Programmable Low Overhead, Run Length Limited and DC-Balanced Line Coding for High-Speed Serial Data Transmission

Julien Saade

UNIVERSITÉ GRENOBLE ALPES

THÈSE

Pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ GRENOBLE ALPES

Spécialité : **Mathématiques et informatique**

Arrêté ministériel : 7 août 2006

Présentée par

Julien Saadé

Thèse dirigée par **M. Frédéric Pétrot**

préparée au sein du **Laboratoire TIMA, CNRS/Grenoble INP/UJF (CIFRE STMicroelectronics)**
dans **l'École Doctorale Mathématiques, Sciences et Technologies de l'Information, Informatique (MSTII)**

Encodage de donnée programmable et à faible surcoût, limité en disparité et en nombre de bits identiques consécutifs

Thèse soutenue publiquement le **« 3 juin 2015 »**,
devant le jury composé de :

M. Bernard TOURANCHEAU
Professeur université Grenoble Alpes (Président)
M. Michel PAINDAVOINE
Professeur université de Bourgogne (Rapporteur)
M. Olivier SENTIEYS
Professeur université Rennes 1 (Rapporteur)
M. Thierry DIVEL
Ingénieur Fogale Sensation Suisse (Membre)
M. Joel HULOUX
Ingénieur STMicroelecronics Grenoble (Membre)
M. Frédéric PETROT
Professeur Institut Polytechnique Grenoble (Membre)

Programmable Low Overhead, Run Length Limited and DC-Balanced Line Coding for High-Speed Serial Data Transmission

By

Julien Saadé

Supervised by Prof. Frédéric Pétrot

In collaboration with

TIMA Laboratory, Université Grenoble-Alpes

and

STMicroelectronics

A thesis submitted for the degree of

Docteur de l'université Grenoble Alpes

May 2015

Abstract

Thanks to their routing simplicity, noise, EMI (Electro-Magnetic Interferences), area and power consumption reduction advantages over parallel links, High Speed Serial Links (HSSLs) are found in almost all today's System-on-Chip (SoC) connecting different components: the main chip to its Inputs/Outputs (I/Os), the main chip to a companion chip, Inter-Processor Communication (IPC) and etc… Serial memory might even be the successor of current DDR memories.

However, going from parallel links to high-speed serial links presents many challenges; HSSLs must run at higher speeds reaching many gigabits per second to maintain the same end-to-end throughput as parallel links as well as satisfying the exponential increase in the demand for throughput. The signal's attenuation over copper increases with the frequency, requiring more equalizers and filtering techniques, thereby increasing the design complexity and the power consumption.

One way to optimize the design at high speeds is to embed the clock within the data, because a clock line means more routing surface, and it also can be source to high EMI. Another good reason to use an embedded clock is that the skew (time mismatch between the clock and the data lanes) becomes hard to control at high frequencies. Transitions must then be ensured inside the data that is sent on the line, for the receiver to be able to synchronize and recover the data correctly. In other words, the number of Consecutive Identical Bits (CIBs) also called the Run Length (RL) must be reduced or bounded to a certain limit.

Another challenge and characteristic that must be bounded or reduced in the data to send on a HSSL is the difference between the number of '0' bits and '1' bits. It is called the Running Disparity (RD). Big differences between 1's and

0’s could shift the signal from the reference line. This phenomenon is known as Base-Line Wander (BLW) that could increase the BER (Bit Error Rate) and require filtering or equalizing techniques to be corrected at the receiver, increasing its complexity and power consumption.

In order to ensure a bounded Run Length and Running Disparity, the data to be transmitted is generally encoded. The encoding procedure is also called line coding. Over time, many encoding methods were presented and used in the standards; some present very good characteristics but at the cost of high additional bits, also called bandwidth overhead, others have low or no overhead but do not ensure the same RL and RD bounds, thus requiring more analog design complexity and increasing the power consumption.

In this thesis, we propose a novel programmable line coding that can perform to the desired RL and RD bounds with a very low overhead, down to 10 times lower that the existing used encodings and for the same bounds. First, we show how we can obtain a very low overhead RL limited line coding, and second we propose a very low overhead method which bounds the RD, and then we show how we can combine both techniques in order to build a low overhead, Run Length Limited, and Running Disparity bounded Line Coding.

Dedication

To my Mother and Father

For everything.

Acknowledgments

I would like to warmly thank my four thesis supervisors, in scrambled order, André Picco, head of the High-Speed Links team at STMicroelectronics, and Frédéric Pétrot, SLS team leader at TIMA laboratory, for welcoming me, directing my thesis and always giving me advices that helped me stay on the right path. I thank Joel Huloux, MIPI Alliance's chairman, for taking the time to advise me and giving me the opportunity to participate to MIPI's PHY and LLI Working Groups, assist and contribute to discussions with experts from the leading semiconductor companies all over the world, from whom I've learned and gained a lot of experience. I want to thank Abdelaziz Goulahsen, MIPI's LLI Working Group chairman, for all his help and expertise, the many technical meetings we had and all the things I learned from him. I consider myself lucky being supervised by such experienced people.

I also must thank Erwan Le-Saint for offering me this opportunity and welcoming me in his team.

I thank my thesis committee members, Bernard Tourancheau, Olivier Sentieys, Michel Paindavoine and Thierry Divel for accepting to review and comment my thesis.

Many thanks to Steve Kwiatkowski, Jérôme Deroo, Mohamed Daoudi, Klodjan Bidaj and Gilles Ries, experts and engineers at STMicroelectronics, for the many fruitful technical discussions we had and the help they have provided me.

Last but not least, I thank my parents and my brothers for all the love and support they have provided me throughout my thesis, and my entire life. This thesis, for all its worth, is dedicated to them.

