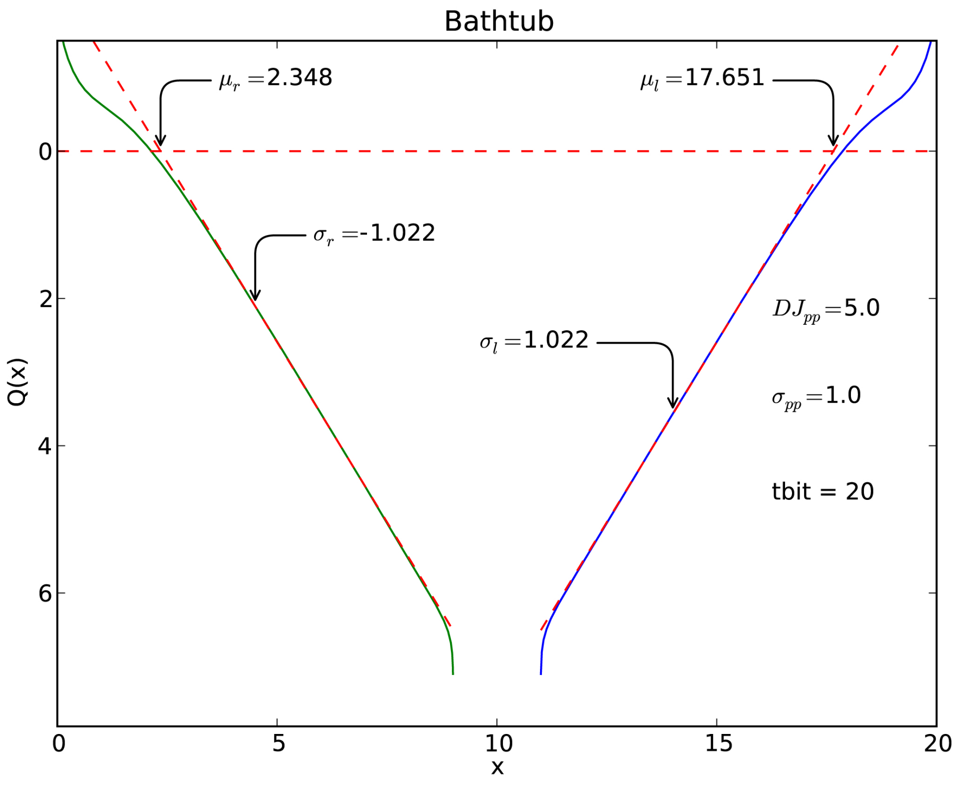


Figure 2.16: Q-scale version of a bathtub plot

Resulting on

$$BER_l(Q) = K_l \frac{1}{\sigma_{l_{\delta\delta}}\sqrt{2\pi}} \oint_Q^\infty exp\left[-\left(\frac{Q'}{\sqrt{2}}\right)^2\right] dQ' \tag{2.23}$$

The complementary error function is given by

$$erfc(x) = \frac{2}{\sigma\sqrt{2\pi}} \oint_x^\infty exp(-u^2)du \tag{2.24}$$

so that eq. 2.22 can be writen

$$\begin{aligned} BER_l(Q) &= K_l erfc\left(\frac{Q}{\sqrt{2}}\right) \\ &= K_l\left(1 - erf\left(\frac{Q}{\sqrt{2}}\right)\right) \end{aligned} \tag{2.25}$$

The reason why this is being done can be found on figure 2.16, where $BER(x)$ is mapped on a bathtub plot resulting on a representation of the Gaussian distributions as straight lines. To obtain

$Q(BER)$ it is necessary to invert eq. 2.25, resulting on

$$\begin{aligned} \frac{BER_l}{K_l} - 1 &= -erf\left(\frac{Q}{\sqrt{2}}\right) \\ erf\left(\frac{Q}{\sqrt{2}}\right) &= 1 - \frac{BER_l}{K_l} \\ Q(BER_l) &= \sqrt{2}erf^{-1}\left[1 - BER_l \cdot A_l\right] \end{aligned} \tag{2.26}$$

Where erf^{-1} indicates the inverse error function and $A_l = \frac{1}{K_l}$. Until this point it is assumed that jitter follows a Gaussian distribution, but now that $Q(BER_l)$ was described it is possible to generalize it to any case. Taking into consideration the dependence of BER_l on the time-delay position of the sampling point, it is possible to express such value as a dependence of time, so replacing BER_l by $BER_l(x)$ on eq. 2.26, a general definition of Q_l that does not rely on the form of the jitter distribution can be described as:

$$Q_l(x) = \sqrt{2}erf^{-1}\left[1 - BER_l \cdot A_l\right] \tag{2.27}$$

Since $Q_l(x) = \frac{x - \mu_{l_{\delta\delta}}}{\sigma_{l_{\delta\delta}}}$, the designer will have to tune A in order to achieve a good linear approximation of $Q(x)$ to a straight line. From the linear approximation it will be possible to obtain μ as the time where $Q(x) = 0$ and σ, as the inverse of the approximated line slope. This method follows the same concept as the ones described in [7] and [8].

Q-scale allows the designer to decompose total jitter into deterministic jitter and random jitter. Still it is also necessary to characterize total jitter in relation to a bit error ratio, from Q-scale and replacing $BER(x)$ by the desired BER:
$Q_{BER} = \sqrt{2}erf^{-1}\left[1 - BER \cdot A\right] = \frac{x - \mu_{l\delta\delta}}{\sigma_{l\delta\delta}}$
Resolving in relation to x, results on

$$x_l = Q_{l\,BER} \cdot \sigma_{l\delta\delta} + \mu_{l\delta\delta} \tag{2.28}$$

To obtain the $TJ(BER)$ it is necessary to consider both sides of the total jitter distribution, since $BER_l(x)$ is equal to $BER_r(-x)$ in terms of equation, resulting on

$$TJ(BER) = Q_{l\,BER} \cdot \sigma_{l\delta\delta} + \mu_{l\delta\delta} - Q_{r\,BER} \cdot \sigma_{r\delta\delta} - \mu_{r\delta\delta} \tag{2.29}$$

Figure 2.17 shows an example of a Q-scale waveform obtained from a total jitter histogram with $DJ_{pp} = 5$ and $RJ = 1$, as expected $\sigma_{r\delta\delta} < 0$ since the slope is negative, resulting on

$$TJ(BER) = Q_{l\,BER} \cdot \sigma_{l\delta\delta} + (Q_{r\,BER} \cdot \sigma_{r\delta\delta}) + |\mu_{l\delta\delta} - \mu_{r\delta\delta}| \tag{2.30}$$

Based on eq. 2.30, deterministic jitter can be defined as $DJ_{\delta\delta} = |\mu_{l\delta\delta} - \mu_{r\delta\delta}|$. In terms of random jitter and following [9] the contribution of each σ can represent a value different than 50% due to that fact that $\sigma_{\delta\delta}$ includes the contribution of DJ not only RJ, considering we_l and we_r as the

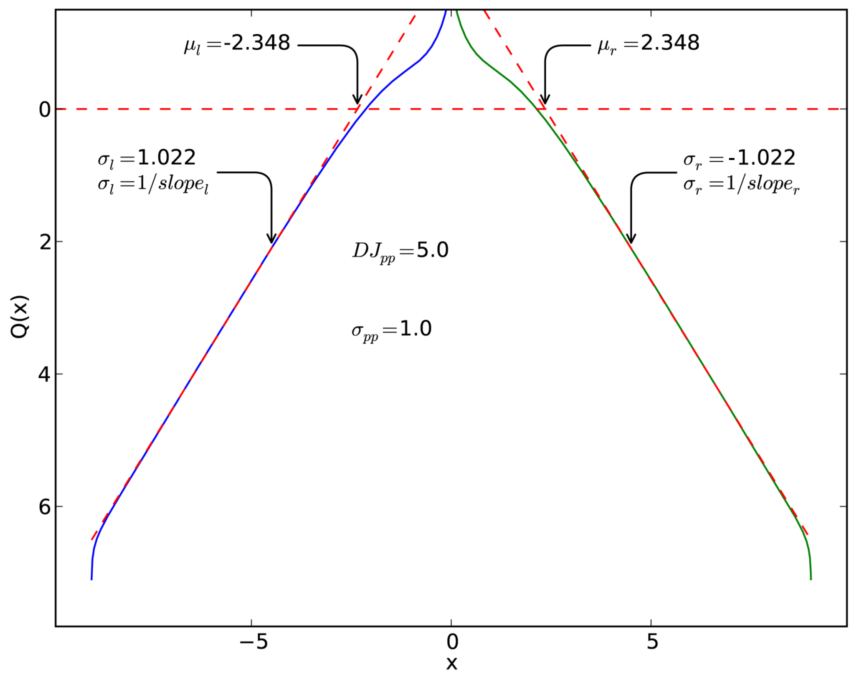


Figure 2.17: Q-scale

multiplication factors where $w_l + w_r = 2$ equation 2.30 would be:

$$TJ(BER) = Q_{l\,BER} \cdot \sigma_{l\delta\delta} \cdot we_l + Q_{r\,BER} \cdot \sigma_{r\delta\delta} \cdot we_r + |\mu_{l\delta\delta} - \mu_{r\delta\delta}| \tag{2.31}$$

Figure 2.18 represents the total jitter Empirical density function (discrete PDF) and the obtained dual-Dirac Gaussian distributions, showing the accuracy of Q-scale method.

2.6 Integrated circuits development process

Automatic characterization of differential high-speed digital interfaces can be divided in two big groups: functional characterization and electrical characterization.

Functional characterization covers the signaling schemes, which must be addressed by the testbench, to comply with a predefined operation mode. Different designs can be impaired and in all cases the end user should obtain a similar result. As an example, for a TV standard like HDMI, functional characterization would include: supported video modes, support for HDCP encoding, correct color mapping, sound extraction, correct encoding/decoding, HPD operation, etc.

Electrical characterization is related with signal quality: rise and fall times, differential voltage, termination value, power down termination value, inter-pair skew, intra-pair skew, jitter, single ended high voltage and low voltage, etc. Signal quality is typically dictated by transmitter

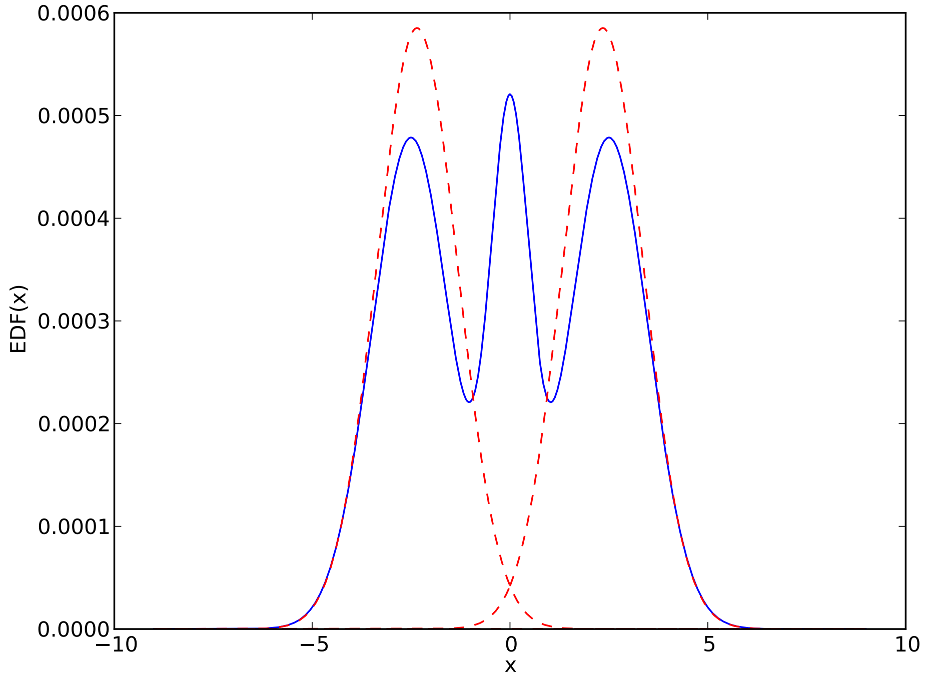


Figure 2.18: Total jitter with application of dual-Dirac method

devices, but receiver devices need to comply with the worst signal defined. On the transmitter side, the signal quality is evaluated; on the receiver side, the signal tolerance is evaluated.

A great number of characterizations/verifications are done over a physical device on laboratories, using expensive test equipment. Sometimes the necessary software to proceed with an automatic characterization for a specific standard is not included by default in these equipments, requiring an extra cost to have such software.

Development phase of new devices is divided in three major groups: architecture definition, implementation and verification. Focusing our attention on front-end devices (OSI Layer 1 - PHY), it is even possible to divide it into digital and analog circuits. Each one of these circuits has their own implementation/verification languages.

Digital circuits are described in Verilog or VHDL language. Simulations are performed using the same languages or high level languages like SystemVerilog, system C or even pure C. Digital tools allow the designer to perform functional verifications over the entire circuit when analog circuit behavioral model is present; or over only the digital part when such model is absent.

Analog circuits are described in Spice or Spectre language. Simulations are performed using these languages together with Verilog-A.

2.7 Verification environment

Current products (devices) verification environments are no longer pure digital or analog. Today's verification environments are complex, supporting mixed-mode simulations performed on a late development stage.

Simulations take about 80% of the development time. The demand for faster development cycles implies faster verification environments, which explains the attraction for SystemVerilog and Verilog-A on digital and analog side respectively.

Results extracted from simulations should allow the designers to make the necessary corrections on the hardware description, increasing the need for complex verification tools like the ones used on characterization phase.

Time involved on simulation is very different from the time involved in characterization. For example, a full spice circuit simulation of a 3ms period can take more than one month to be concluded. Today's developments use a mixed mode environment, which speeds up the verification phase, demanding for higher accuracy for electrical signal characterization at the simulation environment.. Faster simulations are commonly obtained through a reduction of the simulation accuracy.

The solution for a good match between real circuit and simulation model is the use of worst case conditions, combined with parameters extracted from previous designs, at the verification phase.

2.8 System architecture simulation environment

Architecture definition is the first task of a new development phase. It is important to understand how the system requirements can be achieved, the implication in terms of area, power consumption and development team members (number, expertise, effort).

During system architecture definition, system engineers perform successive trials in order to find the pros and cons of the different possible architectures, allowing the engineers to start defining a high level specification (top-down approach).

System engineers tend to use high level simulation languages/tools instead of Verilog or Spice, increasing the simulation speed. For example, a low pass filter can be defined as a transfer function block on MATLAB Simulink. If Spice was used instead, it would be necessary to design a complete circuit with RLC elements.

Signal integrity analysis is also performed at this stage, requiring the availability of cable models, PCB models, driver model, equalizer model and jitter budget. Different tools allow the designer to perform such simulations.

2.9 System modeling tools

MATLAB and ADS (Advanced Design System) are the most used tools on system level architecture definitions. *"MATLAB is a high-level language and interactive environment that enables you to perform computationally intensive tasks faster than with traditional programming languages such as C, C++, and Fortran"*. This definition can be found at the MATLAB site. On ADS web site it is possible to read: *"Advanced Design System is the leading electronic design automation software for RF, microwave, and high speed digital applications"*. Despite the differences both tools can be used to perform architecture simulations.

Cables have a big impact on the overall system architecture definition, influencing the equalizer frequency response, the receiver gain, etc. Unfortunately the designer cannot find such models on books or on the internet; they must be obtained through cable characterization using s-parameters provided by vector analyzers or TDR parameters provided by TDR equipment. ADS has the ability to work with s-parameters, removing from the user the need to extrapolate the frequency response to achieve a good accuracy.

EEsoft Agilent software library provides a set of tools, aimed to support system engineers on the design of communication products. It is important to notice that the use of Agilent EDA software tools allows the designers to understand what will happen at the characterization phase, since similar software will be present on test equipment.

ADS and MATLAB have internal libraries for signal processing, but these are paid tools. A single license can be affordable, but when multiple licenses are needed the overall cost can force the development teams to use different tools, or to create their specific tools.

Jitter decomposition is important for system engineers, allowing them to easily understand which parts of the system are introducing such components. It is also useful for designers during the bit error ratio estimations, since random jitter increases with the number of transmitted bits.

Test equipment provided by LeCroy, Agilent and Tektronix, for example, provide separate software that connected to the oscilloscope allows the user to extract jitter and perform jitter decomposition. These tools usually have to be paid separately (not included in the oscilloscope package). Furthermore, only Agilent software can be used outside the oscilloscope.

Free tools are also available: GNU Octave (MATLAB clone), Stateye [10] (designed for channel characterization). However, it will require training to start using them, and for example clock recovery algorithms will have to be implemented by the user.

2.10 Software selection for signal analysis

2.10.1 Existent solutions

The development of new Integrated Circuits (IC) starts with the creation of behavioral models for such circuits. These allow the system engineers to identify the correct platforms/solutions that must be implemented.

During the platform specification, system engineers define the blocks to implement as well as their parameters, such as: gain, frequency response, bandwidth, maximum response delay, etc. The accuracy of the behavioral models will dictate the correctness of the block specification . At this point system engineers do not use languages like Spice or even Verilog-A as they will require unnecessary work which will reduce the ability to choose between different platform solutions.

System engineers tend to use MATLAB or even ADS (Advanced Design System) tools providing an abstraction layer to the real implementation. This allows the development of accurate behavioral models. Such tools require depth knowledge of the proprietary programming languages supported. This needs extensive training from the engineers before starting the behavioral model creation. The use of proprietary languages reduces the number of engineers which have the skills to work as system engineers. MATLAB and ADS licenses are expensive which reduces even more the number of engineers that have access to them. The ideal solution will be to have the ADS/MATLAB engine on a form of free programming language allowing a bigger number of engineers to understand and work at system level.

2.10.2 Programming language requisites

There are different programming languages suitable to use on the tool development. Octave is one of them. Octave is a MATLAB clone that supports the majority of MATLAB instructions. But like MATLAB the number of users is restricted and it will also require previous extensive training. Octave is not fast [11] and has limited support, reducing the ability to produce an interesting tool.

The programming language that will be used on the software tool development will have to respect the following requisites:

- Free
- Relatively fast
- Built-in signal processing functions
- Built-in plot functions
- Built-in probability functions
- Built-in functions for "electronic engineers"
- Support for interactive debugging
- Support for multi-threading
- Support for graphical interface
- Easy to use

2.10.3 Python programming language

Scientific groups are trying to use open source software to address the new projects. The use of Python for scientific purposes has been increasing. Due to this fact new libraries are being constantly added to python making it a really scientific tool.

Python is very well spread with good support forums in the internet making it easier to find documentation and tutorials. One of the major advantages is the portability. Python does not need to be compiled and the same code will work in Windows, Linux and MacOS, reducing the interoperability problems. The user only needs to install the python interpreter on his machine to be able to run python code.

Python packages extend the base python functions giving to the end user a powerful engine to produce scientific tools. Numpy and scipy packages extend python to support: statistics, numerical integration, linear algebra, Fourier transforms, signal processing, image processing, special functions, powerful N-dimensional array object and random number capabilities among others.

Numpy and scipy were built in C language ensuring good performance. The matplotlib package is a python 2D/3D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms. Pyrex package is a language specially designed for writing Python extension modules. This package is designed to bridge the gap between the nice, high-level and easy-to-use world of Python and the messy and low-level world of C.

Python also has an user friendly command line interpreter. Graphical user interface development can be done using the PyQt package which has a substantial set of GUI widgets. Multiprocessing is a package that supports spawning processes using an API similar to the threading module. The multiprocessing package offers both local and remote concurrency which effectively side-steps the Global Interpreter Lock by using sub processes instead of threads. Thanks to this the multiprocessing module allows the programmer to fully leverage multiple processors on a given machine. There is also the possibility to perform calculations on graphics cards thanks to the pygtk package .

Python is very easy to use with object oriented support. Variable manipulation does not require a type definition (int, real, string, char) as such type will be dynamically allocated.

2.10.4 Python configuration

Python 2.6 version was used due to the code stability and bigger support on the internet. This version was also selected due to the extensive support and packages availability. Pythonxy was used to provide a complete and stable environment removing the need to install the different packages one-by-one.

Pythonxy is a free scientific and engineering development software for numerical computations, data analysis and data visualization based on Python programming language, Qt graphical user interfaces, Eclipse integrated development environment and Spyder interactive scientific development environment. Pythonxy can also be defined as a scientific-oriented Python Distribution based on Qt and Eclipse. Its purpose is to help scientific programmers to switch from interpreted languages (such as MATLAB) or compiled languages (C/C++ or Fortran) to Python.

Pythonxy supports interactive debugging, allowing the user to place breakpoints across the code under development, as the possibility to dynamically check the variables value. Pythonxy interactive mode provides the necessary support during the development phase.

2.11 Conclusion

Jitter can assume different forms leading to different conclusions. Today's developments pay a big attention to random jitter, since the effect is proportional to the number of transmitted bits.

There are different types of decomposition algorithms, although the dual-Dirac method is becoming the most popular, allowing the designers to estimate that effect over the data channel.

A correct jitter decomposition increases the circuit robustness, leading to a high quality device.

Signal analysis is indispensable for the development of the today's interconnection devices, since with giga bit interfaces, the signals can suffer from multiple effects leading to wrong behaviors. Signal characterization tools, allows the designer to have an idea of the signal characteristics after physical implementation.

In this chapter, different approaches for automatic characterization software were presented. The main focus was on the most used applications, a few more could be presented; this was not done since they are very specific.

Signal analysis tool will be implemented under python language. The main reasons for that choice are explained in this chapter.

Chapter 3

Jitter Extraction Tool

3.1 Introduction

One of the most important analyses used to predict a system behavior for long bit sequences is the jitter decomposition. Usually this type of analysis is preformed over IC samples using expensive laboratorial equipment. The intent of this work is to present a tool capable of doing it but free.

The extract_jitter tool uses the Q-scale method to extract the deterministic and random jitter components. The user just needs to provide a jitter histogram file on a csv (comma separated values) format to the gen_hist.exe executable file. The result will be a pdf file with the jitter extracted components.

3.2 Q-scale on discrete events

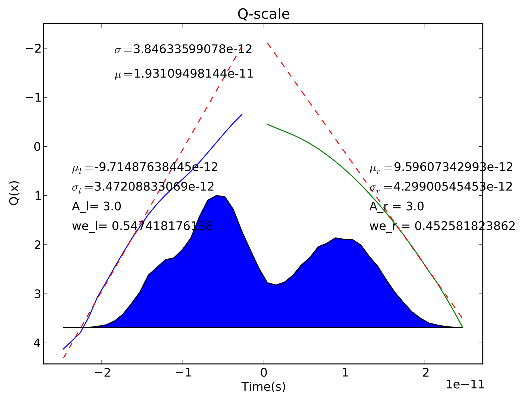
As described on chapter 2.5.2 Q-scale can be expressed as:

$$Q(x) = \sqrt{2} erf^{-1} \left[1 - BER(x) \cdot A\right] \tag{3.1}$$

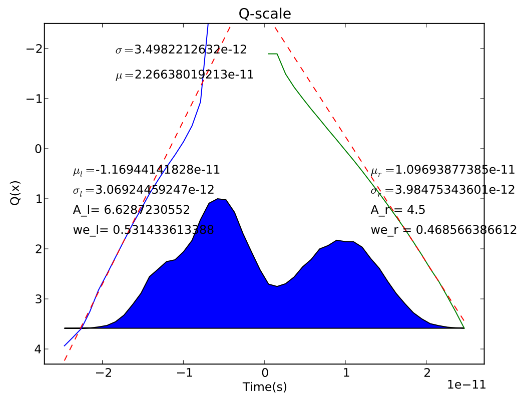
Total jitter is determined by integrating the probability density function (PDF) separately from left and right to determine the symmetric cumulative density function (CDF). The width of this curve at the specified BER (or confidence interval) gives the total jitter. Meaning that $BER(x) = CDF(x) = \oint_{x'=-\infty}^{x'=\infty} PDF(x')dx'$. $CDF(x)$ is purely theoretical obviously , however it can be calculated using the EDF (Empirical Distribution Function), summing the jitter histogram from the left extreme to the desired value of x.

$$EDF(x = h_i) = \frac{\sum\limits_{k=0}^{k=i-1} H_k}{\sum\limits_{k=0}^{k=N-1} H_k} = \frac{1}{P_{total}} \sum_{k=0}^{k=i-1} H_k \tag{3.2}$$

For the purpose of calculating $Q(x)$, following the same idea of previous jitter discussions where the designer is interested in both variations in timing jitter, before and after the mean timing



(a) Non-improved linear approximation



(b) Improved linear approximation

Figure 3.1: Different values for A resulting on different jitter values

value, it is necessary to calculate the left and right sides of the Q-scale. The first thing to do is to obtain the mean position value and then calculate the respective BER:

$$BER_l(x) = EDF_l(x = hi) = \sum_{k=0}^{k=i_{mean}-1} H_k \tag{3.3}$$

$$BER_r(x) = EDF_r(x = hi) = \sum_{k=N-1}^{k=i_{mean}} H_k \tag{3.4}$$

Practically these two functions are joined at the median of the histogram (the bin containing the median, or the bin which 50% of the total population is in that bin or those with lower index). Consequently, 50% of the total population also falls within that bin and those bins with higher index.

3.3 Q-scale linearization

Jitter decomposition tool is based on Q-scale method using a linear approximation to extract random and deterministic jitter values. Since $Q(x) = \frac{x-\mu}{\sigma}$, the relation between $Q(x)$and x is given by a line, with $slope = \frac{1}{\sigma}$ and $y - intersept = \frac{-\mu}{\sigma}$. The intent of extract_jitter tool is to find such values.

During the linearization process, variable A will be changed to reduce the error between $Q(x)$ and the obtained linear approximation. The designer must be aware that a wrong value of A can result on wrong random and deterministic jitter extraction values. On Figure 3.1 the difference on the extracted jitter values can be clearly observed . These differences are caused by a bad linearization because $Q(x)$ is far from a straight line (the designer should only take into to consideration values where $Q(x) > 0$).

The extract_jitter tool divides the jitter histogram in two parts (left and right) and then process the data of each part independently. It is important to notice that each part has their own coefficient

,A_r or A_l, allowing a correct linearization of each side. The use of two independent coefficients is mathematically accurate because each side has their own deterministic jitter. Different coefficients allow the consideration of distinct DJ profiles on each side.

In terms of linearization it is also interesting to notice that to correctly extract the jitter components it is necessary to consider only low probability events. The reason for this is that the Q-scale method assumes that RJ follows a Gaussian distribution and DJ can be defined by two Dirac functions. If the designer uses high probability values the obtained results will be certainly wrong since the idea behind this method is no longer valid.

A golden value can be obtained following this approach: Let's consider 15 as the maximum number of consecutive equal bits transmitted. Using a PRBS function to generate the random bits results on an event probability of: $\frac{1}{2^{15}} = 3.05e^{-5}$. Mapping it into the Q-scale results on $Q(3.05e^{-5}) = 4.008$. In conclusion, in order to obtain accurate jitter values the provided jitter distribution function must generate $Q(x)$ function with values bigger than 4.008.

3.4 The extract_jitter tool

3.4.1 Linearization

The linear approximation is the tool core since everything will depend of it. The idea was to change the coefficient A limiting the minimum and maximum values, using the Newton's method to find the next A value that reduces the quadratic error between the obtained line and $Q(x)$, the error is only considered for $Q(x) > 0$.

The Newton's method can have convergence issues which were addressed by defining a maximum number of iterations. After that number, the linear approximation stops and the A value that returns the lower error is used . The algorithm also stops when the error is bellow than 0.001.

The number of interactions needed to find a coefficient that reduces the error of the linear approximation was reduced by using the second order derivation function (when possible). A_{i+1} can be defined as:

$$A_{i+1} = Ai - \frac{\frac{dQ(x)}{dx}}{\frac{d^2Q(x)}{dx}} \tag{3.5}$$

Proper linearization was also guaranteed by ensuring that the linearization error differences are only considered when Q(x) has values lower than zero. This ensures that Q(x) can be described as the convolution of two dual-Dirac functions with a Gaussian distribution.

3.4.2 Multi-thread

One of the major advantages of using a programming language to create this software tool is the possibility to use threads and queues. Since the jitter extraction can be divided in two parts, the jitter extraction can be performed on each side independently. Based on this fact, the software also divides the work in two threads: one calculates the jitter of the left side and the other calculates the jitter of the right side.

Current processors have multi-threading capabilities. The introduction of multi-thread native support on extract_jitter tool allows the designers to obtain jitter decomposition values faster making use of the capabilities of current processors.

3.4.3 How to use

Jitter extraction software can be found under: `http://paginas.fe.up.pt/~ee10005/soft/extract_jitter.rar` The user needs to have a software capable of opening RAR files, like 7-Zip, in order to be able to start working. Next it is necessary to provide the jitter histogram. The extract_jitter.rar file also provides three examples that can be found under examples directory. The user can create its own file. In order to do that it is just necessary to follow these steps:

- Create a file called <file_name>.csv
- Inside <file_name>.csv separate the histogram hits by new lines
- Each histogram hit (line) should have the time and hit value separated by comma: time, hit
- Once the histogram is completely described it is just necessary to save and close the file
- On the command line and inside extract_jitter directory perform the following command: gen_hist.exe <file_name>.csv
- The jitter tool will create a file named <file_name>_Qscale.pdf containing the jitter extraction values

In order to make the software more generic there is no indication of the total jitter value for the desired BER, but the user can obtain this value through:

$$TJ(BER) = Q_{r\,BER} \cdot \sigma_r \cdot we_r + |Q_{l\,BER} \cdot \sigma_l| \cdot we_l + |\mu_r - \mu_l| \tag{3.6}$$

Where:

$$\begin{aligned} Q_{r\,BER} &= \sqrt{2}erf^{-1}[1 - BER \cdot A_r] \\ Q_{l\,BER} &= \sqrt{2}erf^{-1}[1 - BER \cdot A_l] \end{aligned} \tag{3.7}$$

3.4.4 Results

Tables 3.1, 3.2 and 3.3 summarize the obtained results using different types of jitter with different number of acquisitions. Jitter distribution functions were obtained through the application of the math formulas with the application of the convolution when needed. The result is accurate expected jitter values, allowing an accurate comparison between the expected and obtained results.