TABLE OF CONTENTS

Contributions

Patents:

"Serial Transmission Having a low level EMI", 2013
Inventors: **J. Saadé**, A. Goulahsen

"Polarity-Bit data Encoding Method using Aperiodic Frames", 2014
Inventors: **J. Saadé**, A. Goulahsen

Conference papers and oral presentations:

"A System-Level Overview and Comparison of Three High-Speed Serial Links: USB 3.0, PCI Express 2.0 and LLI 1.0", IEEE 16th Symposium on Design and Diagnostic of Electronic Circuits and Systems (DDECS 2013) – Karlovy Vary, Czech Republic
Authors: **J. Saadé**, F. Pétrot, A. Picco, J. Huloux, A. Goulahsen

"A Scalable Low Overhead Line Coding for Asynchronous High Speed Serial Transmission", IEEE 18th Workshop on Signal and Power Integrity (SPI 2014) – Gent, Belgium
Authors: **J. Saadé**, A. Goulahsen, A. Picco, J. Huloux, F. Pétrot

"Low Overhead, DC-Balanced and Run Length Limited Line Coding", IEEE 19th Workshop on Signal and Power Integrity (SPI 2015) – Berlin, Germany
Authors: **J. Saadé**, A. Goulahsen, A. Picco, J. Huloux, F. Pétrot

Other Participations:

"Latest Version of Interface Protocol Speeds Mobile Device Development, Lowers e-BoM. MIPI Alliance's Low Latency Interface Working Group Delivers LLI v2.1" Published in Design & Reuse Magazine
Authors: A. Goulahsen (STMicroelectronics) and V. Leonov (Intel)
Thanked contributors: B. Balakrishnan (Ericsson), U. Leucht-Roth (Intel) and **J. Saadé** (STMicroelectronics)

List of acronyms

HSSL	High Speed Serial Link
SoC	System on Chip
I/O	Input/Output
IPC	Inter Processor Communication
DDR	Double Data Rate SDRAM
EMI	Electro-Magnetic Interferences
CIB	Consecutive Identical Bits
RL	Run Length
RD	Running Disparity
BLW	Base-Line Wander
BER	Bit Error Rate
AP	Application Processor
RFIC	Radio Frequency Integrated Circuit
NRZ	Non-Return to Zero
Mbps	Megabits per second
Gbps	Gigabits per second
MLT-3	Multi-Level 3
PAM	Pulse Amplitude Modulation
MIPI	Mobile Industry Processor Interface
LLI	Low Latency Interface

UniPro	Unified Protocol
DigRF	Digital RF
RF	Radio Frequency
UFS	Universal Flash Storage
CSI	Camera Serial Interface
DSI	Display Serial Interface
PCIe	Peripheral Component Interconnect express
M-PCIe	Mobile-PCIe
SSIC	SuperSpeed Inter-Chip
ISO	International Standards Organization
OSI	Open Systems Interconnection
PHY	Physical Layer
PLL	Phase Locked Loop
Tx	Transmitter
Rx	Receiver
Dp	Differential positive
Dn	Differential Negative
CDR	Clock and Data Recovery
CRC	Cyclic Redundancy Check
e-BoM	electronic Bill of Materials
LFSR	Linear Feedback Shift Register
PRBS	Pseudo-Random Binary Sequence

List of Figures

Chapter 2 :

Chapter 3:

Chapter 4:

Chapter 5:

Chapter 6:

Chapter 7:

Chapter 8:

List of Tables

Chapter 3:

Chapter 5:

Chapter 6:

Chapter 7:

Chapter 8:

Chapter

1 Introduction

Smartphones and tablets have emerged in the last decade as an essential part of our lives. The number of applications handled is increasing and the quality of service provided to the user is still improving, resulting in more and more on-board hardware components, design complexity and bandwidth increase. One of the main challenges is then the power consumption, especially when focusing on a mobile device and its battery life, in addition to the worldwide environmental impact of the power consumption when expecting 4 billion smartphones and tablets by 2017 [1].

Essential elements that directly affects the performance of mobile devices are High Speed Serial Links (HSSLs). HSSLs connect the different components of a mobile device; the Application Processor (AP) to the modem or a companion chip, the AP to the camera or the display, the AP to the mass storage device, the RFIC (Radio Frequency Integrated Circuit) to the modem and etc… HSSLs are also used in laptops and computers as well as in networking. This results in a variety of HSSLs because each application have different requirements, and different protocols are designed to fulfill their needs.

In this thesis, a system-level overview on high-speed serial links is made, with special focus on three protocols: the Universal Serial Bus (USB), the Peripheral Component Interconnect express (PCIe) and the Low Latency Interface (LLI). We will make a comparison between the different parameters and justify their field of use.

With the increasing demand for bandwidth, the speed of HSSLs is doubling every two to three years presenting many challenges to the designers in terms of complexity and power consumption. The design must then be optimized as much as possible.

One of the parameters that directly affects the bandwidth and the performance of a HSSL is the line coding. In many, if not most of the HSSLs, the data to transmit on the link is encoded to ensure two main characteristics: a bounded Run Length (RL), which means that a certain number of consecutive identical bits must not be exceeded so the data contains enough transitions. The receiver benefits from the transitions to synchronize and recover the clock and the data correctly. The second characteristic that the encoding must bound is the Running Disparity (RD), which means that the difference between the numbers of transmitted 0's and 1's must not exceed a specific limit to reduce the BaseLine Wander (BLW) which is the signal shifting from the zero reference. The BLW closes the eye diagram (which is the superposition of all the bits of a signal) and might create sampling errors when recovering the data.

For those reasons, the line coding intervenes to present solutions. However, Line coding comes at the cost of added bits also called overhead, affecting the throughput. Over time, many encodings have been used in the standards, some present very good characteristics but at the cost of high overhead, reducing the bandwidth efficiency of the link. Other encodings have low overhead but do not ensure the same bounds for RL and RD and require analog components such as filters and equalizers to compensate. This means more design complexity and power consumption.

In this thesis, an overview on the existing methods which bound the RL and the RD is made. We will highlight their advantages and their drawbacks. Then we will present an optimized low overhead method that bounds the Run Length. Another main contribution of this thesis is a low overhead method that bounds

the Running Disparity with an overhead down to 10 times lower than the existing methods, and for the same bounds. After presenting both methods separately, we will show how we can combine them to build a low overhead, run length limited and running disparity bounded line coding.

In addition to its low overhead characteristic, other advantages of the line coding proposed in this thesis will be highlighted such as providing interoperability between links with different RL and RD requirements as well as early errors detection.

Thesis Organization

The remainder of this thesis is organized as follows:

Chapter 2, "Problem Statement", explains in details today's High Speed Serial Links challenges. We will focus on the line coding's effect on the performance of HSSLs and the need for a new line coding.

Chapter 3, "State of the art", is divided into two main sections; the first one presents the state of the art of HSSLs focusing on three of today's HSSLs' protocols. The second section presents the state of the art of the encodings that were proposed and used in HSSLs, we will name their advantages and drawbacks and show the overhead-performance tradeoff.

In Chapter 4, "Low EMI encoding method", we present a line coding that ensures reduced EMI that could be caused by the data.

In Chapter 5, "Low overhead run length limited encoding Method", we will present an overhead-optimized line coding to limit the Run Length and evaluate its advantages over existing equivalent methods.

In Chapter 6, "Low Overhead DC-Balanced encoding method", we will present an overhead-optimized line coding, but this time to bound the Running

Disparity. A comparison will also be made with the existing equivalent methods.

Chapter 7, “DC-Balanced and run length limited line coding” presents a method to combine both encoding methods presented in chapters 5 and 6, to build a low overhead, RL limited and RD limited Line Coding.

In Chapter 8, “Experimental results”, we present the overhead results of the proposed line coding based on simulation, we show the resulting eye diagrams, the VHDL model and the gate count estimation, we compare those results with other encodings and highlight the advantages of our proposal.

In Chapter 9 we conclude and summarize the work presented in this thesis.

Chapter

2 Problem Statement

2.1 *Chapter's* Introduction

With the increase demand for throughput, High Speed Serial Links are now facing important challenges to transmit the data over a channel. In less than 15 years, the frequency has drastically increased from 500 Mbps (Megabits per second) to 16 Gbps (Gigabits per second) as we can see in figure 2.1 and copper-based channels are still used in most HSSL as transmit medium because of their many advantages in terms of area and cost over optical links.

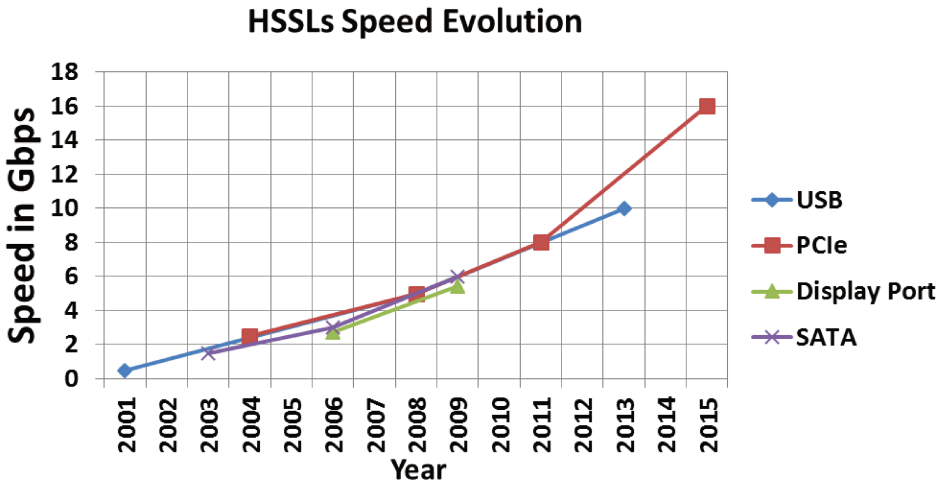


Figure 2.1 Some High Speed Serial Links speed evolution

Another characteristic that is still used and scheduled for the future generations of HSSL is NRZ (Non-Return to Zero) signaling [2], which means that the binary information is sent with two voltage levels, i.e. V1 to indicate a 1 bit and V0 to indicate a 0 bit. NRZ has gained success because of its coding/decoding simplicity and its low voltage swing requirement that enables lower power consumption when compared to multi-level signaling. When power consumption is not the main constraint, multi-level signaling could be used such as MLT-3 (Multi-Level Transmit 3), PAM-4 (Pulse Amplitude Modulation 4), PAM 8, PAM 16 and etc… enabling better throughput at the same frequency, e.g. for PAM-4, 4 voltage levels are assigned for 00b, 01b, 10b and 11b. Thereby, 2 bits are sent in each clock period instead of 1 bit. Multi-level signaling is used in protocols such as Ethernet, where power consumption is not the main constraint. In this thesis, we will mainly focus on copper channels and NRZ signaling.

This chapter's purpose is to explain the functioning of HSSLs, the layering model and the role of each layer. We will then focus on the line coding and its effect on NRZ signaling over copper, and on the bandwidth efficiency as well.

2.2 High Speed Serial Links

2.2.1 *High Speed Serial Links' variety*

High Speed Serial Links (HSSLs) are found in almost all data communication processes from small channels lengths of few centimeters to long distances of tens of meters as we can see in Figure 2.2. From here, several HSSLs' protocols have been designed to fulfill the needs of each situation.

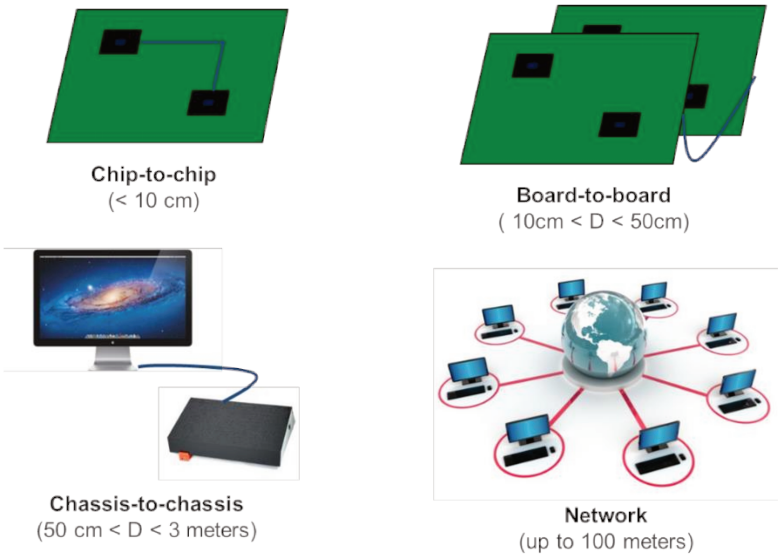


Figure 2.2 HSSLs different domains of application

In addition to the difference in the applications' distances, differences are also found in the applications' type; e.g. a display link have different requirements than a mass storage link or a modem link and etc… resulting in a variety of protocols. More details will be provided in chapter 3.

To reduce the variety and complexity of HSSLs, mobile components manufacturing companies joined their forces in the MIPI® Alliance (Mobile Industry Processor Interface). Founded in 2003 by ARM, Intel, Nokia, Samsung, STMicroelectronics and Texas Instruments, the MIPI alliance's aim is to define and standardize interfaces for connecting the different components

of a mobile device as we can see in Figure 2.3 and now joins more than 280 companies.