Jitter Type	DJ real	RJ Real	TJ(1e-14)real	Tj(1e-12)real
Gaussian only	0	4.000	61.2	56.24
Dual Gaussian	0	4.472	68.42	62.88
Dual Dirac Gaussian	10	3.000	55.9	52.18
Periodic Gaussian	16	2.000	46.6	44.12
Square Gaussian	17	2.500	55.25	52.15

Table 3.1: Total jitter expected for BER=1e-14 and BER=1e-12

Jitter Type	DJ Obtained	RJ Obtained	TJ(1e-14)Obtained	Error(%)
Gaussian only	0.514	4.174	65.88	17.14
Dual Gaussian	0.761	4.840	76.61	21.84
Dual Dirac Gaussian	9.379	3.098	58.3	11.73
Periodic Gaussian	10.963	2.344	48.13	9.09
Square Gaussian	10.926	2.876	56.53	8.4

Table 3.2: Total jitter obtained for BER=1e-14

Jitter Type	DJ Obtained	RJ Obtained	TJ(1e-12)Obtained	Error(%)
Gaussian only	0.514	4.174	61.29	8.97
Dual Gaussian	0.761	4.840	70.79	12.58
Dual Dirac Gaussian	9.379	3.098	54.61	4.66
Periodic Gaussian	10.963	2.344	45.34	2.77
Square Gaussian	10.926	2.876	53.11	1.83

Table 3.3: Total jitter obtained for BER=1e-12

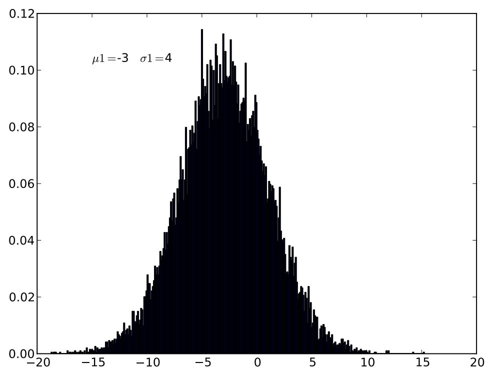
The jitter histograms with lower probability values tend to provide more accurate values after jitter decomposition. This fact is expected due to the intrinsic characteristics of the dual-Dirac method.

Figures 3.2, 3.3, 3.4, 3.5 and 3.6 contain the waveforms used to obtain the results of tables 3.1, 3.2 and 3.3.

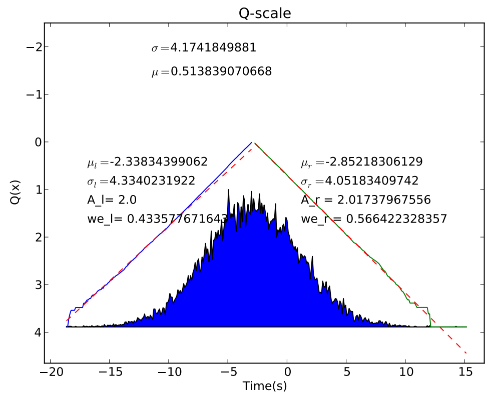
3.5 Conclusion

The extract_jitter tool is very easy to use and provides good results. It is always necessary to understand that $DJ_{\delta\delta}$ is always lower than DJ_{pp}, but $TJ_{\delta\delta}$ is higher than TJ_{pp} for low BER. This tool allows the designers to predict the IC behavior on an early stage or even to reduce the cost with a tool to extract jitter.

The accuracy of the results grows along with the increase of the Q(x) maximum value. This is mainly due to the fact that for low probability values the influence of deterministic jitter is negligible but at higher probability values the Q-scale technique does not provide accurate results. This

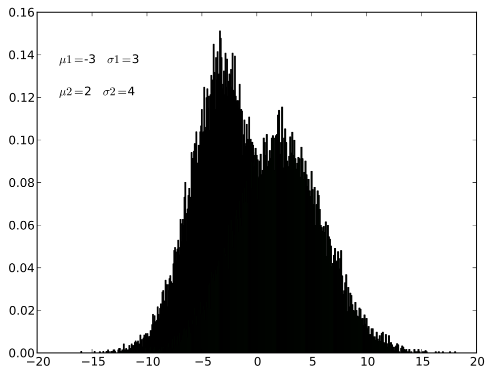


(a) Gaussian jitter distribution

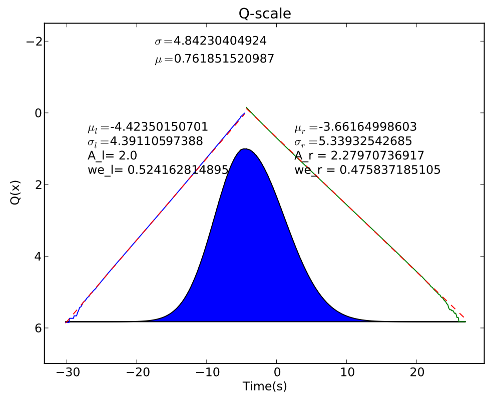


(b) Q-scale

Figure 3.2: Gaussian jitter decomposition using extract_jitter tool

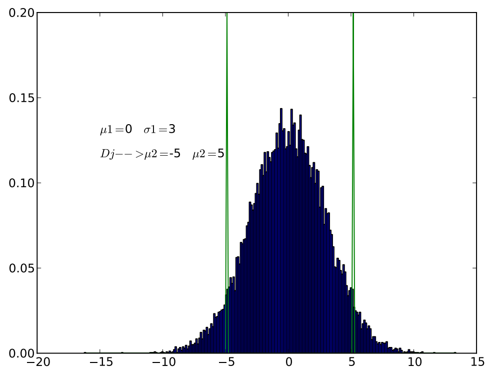


(a) Dual Gaussian jitter distribution

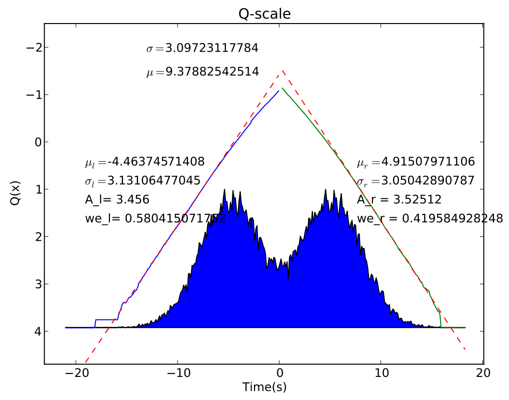


(b) Q-scale

Figure 3.3: Dual Gaussian jitter decomposition using extract_jitter tool

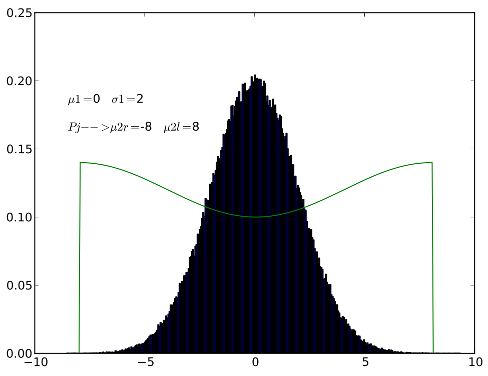


(a) Dual-Dirac Gaussian jitter distribution

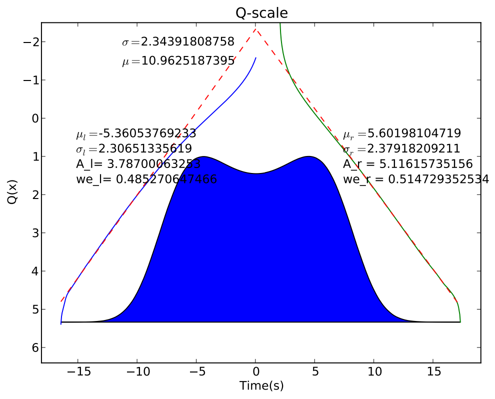


(b) Q-scale

Figure 3.4: Dual-Dirac Gaussian jitter decomposition using extract_jitter tool

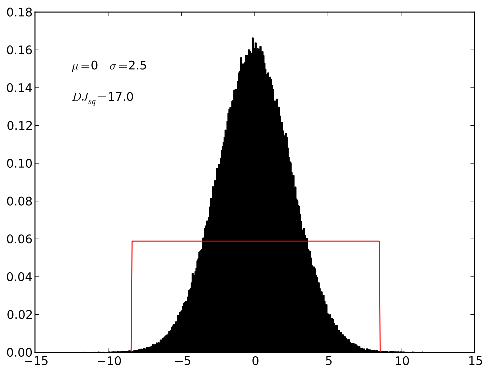


(a) Periodic and Gaussian jitter distribution

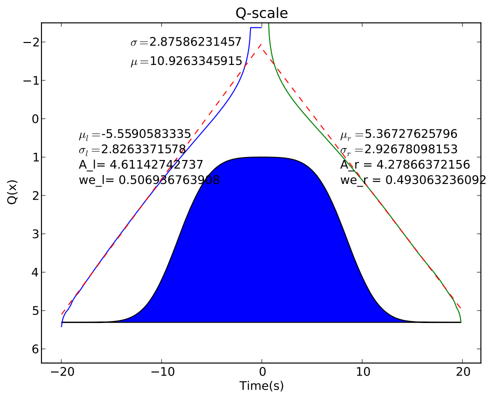


(b) Q-scale

Figure 3.5: Periodic and Gaussian jitter decomposition using extract_jitter tool



(a) Square and Gaussian jitter distribution



(b) Q-scale

Figure 3.6: Square and Gaussian jitter decomposition using extract_jitter tool

happens even when the linearization is correctly performed since deterministic jitter is mixed with Gaussian jitter resulting on overestimated random jitter values (by nature Q-scale overestimates random jitter).

The availability of this software on early stages of the design gives the designers an idea of the system's overall jitter allowing them to take preventive actions and therefore increasing the robustness of the IC.

Chapter 4

Signal Analysis

4.1 Introduction

System level engineers are responsible for the top level blocks definition, following a typical top down approach. New communication standards demand for accurate signal modeling, since jitter budget is decreasing from standard to standard while, in opposition, the frequency of operation is becoming incredibly high.

There are multiple sources of jitter that need to be accurately addressed, during the specification phase by system level engineers, allowing the correct specification of the internal blocks.

Current standards demand high data rates with low BER, reducing the margin for errors. The overall margin needs to take into consideration the technology's internal variations since the same product can be implemented in different technologies.

In terms of signal integrity, the major contributions come from the interconnection layer, between the transmitter and the receiver.

The interconnection layer can vary from a few centimeters of PCB trace to 1/5 meters cables, resulting on a different characteristic of the signals involved.

Turning our attention to a 10G-XAUI interface, it is easier to notice that the cable characteristics will influence the system performance. Since it is very hard to have a constant response of the cable at all frequencies (higher frequencies imply higher attenuation), resulting on very closed eye diagrams after the cable, leading to complex signal recovery algorithms.

Cables represent a significant part of the system signal degradation but, unfortunately, there are many other sources of external degradation like: PCB traces, package, driver bandwidth, interconnection capacitance, cable connectors, temperature, voltage supply variance, to name a few. Such variations will be translated into jitter. There are also internal jitter sources, like the PLL's VCO jitter, reference clock jitter, power supply jitter. All these combined will constrain the maximum data transfer rate between transmitter and receiver circuits.

4.2 Frequency domain to time domain

Frequency and time domains are related. For example, one of the most important properties that relate both domains is the equivalence between convolution in one domain and multiplication in the other.

Frequency domain representation describes the system's response to the input signal's frequency; such representation is obtained by applying multiple sinusoids to the inputs of the system and comparing the output and input signal's in terms of amplitude and phase. Such representation allows the designer to understand, at which frequency, the system will, theoretically, "break". The frequency domain transfer function can be obtained through Fourier transform or by using sophisticated capturing equipment. The Fourier transform decomposes a function into a sum of multiple sinusoidal functions, each with an associated amplitude, phase and frequency, allowing the direct representation on the frequency domain. Fourier transform on the continuous domain is defined by eq. 4.1, while eq. 4.2 represents the Fourier transform for the discrete domain.

$$F(j\omega) \equiv \oint_{-\infty}^{\infty} f(t)e^{-j\omega t}dt \tag{4.1}$$

$$X_k = \sum_{i=0}^{N-1} x_i e^{-j2\pi ik/N} \tag{4.2}$$

The Fourier transform can be applied to generate the sinusoidal coefficients, but it is also possible to do the opposite and obtain $f(t)$ from it is frequency response, using, the inverse Fourier transform. Eq. 4.3 defines the inverse Fourier transform on the continuous domain, while eq. 4.4, on the discrete domain.

$$f(t) \equiv \frac{1}{2\pi}\oint_{-\infty}^{\infty} F(j\omega)e^{j\omega t}d\omega \tag{4.3}$$

$$x_j = \frac{1}{N}\sum_{k=0}^{N-1} X_k e^{j2\pi ik/N} \tag{4.4}$$

Time domain impulse response can then be obtained through the use of inverse discrete Fourier transform (IDFT). IDFT requires $O(N^2)$ operations to complete, thus making it a slow operation, even with today's computers. Fast computing algorithms like FFT and IFFT, are suitable for signal analysis applications, requiring only $O(NlogN)$ operations. Once the impulse response function is obtained, the overall system response can then be obtained by convolving the input signal with the impulse response.

4.2.1 Impulse response based on IFFT - considerations

Frequency domain functions are commonly described for $f = [0 : f_s]$, but the IFFT algorithm receives as input, a function with domain $f = [-f_s : f_s]$, resulting on the need to, first, obtain the function with the desired domain (causality ease this task). Sampling time (t_s) can be obtained from the sampling frequency (f_s) due to the Nyquist theorem, resulting in $t_s = 1/(2 \cdot f_s)$. Impulse

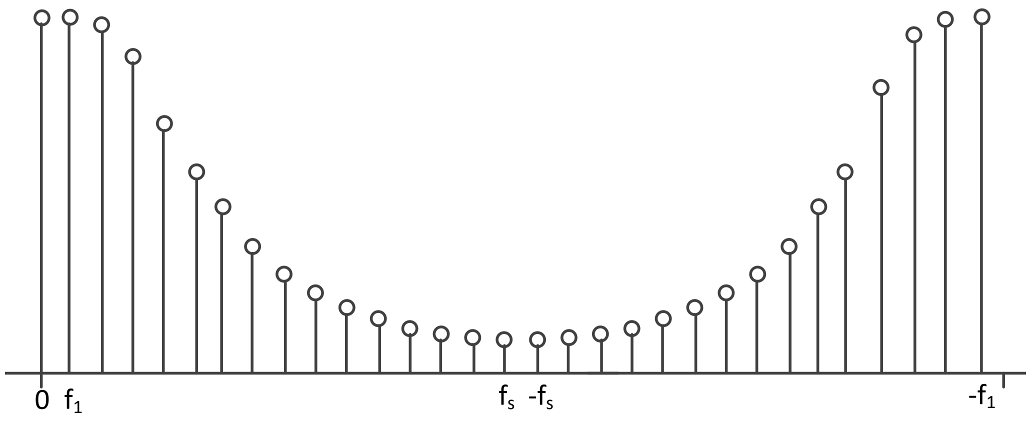


Figure 4.1: FFT amplitude signal vector representation

response's accurate representation depends of f_s. When frequency response is described by an equation, it is easier to extrapolate the frequency response for high f_s values. If it is obtained from lab equipment, the same statement cannot be applied, leading to frequency extrapolation algorithms, based on the causal behavior of the signals under study. The correct description of the possible processes can be a theme for a new thesis, since it is very complex and difficult to obtain. Signal analysis tool performs such extrapolation. A more detailed analysis can be found under chapter 4.3.

Inverse Fast Fourier Transform algorithms, require an input vector with the frequency response ordered on the following order:$[0, f_1, ..., f_s, -f_s, ..., -f_1]$ requiring a previous shift on the frequency response vector (see Figure 4.1).

4.2.2 Causality enforcement

All physical time domain responses are causal, meaning that the signal is non-zero only for times greater than zero [12]. Physical system cannot predict the future. Instead, they react to the input, resulting on an output different than zero only for times greater than the time at which the input signal was applied (causal signal). Signals under analysis are real on the time domain but of complex format on the frequency domain.

Causal signals represented on frequency domain are related by the Hilbert Transform [12] and exhibit Hermitian symmetry, i.e: $X(f) = X^*(-f)$, where "*" denotes the complex conjugate, and the time domain functions are, accordingly, real-valued. Such symmetry should be taken into consideration on the frequency vector generation for IFFT operation.

4.3 Frequency extrapolation

Frequency domain to time domain transformation requires accurate frequency domain representations, maximum frequency will affect the accuracy of the time domain representation since it

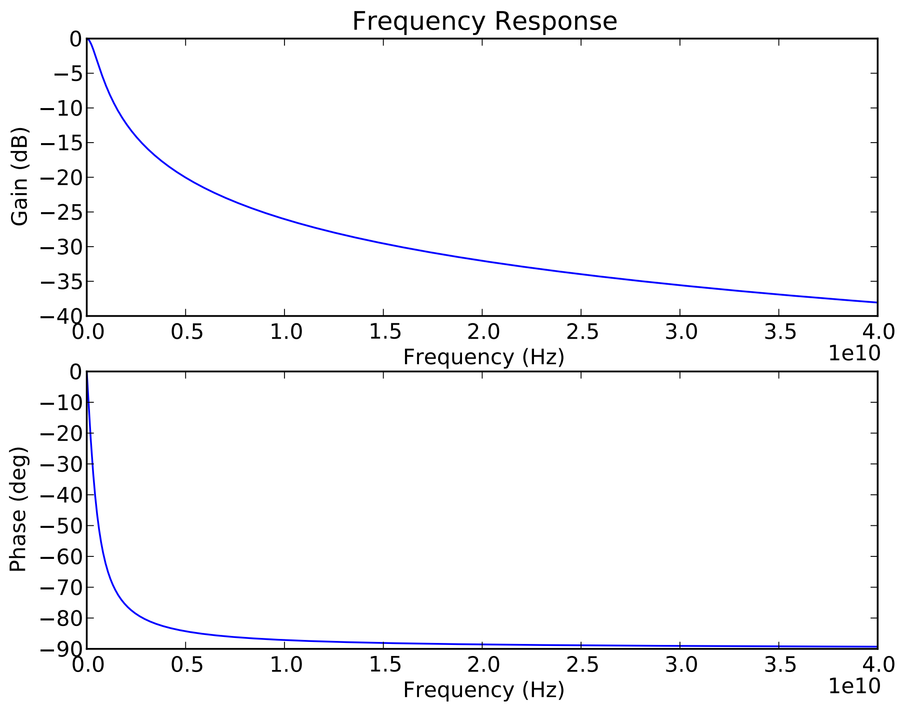


Figure 4.2: Frequency response of signal $e^{-t*500e6}u(t)$

affects time step. To better understand the relation between f_s and time domain accuracy, let's consider a frequency domain representation of the time domain signal $e^{-t*500e6}u(t)$. Using Laplace transform ($F(s) = F(j\omega)$):

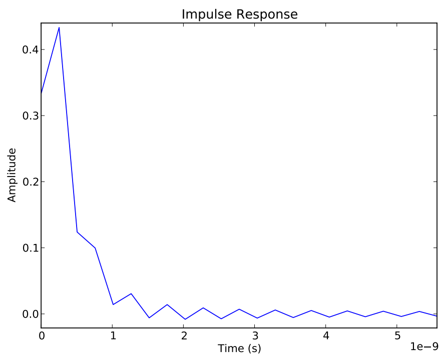
$$F(j\omega) = \frac{500e^6}{j2\pi f + 500e^6} \tag{4.5}$$

Figure 4.2 is in fact a form of a first order low pass filter (as expected), with a cutoff frequency of $500e^6$ Hz. At high frequencies, the amplitude value of the correspondent sinusoid is small, but not zero, leading to accuracy errors when discarded. Figure 4.3 exemplifies the effect of discarding high frequency coefficients. Low f_s values will result in bigger time steps leading to imprecisions on the frequency to time domain transformation.

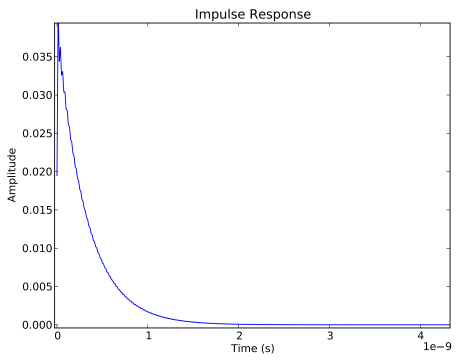
The impulse response cannot be directly compared due to the difference on the time step but, instead, the comparison should be done against exponential signal ($e^{-t*500e6}u(t)$). The impulse response can be converted into the desired signal through convolution with $u(-t)$ signal. Figure 4.4 compares an exponential signal with the convolution signals from impulse response with a maximum frequency of 2GHz and 40GHz, respectively.

System characterization is not done only with equations, since there are systems with huge complexity, which cannot be accurately defined by simple equations. There are other occasions where the system equations are not available at all. For example cable characterization.

Imagine that a designer buys a cable or a set of cables whose frequency response might not be available. The need exists to extract it. Designers typically use network analyzers to obtain



(a) Impulse response ($f_s = 2GHz$)



(b) Impulse response ($f_s = 40GHz$)

Figure 4.3: Impulse response of the time domain exponential signal

the system's frequency response. As discussed above, the maximum frequency used on the characterization will affect the time domain accuracy. Today's equipments can capture signals up to 100GHz, but even that value can be low when compared with the maximum operation frequency of the communication standard that will be addressed with the behavioral model. All these factors result on the need to have models capable of producing accurate frequency extrapolations.

Frequency extrapolation allows the designer to increase the overall accuracy of the model but, unfortunately, such process is not trivial, unless the designer has the knowledge of the system transfer function. On all other cases, some type of frequency extrapolation should be used to increase the accuracy.

Time step accuracy is dictated by f_s. The zero padding method [13] uses this notion to produce the frequency extrapolation. It consists on increasing the number of elements of the frequency response vector, by adding zeros. The number of zeros added to the vector is equal to the frequency step, divided by the difference between the new maximum frequency (f_{ext}) and the vector maximum known frequency (f_s). The basis of this method relates to the low pass behavior of physical systems so, for high frequencies the sinusoidal components coefficients will tend to zero. Signal "Imp_resp(40GHz) - f > 2e9 = 0" on Figure 4.5 was obtained by applying the zero padding method to the frequency coefficients vector with $f_s = $ 2GHz $f_{ext} = $ 40GHz.

Polynomial fitting can also be used to produce the frequency extrapolation but real systems could not produce frequency responses described as perfect lines, leading to high polynomial orders, and a big number of polynomial equations in the case of using multiple polynomial to describe the frequency response. Curve fitting will have to take into consideration the real and imaginary part of the signal, making the fitting process even harder. In [14] the basis for polynomial fitting is presented, the idea is to find a low order polynomial, which fits the magnitude of the frequency response curve, and then through Hilbert transform find the extrapolated real vector value (r_{vector}) that:

$$r_{vector}^2 + H(r_{vector})^2 = Amplitude^2 \tag{4.6}$$

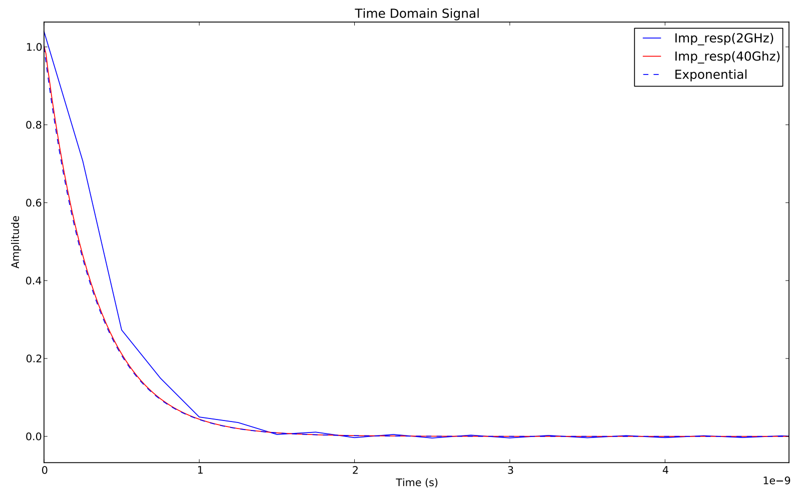


Figure 4.4: f_s effect on time domain representation

Where $H(r_{vector})$ represents the Hilbert transform of the real part of the extrapolated frequency response signal. Such method can only be applied to causal signals, f(t) = 0 t < 0.

Real and imaginary components of the frequency response, in causal systems, are related to each other, such relation is specified by the Hilbert transform. If the designer knows the real part then the imaginary part can be calculated and vice versa using Hilbert transform. This relation is also known as Kramers-Kronig relation.

Polynomial fitting can be hard to implement. On the tentative to bring something new to this thesis, it was developed a new method taking in to consideration the relation between real and imaginary parts of the frequency response through Hilbert transform. Let's call it Hilbert method. Similar methods can be found in [15], [16], [17] and [18].

Proposed Hilbert method is based on [19]. Real and imaginary parts are related by Hilbert and inverse Hilbert transforms. The imaginary part can be obtained through Hilbert transform of real part ($H(r)$), and real part can be obtained through inverse Hilbert transform of imaginary part ($H(i)^{-1}$). It was assumed that the measured frequency response is accurate on the band $]0, f_s]$.

Interactive method can be used to obtain a preliminary frequency response extrapolation. Each iteration starts with a calculation of the imaginary part through $H(r_{vector})$. The obtained result, on the band of $]0, f_s]$, is replaced by the imaginary part of the measured values resulting on i_{vector}. Next action is to obtain the real part by applying inverse Hilbert transform to i_{vector} ($H(i_{vector})^{-1}$). To ensure a correct relation between extrapolated and measured signals, the obtained real part on the band of $]0, f_s]$ is replaced by the real part of the measured values resulting on r_{vector}. This process is repeat about 100 times resulting on $r_{hilbert}$ and $i_{hilbert}$. Initial vectors $r_{hilbert}$ and $i_{hilbert}$ are equal to the measured ones on the band $]0, f_s]$, and zero on the band of $]f_s, f_{ext}]$

The boundary between the measured values and the extrapolated ones should be smooth, $r_{hilbert}$ and $i_{hilbert}$ on the band of $[0.95 \cdot f_s, 1.05 \cdot f_s]$ are replaced by a 4 order polynomial curve fit to reduce the abrupt ups and downs. In order to reduce the high frequency noise in the crossing boundary, between real and extrapolated data, the obtained signal was filtered with a 4 taps

Algorithm	Error	Error_Diff
Equation ($f_s = 40GHz$)	$4.778e-3$	$9.273e22$
Zero padding (from $2GHz$ to $40GHz$)	$2.931e-2$	$2.081e25$
Hilbert fit (from $2GHz$ to $40GHz$)	$8.705e-3$	$-2.216e24$

Table 4.1: Error of different frequency extrapolation methods

low pass filter. The low pass filter with $[0.35, 0.25, 0.20, 0.15]$ as coefficients was applied to $[0.95 \cdot f_s, 1.05 \cdot f_s]$ band. The obtained values inside $[1.05 \cdot f_s, f_{ext}]$ band was replaced by a second order polynomial fit, reducing the high speed variations that are not expected on physical systems.

Figure 4.5 shows the differences between hilbert, zero padding and equation methods. On table 4.1 are described the errors for each method, where the error column was obtained by performing the quadratic error between the expected exponential signal ($Error\_Diff = \sum(expected - obtained)^2$) and the obtained for the different methods. Error_Diff represents the linear error divided by the expected value ($Error\_Diff = \sum(expected - obtained)/expected$). It is interesting to notice that the ideal signal and the obtained signal through hilbert method are closely related.

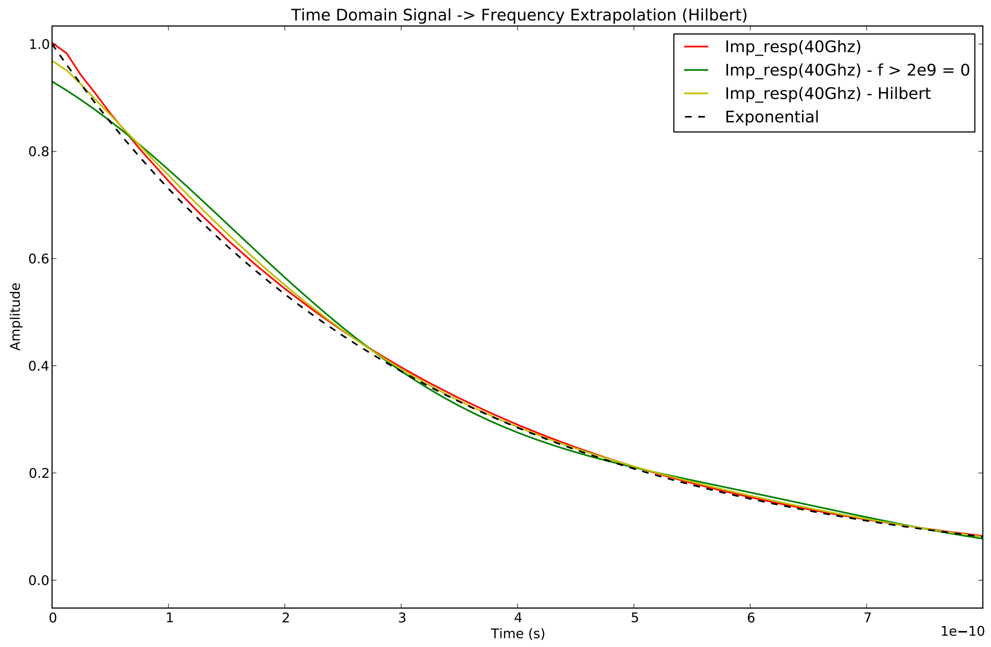


Figure 4.5: Frequency extrapolation (different methods) effect on time domain representation

4.4 Clock generation

Clock signals are the base of work on digital and mixed-signal circuits. Jitter characteristics of clock signals will be present on data signals, which, in turn, are transferred from transmitter to receiver blocks. The ability to correctly characterize clock signals is crucial, for a correct characterization of the entire system.