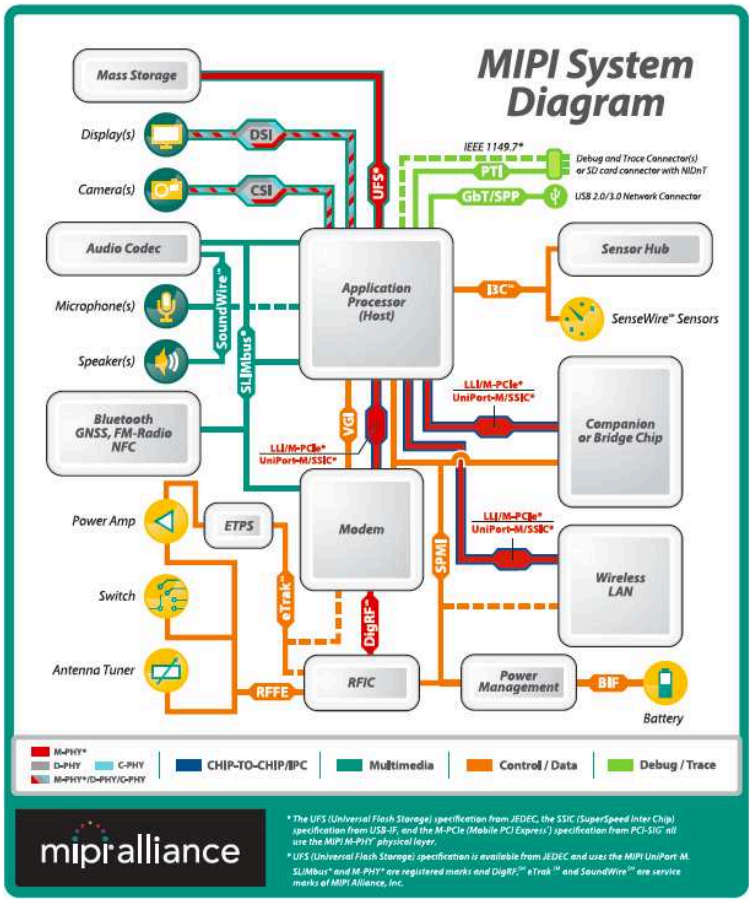


Figure 2.3 MIPI® System Diagram for mobile devices [3]

In Figure 2.3, we can find the different HSSLs connecting the components: the LLI (Low Latency Interface), the UniPro (or UniPort, Unified Protocol), the DigRF (Digital RF), CSI (Camera Serial Interface), DSI (Display Serial Interface), M-PCIe (Mobile Peripheral Component Interconnect express, also called low power PCIe), and SSIC (SuperSpeed Inter-Chip, or the low power USB 3.0). Those protocols sometimes use different physical layers.

2.2.2 *HSSLs' layering model*

High Speed Serial Links' layering model follows the Open Systems Interconnection (OSI) reference model provided by the International Standards Organization (ISO) which is essentially constituted of 7 layers. Not every system uses all the layers, the functions and protocols which support the forwarding of data are then provided in the lower layers. HSSLs generally define the three to four lowest layers (See figure 2.4).

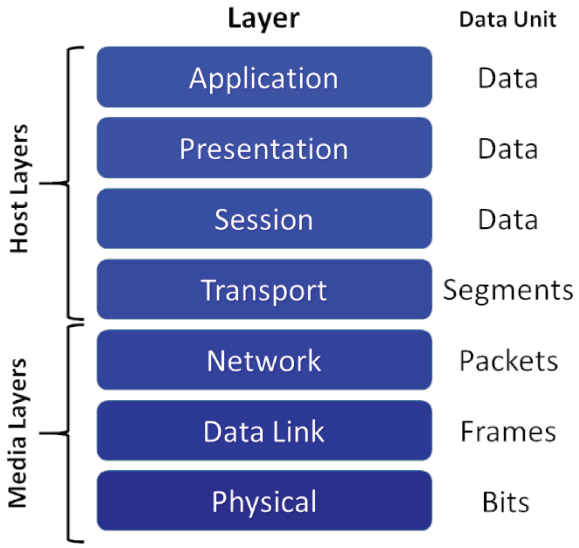


Figure 2.4 Open Systems Interconnection (OSI) Layers

The seven OSI layers' functioning can be summarized as follows [4]:

7. **Application:** The application layer serves as the window for users and application processes to access network services. This layer contains a variety of commonly needed functions.

6. **Presentation:** formats the data to be presented to the application layer. It can be viewed as the translator for the network. This layer might also provide compression and encryption such as password encryption.

5. **Session:** allows session establishment, maintenance and termination: allows two application processes on different machines to establish, use and terminate a connection.

4. **Transport:** provides end to end communication control, splits the message into smaller units (if not already small enough), and passes the smaller units down to the network layer. This layer can also provide message acknowledgment, traffic control and session multiplexing when there's many.

3. **Network:** controls the operation of the subnet, deciding which physical path the data should take based on network conditions, priority of service, and other factors.

2. **Data Link:** provides error-free transfer of data frames from one node to another over the physical layer by errors checking and sometimes correction. This layer also provides link establishment and termination, frame traffic control, sequencing, acknowledgement, and delimiting.

1. **Physical:** describes the electrical/optical, mechanical, and functional interfaces to the physical medium, and carries the signals for all of the higher layers. This layer provides data encoding and physical medium attachment.

HSSL's role in a system is then to route the different components and provide reliable data transmission and reception at the desired speed over the channel.

2.2.3 Focusing on the physical layer

In this paragraph, we will focus on the lowest layer of HSSLs. In figure 2.5 we can see a simplified schematic of the Physical Layer (PHY).

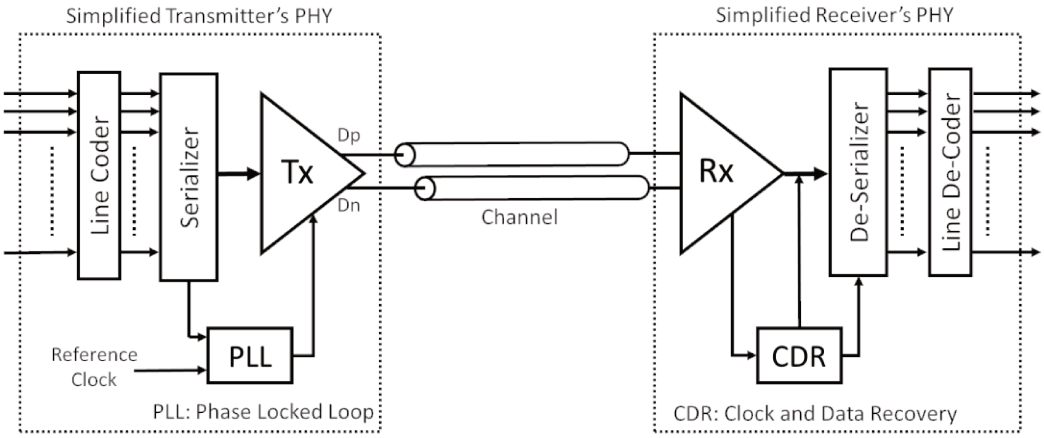


Figure 2.5 Simplified block diagram of HSSLs Physical Layer

The essential elements that constitutes the PHY of a HSSL are represented in the figure above; the data coming from the higher layers comes in parallel (inside the HSSL IP) at the PHY layer to be transmitted in series to the peer destination. The PHY has to reliably, serially transmit the data. The digital part of the PHY is the Line Coder; the line coder encodes the data before transmission and plays a major role in ensuring the reliability of the transmission. This will be detailed in the next paragraph.