Clock signals can be generated through a sine function, with phase modulation to better represent the real clock signals. The implemented clock generation algorithm has 4 inputs, which must be defined by the user: frequency (F_s), maximum phase error (A_j), jitter frequency (F_j) and the stop time. Eq. 4.7 describes the implemented clock generation function.

$$Clock = sin(2\pi F_s \cdot t + A_j \cdot 2\pi \cdot sin(2\pi F_j \cdot t)) \tag{4.7}$$

4.5 Edges extraction

Analysis of digital signals relies in accurate edge extraction algorithms, since a great part of the measurements are done over the signal transition points. The use of inaccurate edge detection algorithms result in higher jitter values, incorrect data recovery, incorrect eye diagrams, incorrect rise and fall time measurements, etc.

Analog signals can have more than one transition on the zero differential point, but all these can be related with only one transition from 0 to 1 or 1 to 0 on digital form. Due to this fact the edge detection algorithm needs to have hysteresis, meaning that the threshold on transitions from 1 to 0 is symmetrical to the 0 to 1 transitions. The implemented algorithm allows the user to specify the threshold, resulting on a robust algorithm.

Signal edges extraction is data dependent; the implemented algorithm allows the user to define the type of data: clock or other types. It is necessary to distinguish clock signals from other types of data, since on clock signals the user is only interested on the rise transitions (period information). In generic data signals, the designer is interested on rise and fall transitions since both are valid data states.

4.6 Data generation

Data generation should replicate, as exactly as possible, the analog signal behavior. Usually the analog conversion of digital signals behave as a line near the transition point. Due to this fact, part of the analog behavior can be defined by straight lines. Figure 4.6 represents one possible signal conversion from digital to analog.

Current implementation divides each signal transition in three parts: bottom, middle and top. Since the middle part can be described by a line, it is just necessary to know the start point, end point and time duration to correctly design the middle waveform. Since rise-fall times are measured from 10% to 90%, the middle waveform was designed to map into it. The boundaries of figure 4.6 T_A<->T_B and T_B<->T_C are defined as 10% and 90% of the signal swing respectively. User will provide rise and fall times of the signal for the middle waveform equation to be described by eq. 4.8, where V_{HIGH} represents the High voltage (digital '1'), V_{LOW} represents the Low voltage (digital '0'), $t_{A<->B}$ represents the boundary between T_A and T_B in time and t_{rise} represents the rise

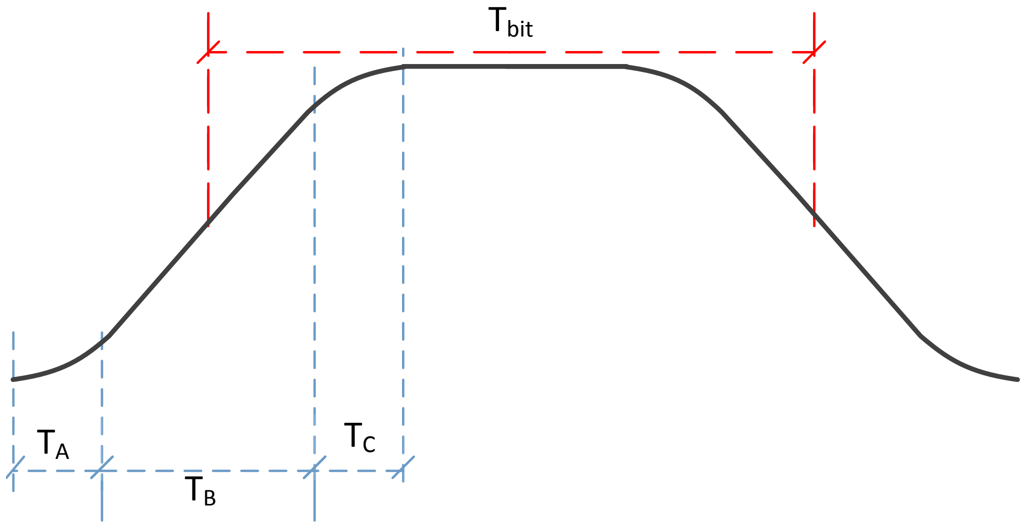


Figure 4.6: Signal generation waveform

time.

$$middle\_waveform(t) = V_{LOW} + (V_{HIGH} - V_{LOW})\left(0.1 + \frac{0.8\,(t - t_{A<->B})}{t_{rise}}\right) \tag{4.8}$$

The non-linear parts (top and bottom) of the analog signal were described as exponential waveforms, with a constant decay, $\tau_{top} = \tau_{bottom} = t_{A<->B}/5$. The transition map into the analog waveform starts at the transition time resulting on a delayed analog waveform in comparison with the digital one. $T_{A<->B}$ boundary time was defined as 3/4 (tfi_tr_rel) of the rise time, ensuring that the exponential waveform has reached the end point. Following the same approach, $T_{B<->C}$ boundary time was defined as $T_{A<->B} + t_{rise}$, the waveform transition modulation effect ends at $T_{B<->C} + tfi\_tr\_rel \cdot t_{rise}$. Eq.s 4.9 and 4.10 describe the waveform generation formula for bottom and top parts. For t (time) higher than $t_{rise}(1 + 2 \cdot tfi\_tr\_rel)$ the waveform is equal to V_{HIGH} till the next transition, the time distance between bits (Bit time - t_{bit}) is directly extracted from the provide clock. The presented formulas are applied to digital transitions from '0' to '1'. Transitions from '1' to '0' follow the same approach with the necessary conversions.

$$bottom\_waveform(t) = V_{LOW} + 0.1 \cdot (V_{HIGH} - V_{LOW})\,exp\left(\frac{t - T_{A<->B}}{\tau_{bottom}}\right) \tag{4.9}$$

$$top\_waveform(t) = V_{HIGH} - 0.1 \cdot (V_{HIGH} - V_{LOW})\,exp\left(\frac{T_{B<->C} - t}{\tau_{top}}\right) \tag{4.10}$$

The analog conversion needs digital data to produce analog waveforms. Current implementation uses the clock edges transitions to generate a digital vector with the same length, and then it is just necessary to feed the digital to analog conversion block with the digital vector and clock edges vector. In order to support as much as possible current communication standards, the implementation allows the user to produce the following data types: Clock and PRBS (Pseudo Random binary sequence)- also known as LFSR (linear feedback shift register)- signals. Since there are different

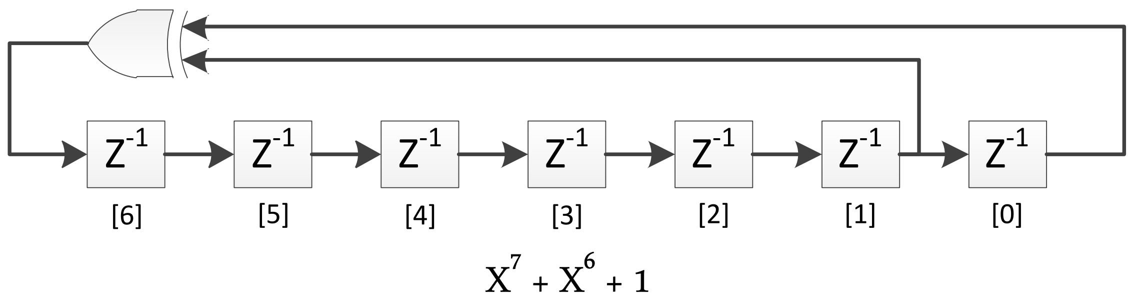


Figure 4.7: PRBS mod7 block diagram

types of PRBS, the user can select one of the following types: mod 5, mod 7 (good approximation for 8b/10b encoded data), mod 15, mod 23 (good approximation for TMDS encoded data), and mod 31. Each type is characterized by a maximum run length ($2^N - 1$, where N is the PRBS mod), a maximum number of consecutive ones (N) and a maximum number of consecutive zeros (N-1). Figure 4.7 represents the block diagram of a PRBS7 sequence.

PRBS sequences are good options to use in channel characterization, since the implementation results on a simple logic, reducing the timing constraints issues in typical IC design flow (a single xor gate between flip-flops). The resulting ISI is similar to a signal generated from an encoder, with the same number of consecutive equal bits. The PRBS generation requires an initial valid state, typically, it is chosen the all ones state as initial point for the flip-flops, represented on figure 4.7 as simple unitary delays.

Channel attenuation is sometimes compensated on transmitter side using pre-emphasis, meaning that during one bit period the signal amplitude is increased, but rise and fall times still the same (slew-rate increases). The idea behind pre-emphasis is to increase the gain at high frequencies. Due to that reason, the bits equal to the previous one will return to the typical analog value (V_{HIGH} or V_{LOW}).

Sometimes, there is interference from adjacent channels or some jitter noise is coupled to the output signal. The signal generation algorithm also allows the modulation of these factors. Such jitter will affect the amplitude of the analog converted signal, in order to make the implementation of such phenomenon; the user can define deterministic and random jitter values, which will be directly applied to the waveform voltage generation.

4.7 Convolution

One of the key aspects of signal analysis, is the capacity to accurately reflect the frequency response models into the discrete time waveforms. On chapter 4.2 it was described the technique to obtain an impulse response from the system's frequency response. The impulse response represents the system's response to a "Dirac signal"[1]. The analog waveforms described on the time

[1]The use of Dirac sentence refers to the analogy, between a pure Dirac function and single event function with unitary amplitude, and a width of a time step.

domain can be described by multiple "Dirac functions". From this sentence, it is easy to understand that the best way to represent the system effect is through convolution.

The analog signal and impulse response should have the same time step, since the discrete convolution requires the same time base on both signals. Signals under study are causal, i.e., the output $y(t_0)$ only depends on the input $x(t)$ for values of $t < t_0$,the system reacts to applied signals not predicting future events. Both waveforms (impulse response and analog signal) have to be equal to zero for $t < 0$. Due to this fact both waveforms are described only for times greater than zero. Resultant convolution vector will have a number of elements equal to the sum of the number of elements of the input signals (analog signal and impulse response), but should only be considered the elements between the position zero and position equal to the length of the analog signal.

4.8 Clock recovery unit

Clock recovery is very important for channel characterization since without this algorithm, the jitter values obtained by the developed tool will be very different from the real ones present at the CDR circuit.

All signals, inside digital circuits, are generated from a clock source on the transmitter side; such clock can be shared between transmitter and receiver, allowing an easier clock extraction. Some communication standards do not have a separate channel to transmit the clock signal (like USB standard), resulting on a more complex receiver circuit, since it will be necessary to extract the clock from the transmitted data signals. On this type of circuits, the clock frequency bandwidth is very limited, because without such a tight constraint, it would end up with very complex circuits.

Systems that do not share the clock signal between transmitter and receiver have to use data encoder algorithms that guarantee a sufficient number of transitions for clock recovery. For example, USB3.0, operating at 5Gbps, uses a 8b/10b encoder algorithm to ensure a maximum number of equal consecutive bits of 5.

Clock sharing interfaces typically support a wide range of frequencies, for example: HDMI standard supports frequencies from 25MHz to 340MHz. Since the clock is shared between the transmitter and receiver circuits, the frequency range is easily extracted on the receiver side. In terms of data encoding algorithms, these type of standards can have a lower number of transitions, higher number of equal bits transmitted, thus reducing the electromagnetic interference in adjacent channels but, effectively, increasing the ISI jitter. Typically, these type of systems tend to have higher deterministic jitter values.

The clock recovery unit - sometimes called "golden PLL" [20] - is used to accurately extract the golden clock from the signals under analysis, modeling the effect of ideal PLL circuits; this means that no jitter is added to the system due to the use of a "golden PLL". Instead, the use of real PLL circuits can introduce random and deterministic jitter components to the overall system jitter. CRUs are used on the characterization of channels, ensuring that the extracted jitter comes from the channel and not from the equipments themselves.

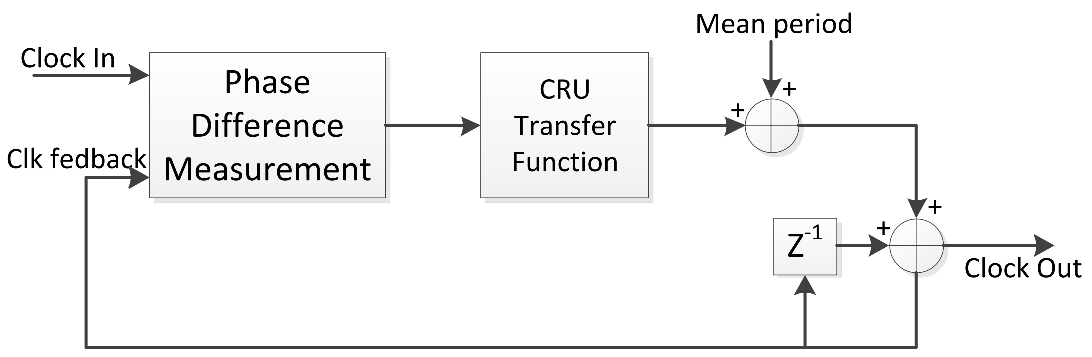


Figure 4.8: Clock recovery unit block diagram

The CRU receives as input a clock signal (Clock In), or a data signal with an embedded clock. It outputs a clock recovered signal (Clock Out). At each rising edge of Clock In it is measured the difference between the rising edge time location of Clock feedback (same as Clock Out), which represents the signals' phase difference. It is, then, used to increase or decrease the period of Clock Out, following a typical negative feedback loop compensation system. Phase difference value is not directly summed to the mean period (golden period), it is necessary to reflect the CRU transfer function on this process. Phase compensation on a real circuit does not occur on the next cycle. The CRU transfer function (CRU-TF) is represented on the frequency domain, since the CRU works on the time domain, it is necessary to represent the CRU-TF as an impulse response with points on the time domain separated by mean period. Phase difference value is then "convolved" with the CRU-TF impulse response, the resultant value is them added to the mean period resulting on a new clock period.

CRU has to provide an indication of the Clock Out edge position, it is necessary to add the current period to the previous edge location of Clock Out generating on this way, a clock recovered signal. Figure 4.8 represents the CRU block diagram.

4.8.1 Clock recovery unit transfer function considerations

A clock Recovery unit can be described as a low pass filter, since the transfer function can be limited in frequency, due to the attenuation at high frequencies the resultant impulse response can also be limited in terms of time. Taking this fact in consideration, one of the solutions is to implement such transfer function as a FIR (Finite Impulse Response) filter. Coefficients for the FIR filter can be obtained through undersampling of the impulse response obtained from the CRU-TF (see figure 4.9). The undersampling can be done by adding the hits contained inside of each time step equal to the mean period of Clock In.

4.8.2 Plesiochronous system

In a plesiochronously communication system transmitter and receiver operate at the same nominal frequency but with small frequency variations [21]. USB3.0 is an example of plesiochronous

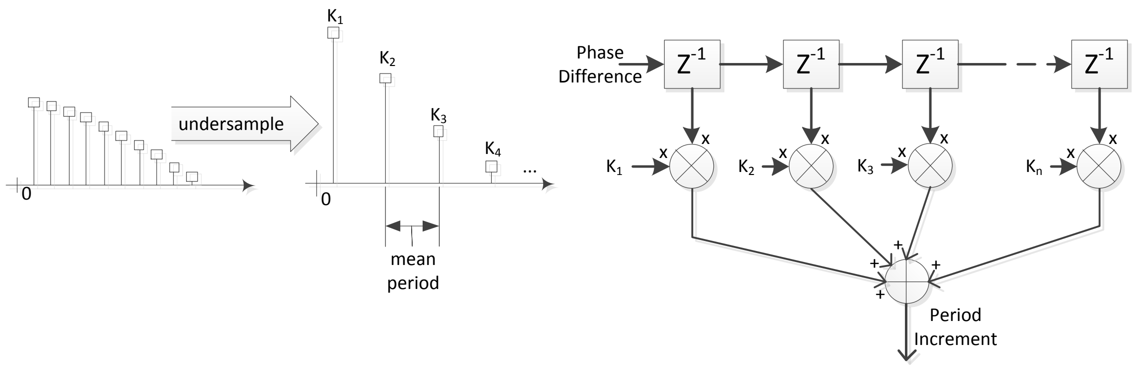


Figure 4.9: Clock recovery unit transfer function block diagram

systems, where the absence of a dedicated clock connection between transmitter and receiver, reduces the interconnection costs.

Transmitter and receiver can have a frequency mismatch which, although small, might lead to errors if not taken in to account. Systems intended to support such feature, must be developed to ensure the availability of a clock recovering scheme based on the transmitted data. Data codification algorithms must be used to ensure the availability of "clock synchronization periods" separated by a few number of bits. Support for plesiochronous clock recovery requires a different, although similar, algorithm to the presented above. The phase error cannot be calculated for all clock feedback pulses since the input is not a clock signal but a data signal. The mean period cannot be obtained through the average of the input signal period, it must be provided to the CRU algorithm.

The implemented algorithm supports two different approaches regarding phase error calculation. Phase error is calculated for input edges, resulting in a period increment after the application of the CRU-TF. The period increment at the input signal transitions can then be added to the mean period on the next iterations, till a new transition appears on the input signal (mode 0). Or it can be added only at the input signal transition iteration and the subsequent iterations will suffer from zero period increment, till the availability of a transition on input signal (mode 1). It is Important to notice that CRU-TF effect is only performed at the iterations with transitions on input signal, modeling the real behavior of a clock recovery circuit designed to support plesiochronous interfaces. Is up to the user the selection between algorithms.

4.9 Cable model

Current commercial interface products like USB3.0 require a very well characterized interconnection medium, typically cables, whose specification is also well defined, just like the communication standard itself.

Cables are becoming more and more important for the system performance, even using digital signaling. Errors can always occur but, bad cables will result in lower transmission rates which,

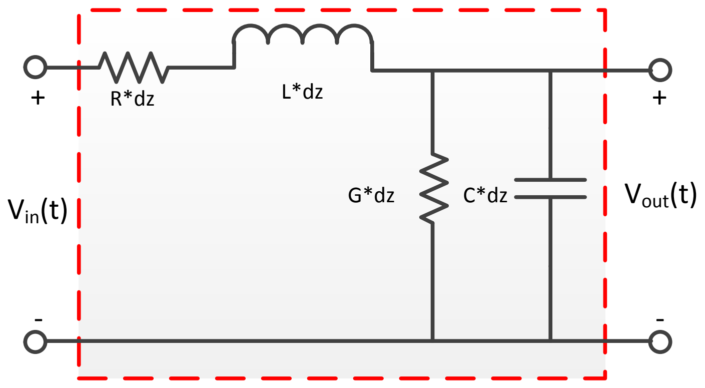


Figure 4.10: Cable lumped devices model

for example, in USB and, HDMI interfaces can result in image flashing, loss of horizontal synchronism, due to transmission errors.

Costumers tend not to pay attention to the cable, resulting on an even higher pressure to the TV maker companies, because in case of bad image quality due to the cable characteristics, the consumer will change the TV... not the cable. This fact leads the receiver systems to have to support different types of cable, with different type of frequency responses. This requisite demands for cable characterization tools.

Figure 4.10 represents an equivalent lumped element circuit [22, p. 56-57] for cable characterization (rectangle in red), where dz represents the cable element length, R represents the series resistance per unit length (Ω/m), L represents the series inductance per unit length (H/m), G represents the shunt conductance per unit length (S/m), and C represents the shunt capacitance per unit length (F/m). Long cables can be described by multiple equivalent circuits connected in series.

The cable model described above needs to be converted to a form of impulse response, allowing the use of the convolution, to replicate the cable influence on the signal being transmitted. The idea is to convert the lumped elements to a frequency response waveform and then extract the impulse response.

ABCD parameters can direct map the lumped elements into a frequency response. The ABCD matrix is defined for a two-port network in terms of the total voltages and currents as shown in Figure 4.11, can be described as:

$$V_1 = A_1 V_2 + B_1 I_2,$$
$$I_1 = C_1 V_2 + D_1 I_2$$

or in matrix form as

$$\begin{bmatrix} V_1 \\ I_2 \end{bmatrix} = \begin{bmatrix} A_1 & B_1 \\ C_1 & D_1 \end{bmatrix} \begin{bmatrix} V_2 \\ I_2 \end{bmatrix} \tag{4.11}$$

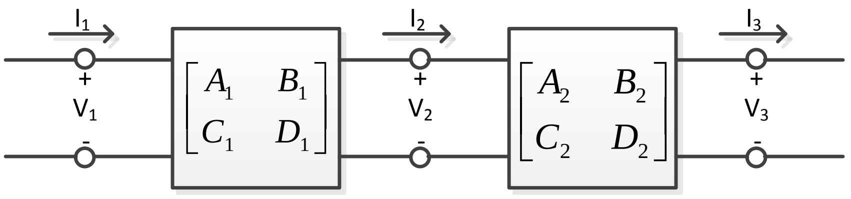


Figure 4.11: A cascade connection of two-port networks

In the cascade connection of two two-port networks as shown on Figure 4.11:

$$\begin{bmatrix} V_1 \\ I_2 \end{bmatrix} = \begin{bmatrix} A_1 & B_1 \\ C_1 & D_1 \end{bmatrix} \begin{bmatrix} V_2 \\ I_2 \end{bmatrix} \tag{4.12a}$$

$$\begin{bmatrix} V_2 \\ I_2 \end{bmatrix} = \begin{bmatrix} A_2 & B_2 \\ C_2 & D_2 \end{bmatrix} \begin{bmatrix} V_3 \\ I_3 \end{bmatrix} \tag{4.12b}$$

Substituting (4.12b) in (4.12a) gives

$$\begin{bmatrix} V_1 \\ I_2 \end{bmatrix} = \begin{bmatrix} A_1 & B_1 \\ C_1 & D_1 \end{bmatrix} \begin{bmatrix} A_2 & B_2 \\ C_2 & D_2 \end{bmatrix} \begin{bmatrix} V_3 \\ I_3 \end{bmatrix} \tag{4.13}$$

Equation 4.13 shows that the ABCD matrix for cascade connection of the two networks is equal to the product of the ABCD matrices representing the individual two-ports. Note that the order of multiplication of the matrix must follow the order in which the networks are arranged, once commutativity is not a property of matrix multiplication.

The cable's ABCD matrix is given by eq. 4.14. Such matrix was obtained from ABCD conversion matrix table [22, p. 205-208]. It is just necessary to convert the matrix to a frequency response transfer function. Frequency response transfer function assumes that V_1 is the voltage value at the terminals of a voltage source with source impedance equal to $Z0$, resulting on eq. 4.15
.

$$\begin{bmatrix} 1+(R+j\omega Ldz)\cdot(Gdz+j\omega Cdz) & (R+j\omega Ldz) \\ Gdz+j\omega Cdz & 1 \end{bmatrix} \tag{4.14}$$

$$cable_{TF}(\omega) = \frac{2}{A+B/Z0+C\cdot Z0+D} \tag{4.15}$$

4.9.1 PCB FR-4 model

Interface circuits are connected using PCB or cables, depending of the distances involved, but even on connections with cables a few centimeters are done via PCB, it is necessary to connect the integrated circuit to the cable connector.

FR-4 PCB Laminate is the most commonly used base material for printed circuit boards. The "FR" means Flame Retardant (to UL94V-0), and Type "4" indicates woven glass reinforced epoxy resin.

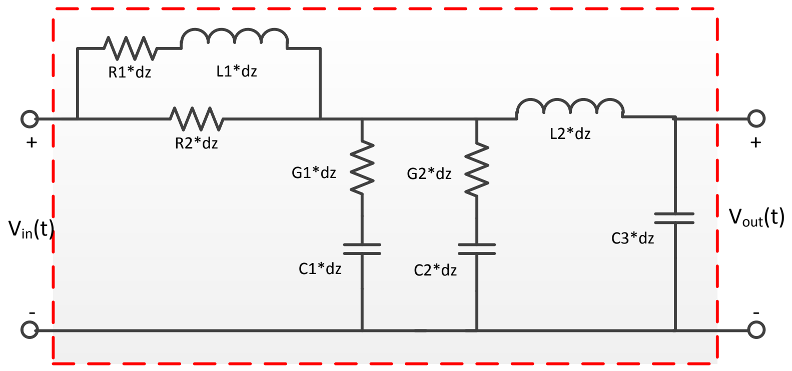


Figure 4.12: PCB FR4 lumped model

FR-4 PCB's are standard, making possible the creation of a model that is more or less independent of the supplier. An equivalent circuit is presented in figure 4.12, such model was presented on [1] using 1 inch as unit length. One key aspect on PCB designs is the characteristic impedance, $Z_0 = \sqrt{L/C}$, that will be tuned to match the load impedance. Such tuning is done by changing the coper traces width (w). L and R are inversely proportional to w ($L,R \propto 1/w$), in opposition C and G are proportional to w ($C,G \propto w$), resulting on $Z_0 \propto \frac{1}{w}$.

Element	$Z_0 = 50$	$Z_0 \neq 50$
R1	$5.91\ \Omega/m$	$5.91\ \frac{Z_0}{50}\ \Omega/m$
L1	18.5 nH/m	18.5 $\frac{Z_0}{50}$ nH/m
R2	$219\ \Omega/m$	$219\ \frac{Z_0}{50}\ \Omega/m$
G1	$19.7\ m\Omega^{-1}/m$	$19.7\ \frac{50}{Z_0}\ m\Omega^{-1}/m$
C1	7.872 pF/m	7.872 $\frac{50}{Z_0}$ pF/m
G2	$0.394\ \Omega^{-1}/m$	$0.394\ \frac{50}{Z_0}\ \Omega^{-1}/m$
C2	3.15 pF/m	3.15 $\frac{50}{Z_0}$ pF/m
L2	$3.04\ nH/m$	$3.04\ \frac{Z_0}{50}\ nH/m$
C3	1.22 pF/m	1.22 $\frac{50}{Z_0}$ pF/m

Table 4.2: PCB FR-4 lumped elements value per unit length of equivalent model

Table 4.2 specifies the model elements value per meter referring to figure 4.12 with $Z_0 = 50$ and $Z_0 \neq 50$ cases. To use this model with the cable model defined before, it is necessary to convert the lumped model to an ABCD matrix making the cascading of PCB with cable effects done through multiplication of the correspondent ABCD matrixes. Eq. 4.16 describes the equivalent ABCD

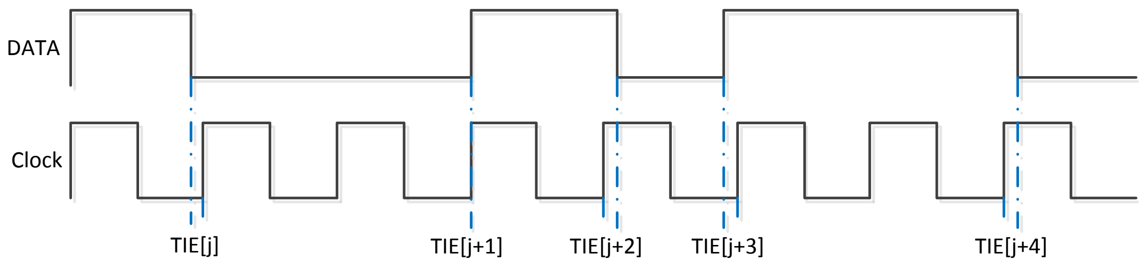


Figure 4.13: Time interval error

matrix for PCB FR-4 simulation model.

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix} = \begin{bmatrix} 1 & \frac{(R1+j\omega L1)\cdot R2}{R1+j\omega L1+R2} \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ \frac{1}{1/G1+1/j\omega C1} & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ \frac{1}{1/G2+1/j\omega C2} & 1 \end{bmatrix} \begin{bmatrix} 1 & j\omega L2 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ j\omega C3 & 1 \end{bmatrix} \tag{4.16}$$

4.10 Zero pole gain frequency domain representation

Frequency domain characterization can also be done by zero-pole functions. There are cases where the designer has a clear picture of the system frequency transfer function. For those cases it will be helpful to provide a way of representing such system, allowing the extraction of the impulse response.

Equation 4.17 describes the zero pole gain function implementation on the frequency domain, where k represents the gain, z_i transfer function zeros, p_j transfer function poles with $s = j \cdot 2\pi f$.

$$ZPK_{TF}(f) = k\frac{\prod_{i=1}^{N}(s-z_i)}{\prod_{j=1}^{M}(s-p_j)} \tag{4.17}$$

4.11 Time interval error measurement

Digital interfaces have to be compliant with jitter maximum values for current communication standards. One of the common techniques used to extract such jitter values is based on Time Interval Error measurements (TIE) - as represented on Figure 4.13. TIE represents the difference between the data edge position in relation to the ideal edge position, resulting on complex algorithms for the ideal edge position extraction (CRU).