After encoding, the data goes to the serializer that uses a PLL (Phase Locked Loop) to multiply the reference parallel clock up to the serial frequency. The data goes then to the Tx (Transmitter) Driver that sends the Dp/Dn (Differential positive/negative) signals over the channel. Differential signaling is used to reduce noise [5] and radiated emissions.

We note that the described mechanism above is a simplified mechanism and other components could exist, e.g. a pre-driver might exist before the driver to do the pre-emphasis, which is a mechanism that amplifies high frequencies as a prevention from the channel's attenuation.

At the receiver, the Rx driver receives the serial data and the CDR unit (Clock and Data Recovery) recovers the clock and the data. The serial data is

then made parallel by the de-serializer, de-encoded, and then forwarded to the upper layer.

2.3 Line *Coding's effect on data transmission*

2.3.1 Introduction

The most important measures to evaluate the performance of HSSLs are the BER (Bit Error Rate) and the eye diagram, which is the plot of the superposition of all the bits during transmission as we can see in figure 2.6. The eye diagram is judged by its vertical and horizontal opening. The protocol specification defines the minimum opening required at the receiver. The transmission should respect the specification so the system could ensure the defined BER.

Timing Jitter and the Signal-to-Noise Ratio (SNR) are two of the factors that affect the BER and the eye diagram's opening. Data encoding has a direct impact on both and in this section we're going to see how. Transmitted data can also contribute to increase Electro-Magnetic Interferences (EMI), causing errors in neighboring lanes or even neighboring devices. We will start by explaining how the data can increase EMI, and then we'll show the impact of the RL and RD of the data on the transmission.

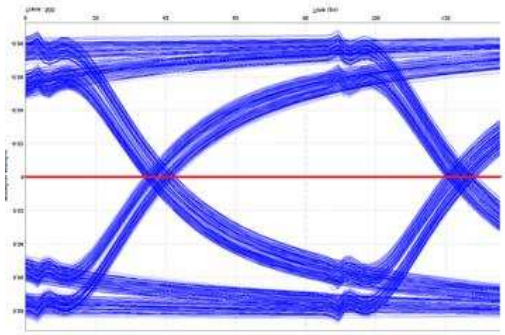


Figure 2.6 Eye diagram example

2.3.2 Data*'s impact on EMI*

According to Maxwell, whenever electrons are moved, magnetic fields will be generated. Those fields could interfere with Radio Frequency (RF) components sending radio signals nearby. This is what we call Electro-Magnetic Interferences (EMI). Radiations could also aggress neighboring lanes of the same link creating errors. With the ascending data rates in electronics, care must be taken to mitigate EMI.

One of the reasons why HSSLs use differential signaling is EMI reduction. In a perfect world, the differentials create opposite fields that will cancel each other resulting in EMI suppression. But imbalance between differentials is inevitable and results in common mode voltage (V_{cm}) that will be source to EMI [6]. The common mode voltage is represented in figure 2.7.

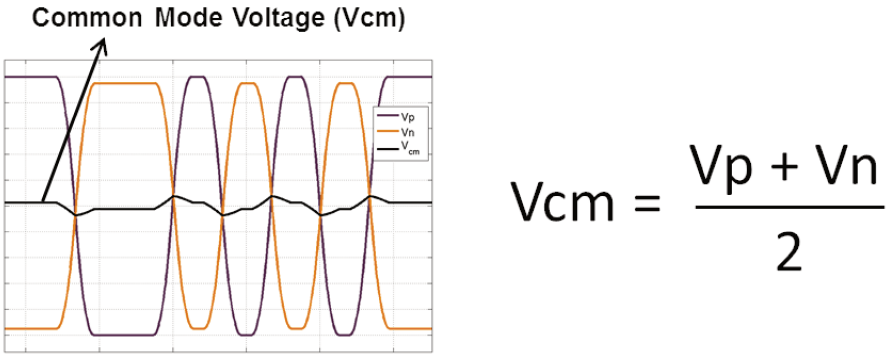


Figure 2.7 Common mode voltage representation (voltage mismatch = 5%, time mismatch = 5%)

EMI are judged then on the Power Spectral Density of the common mode voltage [7] resulting from sending the data on the link. Repetitive data (e.g. repetitive bytes) can generate EMI [8] and will create peaks in the spectrum of the V_{cm} as we can see in figure 2.8. In this example, the peaks are due to repeated pixels data. To mitigate EMI, engineers work on reducing the peaks and spreading the spectrum of the V_{cm}. Many techniques exist to reduce EMI [6 – 9] and line coding is one of them. Line coding helps in reducing the redundancy

in the data and spreading the spectrum of the V_{cm} as we are going to see in chapter 3.

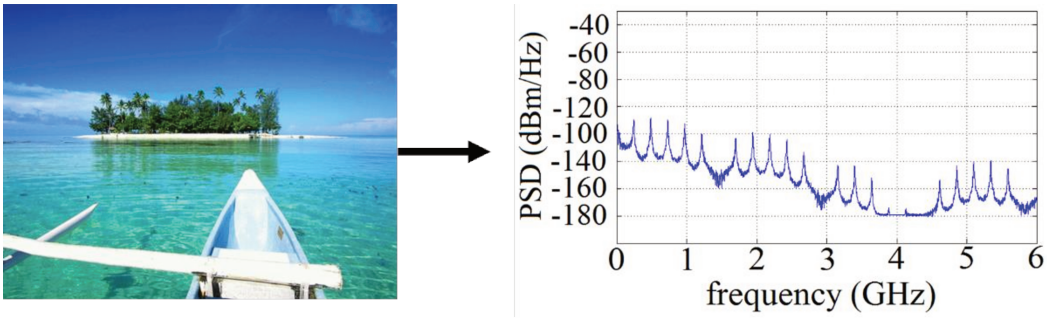


Figure 2.8 Power Spectral Density example of the Vcm of raw picture data at 1.4 GHz

2.3.3 *Data's* Run Length impact

The Run Length (RL) is a major factor in Clock and Data Recovery (CDR) unit at the receiver's side (see figure 2.5). A simplified schematic of the CDR unit is shown in figure 2.9 as follows:

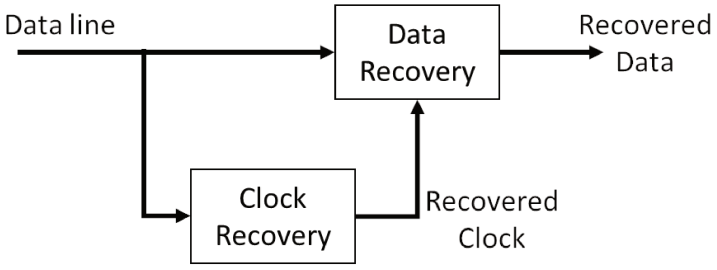


Figure 2. 9 Clock and Data Recovery simplified schematic

The clock recovery part of the CDR detects the transitions on the data line and recovers the clock that is synchronized with the received data. The recovered clock is then used to sample the data at the right moment and deduce the bits value.

Data transitions have a direct impact on the clock recovery part. Most of clock recovery in HSSLs are Phase Locked Loop based (PLL) because of many

advantages that are out of scope of this thesis. A simplified schematic of the PLL-based Clock Recovery is shown in figure 2.10.

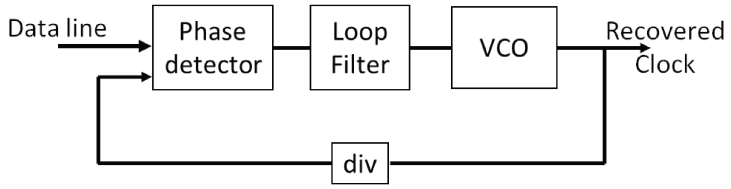


Figure 2.10 PLL-based Clock Recovery simplified schematic

The PLL acts like a low pass filter to its input's jitter, and as a high pass filter to the VCO's (Voltage Controlled Oscillator) jitter. The PLL is characterized by its cutoff frequency which could be written according to [10] as follows:

$$f_{-3dB} \cong \frac{K_P \cdot (TD \cdot K_{PD}) \cdot K_{VCO}}{2 \cdot \pi} \qquad (2.1)$$

where TD is the transitions density and equals 1/(Run Length)

and K_p, K_{PD}, and K_{VCO} are coefficients that are related to the loop filter, the phase detector and the VCO, that are out of scope of this thesis.

As we can see from equation (2.1), the transition density affects in first place the phase detector's output, making the PLL's bandwidth RL-dependent and not predictable. While for many reasons the PLL's bandwidth should be carefully chosen [11], this RL dependency could have many consequences. The increase in RL (decrease in TD) also means PLL bandwidth decrease which results in jitter peaking [12] and degradation of the receiver jitter tolerance characteristic (increase system Bit Error Rate).

Designers have addressed the problem of pattern dependency of the phase detector [12] [13] [14] but have done so with a significant increase, about twice, in hardware [10].

While PLL-based CDR architectures already suffer from a relatively large total layout area [15], adding more complexity might not be a good choice.

2.3.4 *Data's* Running Disparity impact

The Running Disparity (RD) also called the Cumulated Bits Difference (CBD) is known as the difference between the 1's and 0's in the data to send over the link. The RD is counted by adding +1 for a '1' bit and -1 for a '0' bit. Figure 2.11 shows a simple example.

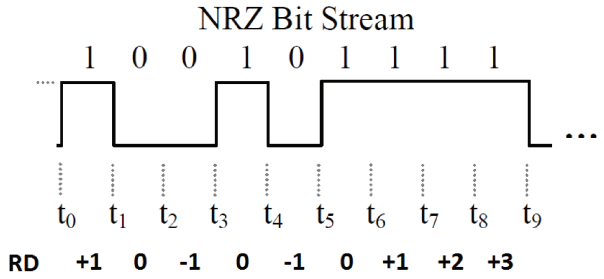


Figure 2.11 Running disparity calculation example for NRZ signaling

Since the early days of data communication, data coding methods have been introduced to limit the RD and bound it to reduce the baseline wander initially due to AC-Coupling at the receiver. AC-coupling is generally used when interconnecting drivers with different voltage thresholds [16]. This results, after a transition period, in differential signaling centered to Zero volts at the receiver whatever the Dp and Dn voltages levels sent by the transmitter. This is illustrated in figure 2.12. AC-coupling is then an advantage when using cables and connectors to connect different devices, allowing interoperability between constructors using different technologies.

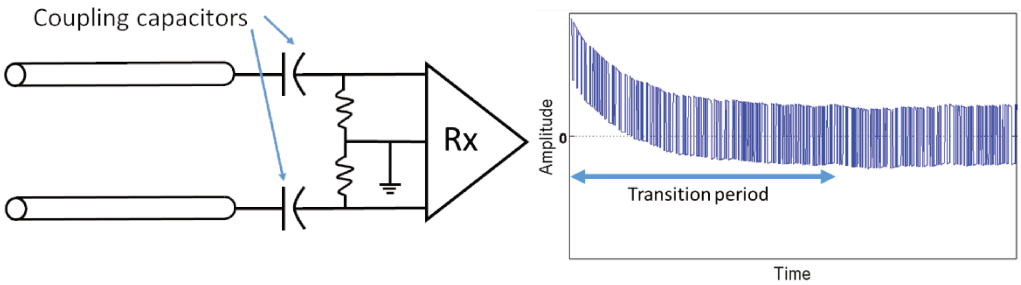


Figure 2.12 AC-coupling and transition period

However, AC-coupling has a big drawback; after the transition period for the signal to stabilize, the capacitive effect can make the signal shift up and down (charging and discharging the coupling capacitor) creating Baseline Wander, closing the eye diagram and degrading the SNR. This could be explained differently; the coupling capacitor forms with the termination resistor a high-pass RC filter that attenuates low frequency components formed by runs of consecutive bits, but more precisely by the difference between 1's and 0's, which is the running disparity. This is why one of the main interests of a line coding is to reduce or bound the RD.

Because it is a capacitance charge/discharge phenomenon, BLW due to the coupling capacitor can be estimated. For the sake of simplicity, we consider a single ended receiver (Dp or Dn). The simplified schematic is shown in figure 2.13.

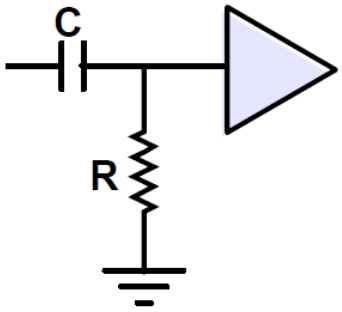


Figure 2.13 Simplified AC-coupling

The BLW also creates timing Jitter as we can see in figure 2.14. This type of Jitter is part of the Pattern Dependent Jitter (PDJ) (also called Data Dependent Jitter (DDJ) or Inter-Symbol Interference (ISI)) and from [17] and [18] we can calculate both the BLW and the PDJ.