Ideal edge positions can be extracted from the clock channel or data pattern using a CRU algorithm. It is important to notice that different CRU transfer functions will result in different TIE probability density functions (PDF), leading to different jitter values.

The TIE measurement is performed for all transitions on data signal, resulting on a TIE vector. This vector can then be converted into a PDF histogram. The requirement for a PDF histogram come from the jitter tool presented in chapter 3, since it should be used to extract the jitter values in terms of random jitter, deterministic jitter and total jitter for a given BER.

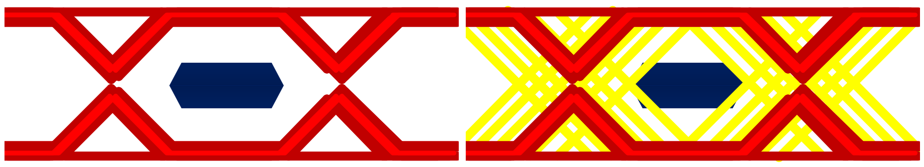

(a) Correct eye diagram generation (b) Incorrect eye diagram generation

Figure 4.14: Eye diagram examples

TIE measurement is not trivial, since between two consecutive data edges can occur multiple ideal edges (clock). Due to this fact, TIE vectors cannot be generated by the difference of data edges position vector and ideal edges position vector. To overcome this fact TIE error measurement should first detect the data edge position and then, the ideal clock edge position within half period of the ideal edge position period. TIE is then obtained subtracting the data edge position by neighbor clock edge position, the resultant value is a measurement of time.

The described procedure should then be performed to all data edges, resulting on a TIE vector with a length equal to the number of data edges.

4.12 Eye diagram generation

Eye diagram characterization is the most important measure in terms of signal integrity, that a designer can have. It is possible to analyze from a single waveform a huge number of parameters like: jitter, rise/fall times, voltage swing, intra-pair skew, etc. Making possible to represent a set of analog signal considerations through a single waveform.

Eye diagram generation requires a good knowledge of the system under test, since it will be very dependent of the ideal clock signal generation. Major task is the extraction of the ideal clock edge locations, since all bits will be mapped into a "single bit" (or two in some cases). The designer can notice in case of wrong clock extraction schemes the eye diagram obtained waveform can be complectly closed (see figure 4.14).

The eye diagram generation algorithm should include a CRU to generate the ideal bit period, depending on the system; ideal clock can be extracted from a clock channel or from the data under analysis. Eye diagram plots are, typically, designed to plot two bit times, since on this way the designer can have a clear picture of the signal integrity prior and after the transition points. Some algorithms also start to plot from -0.5UI (UI- Unit interval, same as bit time) to 1.5UI. The implemented algorithm follows these two guidelines.

Current implementation uses a matrix as base for the eye diagram generation, each two bits will be mapped into a 360x180 matrix (see Figure 4.16). To fill the matrix the eye diagram algorithm performs:

1. Extract the ideal clock edge positions using the CRU

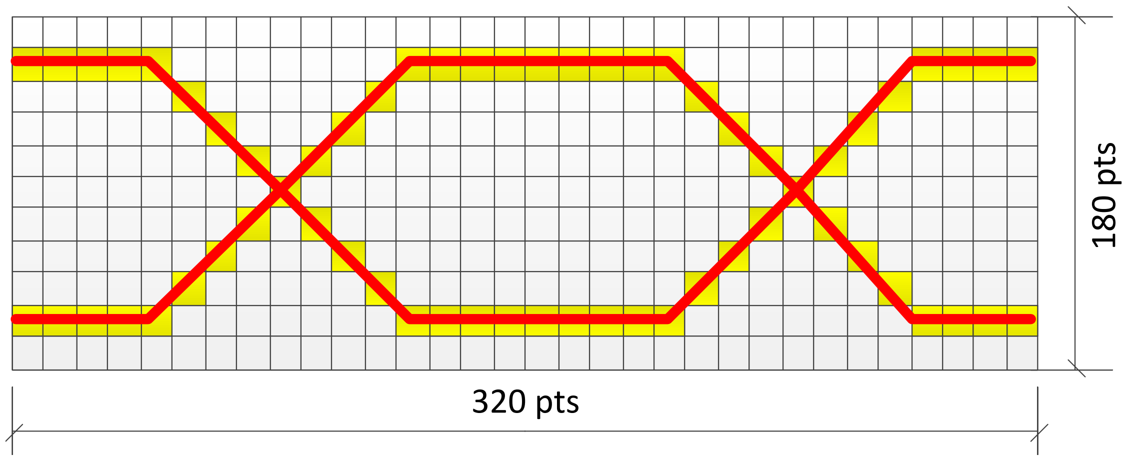


Figure 4.15: Eye diagram matrix creation

2. Collect the start time, matrix x point 1, and end time, matrix x point 320, from the ideal clock edge location period (clock_vec[]). Start time is equal to clock_vec[n], end time is equal to clock_vec[n+2]

3. Find on the data vector the points on the time interval described by clock_vec[n] to clock_vec[n+2] (data_ch[start_t] to data_ch[end_t])

4. Map the data channel analog points into the eye diagram matrix, with matrix x point 1 equal to data_ch[start_t] and matrix x point 320 equal to data_ch[end_t]. Each x point (time) on the eye diagram matrix will have a associated y point representation the channel analog value for the correspondent time, since this algorithm was designed assuming differential signals, y point 1 represents the maximum positive voltage, y point 180 represents the minimum negative voltage (both values are defined by the user).

5. Increments n by 2 and repeats the described procedure until all analog values are mapped

6. Eye diagram matrix points represent the number of times where the signal crosses that point (time and voltage)

Eye diagram also have an eye diagram mask, representing the minimum eye opening. Eye mask varies from standard to standard; the user has to provide the eye mask vertices points to the implemented algorithm.

The mask should be centered on the eye diagram opening (not equivalent to geometrical center). The implemented algorithm searches for the opening on eye diagram where the eye diagram mask can be placed, without touching the channel analog data boundaries.

Eye diagram opening can be detected by a sweep across the x axis and checking the start point where the mask points does not touch the channel data signals representation as shown in Figure 4.16. The eye diagram mask is then positioned with the start point equal to the mean of the valid positions.

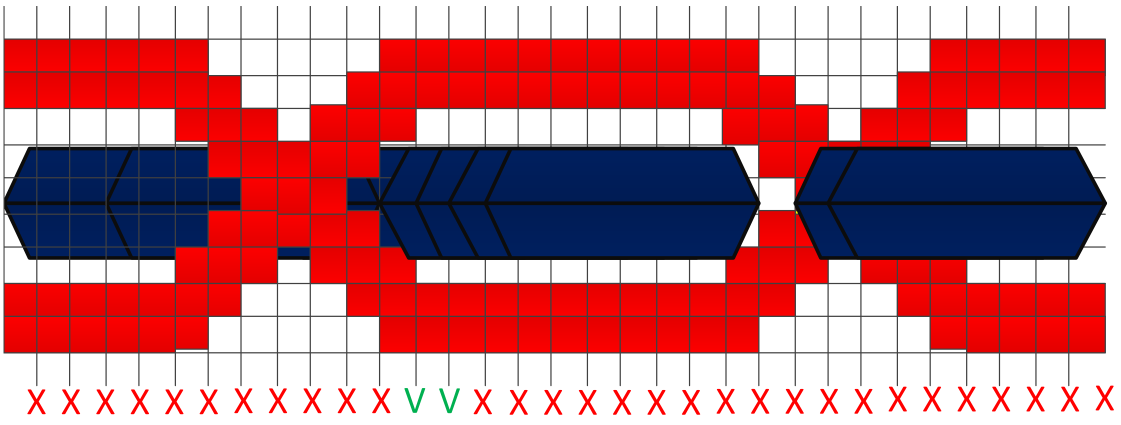

X X X X X X X X X X X V V X X X X X X X X X X X X X X X X X X X X

Figure 4.16: Eye diagram mask positioning detection

4.13 Analog parameters extraction

Current communication standards define a set characteristics for the interconnection signals, these are the most common: minimum rise time, minimum fall time, minimum voltage representing a digital '1' (VIH), maximum voltage representing a digital '0'(VIL).

In terms of digital to analog conversion signals, the eye diagram plot provides enough data to obtain such values. The implemented algorithm perform a sweep across the voltage points (y axis) for the middle of the eye (0.5UI - point 160) and them the maximum and minimum values can be obtained through the following equations:

$$VIH = max(eye\_matrix(160,1:90) > 0) \tag{4.18}$$

$$VIH_{mean} = \frac{\sum_{n=1}^{90} eye\_matrix(160,n) \cdot n}{\sum_{n=1}^{90} eye\_matrix(160,n)}$$

$$VIL = min(eye\_matrix(160,91:180) > 0) \tag{4.19}$$

$$VIL_{mean} = \frac{\sum_{n=91}^{180} eye\_matrix(160,n) \cdot n}{\sum_{n=91}^{180} eye\_matrix(160,n)} \tag{4.20}$$

Current communication standards try to reduce the electromagnetic interference, some standards use spread spectrum clock signals reducing the power at the central frequency others also include restrictions on slew-rate, leading to less interference in adjacent channels once the power at the central frequency is reduced. Almost all current communication standards constraint the minimum rise and fall times, resulting on a bounded frequency spectrum.

The implemented algorithm supports the measurement of rise and fall times, it is common to define a maximum number of evaluate transitions; on current implementation such value was defined as 50000 transitions. The idea is to look for each transition and calculate the time needed to achieve 80% of the final value (differential), the measurement starts when the signal reaches 20% of the final value (differential). The detection of initial value and final value can be done by checking the value at -0.5UI of the transition point as the initial voltage value, and the final value as the analog value 0.5UI after the transition. The bit time is obtained from the ideal clock.

4.14 Eye diagram extrapolation

Eye diagram generation can be performed on simulation and lab data. Major difference between them is the time needed to achieve the same data points. For instance 1 minute of data acquisition on lab can represent months of simulation time. Due to the impracticable duration of such simulations there is the need for a cross relation method, witch correlates simulation with lab results, or in other words, an algorithm that based on lab parameters complement the information obtained from simulation, resulting on a better match between simulation and lab results.

New products tend to use old sub-blocks mixed with new ones, reducing the risk. The combination allows the designers to built-in on new products data from past products, for example: It is very difficult to estimate random jitter in simulation environment, due to the lack of information inside the transistor models (technology nodes under development), but since a great part of the sub-blocks are being reused, it can be assumed that in terms of random jitter the behavior will be similar, allowing the designer to include that information on the simulation environment, leading to more accurate simulation results.

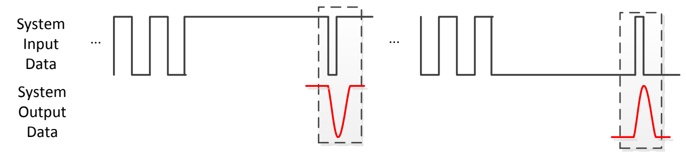
Statistical eye or stateye [10] as it is also known, is a tool capable of representing statistical eye diagrams. Eye diagram extrapolation algorithm implemented on the software tool follows a similar approach, but instead of using the worst case response of the channel it uses the eye diagram information. Signal analysis tool does not have the capability to generate data to the system under test, it is just possible to process the acquired data.

Worst case response defined on Stateye [10] represents the ability of a system to tolerate ISI. The maximum ISI can be obtained by applying the maximum allowed consecutive number of ones, followed by a single zero and then a long sequence of ones. Since the worst ISI could be on transition from a long sequence of zeros to a single one, then it is necessary to put the system on a intermediate state, which can be done by applying a long sequence of '1100' pattern. The process is then repeated for the maximum number of zeros allowed. With these values in hand it is necessary to apply the random jitter effect to produce an accurate worst case condition of jitter/eyediagram representation for a given BER. Figure 4.17 represents the major stateye generation steps.

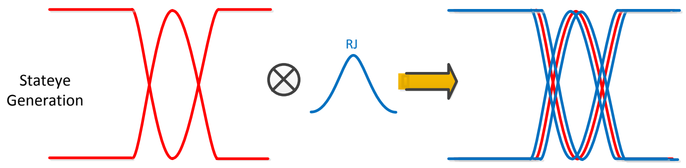
Eye diagram extrapolation algorithm takes advantage of the eye diagram design algorithm (Eye diagram should be created prior); instead of using a predefined captured pattern, the base is the eye diagram itself. Typically it would be needed to send a pattern that produces the worst ISI condition.

The captured data is processed in terms of jitter and eye diagram, such values can then be used to perform the eye diagram extrapolation to the desired BER.

Jitter analysis tool provides two important values: random jitter and deterministic jitter, the last one does not change with BER, resulting on a static influence when enough data is captured. Random jitter follows a Gaussian distribution, allowing a jitter representation on a form of probability density function (PDF), giving the ability to estimate a maximum jitter value for a given BER.



(a) Stateye data acquisition



(b) Stateye waveform plot

Figure 4.17: Stateye waveform generation flow

Figure 4.18 represents the concept described above, the observed jitter value (total jitter observed) on the capture data is the sum of deterministic jitter with the observed random jitter. Eye diagram extrapolation can then be performed by convolving the obtained eye diagram with the random jitter PDF for the desired BER.

Eye diagram extrapolation is obtained through convolution of eye diagram with RJ distribution; due to this fact part of the total random jitter is actually represented on the eye diagram that serves as input. In order to create a correct representation, the observed random jitter should be removed from the random jitter PDF, which will be used on the convolution. Eq. 4.21 describes the maximum jitter value that should be used on the generation of the RJ PDF, where N_{Bits} represents the number of bits used to create the eye diagram waveform, where σ_{ext} represents the extrapolation random jitter and σ_{meas}, represents the random jitter value extracted with the internal jitter decomposition tool.

$$max(RJ) = 2 \cdot \sqrt{2} \cdot erf^{-1}(1 - 2 \cdot BER) \cdot \sigma_{ext} - 2 \cdot \sqrt{2} \cdot erf^{-1}(1 - 2 \cdot \frac{1}{N_{Bits}}) \cdot \sigma_{meas} \ (s) \tag{4.21}$$

Eye diagram extrapolation waveform will follow the same plot algorithm that is used on eye

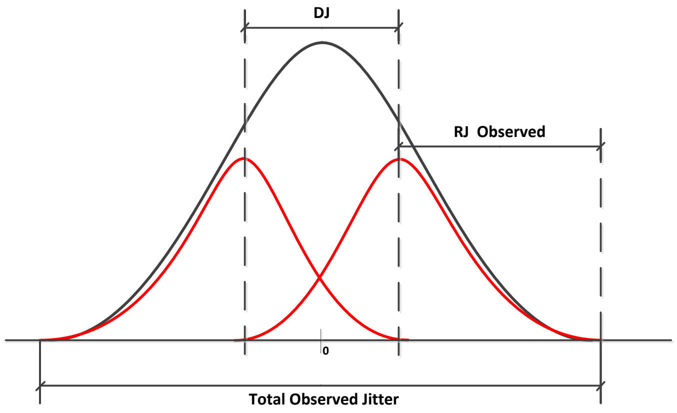


Figure 4.18: Eye diagram dual dirac waveform

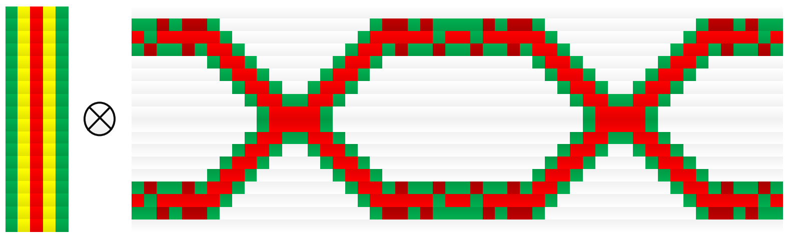


Figure 4.19: Eye diagram extrapolation

diagram generation. Due to this fact is then necessary to describe the RJ PDF on a form of a vector. One of the possible approaches is to use a normalized Gaussian distribution function (maximum jitter must follow eq. 4.21), implying a limit on the vector elements, RJ PDF elements are equal to the integer of the maximum. RJ divided by the eye diagram time step ($int(max(RJ)/t\_step)$).

Figure 4.19 exemplifies the process of obtaining the eye diagram extrapolation. Please note that the convolution must be done for all eye diagram lines, meaning that on the current implementation the algorithm performs 180 convolutions (vertical eye diagram points), resulting on an extrapolated eye diagram in terms of jitter (x-axis).

Eye diagram extrapolation can be hard to understand, but it is very easy to generate since it can be obtained through the convolution of a vector and a matrix, resulting on multiple convolutions of the vector (representing the RJ PDF) with the multiple lines of the matrix (representing the eye diagram color points).

4.15 Conclusion

In this chapter, the signal analysis algorithms that will be implemented were described.

Eye diagram waveform generation was implemented through a matrix, leading to a resolution of 1UI over 320. The eye diagram resolution seems to be good enough. Sometimes, depending of acquiring sample time, there are lines of the eye diagram without information, this does not have impact on the jitter analysis neither the eye mask placement.

The presented algorithms were developed to reduce the number of user defined parameters, removing from the user the need for a deep knowledge of the algorithms.

Chapter 5

Signal Analysis - Software Implementation

5.1 Introduction

Signal analysis functions were implemented in python using python classes. The usage of python classes maximizes the interoperability between functions, since a class is an object storing the internal variables until a delete operation. Objects are easy to use, allowing the creation of more intuitive and human readable code.

On this chapter the major functions/classes, that make the signal analysis tool, will be described. Pictures will be also presented to give visual information, about the task performed by the correspondent function.

5.2 Signal generation

Signal generation functions are the basis of the implemented software, allowing the user to create internal patterns, or to use patterns from external sources. This section describes the implemented signal generation functions:

5.2.1 Clock generation

The clock_gen class provides the interface for clock generation. Clock signals can have low frequency jitter effects, the modulation of such effect is achieve through phase variation.

Phase variation follows a sinusoidal function, described by a central frequency Fj and a maximum amplitude Aj.

Generated clock information is stored inside clock_gen class. The clock_gen.t variable refers to time, clock_gen.clock variable refers to the signal amplitude.

Class input parameters, internal variables and available functions:

clock_gen(Aj, Fs, Fj, tfinal)	
Aj	Maximum phase error
Fs	Frequency of clock signal
Fj	Jitter component frequency
tfinal	Final time

$$clk = cos(2\pi Fs \cdot t + Aj \cdot 2\pi sin(2\pi Fj \cdot t))$$

clock_gen.t	Time vector
clock_gen.clock	Amplitude Vector
clock_gen.plot_clock_fft( )	Function to plot signal FFT
clock_gen.plot_phase_jitter_fft( )	Function to plot signal phase jitter FFT

5.2.2 Edges extraction

The clock_edges_extract class performs the edge extraction task. Digital signals represented in an analog way, does not have information about bit time, or edge location.

Signal analysis algorithms require information about edges location and bit period. This class provides such information. The clock_edges_extract.clk_per[:,0] variable contains the time information about the edge location, clock_edges_extract.clk_per[:,1] variable contains the bit period.

Edges extraction algorithm performs a 2^{nd} order polynomial fitting, which ensures a better time extraction of the zero crossing point. Polynomial fitting is performed over analog vector, when a transition is detected on analog vector (position n), two adjacent indices (positions n-1 and n+1) are used to perform the poly fit.

Jitter addition can be performed at this level. The addition of random jitter or deterministic jitter, will influence the edge time locations and bit period .

Class input parameters, internal variables and available functions:

clock_edges_extract(clock_gen_class, thrs=0.0, d_type="CLOCK")	
clock_gen_class	Analog signal data type
thrs	Zero detection point Threshold
d_type	Data Type ("CLOCK","PRBS5",...)

d_type	Used to select between clock and generic data extraction edges extraction
clock_edges_extract.clk_per[N,2]	Edge location point vector, column 0 refers to time, column 1 refers to period

clock_edges_extract.add_jitter(dj,rj)	Jitter addition to clk_per vector, where rj represents the gaussian jitter σ, dj represents the deterministic jitter

5.2.3 Clock multiplication

Clock multiplication is commonly used in current circuits. The clock_edges_mult class allows the user to generate, a high speed signal based on a clock edges vector.

Class input parameters, internal variables and available functions:

clock_edges_mult(clock_sig_class, mult=1)	
clock_sig_class	clock_edges_extract data
mult	Multiplication factor, final number edges = "mult" * edges number of clock_sig_class. Generated signal period will be "mult" times lower than clock_sig_class for each vector point

clock_edges_mult.clk_per[N,2]	Edge location point vector, column 0 refers to time, column 1 refers to period
clock_edges_mult.add_jitter(dj,rj)	Jitter addition to clk_per vector, where rj represents the gaussian jitter σ, dj represents the deterministic jitter

5.2.4 Data generation

The data_generation class is responsible for the analog signals generation. Based on a clock edges vector, different patterns can be generated.

Pattern sequence used for analog signal generation, needs to be define by the user. Analog signal characteristics, follow the approach presented in chapter 4.6.

Figure 5.1 contains an example of a PRBS5 signal generation. Multiple waveforms were generated to exemplify some of the capabilities, supported by class data_generation, like: Pre-Emphasis, amplitude random jitter addition and amplitude deterministic jitter addition.

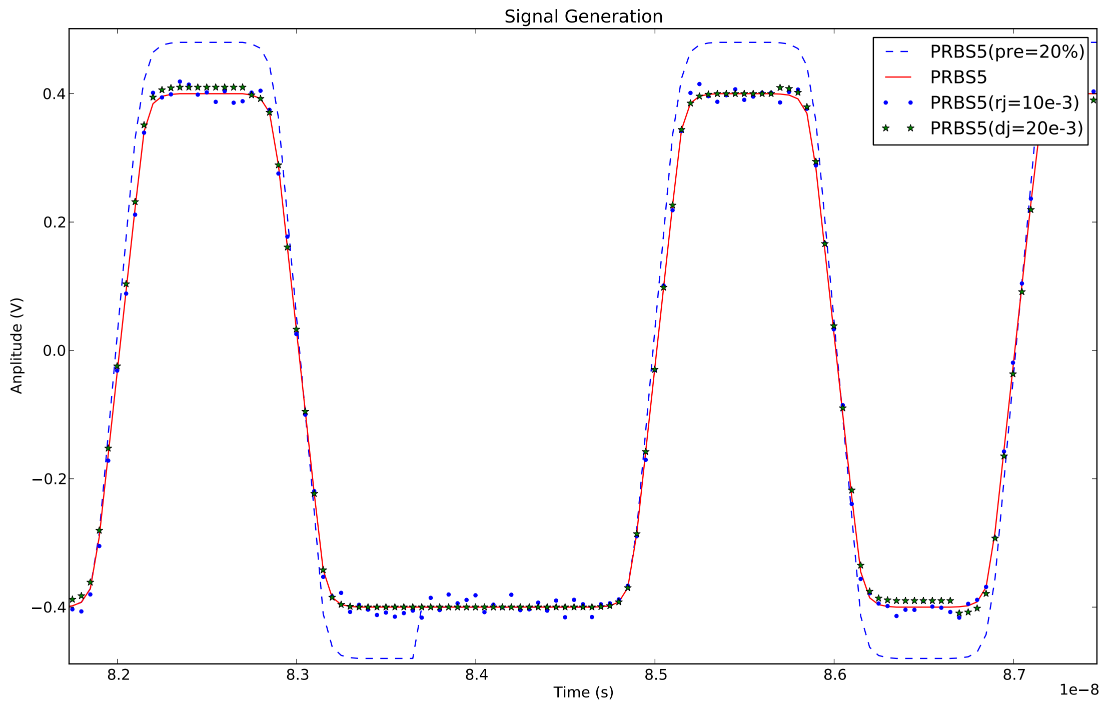


Figure 5.1: Signal generation

Class input parameters, internal variables and available functions:

data_generation(clk, data_type, VHI, VLOW, tr, tf, tfi_tr_rel=3/4., tfi_tf_rel=3/4., dj=0., rj=0., preemphasis=None)	
clk	clock_edges_extract data
data_type	Analog signal data type ("CLOCK", "PRBS5", "PRBS7", "PRBS15", "PRBS23", "PRBS31")
VHI	High voltage value
VLOW	Low Voltage value
tr	rise time
tf	fall time
tfi_tr_rel	relation between rise time and the necessary time to consider that the final value was achieved
tfi_tr_rel	relation between fall time and the necessary time to consider that the final value was achieved
dj	Deterministic amplitude jitter (V)
rj	Random amplitude jitter (V)
preemphasis	Percentage of preemphasis in relation to the regular value

data_generation.ana_data[N,2]	Analog signal vector, column 0 refers to time and column 1 refers to amplitude
data_generation.write_to_file(filename)	Stores Analog signal vector on file:filename in csv format.

5.2.5 External analog data

User can perform signal analysis over real data, provided from simulations for example. The external_ana_data class allows the user, to insert that data into the python environment.

External signals should be stored in .csv (comma separated values) file format. The .csv file should have the time information as the first element; second element should be the signal amplitude value.

Implemented class supports the use of a single .csv file, with multiple analog signals. The user needs to define the analog signal to be extracted. The first line of a .csv file contains the elements name. For example:

time, analog_ch0, analog_ch1

In this case the user can select between signal "analog_ch0" or "analog_ch1".

Class input parameters, internal variables and available functions:

external_ana_data(filename, ind_ana_sig=1, lines_max=None, rem_start_t=True)	
filename	File name to read
ind_ana_sig	Column zero represents time, the analog signal will be extracted from column defined by ind_ana_sig (default is 1 - "analog_ch0" in the example presented above)
lines_max	Maximum number of lines to read (default is to read all)
rem_start_t	Considers initial time as zero (default is true)

external_ana_data.write_to_file(filename)	Stores Analog signal vector on file:filename in csv format.

5.3 Operation over generated signals

On this section the user can find the available functions to perform operation over the generated signals, like: cable effects, frequency domain effects, clock recovery, signal convolution and frequency response extrapolation:

5.3.1 Generic frequency domain to time domain conversion

Frequency domain to time domain conversion is the central core from the presented tool. The s21_param class is the low level class, responsible for impulse response vector generation.

The s21_param is able to extract frequency response vector from .s2p files (only s21 parameter is used), .csv files or python specified functions.

When using .csv files special care is needed, .csv file should have the following 3 elements: frequency(Hz), gain (real part), gain (imaginary part)

Class input parameters, internal variables and available functions:

s21_param(func_filename, plt_figs=False)	
func_filename	Python function or filename
plt_figs	Plot internal operations

s21_param.get_mag( )	Return s21 magnitude vector
s21_param.get_angle( )	Return s21 phase vector
s21_param.get_real( )	Return s21 real part vector
s21_param.get_imag( )	Return s21 imaginary part vector
s21_param.get_frequency( )	Return s21 frequency vector
s21_param.get_extrapol( )	Return s21 frequency response extrapolation vector [frequency, s21 magnitude]
s21_param.extrapol_re_img( )	Frequency response extrapolation $F_{ext} = F_{max}$, F_{max} can be overridden by the user
s21_param.get_impulse_resp( )	Impulse response generation
calc_inp_coef.plot_freq_resp(fmax=None, data_type=None)	Plot frequency response. Frequency plot will be limited by fmax value, data_type refers to the signal name
calc_inp_coef.plot_imp_resp(data_type=None)	Plot impulse response data_type refers to the signal name

5.3.2 CRU to impulse response generation

The cru_func_to_ir is a wrapper class. Internally s21_param class is used to provide the impulse response vector.

HDMI communication standard specifies the frequency response function, to be used on a clock recovery unit. The cru_func_to_ir allows the user to select between a generic transfer function, defined by cru_gen_func input parameter or the HDMI pre-defined frequency response (refer to http://www.miko.com.hk/SDA_HDMI.htm or www.dybkowski.comule.com/download/hdmi/hdmi_spec_1.3_gm1.pdf for more information).

Class input parameters, internal variables and available functions:

cru_func_to_ir(func_type, cru_gen_func=None)	
func_type	Function type: "HDMI", "Generic"
cru_gen_func	Frequency response function, should be defined for Generic func_type, .csv or .s2p files are not supported

cru_func_to_ir.plot_freq_resp(fmax=None, data_type=None)	Plot frequency response. Frequency plot will be limited by fmax value, data_type refers to the signal name
cru_func_to_ir.write_to_file(filename, fmax)	Write frequency response to file (filename), frequency response will be store for frequency values lower than fmax

5.3.3 HDMI reference equalizer

The hdmi_ref_eq class contains the frequency response of the equalizer defined in HDMI spec (please refer to www.dybkowski.comule.com/download/hdmi/hdmi_spec_1.3_gm1.pdf for more information).

HDMI equalizer frequency response function is internally defined, allowing the use of s21_param class to extract the impulse response. Such action is performed automatically.

Class input parameters, internal variables and available functions:

hdmi_ref_eq( )	
hdmi_ref_eq.plot_freq_resp(fmax=None, data_type=None)	Plot frequency response. Frequency plot will be limited by fmax value, data_type refers to the signal name
hdmi_ref_eq.write_to_file(filename, fmax)	Write frequency response to file (filename), frequency response will be store for frequency values lower than fmax

5.3.4 Impulse response generation

The calc_inp_coef class is used to obtain the impulse response of a frequency response, defined by a python function.

Class input parameters, internal variables and available functions:

calc_inp_coef(gen_func=None)	
gen_func	Frequency response function, .csv or .s2p files are not supported

calc_inp_coef.plot_freq_resp(fmax=None, data_type=None)	Plot frequency response. Frequency plot will be limited by fmax value, data_type refers to the signal name
calc_inp_coef.write_to_file(filename, fmax)	Write frequency response to file (filename), frequency response will be store for frequency values lower than fmax
calc_inp_coef.plot_imp_resp(data_type=None)	Plot impulse response data_type refers to the signal name

5.3.5 Clock recovery

Clock recovery unit algorithm is implemented by clok_recovery class.

The clok_recovery class requires as input (cru), an object of type cru_func_to_ir class.

Clock extraction from data signals is supported by clok_recovery class. The user will have to define the data_type to a signal name (different from 'CLOCK'), the mean period and the CRU mode. For more information about mode, please take a look at chapter 4.8.2.

Figure 5.2 shows the effect of clock recovery circuit when applied to a clock signal with a frequency of 10 GHz, random jitter of 0.5ps, and frequency jitter described by $Aj = 0.2$ and $Fj = 500KHz$.

Class input parameters, internal variables and available functions:

clok_recovery(clock_sig, cru, data_type='CLOCK', mean_per=None, mode=None)	
clock_sig	Input signal edges
data_type	Data signal type: "CLOCK", "PRBS5", ...
mean_per	Mean Period, used in plesiochronous interfaces

mode	Clock recovery mode for clock recovery in plesiochronous interfaces, mode=0 - phase is incremented on all iterations, mode=1 - phase is incremented only on iterations with transitions in both signals (input and feedback signals)

clok_recovery.clk_per[N,2]	Edge location point vector of recovered clock , column 0 refers to time, column 1 refers to period
clok_recovery.ind_start	Number of items to discard on clok_recovery.clk_per vector

5.3.6 Convolution

Convolution between impulse response and analog signal is performed in ana_convolve class. The impulse response, defined as ir input parameter, is an object of type calc_inp_coef class. The analog signal, defined as ana_sig input parameter, is an object of external_ana_data type or data_generation type.

The ana_convolve class, stores the convolution result in an internal variable (ana_convolve.ana_data), similar to the defined in data generation classes.

The resultant vector of a convolution has more elements than, each one of the vectors used as input. The ana_convolve internally performs the vector casting.

Class input parameters, internal variables and available functions:

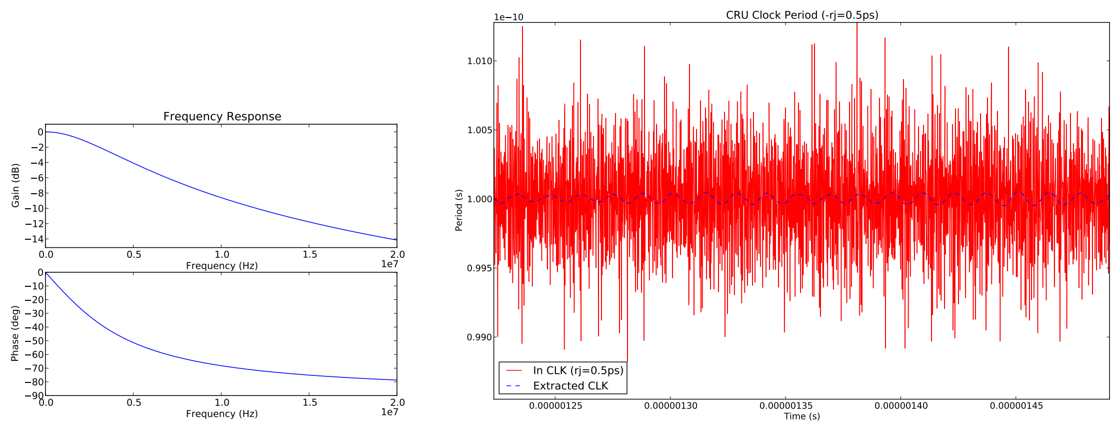
ana_convolve(ir, ana_sig)

ir	Impulse response
ana_sig	Analog signal obtained from data generation functions

ana_convolve.ana_data[N,2]	Analog data vector after convolution, column 0 refers to time and column 1 to the analog signal value

5.3.7 Generic filter

The generic_filter class allows the user to specify a frequency response signal trough zero-pole-gain vectors.



(a) CRU TF

(b) Input clock Vs Recovered Clock

Figure 5.2: Input clock with rj=0.5ps Vs recovered clock

Conversion from zero-pole-gain function to impulse response can be done through calc_inp_coef class.

Class input parameters, internal variables and available functions:

generic_filter(z, p, k)	
z	Zeros vector
p	Poles vector
k	Gain

5.3.8 Cable model

Cable and PCB FR-4 frequency response models are implemented in class cable_model.

The effect of a cable, can be introduced to the cable_model class through cable_model.add_element() function. User will have to define the R, L, G, C constants per meter. Cable length is then reflect in dz parameter.

PCB effects are introduced to the cable_model class through cable_model.add_pcb_element(). User will have to provide the PCB characteristic impedance through Z0 parameter. The PCB length is defined by dz parameter.

PCBs and cables does not have smooth transfer functions, cable_model class allows the user to add random jitter to the frequency response, making the model more realistic.

Class input parameters, internal variables and available functions:

cable_model(Z0=50)	
Z0	Source impedance

cable_model.add_element(r, c, g, l, dz)	Add a new cable element, with impedance r, capacitance c, inductance l, admittance g and length dz(in meters)
cable_model.add_pcb_element(	Add a new PCB FR-4 element, with characteristic

5.4 Signal analysis

Signal Analysis functions were implemented trough calc_analog_params( ) class. The implemented class allows the user to extract jitter, plot eye diagram, measure rise and fall times and eye diagram extrapolation.

5.4.1 Eye diagram mask

Eye diagram minimum opening is defined through an eye diagram mask. No signal should cross the eye diagram mask. Bad signals sometimes cross that mask, resulting on a failing measure indication.

An eye diagram mask is related with signal integrity, specifying jitter, voltage swing/amplitude and rise/fall times.

Eye diagram mask points are described in more detail, in figure 5.3. The eye_mask class, contains the information about an user defined eye diagram mask .

Class input parameters, internal variables and available functions:

	eye_mask( )
eye_mask.bot_top	Maximum amplitude of the eye diagram waveform (Volts)
eye_mask.bot_top_mask	Eye diagram mask amplitude (Volts)
eye_mask.zero_mask	Eye diagram mask point for 0V amplitude mask edge (UI)
eye_mask.top_bot_mask	Eye diagram mask point for bot_top_mask amplitude mask edge (UI)

5.4.2 Analog parameters extraction

The calc_analog_params class implements: jitter analysis, eye diagram generation, eye diagram extrapolation and analog parameters measurements. It requires as input parameters a clock edges class (clock_edges_mult, clock_edges_extract or, clok_recovery), an analog signal class (data_generation or external_ana_data), the indication of the position of the first valid element of clk_edges class (usually clok_recovery.ind_start assuming that clk_edges is of type clok_recovery) and the data_type information (’CLOCK’, ’PRBS7’, etc.)

Jitter analysis are performed through calc_analog_params.extract_jitter() function.

Eye diagram generation and analog parameters measurements (min rise/fall time, minimum differential VIH/VIL and average VIH/VIL) are done through calc_analog_params .plot_eye_diagram() function. This function requires as input parameter the eye diagram mask, defined in class eye_mask

Eye diagram extrapolation is performed through calc_analog_params.extrapol_eye_diagram() function. This function requires as input parameters the eye diagram mask, defined in class eye_mask; the BER target and an indication related with jitter. The only_dj input parameter should be set to 1, when the extrapolation is to be performed, considering all the jitter presented on eye diagram as deterministic.

Eye diagram jitter measurement is available through calc_analog_params.plot_jitter_eye() function.

An example of the capabilities available through this class can be found in figure 5.4.

Class input parameters, internal variables and available functions:

calc_analog_params(clk_edges, ana_sig, valid_ind, data_type ='PRBS7')	
clk_edges	Clock recovery edges position - from clock recovery function
ana_sig	Analog signal
valid_ind	clock_edges elements to discard
data_type	Analog signal data type: "CLOCK", "PRBS15", ...

calc_analog_params.extract_jitter(ber=1e-12)	Jitter extraction, total jitter value will be calculated for the input ber value
calc_analog_params.plot_eye_diagram(class_eye_mask)	Plot eye diagram, necessary to provide the eye diagram mask
calc_analog_params. extrapol_eye_diagram(class_eye_mask, ber_e=1e-9, sigma_e=None, only_dj=None)	Plot eye diagram extrapolation for a desired ber assuming the previous measured random jitter or a specified one (sigma_e), only_dj=1 ensures that the extracted jitter will be considered as dj only
calc_analog_params.plot_jitter_eye( )	Plot eye diagram jitter
calc_analog_params.vlow	Analog signal - Average low voltage value
calc_analog_params.vhigh	Analog signal - Average high voltage value
calc_analog_params.vlow_max	Analog signal - Maximum voltage of "digital 0"
calc_analog_params.vhigh_min	Analog signal - Minimum voltage of "digital 1"
calc_analog_params.min_rise_time	Analog signal - Minimum rise time
calc_analog_params.min_fall_time	Analog signal - Minimum fall time

5.5 Report file generation

Signal analysis tool creates a html file, reporting the results provided by the implemented python classes presented before.

All simulation environments require the creation of the report class, this process is done through c_html.create() call.

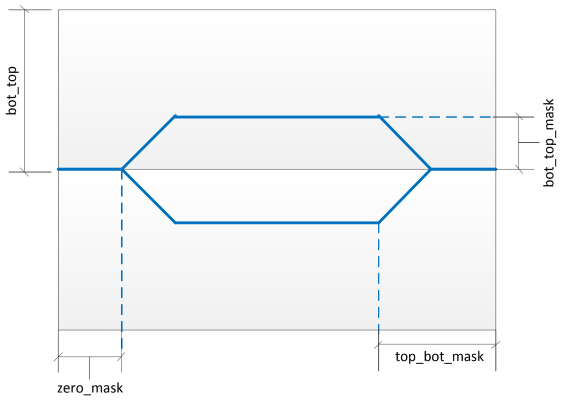
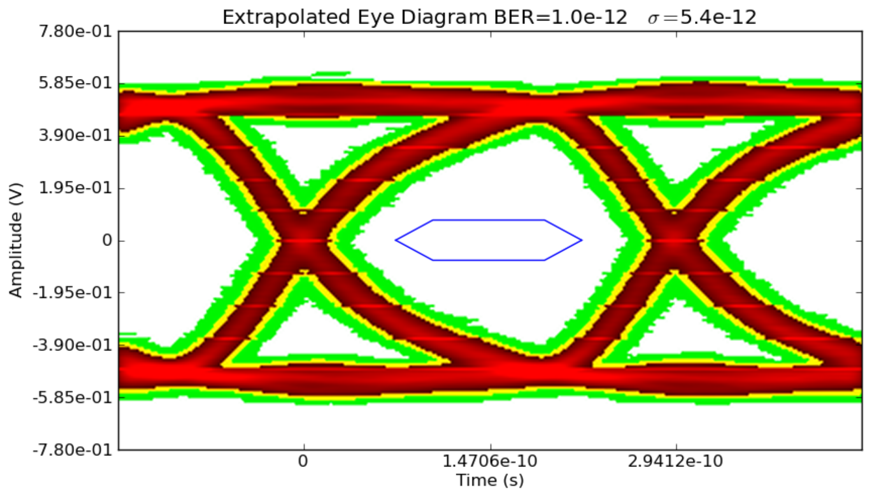
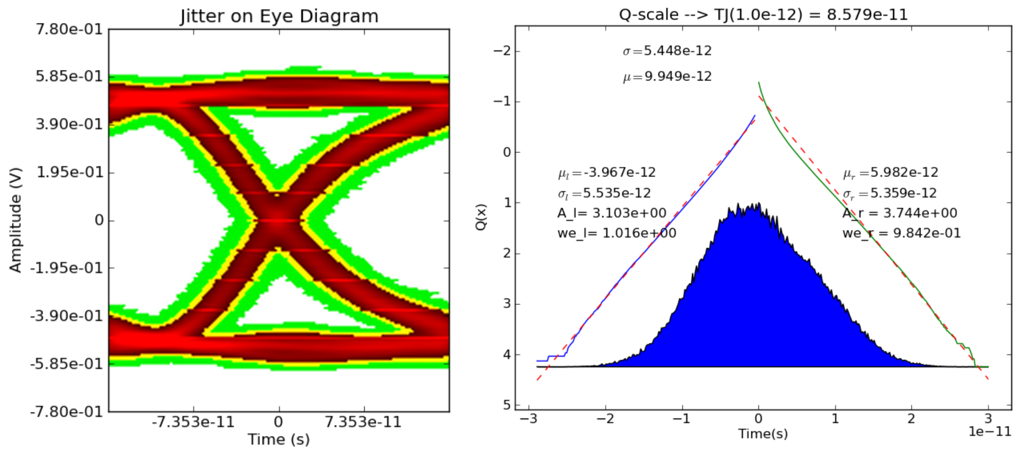


Figure 5.3: Signal generation



(a) Eye diagram



(b) Jitter on eye diagram

(c) Extracted jitter

Figure 5.4: Extrapolated eye diagram for $BER = 1e^{-12}$ with $\sigma = 5.4ps$

The c_html class is a global object created in the signal analysis python library, this object is initialized when the signal analysis module is imported to the python environment.

The c_html.close() function show be performed at the end of the simulation, to ensure a correct report file closing procedure.

5.5.1 HTML file generation

The create_html class provides the necessary functions to produce a html report file. This class is self-created in the signal analysis python module, through the object handler c_html.

The internal functions allow the user to save images, tables and to separate the different waveforms with headers.

Class input parameters, internal variables and available functions:

create_html()

create_html.create( )	File creation. The html file header includes the creation time
create_html.title_break(title)	The text inside title variable is added as header 1
create_html.title(title)	The text inside title input parameter is added as header 2
create_html.subtitle(title)	The text inside title input parameter is added as header 3
create_html.table(title, table_data)	Adds a table to the html report file. Title input parameter refers to the cable caption. The table_data input parameter represents the table raw data vector.
create_html.image(title, image_name, file_name)	Adds an image to the html report file. The title input parameter refers to the cable caption. The image_name refers to name that will be presented in html report. The file_name input variable refers to the file were the image to include is stored
create_html.close( )	Close the html file.

5.6 Conclusion

In this chapter the python environment classes and functions, supported by signal analysis tool were presented.

An usage example of the signal analysis tool is presented in annex A.

Chapter 6

Results

6.1 Introduction

The signal analysis tool provides a standard interface for a variety of analysis. Allowing the users to perform different types of architecture modeling.

The accuracy of the tool was compared against commercial solutions, the results of that evaluation are presented in this chapter.

The CRU transfer function affects the jitter value, an incorrect definition can lead to wrong characterization values. Plesiochronous systems are affected, not only by the CRU-TF, but also by the data pattern, an incorrect encoding algorithm can lead to unexpected behaviors.

Channel characterization can make use of all available tool functions, the use of python result on a easier interconnection, reducing the scripting complexity.

6.2 Proposed software tool accuracy

System engineers define the system architecture based on simulation results. EDA (Engineering Design Automation) tools must predict the real circuit is behavior accurately.

Frequency domain to time domain operation is the basis of the proposed tool. It is important to understand how accurate this operation is. In terms of frequency domain to time domain operations the entire behavior can be described by two functions, cable_model and generic_filter. One of the best solutions to proceed with the validation is to implement Spice models describing these functions, and then simulate those using commercial tools. Cable model validation is easy to implement in spice language, since it can be described by ideal lumped elements like resistors, inductors and capacitors. However, the same cannot be applied to ZPK functions, which have to be described using Verilog-A.

Commercial Spice simulators have deep support for spice language, but there are few simulators that also support Verilog-A. The Spice simulator that was used during this work is one of the few that support both languages (this is one of the reasons for being one of the most used).

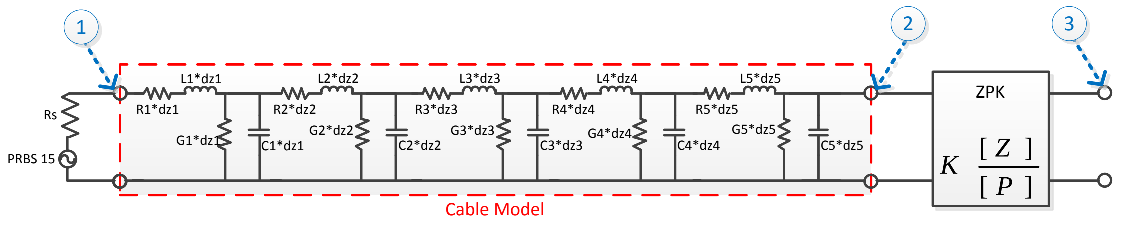


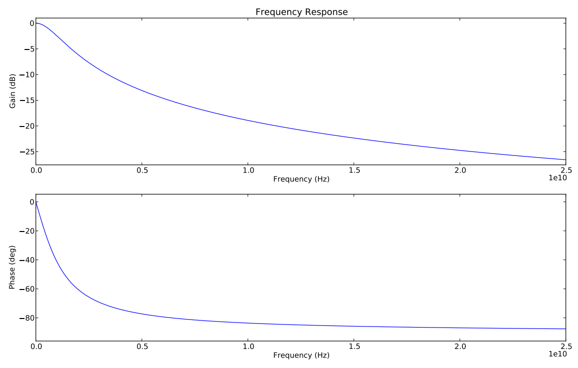
Figure 6.1: Circuit used to validate python algorithms

Circuit used on both simulations is described on figure 6.1. Generic filter is a low pass and can be described by a zero pole gain function as: $Z = [-2\pi 0.9e9]$, $P = [-2\pi 3.5e9, -2\pi 4.5e9]$ and $K = \frac{-2\pi 3.5e9 \cdot -2\pi 4.5e9}{-2\pi 0.9e9}$. Cable model is described by multiple cable elements with: R=[1e-9, 1e-9, 1e-9, 1e-9,1e-8], C=[2e-12, 5e-12, 3e-12, 0.1e-12, 3e-12], G=[1/3e7, 1/3e6, 1./3e5, 1/3e7, 1/3e7], L=[0.1e-12, 0.01e-12, 2e-12, 0.1e-12, 0.5e-12], dz=[0.2, 0.2, 0.2, 0.2, 1.2].

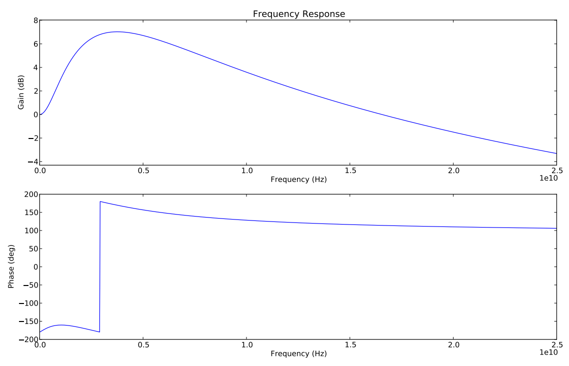
To have a correct measurement the input data was generated on the python tool and then applied to Spice simulations. Signal generation must be modulated as a voltage signal connected to a source termination, in this case Z0 was defined to 50 Ω.

Python algorithms rely on $Fmax$ definition to determine the maximum frequency considered on the frequency domain analysis. Simulations were done with $Fmax = 200e9$, which corresponds to ten times the maximum frequency of the clock used to generate the PRBS15 signal (20GHz). Frequency response waveforms of the cable and generic filter are represented on figure 6.2.

Data used on this process was generated in python and used in spice as well, making possible the direct comparison between environments. Figure 6.3 shows the difference between python and spice at the cable output (point 2 in figure 6.1). Blue waveform is the input signal (point 1 on figure 6.1), red waveform represents the signal from python and black signal represents the signal obtained from spice simulation. Differences can be caused by the approximations done inside the Spice tool, since very-low resistance values are usually discarded, to increase the simulation speed. It is then possible to conclude that the cable model algorithm implemented on python is very accurate when compared with spice even for signals at 20Gbit/s.



(a) Cable transfer function



(b) ZPK Transfer function

Figure 6.2: Cable and generic filter transfer functions

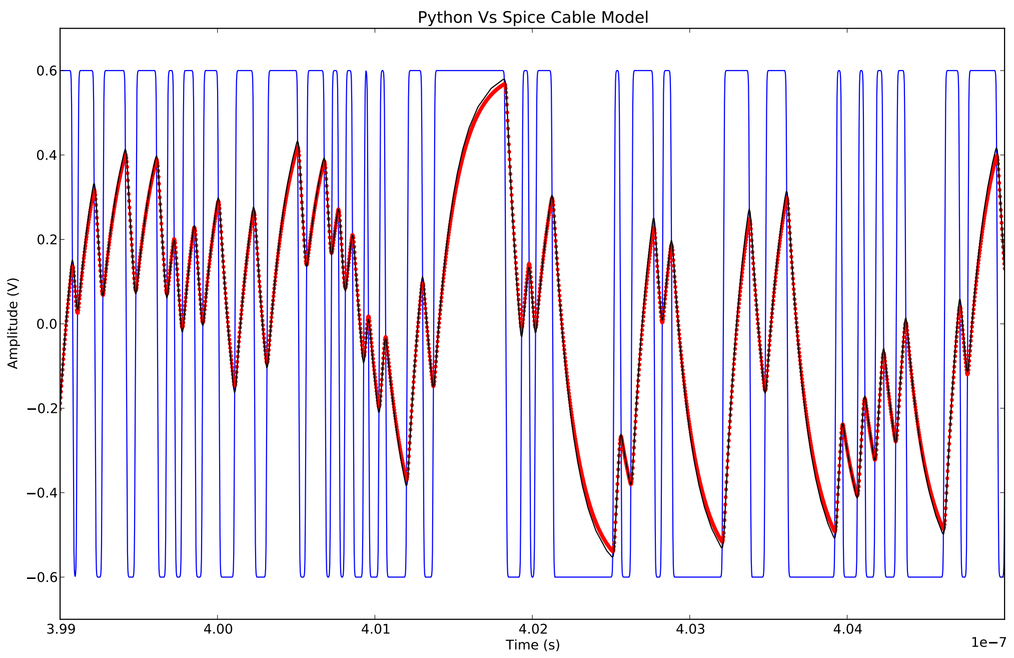


Figure 6.3: Python Vs spice cable model effect on PRBS15 signal

Figure 6.4 represents the signal after generic filter (point 3 in figure 6.1). The signals are represented with the same color mapping as the used above. In terms of accuracy it is impossible to check if the differences are due to the cable model, or to the generic filter model. A second simulation was performed using the signal after cable obtained in python as input to the generic filter in Spice. Results of this simulation can be found on figure 6.5.

The ZPK implementation is as accurate as spice allowing to conclude that when an appropriate $Fmax$ value is used, the obtained results from python simulations are very accurate, giving the necessary confidence to the end user to use frequency response functions on the system model.

PCB FR-4 validation model was done through superposition of frequency response profiles presented on [1] and python implementation, via cascading 30 elements with dz=0.0254m (1in).

Images from python were re-scaled to match the scale of the images on the article, then it was just necessary to align the references to create the final image presented on figure 6.6. The method used is not perfect once it implies a manual action. Differences between python model and the one proposed on the article are very small in terms of gain and similar in terms of phase. In terms of gain the difference is on the order of 1dB. Python implementation represents better the influence at high frequencies.

Based on this analysis it is possible to conclude that python model can be used with confidence for frequencies lower than 10GHz. It is possible to use python model for frequencies till 20GHz when short PCB traces are considered (up to 20cm).

Validation of clock recovery algorithm was done through a different method: instead of compare python results with simulation, the comparison was done against lab results, obtained from commercial and expensive equipment. A real circuit was used to provide the input signals to the

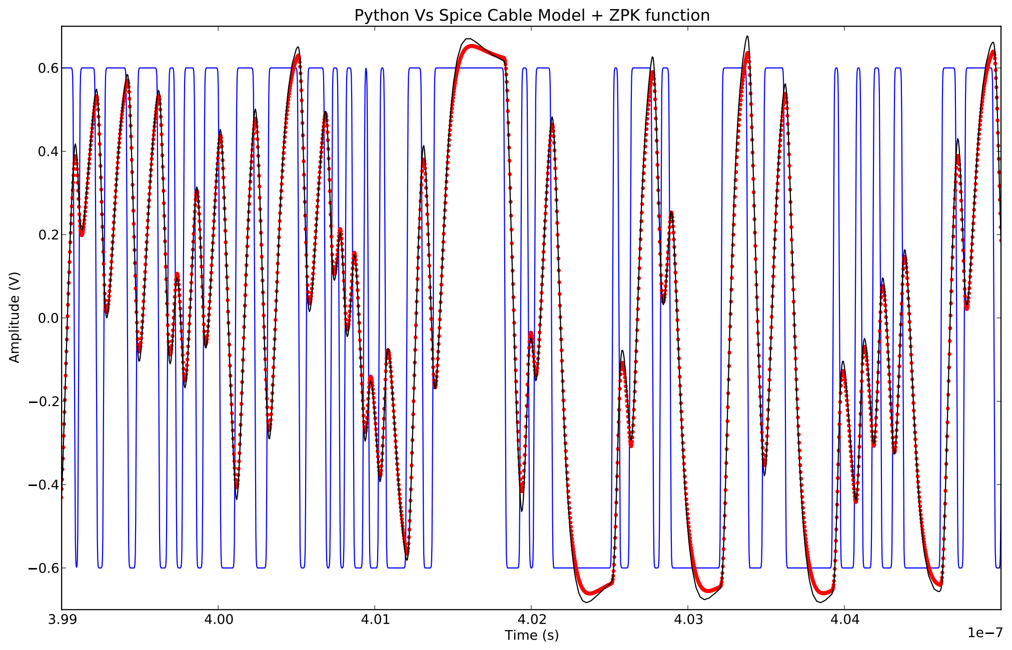


Figure 6.4: Python Vs spice cable model + generic filter effects on PRBS15 signal

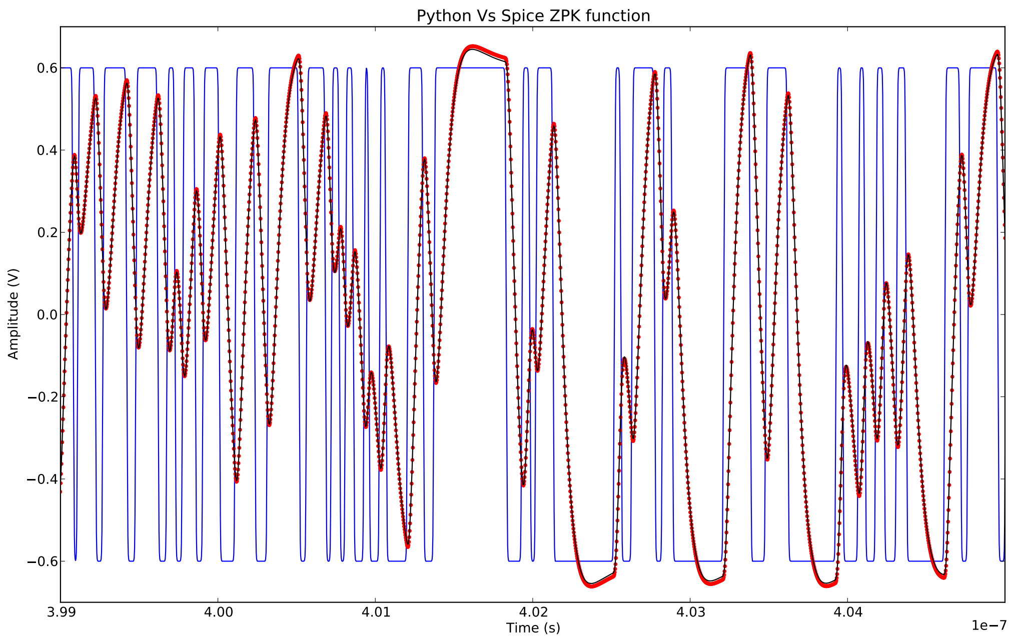
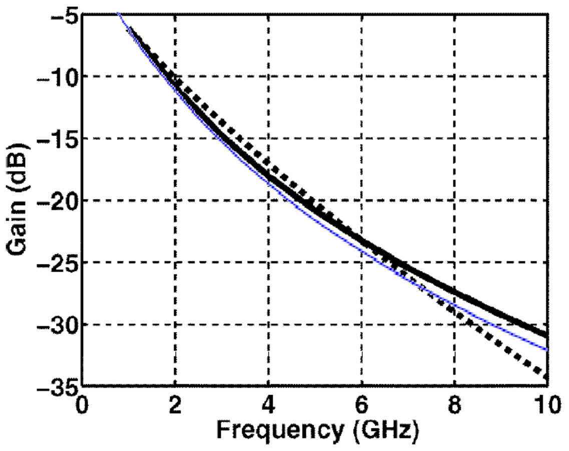
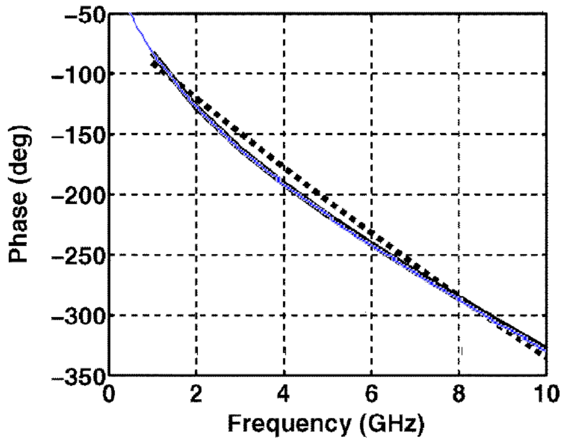


Figure 6.5: Python Vs generic filter effects on PRBS15 signal



(a) FR-4 model gain



(b) FR-4 model phase

Figure 6.6: PCB FR-4 frequency response profiles, microstrip (black dashed lines), and scalable model from [1] (black solid lines) and python implementation (blue solid lines) for a 30-in trace

oscilloscope, responsible for the eye diagram generation at the lab side. Raw data used by the oscilloscope was stored and applied to the python tool.