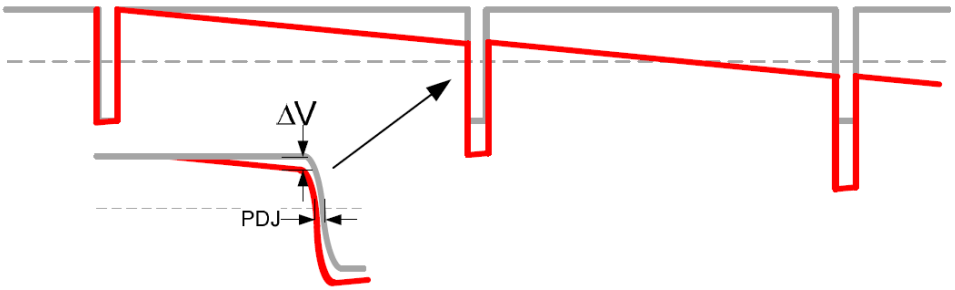


Figure 2.14 Baseline Wander and jitter introduced by the high pass filter [17]

In figure 2.14, ΔV represents the BLW, and PDJ is the Jitter and they can be calculated according to the following equations:

$$BLW = 0.5*V_{pp}(1-e^{-t/RC}) \quad (2.2)$$

$$PDJ = \frac{BLW}{slope} \quad (2.3) \qquad \text{with} \quad slope = Vpp\,\frac{0.6}{Tr} \quad (2.4)$$

Where t is the discharge time

V_{pp} is the peak-to-peak voltage (voltage swing)

R is twice R_t (considering the driver's resistor)

C is the coupling capacitor

and Tr is the rise time (20% to 80% of the signal)

The discharge time of the capacitor is represented by the signal being at the same level for a certain moment, this means consecutive identical 1's. But when the signal goes to 0, this will recharge the capacitor for a certain duration. The charge or discharge time will then be represented by the difference between number of 1's and 0's which is the Running Disparity times the bit duration. The BLW can thus be written as follows:

$$BLW = 0.5*V_{pp}(1-e^{-(RD*Tb)/(R*C)}) \quad (2.5)$$

where RD is the running Disparity

and Tb is the bit time or 1/frequency

Equation (2.3) shows that PDJ can be reduced by reducing the BLW. To reduce BLW, according to equation (2.5), we should increase the values of R and C. The resistor's value should be adapted to the driver and the channel, so its value cannot be simply manipulated. When it comes to the value of the capacitor, the best is to have an infinite value. But the more the capacitor's value gets bigger, the bigger is its surface and harder is the integration in the chip. On-chip capacitance per lane is limited to a few picoFarads (pF) at best in practical real estate of chip area [19]. Another consequence from increasing the capacitor's value is increasing the transition period, creating a high latency. R and C values are then forced by the system's obligations and their negotiation margin is tight. When there's no choice, filters and equalizers are used to counter the BLW's effect adding more complexity, area and power consumption. More details are provided in the next chapter.

Even when the transmitter and the receiver are DC-Coupled, BLW and PDJ exist, due to the channel and other factors, and are affected by the RD as we will observe later on. But it is more complex to get an estimation because it is channel-dependent and case-dependent.

2.4 *Chapter's* Conclusion

As seen in this chapter, the redundancy, Run Length and the Running Disparity of the data have an immediate impact on signal's integrity and system performance. For this reason, encodings have been designed to transform the raw data and limit or reduce the RL and the RD, but this comes at the cost of added bits called bandwidth overhead that sometimes reaches up to 25% of the initial size of the data, reducing the throughput. With the increasing demand for throughput, every bit sent on the link counts. Line coding is then a big challenge;

so is it possible to design a line coding that can bound the RD and the RL to low values with a **low overhead**?

High Speed Links are also applied on a wide range on data communication as we saw earlier in this chapter and a big variety exists. The bounds to the RL and RD requested by the link could be variable and case-dependent. Is it then possible to design a **programmable** low overhead line coding that performs to the desired Run length and Running Disparity?

Chapter

3 State of the Art

3.1 *Chapter's* Introduction

In the previous chapter we saw that a variety of high speed serial links exists to satisfy different types of applications, and then we saw the impact of the non-coded data on a HSSL.

This chapter is divided into two main parts: in the first part we will make a system-level comparison between three HSSLs that are used for three different

kinds of application: the Universal Serial Bus (USB), the Peripheral Component Interconnect express (PCIe) and the Low Latency Interface (LLI). We analyze their different parameters, we show the relation between these parameters and how improving one parameter could result in a degradation of another. Based on this analysis, our conclusion outlines the reason why USB is used for I/Os, PCIe is used for data hungry devices and LLI for memory sharing.

In the second part of this chapter, we overview most of the existing line coding methods that were designed for NRZ signaling. We compare them and show the advantages and the drawbacks of each, then highlight the overhead/performance tradeoff.

3.2 System-level comparison of three HSSLs: LLI, PCIe and USB

3.2.1 The Low Latency Interface (LLI)

One additional challenge in mobile phones industry is to reduce the electronic Bill of Materials (e-BoM). With today's phone peripherals becoming more and more complex, as most of them are having their own CPU-DDR subsystem, reducing BoM is not a simple task. That's why the Mobile Industry Processor Interface (MIPI®) Alliance developed the LLI 1.0 (Low Latency Interface 1.0) [20] [21] which is a serial interface that enables peripherals, like modems for example, to share the system's main DDR located on the application processor's side, which enables mobile phones manufacturers to remove the modem's DDR and reduce the total phone's cost. LLI 2.0 version extended the use of LLI and made it a general chip-to-chip interconnect. LLI is also used for Inter-Processor Communication (IPC).

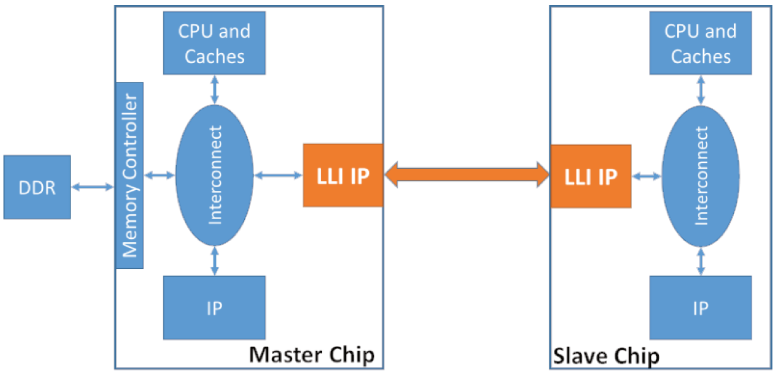


Figure 3.1 Example of an LLI environment (not exhaustive)

As we can see in Figure 3.1, LLI can be considered as a tunnel between two chips and it is memory mapped from both sides. The DDR on the main chip can be then considered as it was on the slave chip. LLI uses the M-PHY^{SM} from MIPI as physical layer. The LLI benefits from M-PHY's three high-speed gears: G1 at 1.4 Gbps, G2 at 2.9 Gbps and G3 at 5.8 Gbps. Gear 4 is expected to be released soon and will run at 11.6 Gbps. Seven low speed gears are featured as well.