The use of an eye diagram allows the comparison of multiple algorithms trough a single picture, like clock recovery algorithm, zero crossing point detection, rise and fall times and jitter.

Python tool processed the raw data and generated an eye diagram that was stored into a .png file. The comparison between lab and python results was done through image superposition. Obtained eye diagrams have a different number of pixels but, if one image is resized until the eye diagram mask matches perfectly the mask on the other image, figures will be perfectly aligned.

Figure 6.7 represents the differences between python and oscilloscope images, python image provides the background, the boundaries of the oscilloscope image are then plotted in blue color over that background. This operation was repeated for three different channels and the results were consistent, allowing to conclude that the clock recovery algorithm and signal analysis functions implemented in python are as accurate as the ones used in commercial equipments.

6.3 Frequency response to time response analysis

The frequency domain to time domain conversion depends of the maximum frequency available inside the frequency response representation. In order to demonstrate this effect, simulations were performed using a switch from Avago (AMMC-2008), with a frequency response representation till 50 GHz as reference.

The simulation environment was designed to check only the influence of the frequency response on the jitter analysis. A PRBS15 sequence was used as data pattern; the clock used to trigger the data has a frequency of 6.5 GHz without jitter components, leading to 6.5Gbps interface.

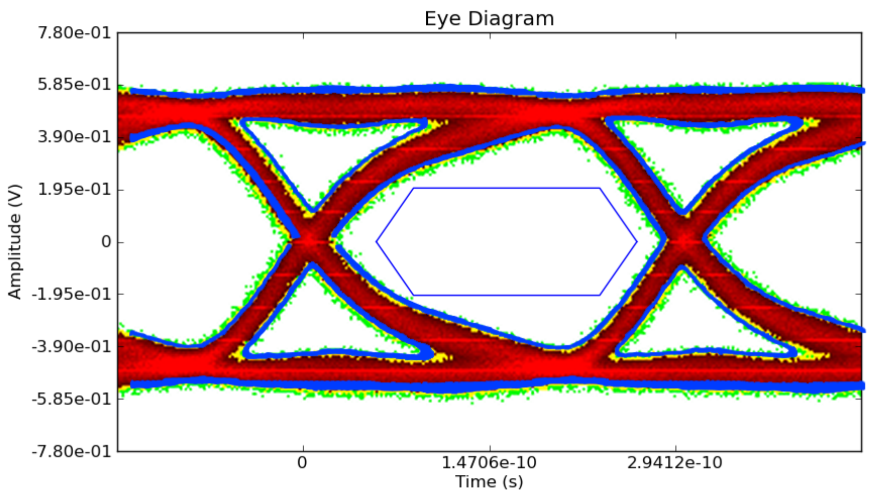
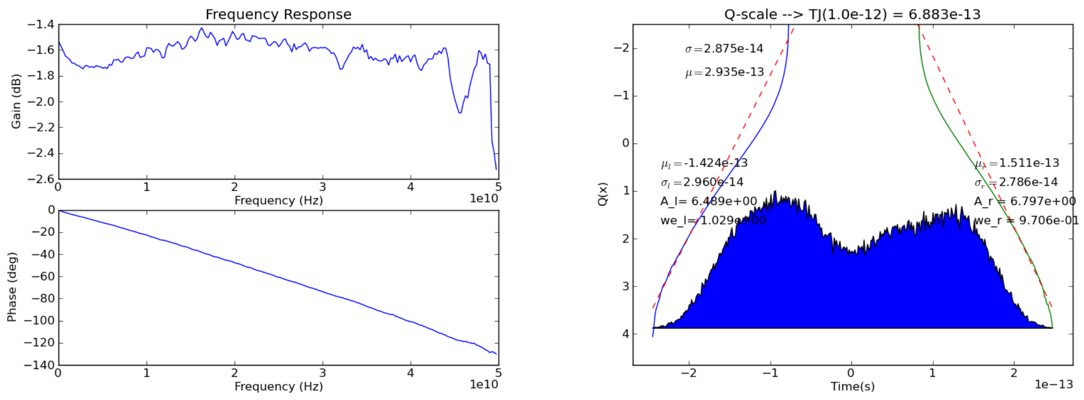


Figure 6.7: Eye diagram comparison between python and lab

Figure 6.8-a) represents the frequency response of the switch obtained from the .s2p file provided by Avago. The jitter analysis presented in figure 6.8-b) were performed using the generation clock for ideal edge positions and $Fmax = 50GHz$. Since no jitter was added, the observed jitter should be considered only deterministic, not varying with the number of acquired bits (assuming at least $2(2^{15} - 1)$ are acquired). The total jitter measured after 260.000 acquisitions was 0.5ps.



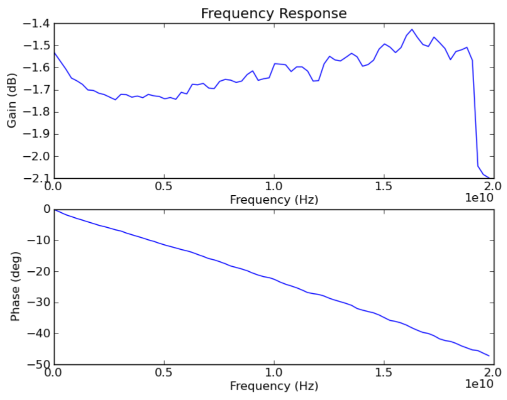
(a) Frequency response

(b) Jitter histogram

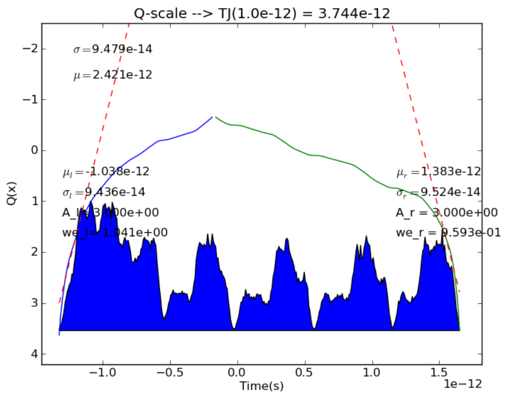
Figure 6.8: Switch effect when using $fs = 50Ghz$

Figure 6.9-a) represents the frequency response for the Avago switch, but this time limited to 20GHz. The obtained jitter presented in figure 6.9-b) was obtained using $Fmax = 20GHz$, leading to a bigger value (2.99ps) when compared with the previous approach.

Figure 6.10-a) represents the frequency response extrapolated from 20GHz to 50GHz for the Avago Switch. Figure 6.10-b) represents the jitter observed in this approach ($Fmax = 50GHz$). The total jitter measured was 0.4ps.

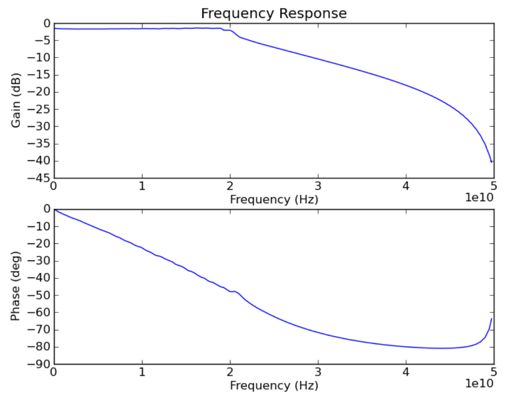


(a) Frequency response

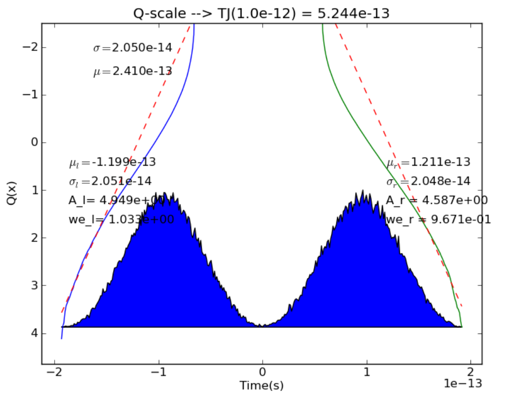


(b) Jitter histogram

Figure 6.9: Switch effect when using $fs = 20Ghz$



(a) Frequency response



(b) Jitter histogram

Figure 6.10: Switch effect when using $fs = 20Ghz$ and proceeding with a frequency response extrapolation to 50Ghz

The frequency response extrapolation was done using the previously presented algorithm (use of Hilbert transform), such approximation assumes a low-pass characteristic that can be noticed in figure 6.10-a). The Avago switch has a flat frequency response at least till 50GHz, leading to an inaccurate frequency extrapolation, although the jitter error is much smaller than doing nothing. The extrapolation results in an error of 20%, but using only 20GHz results in 498% of error.

The frequency band of operation should be carefully chosen, to minimize the error. In this case it was used a data rate of 6.5Gbps leading to a maximum frequency of 3.25GHz, far from the 20GHz used in the worst case.

6.4 Clock recovery topologies analysis

Clock recovery circuits are used to extract the jitter value seen by the data recovery circuit. As referred by Agilent [23], "Any test standards (IEEE 802.3 Ethernet, Fibre Channel etc.) require the use of a Golden PLL (phase locked loop), jitter transfer characteristic or loop bandwidth to control what spectrum of jitter is observed and what is removed from eye-mask and jitter tests. If the loop bandwidth is too wide, too much high-frequency jitter is removed from the observed signal. If the loop bandwidth is too narrow, measurements can be obscured with lower frequency jitter. This jitter is usually less important since receivers easily tolerate it. Testing with an optimal loop bandwidth ensures that good parts do not appear to be bad, and bad parts do not appear to be good".

Clock recovery circuit topologies affect the overall system jitter, the first line of defense against jitter is done by them.

When used, CRU represents an ideal PLL circuit without jitter addition. However, CRU-TF influences the overall measured jitter. PLL can follow accurately low frequency components, leaving the high frequency to be addressed by CDR. High frequency jitter components are seen on the eye diagram, representing the jitter that must be tracked by the CDR. If such jitter profile is bigger than what can be addressed by the CDR, data errors will appear.

6.4.1 CRU-TF influence on clock shared communication systems

Clock sharing communication systems have more immunity in relation to low frequency jitter (when compared with the transfer function bandwidth). The following example was created to demonstrate the influence of this factor on the overall jitter.

PRBS15 data sequence was used, generated via a 10GHz clock with $A_j = 0.2UI$ and $F_j = 500kHz$ with 1ps of random jitter. Data signal was also generated with 5mV of random amplitude noise.

To make the verification more realistic a cable model with 3dB of attenuation at 10GHz (see figure 6.11) was introduced on the characterization system.

CRU-TF was obtained from eq. 6.1. Frequency responses were obtained using ωn = 1 and 13 MHz, with a damping factor of 0.707 (see figure 6.12).

$$CRU - TF(s) = \frac{\omega n^2 + 2s\zeta\omega n}{s^2 + 2s\zeta\omega n + \omega n^2} \tag{6.1}$$

Figure 6.14 shows the jitter histograms for each CRU-TF. Eye diagrams are presented on figure 6.13. Obtained random jitter is bigger than 1ps. This represents in fact the influence of cable attenuation combined with PRBS15 data sequence. It is also necessary to take into consideration the amplitude random noise of 5mV, which will be translated in a few more fs of random jitter.

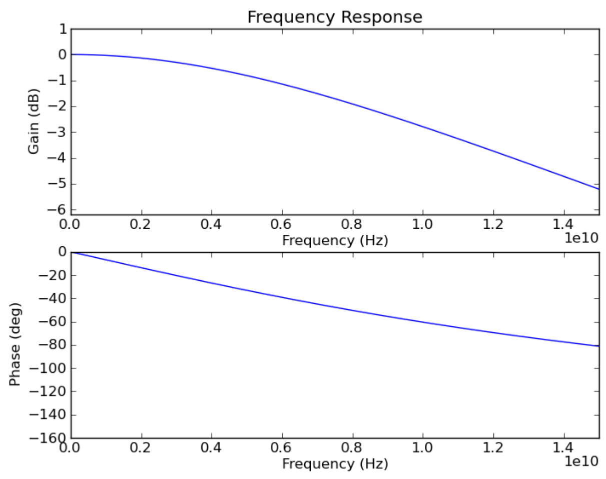
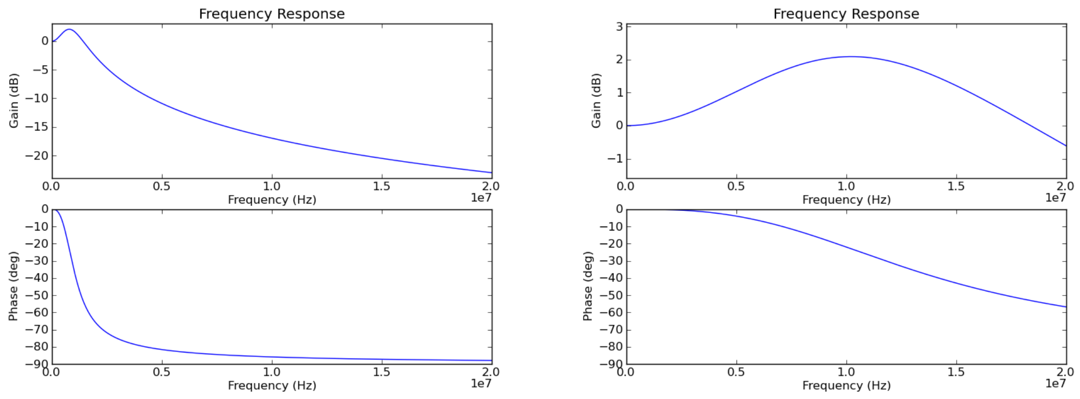


Figure 6.11: Cable transfer function



(a) $\omega n = 2\pi 1e6$, $\zeta = 0.707$ (b) $\omega n = 2\pi 13e6$, $\zeta = 0.707$

Figure 6.12: CRU transfer function

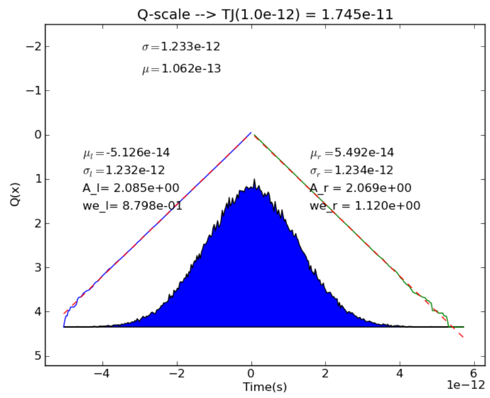
6.4.2 CRU-TF influence on plesiochronous communication systems (without frequency offset)

On this section the influence of clock recovery transfer function over plesiochronous system will be evaluated.

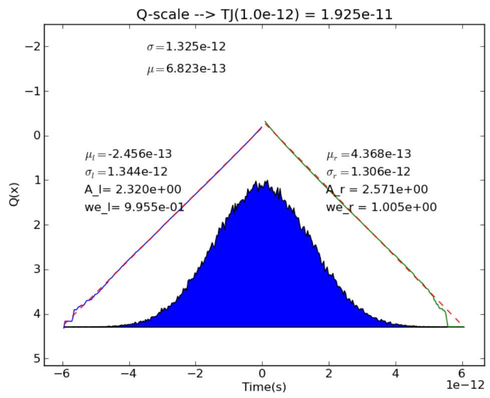
Starting with a simple model, PRBS5 data is generated via a 10GHz clock with $A_j = 0.2UI$ and $F_j = 500kHz$, 1ps of random jitter (only time jitter, amplitude noise was not added).

Figure 6.12 contains the frequency response of two different CRU-TF, 1MHz and 13MHz. Such transfer functions were obtained from eq. 6.1. Jitter histogram and eye diagram can be found on figures 6.15 and 6.16 respectively.

Jitter histogram represented on figure 6.15 shows in fact that the random jitter (1.3ps) is bigger than the present on clock signal used to generate PRBS5 data (1ps). Such fact is related with the clock extraction mechanism, since it is recovered from a PRBS5 signal instead of a clock signal.

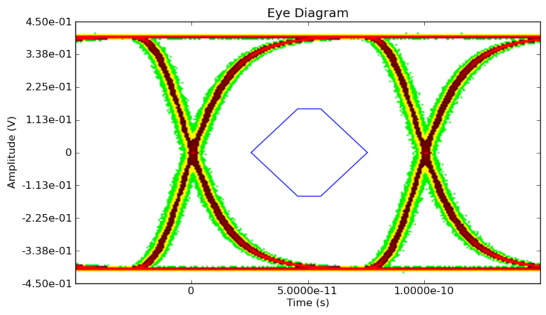


(a) CRU-TF ($\omega n = 2\pi 1e6$, $\zeta = 0.707$)

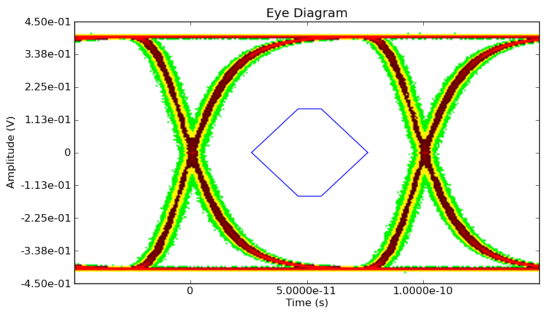


(b) CRU-TF ($\omega n = 2\pi 13e6$, $\zeta = 0.707$)

Figure 6.13: Jitter histogram, clock extracted from clock signal, PRBS15 as data with rj=1ps and 5mV as amplitude noise



(a) CRU-TF ($\omega n = 2\pi 1e6$, $\zeta = 0.707$)



(b) CRU-TF ($\omega n = 2\pi 13e6$, $\zeta = 0.707$)

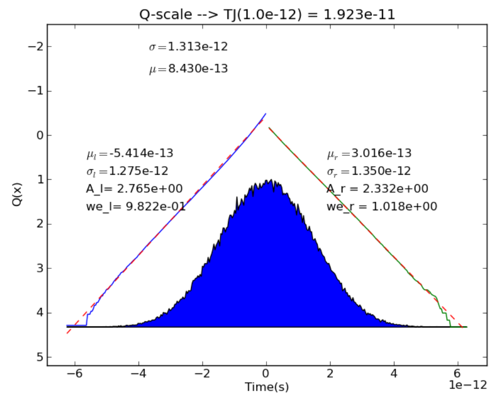
Figure 6.14: Eye diagram, clock extracted from clock signal, PRBS15 as data with rj=1ps and 5mV as amplitude noise

It is also important to notice that even this ideal system has more jitter than a clock sharing system with cable attenuation, supply random noise of 5mv and PRBS15 for data generation.

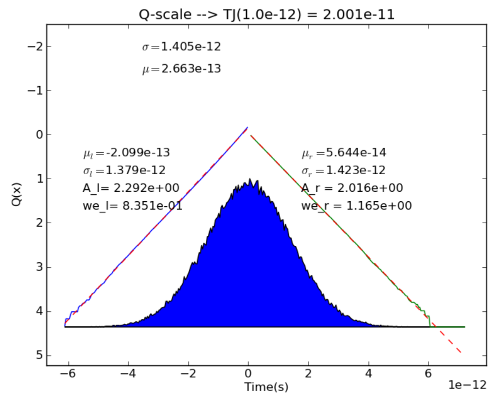
6.4.3 CRU-TF influence on plesiochronous communication systems (with frequency offset)

Leaving the ideal world, it would be good to have an evaluation of how a CRU-TF influences the system jitter when PRBS15 data pattern is used and a frequency offset is present. For this evaluation two data patterns were used: PRBS5 and PRBS15. Both were generated from a clock signal with central frequency of 10GHz, $A_j = 0.2UI$, $F_j = 500kHz$. The frequency offset added to CRU circuit was -2000ppm. Data signals have also 1ps of random jitter, 5mV of amplitude noise and the cable effect represent on figure 6.11.

CRU-TF used in this simulation was different from the previous ones. Instead of using a second order equation a first order was used. Frequency response functions used are represented

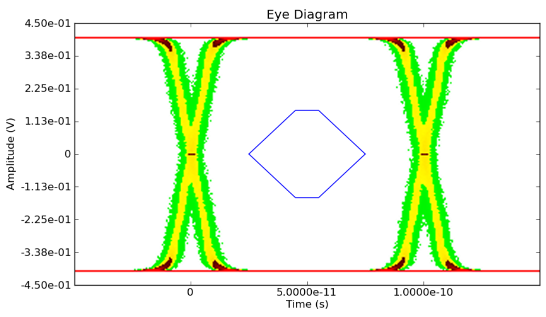


(a) CRU-TF ($\omega n = 2\pi 1e6$, $\zeta = 0.707$)

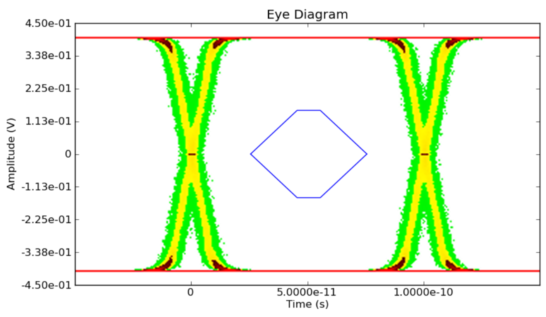


(b) CRU-TF ($\omega n = 2\pi 13e6$, $\zeta = 0.707$)

Figure 6.15: Jitter histogram, clock extracted from PRBS5 data with rj=1ps



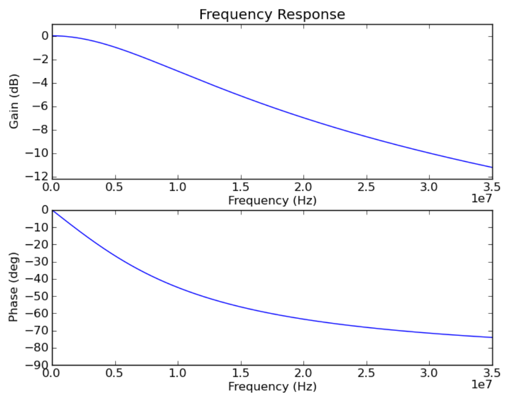
(a) CRU-TF ($\omega n = 2\pi 1e6$, $\zeta = 0.707$)



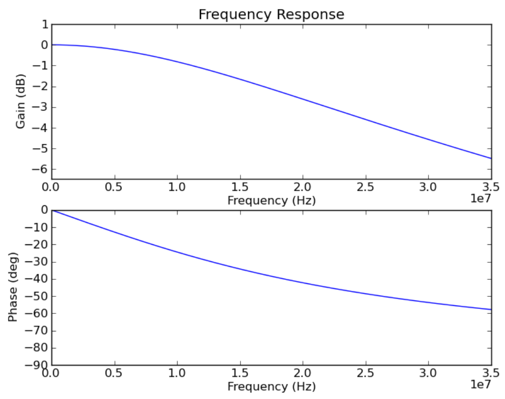
(b) CRU-TF ($\omega n = 2\pi 13e6$, $\zeta = 0.707$)

Figure 6.16: Eye diagram, clock extracted from PRBS5 data with rj=1ps

on figure 6.17. The -3dB frequencies are 10MHz and 22MHz respectively.



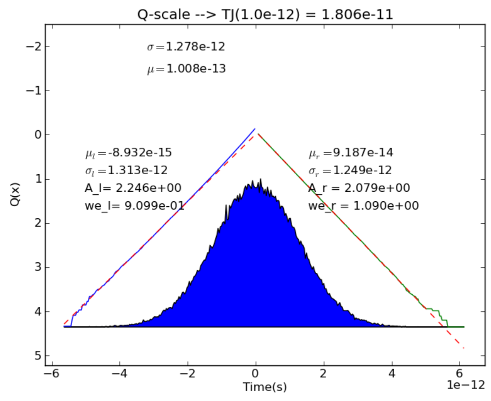
(a) CRU-TF ($f_{3dB} = 10MHz$)



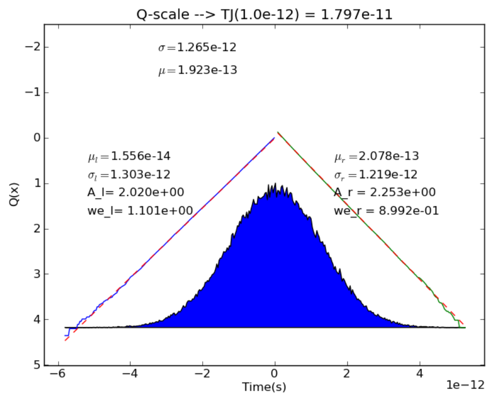
(b) CRU-TF ($f_{3dB} = 22MHz$)

Figure 6.17: CRU transfer function (1st order)

Figures 6.18 and 6.19 show the influence of CRU-TF on the overall system jitter. As expected the jitter values are bigger than when no frequency offset is considered.

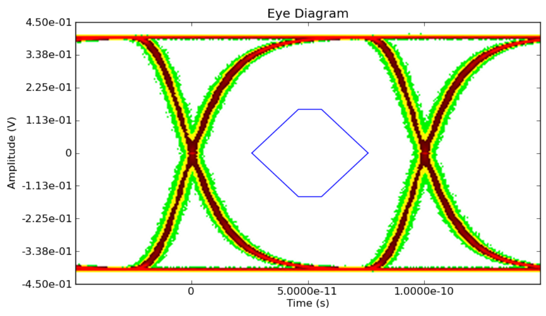


(a) CRU-TF ($f_{3dB} = 10MHz$)

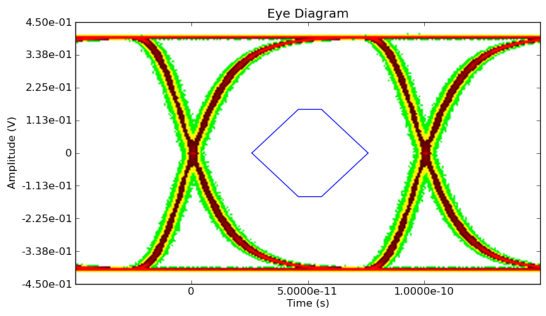


(b) CRU-TF ($f_{3dB} = 22MHz$)

Figure 6.18: Jitter histogram, clock extracted from PRBS5 data with rj=1ps, cable model, 5mV of random amplitude noise and 2000ppm of frequency offset



(a) CRU-TF ($f_{3dB} = 10MHz$)



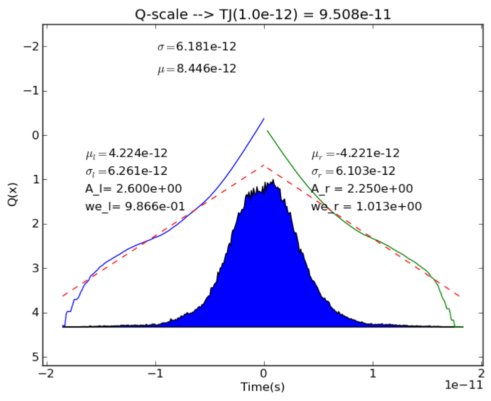
(b) CRU-TF ($f_{3dB} = 22MHz$)

Figure 6.19: Eye diagram, clock extracted from PRBS5 data with rj=1ps, cable model, 5mV of random amplitude noise and 2000ppm of frequency offset

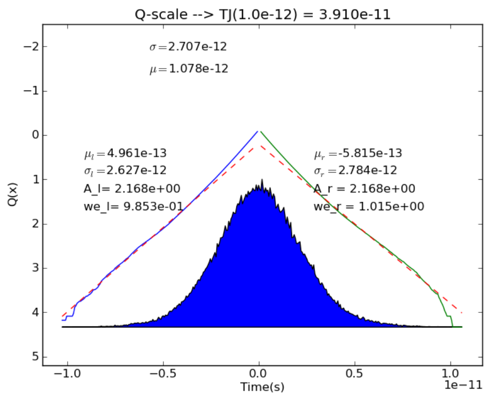
Figures 6.20 and 6.21 represent CRU-TF influence when PRBS15 sequence is used in data generation. Jitter values are bigger than on PRBS5, demonstrating the need for the use of encoders to ensure a low number of equal bits.

6.4.4 CRU-TF influence on clock shared communication systems with high frequency jitter components

An important concept to retain is that clock recovery units are intended to track low frequency jitter components, leaving the high frequency components to the CDR block. It is interesting to see what happens to the eye diagram and jitter histograms, when high frequency jitter components are added to signal under recovery. For this example let use clock shared communication system, due to a better tracking of jitter components present on clock signal.

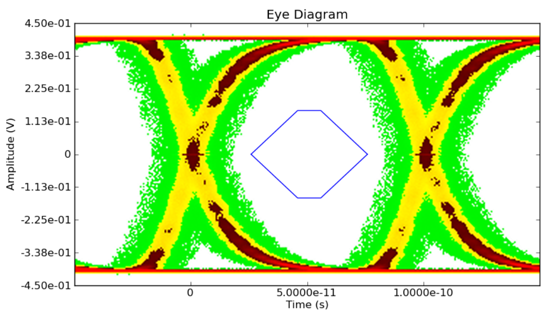


(a) CRU-TF ($f_{3dB} = 10MHz$)

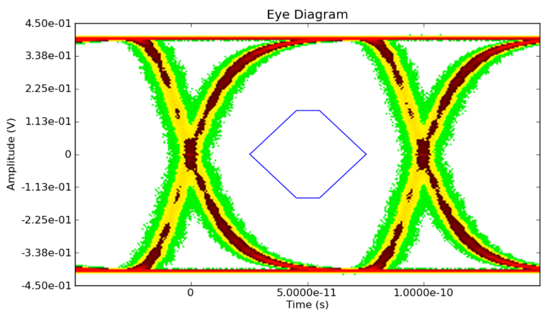


(b) CRU-TF ($f_{3dB} = 22MHz$)

Figure 6.20: Jitter histogram, clock extracted from PRBS15 data with rj=1ps, cable model, 5mV of random amplitude noise and 2000ppm of frequency offset



(a) CRU-TF ($f_{3dB} = 10MHz$)



(b) CRU-TF ($f_{3dB} = 22MHz$)

Figure 6.21: Eye diagram, clock extracted from PRBS15 data with rj=1ps, cable model, 5mV of random amplitude noise and 2000ppm of frequency offset

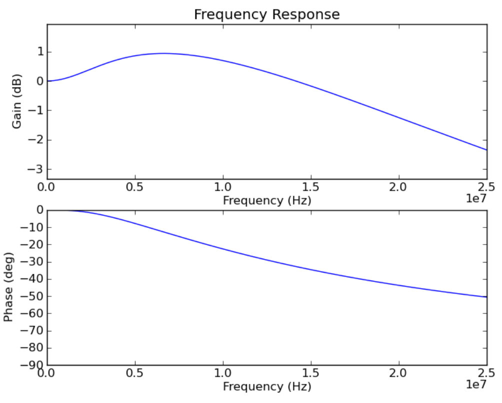
On this example a PRBS15 sequence was used. Clock signal was generated with a central frequency of 10GHz, $A_j = 0.2UI$, $F_j = 100MHz$. In terms of jitter, 1ps of random jitter and 5mV of amplitude jitter were added to the data and clock signals. Frequency response functions were obtained via eq. 6.1.

Figure 6.22 represents the CRU-TF used on this example. Case a) describes the frequency response of a high bandwidth PLL (10MHz). Case b) represents the combined effect of CDR and PLL, this explaining the high bandwidth (100MHz).

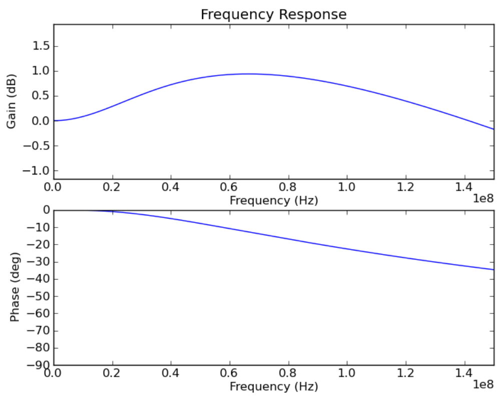
Figures 6.23 and 6.24 a) represent the jitter and eye diagram seen by CDR. This is the jitter that must be tracked by the CDR. The b) represents the jitter and eye diagram after the CDR. It is important to notice that the CDR cannot follow random jitter, and as expected, it will appear on the jitter histogram.

Periodic jitter is clearly seen on jitter histogram figure 6.23 a). This type of jitter is represented in jitter histogram as an "U" shape.

This example shows the importance of a correct definition of the CRU-TF, since a bad definition can be masking jitter components that would be seen in the CDR circuit, leading to wrong characterizations.

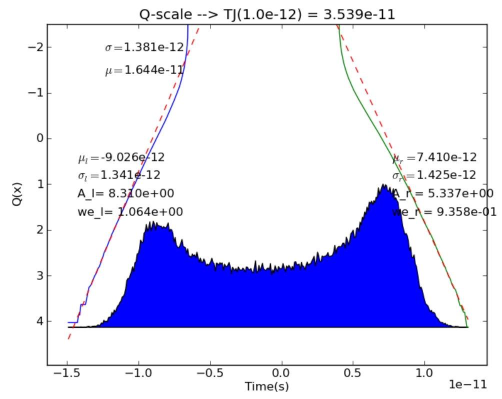


(a) $\omega n = 2\pi 10e6$, $\zeta = 1.2$

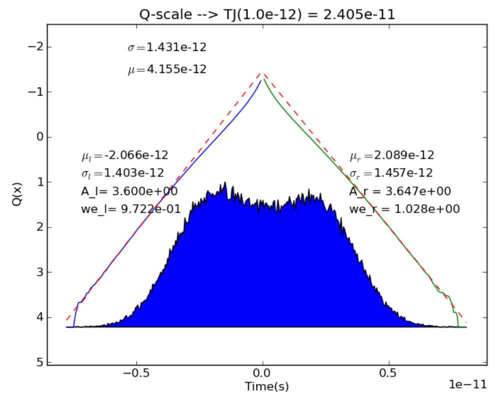


(b) $\omega n = 2\pi 100e6$, $\zeta = 1.2$

Figure 6.22: CRU transfer function for high frequency jitter tracking



(a) CRU-TF ($\omega n = 2\pi 10e6$, $\zeta = 1.2$)

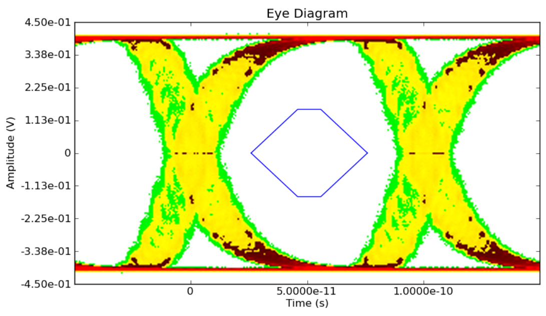


(b) CRU-TF ($\omega n = 2\pi 100e6$, $\zeta = 1.2$)

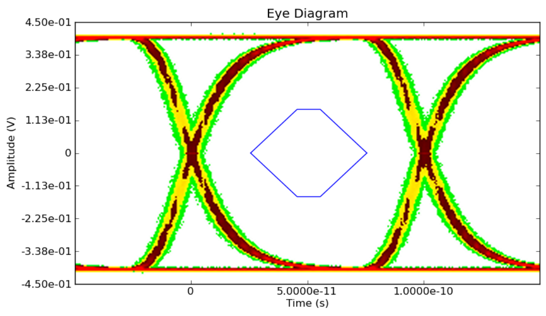
Figure 6.23: Jitter histogram, when 100MHz of jitter frequency is added to the clock signal

6.5 Transmitter characterization

Transmitter characterization is performed using data provided from lab or simulation environment. Such data will have to respect the csv format. The user needs to first read the csv data from the different channels to the software tool. Then it is just necessary to recover the clock signal and proceed with the signal analysis: jitter histogram, eye diagram, min rise and fall times, low and high differential values and minimum and maximum VHIGH and VLOW differential values.



(a) CRU-TF ($\omega n = 2\pi 10e6$, $\zeta = 1.2$)



(b) CRU-TF ($\omega n = 2\pi 100e6$, $\zeta = 1.2$)

Figure 6.24: Eye diagram, when 100MHz of jitter frequency is added to the clock signal

Data signals can be extracted from simulations. It is possible to represent the interconnection effects between device under test and test equipment. Typically the use of a 1st order low pass filter provides a good approximation.

Based on a specific design, a python script was created to ease the characterization of clock sharing transmitter devices. This script can be applied when the clock shared signal has a frequency 10 times lower than the frequency used to generate the serial data.

A configuration file must be provided to the script. The user can define the maximum number of lines that will be evaluated and the signal data column that is used when a single file contains the data from all channels. An example of a configuration file is presented below:

```
lines_max,100e6
channel1,from_sim\waves_ck_d0_d1_d2.csv,3
channel2,from_sim\waves_ck_d0_d1_d2.csv,4
channel0,from_sim\waves_ck_d0_d1_d2.csv,2
clk,from_sim\waves_ck_d0_d1_d2.csv,1
extrapol_eye,2.9411e-7,3.4e-12,1
lpf,7e9
report_name,from_sim\TX_dj_only.html
```

The presented configuration file defines the report file name, the low pass filter characteristic and allow the python script to perform an eye diagram extrapolation, with BER=2.9411e-7, rj=3.4e-12 and the jitter obtained from data channels should be considered as deterministic jitter only. In terms of CRU-TF a 1st order low pass filter with $f_{3dB} = 4MHz$ is used.

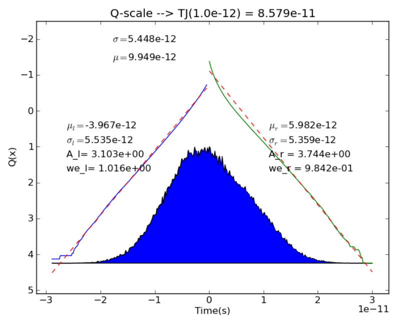
6.5.1 3.4Gbps transmitter characterization based on lab results

The Following configuration file was used to perform the transmitter characterization based on lab data. It is important to notice, that the extrapolated eye will be obtained, using the extracted jitter components (RJ):

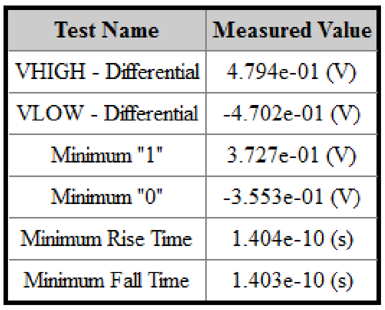
```
lines_max,100e6
```

```
channel0,From_Lab\data0.csv
clk,From_Lab\clk.csv
extrapol_eye,1e-12
report_name,From_Lab\TX.html
```

Figures 6.25, 6.26 and 6.27 were extracted from the reported created with the previously defined python script. The eye diagram jitter obtained with lab data was 57ps and the extrapolated was 83ps, matching the expected result from the jitter extrapolation based on random and deterministic jitter components.



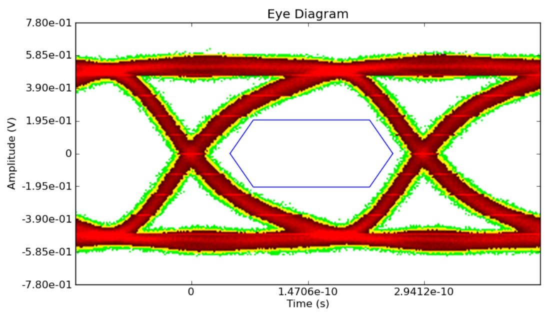
(a) Jitter histogram



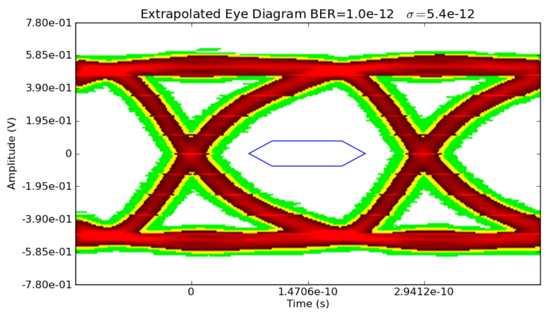
Test Name	Measured Value
VHIGH - Differential	4.794e-01 (V)
VLOW - Differential	-4.702e-01 (V)
Minimum "1"	3.727e-01 (V)
Minimum "0"	-3.553e-01 (V)
Minimum Rise Time	1.404e-10 (s)
Minimum Fall Time	1.403e-10 (s)

(b) Analog parameters table

Figure 6.25: Transmitter characterization based on lab results (jitter histogram, analog parameters)



(a) Obtained eye diagram



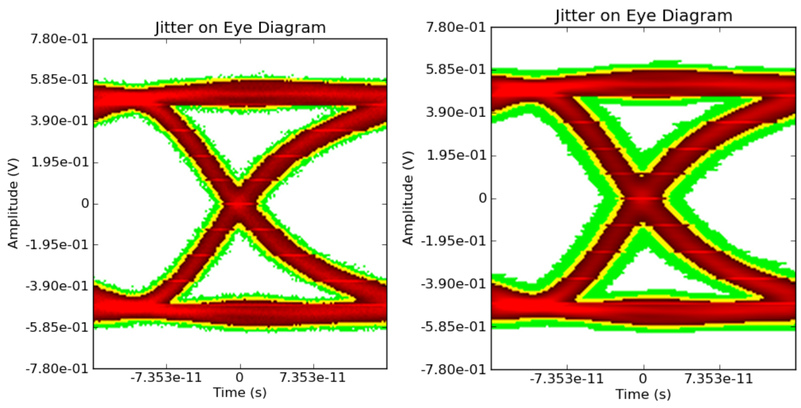
(b) Extrapolated eye diagram (BER=1e-12)

Figure 6.26: Transmitter characterization based on lab results (eye diagram, extrapolated eye diagram)

6.5.2 3.4Gbps transmitter characterization based on simulation results

Simulations were performed over the transmitter circuit under test, instead of acquiring 3.4Mbits as obtained in lab, only 31 kbits were acquired. The difference is related with time taken by simulation to achieve such value.

Simulation environment does not provide the random jitter value, intrinsically defined by the technology process, where the design will be implemented. Jitter observed is due to the PRBS



(a) Obtained eye diagram jitter (b) Extrapolated eye diagram jitter(BER=1e-12)

Figure 6.27: Transmitter characterization based on lab results (eye diagram jitter, extrapolated eye diagram jitter)

(mod 15) sequence used on the data generation. PRBS sequences tend to produce jitter similar to Gaussian distributions, which can lead to wrong extraction jitter values. The only way of overcoming this fact, is trough long acquisitions that will ensure the presence of very low probability events, allowing a correct used of dual-Dirac jitter extraction method.

Two configuration files were used:

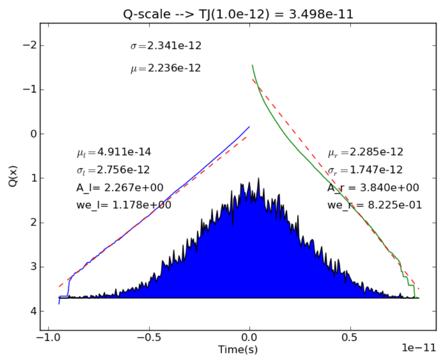
```
lines_max,100e6
channel0,                  from_sim\
sim_waves_ck_d0_d1_d2.csv,2
clk,from_sim\
sim_waves_ck_d0_d1_d2.csv,1
extrapol_eye,2.9411e-7,3.4e-12,1
lpf,7e9
report_name,from_sim\TX_dj_only.html
```

```
lines_max,100e6
channel0,                  from_sim\
sim_waves_ck_d0_d1_d2.csv,2
clk,from_sim\
sim_waves_ck_d0_d1_d2.csv,1
extrapol_eye,2.9411e-7, 5.7e-12
lpf,7e9
report_name,from_sim\TX.html
```

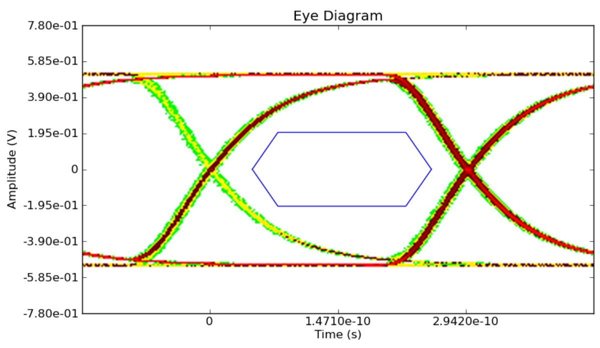
Two different approaches were used for the jitter extrapolation. In a) random jitter value extracted from simulation is taken in consideration for eye extrapolation. In b) Simulation jitter is considered as only deterministic. For a better match between simulation and lab results a low pass filter ($f_{3dB} = 7GHz$) was used, to represent the package, connectors and cable used in lab characterization environment.

Figures 6.28 and 6.29 contain the simulation results. In terms of jitter the maximum observed jitter in simulation was 22ps. Analog parameters extracted from simulation are similar to the ones extracted from lab characterization (the accuracy of the simulation can be increased).

The numbers of bits captured on test characterization were 100 times more than the captured in simulation. This is clearly seen on the eye diagram jitter.

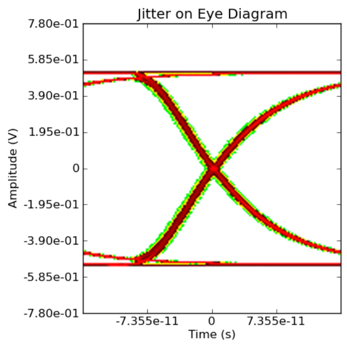


(a) Simulation jitter histogram

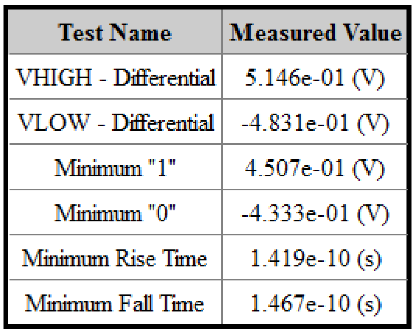


(b) Simulation eye diagram

Figure 6.28: Simulation results - jitter histogram and eye diagram



(a) Simulation eye diagram jitter



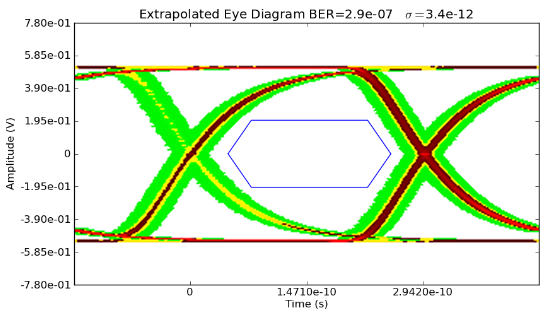
Test Name	Measured Value
VHIGH - Differential	5.146e-01 (V)
VLOW - Differential	-4.831e-01 (V)
Minimum "1"	4.507e-01 (V)
Minimum "0"	-4.333e-01 (V)
Minimum Rise Time	1.419e-10 (s)
Minimum Fall Time	1.467e-10 (s)

(b) Simulation - analog parameters table

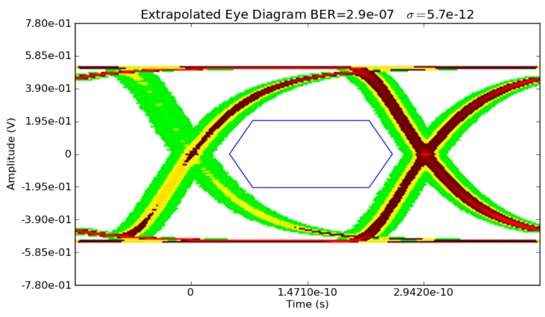
Figure 6.29: Simulation results - jitter on eye diagram and analog parameters table

In order to predict what will be the eye diagram obtained in characterization based only on simulation results, two similar approaches were followed (a) and b)). The difference is related with the eye extrapolation algorithm. In a) the jitter obtained in simulation is considered as deterministic. This jitter will be then convolved with a Gaussian distribution with $\sigma = 3.4ps$. In b) the jitter obtained in simulation is considered as a combination of random and deterministic jitter. The Gaussian distribution that will be convolved with the simulation eye diagram reflects this effect, to ensure that the convolution between the Gaussian distribution and eye diagram results in a random jitter described by a Gaussian distribution with $\sigma = 5.7ps$.

Results for eye diagram extrapolation a) and b) approaches can be found in figure 6.30. The eye diagram was extrapolated to $BER = 2.94e-7$, to be in accordance with the number of bits captured by test equipment. In terms of jitter a) has a total jitter of 53ps and b) of 59ps. The a) approach seems to be more correlated with the eye diagram obtained in lab characterization. This can lead to the conclusion that jitter decomposition obtained from lab results is not accurate. As stated before a correct extraction of jitter based on dual-Dirac method requires the presence of very low probability events, leading to the conclusion that for correct jitter decomposition more bits should be captured in lab.



(a) Simulation eye diagram jitter extrapolation (rj=3.4ps, eye sim jitter considered as deterministic only)



(b) Simulation eye diagram jitter extrapolation (rj=5.7ps)

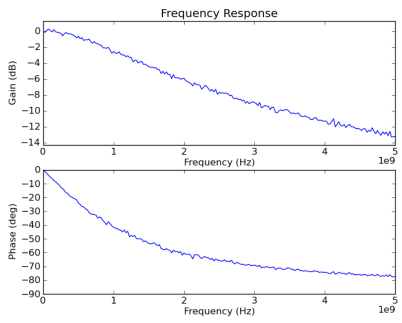
Figure 6.30: Eye diagram extrapolation based on simulation results

6.6 Channel characterization

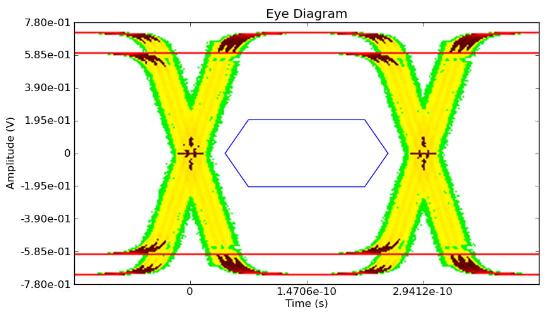
Channel characterization plays an important role in system level architecture definition. From this environment will emerge the specifications for the circuit is implementation, in terms of equalizer transfer function, jitter tolerance, jitter budget, etc.

Cable transfer functions are not always smooth. Presented tool has the ability to add random noise to the defined transfer function. An example of such behavior is presented on figure 6.31-a).

Pre-emphasis can play an important rule on eye opening. Sometimes a simple 20% of pre-emphasis boost allows a transmitter to pass eye diagram restrictions. A representation of such behavior can be found on figure 6.31-b).



(a) Cable transfer function with random noise

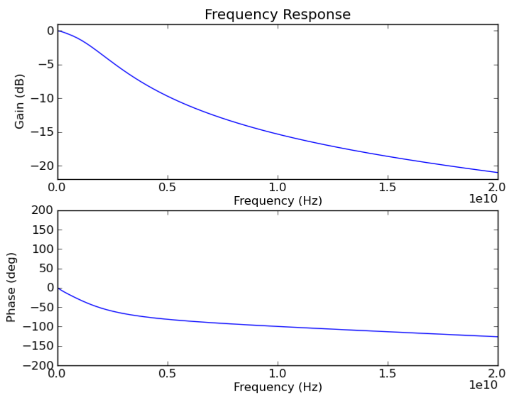


(b) Eye diagram of a signal with pre-emphasis

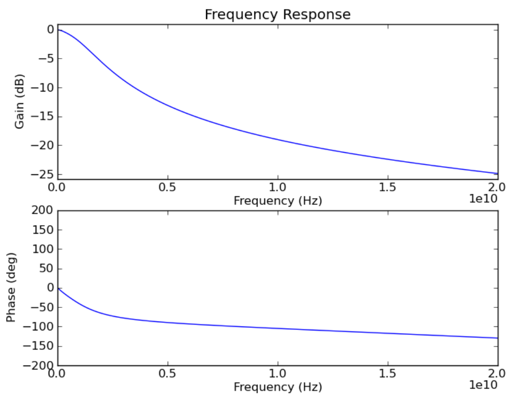
Figure 6.31: Generic examples

Channel characterization starts with the definition of the link transfer function, since the frequency of operation is dictated by the communication standard (nothing can be done in relation to it). Link means cable plus connectors plus PCB traces plus "other" connection elements.

Figure 6.32 describes the transfer functions of two different links, a) 4cm of PCB FR-4 traces, followed by a 1m cable, followed by more 4cm of PCB traces. Case b) 4cm of PCB FR-4 traces, followed by a 2m cable, followed by more 4cm of PCB traces.



(a) Transfer function of 8cm PCB traces with hypothetic 1m cable



(b) Transfer function of 8cm PCB traces with hypothetic 2m cable

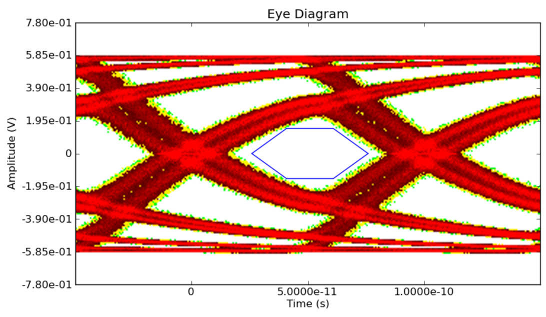
Figure 6.32: PCB and cable transfer function

Figure 6.32 represents the link transfer function from 0 to 20GHz; the frequency of operation will be on the 5GHz band.

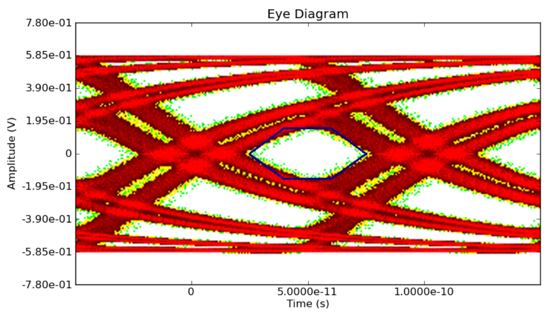
In order to have a good simulation accuracy, $Fmax$ was defined to 100GHz resulting on narrow time steps on the impulse response.

A plesiochronous system was used without frequency offset. A PRBS7 data pattern was generated from a clock signal with 10GHz as central frequency, $F_j = 100kHz$, $A_j = 0.2UI$, 4ps of deterministic jitter and 0.5ps of random jitter.

Figure 6.33 presents the eye diagram after link a) with and without pre-emphasis boost. As stated before, pre-emphasis can increase the eye opening leading to a pass condition. Today's transmitters have support for pre-emphasis boost on most cases.



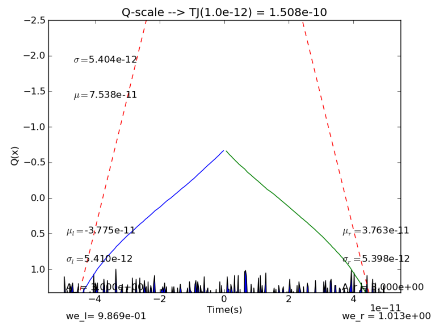
(a) With 20% of pre-emphasis boost



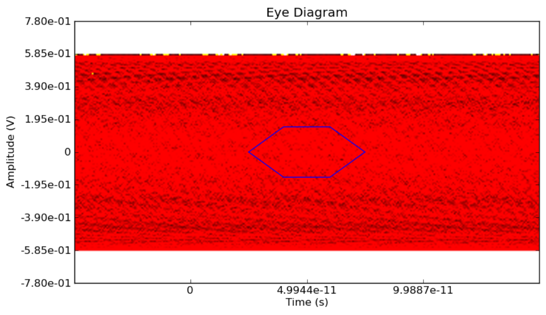
(b) Without pre-emphasis

Figure 6.33: Eye diagram after 1m cable and 8cm of PCB traces

Figure 6.34 represents the effect of link b). There is no eye opening, even with 20% of pre-emphasis boost. The solution for this problem is the use of an equalizer prior to the receiver stage; typically receiver devices support adaptive equalization.



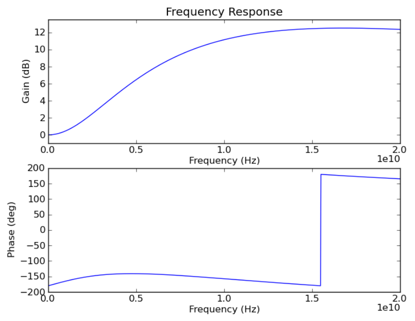
(a) Jitter histogram



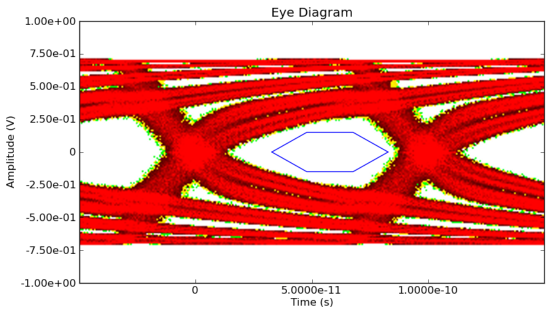
(b) Eye diagram

Figure 6.34: Jitter histogram and eye diagram after 2m cable and 8cm of PCB traces, PRBS7 signal with 20% of pre-emphasis

An adaptive equalizer is an equalizer that automatically adapts the internal characteristics to match the link. Figures 6.35, 6.36, 6.37 and 6.38 represent the eye diagram for different equalizer transfer functions. Optimum equalizer transfer function will be link dependent. This fact leads to adaptive equalizers, which have the ability to select the appropriate transfer function from the available ones automatically. The end user will just plug and unplug different cables/equipment and the overall functionality will be the same.



(a) Equalizer transfer function



(b) Eye diagram

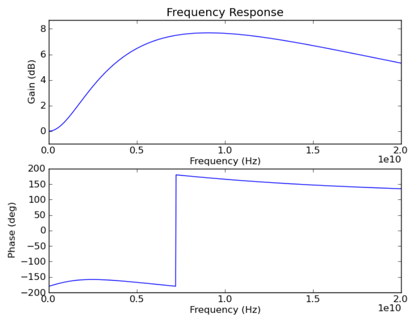
Figure 6.35: Equalizer C effect on eye diagram (under-equalization)

6.7 Conclusion

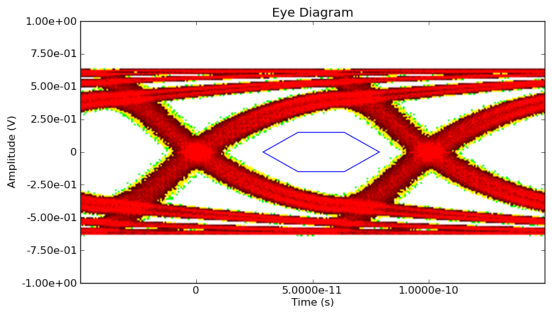
In this chapter, detailed results obtained with the developed tool were presented. It was demonstrated the ability to proceed with a system characterization using the python signal analysis tool.

The use of python reduces the code complexity, allowing faster development stages. System engineers would not need a deep understanding of python to be able to work with this tool.

The advantage of using python is the outstanding support for objects oriented programming, as the availability of free scientific libraries.

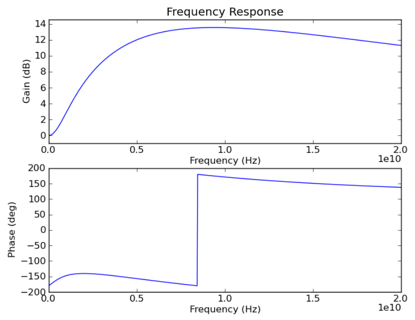


(a) Equalizer transfer function

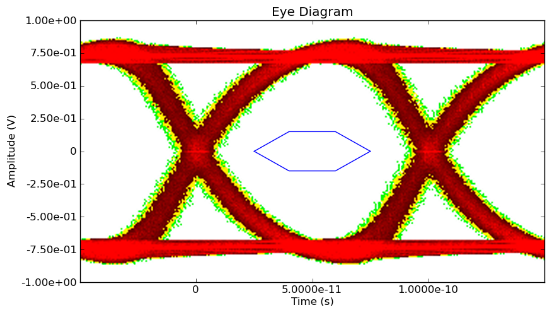


(b) Eye diagram

Figure 6.36: Equalizer A effect on eye diagram (under-equalization)

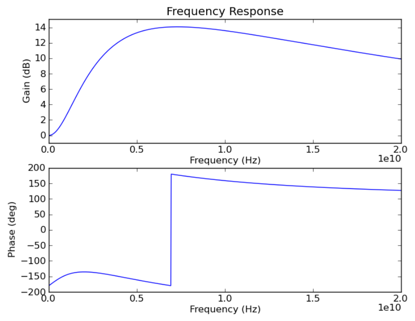


(a) Equalizer transfer function

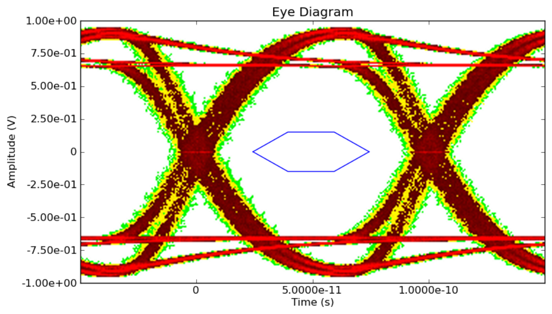


(b) Eye diagram

Figure 6.37: Equalizer B effect on eye diagram (good-equalization)



(a) Equalizer transfer function



(b) Eye diagram

Figure 6.38: Equalizer C effect on eye diagram (over-equalization)

The results presented on this chapter were extracted from simulation reports. Such reports can be found under: paginas.fe.up.pt/ ee10005/docs_page.php.

Chapter 7

Conclusions and Further Work

7.1 Conclusions

In this report a new tool was presented, capable of performing signal analysis at the simulation level. The accuracy is about the same that the user can have when using commercial tools like: Spice or Oscilloscope software.

Traditional signal analysis tools are expensive, requiring extensive training. In opposition the presented tool is based on high level classes, requiring from the user the knowledge of a few list of commands.

Python was the chosen language making the tool independent of the operation system. Another advantage of using such language relates to the scripting capability, providing the ability to run the simulations without the user interference, leading to a productivity increment.

Costs reduction are always in demand, the presented tool can lead to improvements on the development process, reducing the number of top level simulations performed, leading to a cheaper development process.

Frequency domain to time domain, frequency response extrapolation, clock recovery unit, interconnection modeling, jitter extraction and jitter decomposition define the most important actions provided by the presented tool.

Traditional simulation environments doesn't have the ability to perform signal processing tasks, like the ones performed by oscilloscopes at the characterization phase. The presented tool brings to the simulation environment that ability.

The applications of the proposed tool can vary from channel characterization, to transmitter characterization. This tool supports different environments, ensuring the compatibility with different interconnection standards.

7.2 Further work

The goals of the present work were met. Although the signal analysis tool could support more features like: CDR model, real cable models, real PLL/DLL models and a few more signal analysis algorithms. The foundations for future works on this area are done, it is just necessary passion and dedication to increase the functionality of this tool.

The presented tool doesn't have a graphical interface, since the productivity is bigger with scripts than buttons. A mixed-mode solution can be found, reducing the necessary time to start working with this tool.

The frequency extrapolation algorithm that was developed is far from being finished. Extra work should be done to make sure that the algorithm is robust enough. The boundary between real values and extrapolated values should be analyzed with care, some changes could be done to the algorithm in order to ensure a smooth transition (without ups and downs).

CUDA support can be another enhancement, giving to the python tool the ability to perform convolutions on graphical processors will decrease the simulation time. The convolution was just an example, there are other algorithms suitable to run on graphical processors.

During the development phase the idea of writing a scientific paper has emerged. The subject could vary from jitter decomposition algorithm to frequency response extrapolation or about the python tool.

The work is not finished, it just began.

Appendix A

Channel Characterization Example

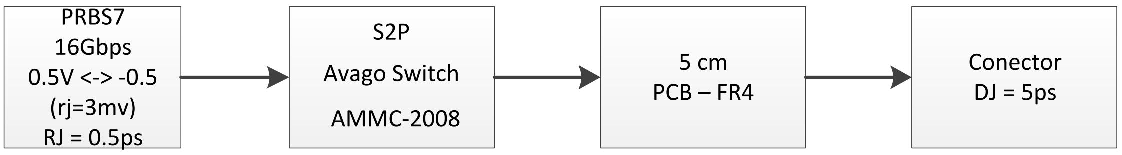


Figure A.1: Channel characteristics

This example defines a channel to be characterized as having the characteristics defined in figure A.1, its a plesiochronous system without frequency offset. The clock signal used to generate the PRBS7 data has a sinusoidal jitter component defined by $Aj = 0.1$ and $Fj = 50KHz$.

The python script starts with the packages import:

```
import os, sys
import sig_analysis as sig
import copy
import numpy as np
import matplotlib.pyplot as plt
```

The signal analysis package, in this example, is imported as "sig"; the package internal functions are reference by "sig.<function>". The simulation will generate a report, which will be stored in html format. The filename is defined through:

```
sig.Report_Filename = 'Channel_Example/Channe_16Gbps.html'
```

Report file should be created at the beginning of the script:

```
sig.c_html.create()
```

The simulation environment needs to have defined the $Fmax$ variable, in this example it is settled to 100MHz:

```
sig.Fmax = 100e9
```

The previous commands have defined the parameters for the simulation. Moving forward, the data generation needs a clock edges vector to produce the data sequence. The clock edges is generated through:

```
clk_c             = sig.clock_gen(0.1,16e9,50e3,2.0e-5)
clock_edges       = sig.clock_edges_extract(clk_c)
```

The PRBS7 analog signal is then generated through:

```
data_s = sig.data_generation(clock_edges,'PRBS7',0.5,-0.5,20e-12,20e-12,rj=3e-3)
```

Next step is to produce the signal after the switch. The frequency response of the switch is defined as a .s2p file, leading to the use of s21_param class to obtain the impulse response:

```
avago_switch = sig.s21_param(filename)
#IR from avago switch
avago_switch.get_impulse_resp()
#signal after switch
data_switch = sig.ana_convolve(avago_switch,data_s)
```

The PCB element has "data_switch" signal as input. The signal after pcb can be generated through:

```
pcb = sig.cable_model()
pcb.add_pcb_element(0.05)
# Get Impulse Response of PCB element
pcb_ir = sig.calc_inp_coef(pcb)
#signal after pcb
data_pcb = sig.ana_convolve(pcb_ir,data_switch)
```

The system is plesiochronous leading to a clock extraction from "data_pcb" signal. In this example a low pass filter with $f_{3dB} = 2MHz$ is used as CRU-TF:

```
#Edges extraction from data_pcb signal
data_edges = sig.clock_edges_extract(data_pcb,0.0,'PRBS7_after_PCB')
# CRU-TF definition
cru_tf  = sig.generic_filter([],[-2*np.pi*2e6],2*np.pi*2e6)
cru  = sig.cru_func_to_ir('generic',cru_tf)
#Golden edges extraction
clok_data_cable = sig.clok_recovery(data_edges,cru, \
                          'PRBS7_after_PCB',1.0/16e9)
```

The ideal edges position are stored in "clok_data_cable" variable. The jitter extraction via TIE method is performed by the following commands:

```
tie_dt          = sig.calc_analog_params(clok_data_cable,data_pcb,\
                     clok_data_cable.ind_start, \
                     data_type='PRBS7_after_PCB')
tie_dt.extract_jitter()
```

The "tie_dt" variable is used to initialize the calc_analog_params class. The extract jitter function is internal to the class and can be called by "." method (tie_dt.extract_jitter).

To proceed with the eye diagram generation, is necessary to previously define the eye diagram mask:

```
class_eye_mask  = sig.eye_mask()
class_eye_mask.bot_top = 0.6 # Volt
class_eye_mask.bot_top_mask = 0.1 #Volt
class_eye_mask.zero_mask = 0.35 #UI
class_eye_mask.top_bot_mask = 0.4 # UI
```

The eye diagram generation function is then called by:

```
tie_dt.plot_eye_diagram(class_eye_mask)
```

In this example an extrapolated eye diagram is generated, assuming $BER = 2^{-11}$ the command is:

```
tie_dt.extrapol_eye_diagram(class_eye_mask_ext,2e-11)
```

The complete python script can be found in chapter A.1, the report file can be found in chapter A.2.

A.1 Python script

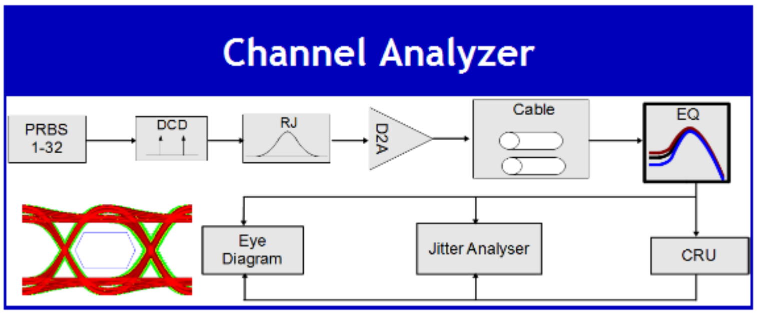
```
import os, sys
import sig_analysis as sig
import copy
import numpy as np
import matplotlib.pyplot as plt
def main():
  # Report File Creation
  data_name = 'Avago_Switch_50G_Extrapol'
  sig.Report_Filename = 'Channel_Example/Channe_16Gbps.html'
  filename = 'Frequency_extrapol/Switch_Avago/ammc2008_2_on_a.s2p'

  sig.c_html.create()
  # Maximum frequency definition
  sig.Fmax = 100e9
```

```
# Signal Generation
clk_c            = sig.clock_gen(0.1,16e9,50e3,2.0e-5)
clock_edges      = sig.clock_edges_extract(clk_c)
#Random jitter addition 0.5ps
clock_edges.add_jitter(0,0.5e-12)
#Deterministic jitter addition 5ps
clock_edges.add_jitter(5e-12,0)
#Signal generation with amplitude random jitter =3mV
data_s           = sig.data_generation(clock_edges,'PRBS7',\
                   0.5,-0.5,20e-12,20e-12,rj=3e-3)

# Header in Report
sig.c_html.title_break('Switch Characteristics')

#s2p file import
avago_switch = sig.s21_param(filename)
#IR from avago switch
avago_switch.get_impulse_resp()
plt.close('all')
#Plot Frequency Response
avago_switch.plot_freq_resp(data_type=data_name)
plt.close('all')
#Plot Impulse Response
avago_switch.plot_imp_resp(data_type=data_name)
plt.close('all')

#signal after switch
data_switch = sig.ana_convolve(avago_switch,data_s)

sig.c_html.title_break('PCB Characteristics')
# 5cm PCB
pcb = sig.cable_model()
pcb.add_pcb_element(0.05)
pcb.plot_freq_resp(data_type='5 cm of PCB')
plt.close('all')
# Get Impulse Response of PCB element
pcb_ir = sig.calc_inp_coef(pcb)

#signal after pcb
```

```
data_pcb = sig.ana_convolve(pcb_ir,data_switch)

sig.c_html.title_break('Signal Analysis')

data_edges = sig.clock_edges_extract(data_pcb,0.0,'PRBS7_after_PCB')



cru_tf                 = sig.generic_filter([],[-2*np.pi*2e6],2*np.pi*2e6)
cru                    = sig.cru_func_to_ir('generic',cru_tf)
cru.plot_freq_resp(20e6,'CRU-TF')
plt.close('all')
clok_data_cable        = sig.clok_recovery(data_edges,cru, \
                         'PRBS7_after_PCB',1.0/16e9)

#Eye diagram mask definition
class_eye_mask  = sig.eye_mask()
class_eye_mask.bot_top = 0.6 # Volt
class_eye_mask.bot_top_mask = 0.1 #Volt
class_eye_mask.zero_mask = 0.35 #UI
class_eye_mask.top_bot_mask = 0.4 # UI

#Signal Analysis
tie_dt          = sig.calc_analog_params(clok_data_cable,data_pcb,\
                  clok_data_cable.ind_start, \
                  data_type='PRBS7_after_PCB')
tie_dt.extract_jitter()
plt.close('all')
tie_dt.plot_eye_diagram(class_eye_mask)
plt.close('all')
tie_dt.plot_jitter_eye()
plt.close('all')

# Eye diagram extrapolation for BER=2e-11
class_eye_mask_ext  = sig.eye_mask()
class_eye_mask_ext.bot_top = 0.6 # Volt
class_eye_mask_ext.bot_top_mask = 0.05 #Volt
class_eye_mask_ext.zero_mask = 0.37 #UI
class_eye_mask_ext.top_bot_mask = 0.42 # UI
tie_dt.extrapol_eye_diagram(class_eye_mask_ext,2e-11)
```

```
  sig.c_html.close()
if __name__ == "__main__":
  sys.exit(main())
```

A.2 Simulation html report file



Channel_Example/Channe_l6Gbps Report

2011-06-26 15:22

Switch Characteristics

- **Frequency Response function of Avago_Switch_50G_Extrapol**

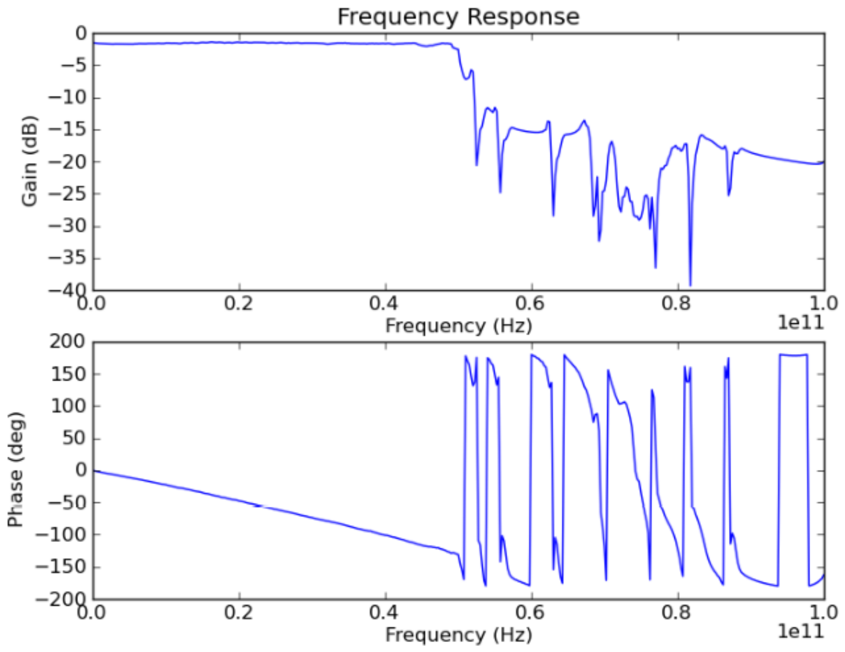


Fig:1 Frequency Response function of Avago_Switch_50G_Extrapol

- **Impulse Response function of Avago_Switch_50G_Extrapol**

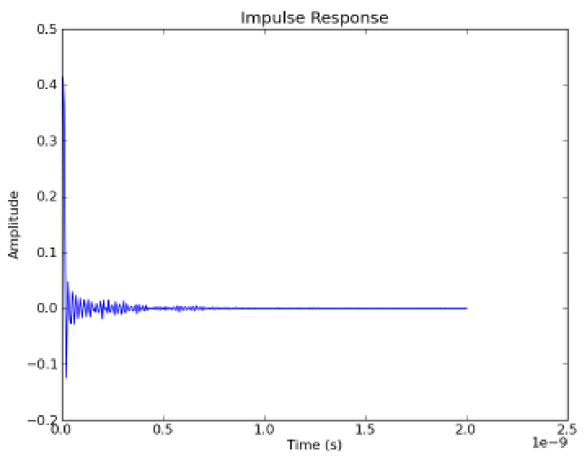


Fig:2 Impulse Response function of Avago_Switch_50G_Extrapol

PCB Characteristics

- **Frequency Response 5 cm of PCB**

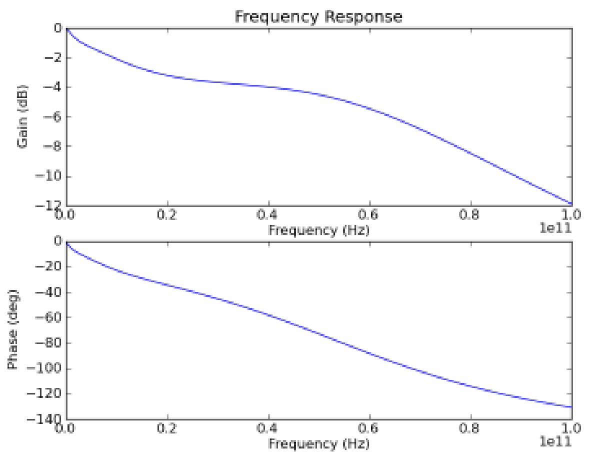


Fig:3 Frequency Response 5 cm of PCB

Signal Analysis

- **generic - CRU Frequency Response function of CRU-TF**

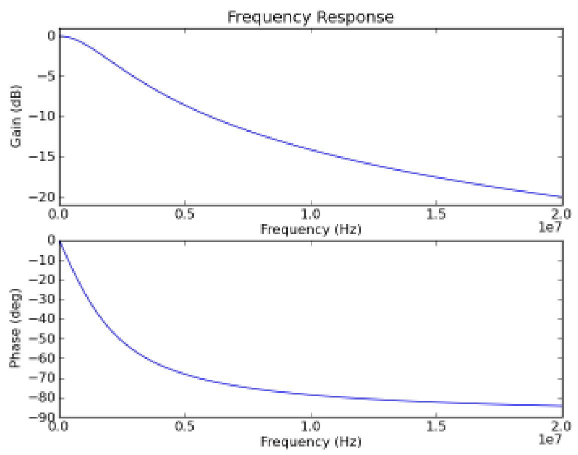


Fig:4 generic - CRU Frequency Response function of CRU-TF

Analog Parameters on PRBS7_after_PCB channel

- Jitter Values on PRBS7_after_PCB

Maximum Measured Data Jitter	TJ(BER=1.000e-12)
2.25930e-11 (s)	3.70576e-11 (s)

- Jitter Components on PRBS7_after_PCB

TJ	DJ	RJ	Qber_r	Qber_l	A_r	A_l
3.7058e-11 (s)	4.9347e-12 (s)	2.3068e-12 (s)	6.9777e+00	6.9468e+00	9.9593e-01	1.0041e+00

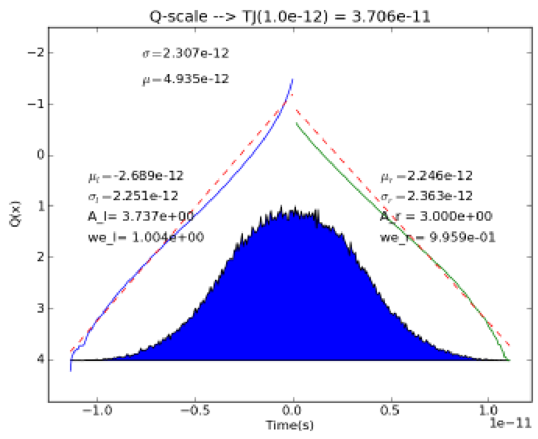


Fig:5 PRBS7_after_PCB Jitter PDF

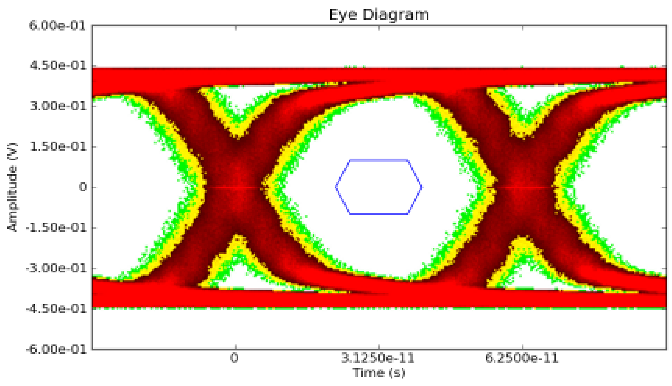


Fig:6 PRBS7_after_PCB Eye Diagram

- Analog Signal Analysis PRBS7_after_PCB

Test Name	Measured Value
VHIGH - Differential	3.993e-01 (V)
VLOW - Differential	-3.856e-01 (V)
Minimum "1"	3.333e-01 (V)
Minimum "0"	-3.200e-01 (V)
Minimum Rise Time	1.310e-11 (s)
Minimum Fall Time	1.351e-11 (s)

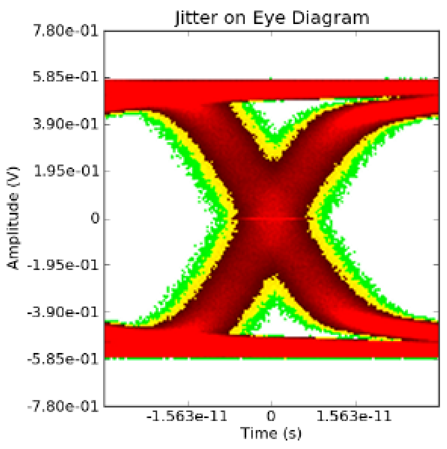


Fig:7 PRBS7_after_PCB Jitter on Eye Diagram Obt. Jitter = 2.031e-11 (s)

- **Eye Diagram Extrapolation Considering that 5.00e+10 bits were captured**

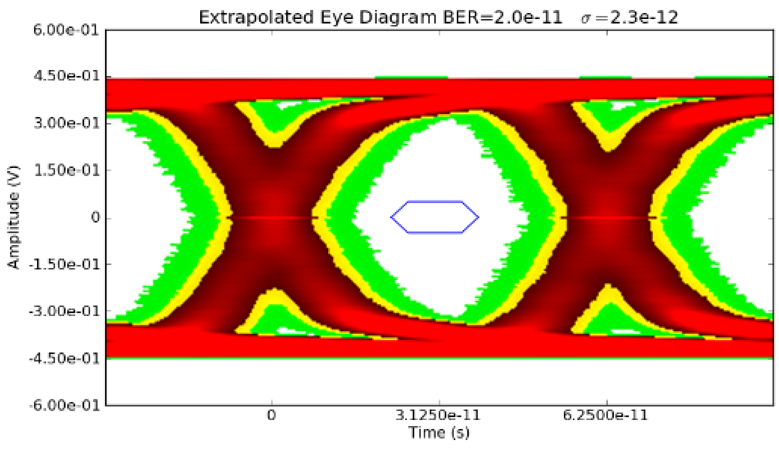


Fig:8 PRBS7_after_PCB Eye Diagram Extrapolated (RJ=2.3e-12)

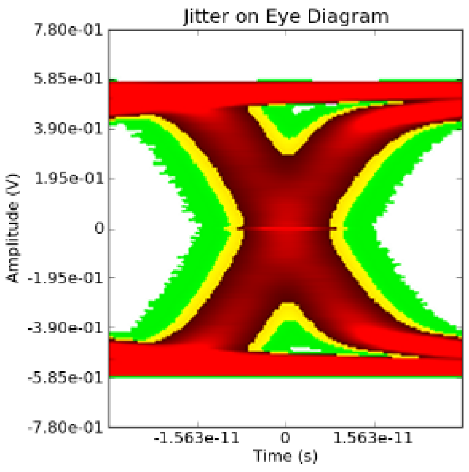


Fig:9 PRBS7_after_PCB Jitter on Eye Diagram Obt. Jitter = 2.969e-11 (s)

References

[1] S. Gondi and B. Razavi, “Equalization and clock and data recovery techniques for 10-gb/s cmos serial-link receivers,” *Solid-State Circuits, IEEE Journal of*, vol. 42, no. 9, pp. 1999 –2011, sept. 2007.

[2] N. Ou, T. Farahmand, A. Kuo, S. Tabatabaei, and A. Ivanov, “Jitter models for the design and test of gbps-speed serial interconnects,” *Design Test of Computers, IEEE*, vol. 21, no. 4, pp. 302 – 313, july-aug. 2004.

[3] T. Yamaguchi, M. Soma, M. Ishida, T. Watanabe, and T. Ohmi, “Extraction of instantaneous and rms sinusoidal jitter using an analytic signal method,” *Circuits and Systems II: Analog and Digital Signal Processing, IEEE Transactions on*, vol. 50, no. 6, pp. 288 – 298, june 2003.

[4] N. Kiddinapillai and T. Kwasniewski, “Jitter tolerance estimation of a 3x oversampling cdr using event-driven simulation,” in *Microsystems and Nanoelectronics Research Conference, 2009. MNRC 2009. 2nd*, oct. 2009, pp. 136 –139.

[5] R. Stephens, “Analyzing jitter at high data rates,” *Communications Magazine, IEEE*, vol. 42, no. 2, pp. S6–10, feb 2004.

[6] R. Stephens, “Jitter Analysis: The dual-Dirac Model, RJ/DJ, and Q-Scale,” A. Technologies, Ed. White Paper, Agilent Technologies, 2004.

[7] M. Miller, “Normalized Q-scale analysis: Theory and background,” *EDN*, 2007.

[8] D. Hong and K.-T. Cheng, “An accurate jitter estimation technique for efficient high speed i/o testing,” in *Asian Test Symposium, 2007. ATS ’07. 16th*, oct. 2007, pp. 224 –229.

[9] M. S. Martin Miller, “A comparison of Methods for Estimating Total Jitter Concerning Precision, Accuracy and Robustness,” *DesignCon*, 2007.

[10] Stateye, “Statistical eye,” 2007, http://www.stateye.org/.

[11] B. G. J. G. S. A. A. K. J. U. A. C. A. W. Juan Carlos Chaves, John Nehrbass and O. Siddharth Samsi Ohio Supercomputer Center, Columbus, “A High-Level Scripting Languages Productivity and Performance Evaluation,” *HPCMP Users Group Conference (HPCMP-UGC’06)*, 2006, http://www.osc.edu/research/cse/projects/octave_python.pdf.

[12] S. Narayana, G. Rao, R. Adve, T. Sarkar, M. Wicks, and S. Scott, “Interpolation/extrapolation of frequency domain responses using the hilbert transform,” in *Signals, Systems, and Electronics, 1995. ISSSE ’95, Proceedings., 1995 URSI International Symposium on*, oct 1995, pp. 335 –338.

[13] R. A. M. W. Sharath Narayana, Tapan K. Sarkar and V. Vannicola, “A Comparison of Two Techniques for the Interpolation/Extrapolation of Frequency Domain Responses,” *Digital Signal Processing*, 1996.

[14] F. R. Colin Warwick, “S-parameters Without Tears,” *EETimes Design*, 2010.

[15] T. Brazil, “Causal-convolution-a new method for the transient analysis of linear systems at microwave frequencies,” *Microwave Theory and Techniques, IEEE Transactions on*, vol. 43, no. 2, pp. 315 –323, feb 1995.

[16] G. C. R. M. E. E. M. S. S. N. Lalgudi, K. Srinivasan and Y. Kretchmer, “Causal Transient Simulation of Systems Characterized by Frequency-Domain Data in a Modified Nodal Analysis Framework,” *IEEE Electrical Performance of Electronic Packaging*, 2006.

[17] S. N. Lalgudi, K. Srinivasan, G. Casinovi, R. Mandrekar, E. Engin, M. Swaminathan, and Y. Kretchmer, “Causal transient simulation of systems characterized by frequency-domain data in a modified nodal analysis framework,” in *Electrical Performance of Electronic Packaging, 2006 IEEE*, oct. 2006, pp. 123 –126.

[18] M. Kavehrad, J. Doherty, J.-H. Jeong, A. Roy, and G. Malhotra, “10 gbps transmission over standard category-5 copper cable,” in *Global Telecommunications Conference, 2003. GLOBECOM ’03. IEEE*, vol. 7, dec. 2003, pp. 4106 – 4110 vol.7.

[19] T. Iwata, H. Shibata, and T. Araki, “Extrapolation of band-limited frequency data using an iterative hilbert-transform method and its application for fourier-transform phase-modulation fluorometry,” *Measurement Science and Technology*, vol. 18, no. 1, pp. 288–294, 2007. [Online]. Available: http://stacks.iop.org/0957-0233/18/i=1/a=035

[20] M. Schnecker, “Clock Recovery Methods for Jiter Analysis,” Technical Brief, Lecroy, 2005.

[21] Wikipedia, “Plesiochronous,” April 2011, http://en.wikipedia.org/wiki/Plesiochronous.

[22] D. M. Pozar, *Microwave Engineering*, Second ed. Wiley, 1998.

[23] Agilent, “Agilent 83496B Clock Recovery Module With PLL Analysis,” www.agilent.com/find/83496B, Agilent.