3.2.2 The Peripheral Component Interconnect express (PCIe)

PCIe [22] [23] is a serial interconnect developed as a replacement to the parallel PCI bus. PCIe 1.0 runs at 2.5 Gbps, PCIe 2.0 at 5 Gbps and PCIe 3.0 runs at 8 Gbps. As shown in figure 3.2, the PCI Express is a central element of a modern computer and connects many of its components.

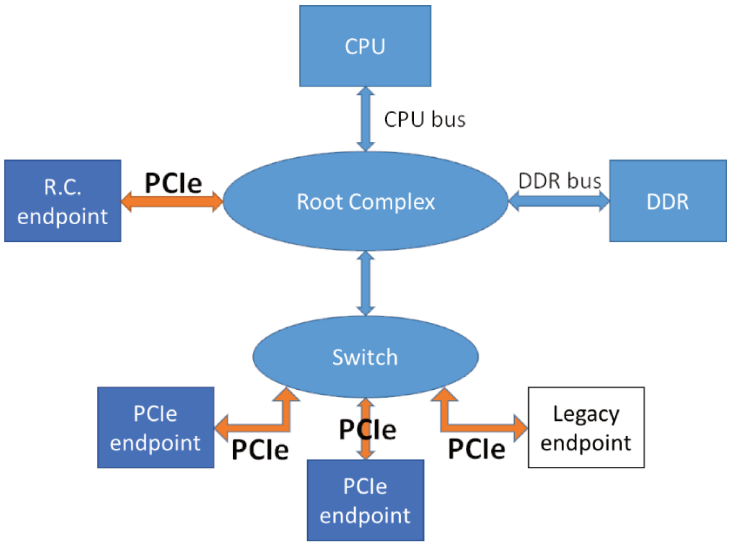


Figure 3.2 Example of PCIe link environment

In figure 3.2 we can see the following components:

- The 'Root Complex' (R.C., North Bridge) primary usage is to connect the CPU and the main DDR, and is also used for high throughput devices that need to access the DDR.
- The 'Switch' (South Bridge in PCI) is used to connect I/Os on the system.
- The R.C. Endpoints e.g. external Graphics Card, are directly routed to the R.C. because they need maximum performance and the root complex allows the Graphic Card to share the system's main DDR.
- PCIe Endpoints are PCIe native devices. They are routed to the switch.
- Legacy Endpoints: e.g. USB or SATA controllers, they are not PCIe devices but their controllers communicate with the switch using the PCIe protocol.

PCI express's instructions are Memory Mapped.

PCIe features Hot Plug.

3.2.3 The Universal Serial Bus (USB)

The Universal Serial Bus (USB) [24] is a high-speed serial interface developed to connect I/O peripherals to a computer. USB3.0 runs at 5 Gbps. Figure 3.3 shows the USB structure. The host is the intelligent element, it initiates all the transfers, and so, unlike PCIe and LLI, the USB devices cannot initiate a transaction (but they can ask the host to reschedule the transaction when they are not ready and they ask the host to reinitiate when they are ready). The Hub allows the extension option; it acts like a device in the Host's point of view.

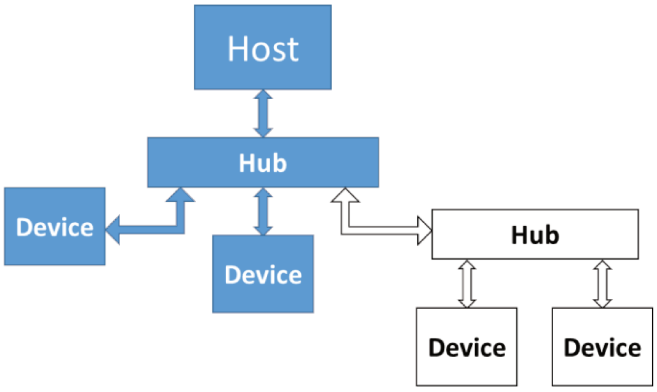


Figure 3.3 USB Structure

As mentioned in the PCIe's section above, the USB's Host is connected to the Southbridge in a system which might be the PCIe Switch.

USB's instructions are not Memory Mapped; The CPU does not have direct access to devices, the Host must have an integrated memory to temporary store devices' data and the CPU comes later on to retrieve data from the Host's memory.

USB features Hot Plug.

3.2.4 Layering model comparison

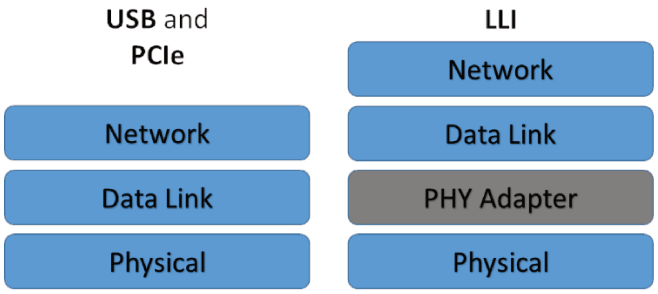


Figure 3.4 USB, PCIe and LLI Layering model comparison

The USB and PCIe layers are designed according to the OSI model. They both have 3 layers: the Transaction Layer (TL), the Data Link Layer (DLL) and the Physical Layer (PL). The TL is the upper layer; its duty is to generate header and data packets. The DLL adds CRC (Cyclic Redundancy Check) info to the header and data packets on the transmit side, checks the CRC on the receive side, generates flow control packets to the remote data link layer and manages the local flow control. The PL makes the coding/encoding and sends the data on the link.

In comparison to USB and PCIe layering model, the LLI has one additional layer: the PHY Adapter Layer (PA).

The LLI's transaction and physical layers does the same main duties such as in USB and PCIe. The DLL adds flow control on each data packet on the transmit side, and manages flow control on the receive side. The PA layer adds CRC and sequence number for each transmit packet and does the CRC and sequence number checking on the receive side.

3.2.5 Other parameters Comparison

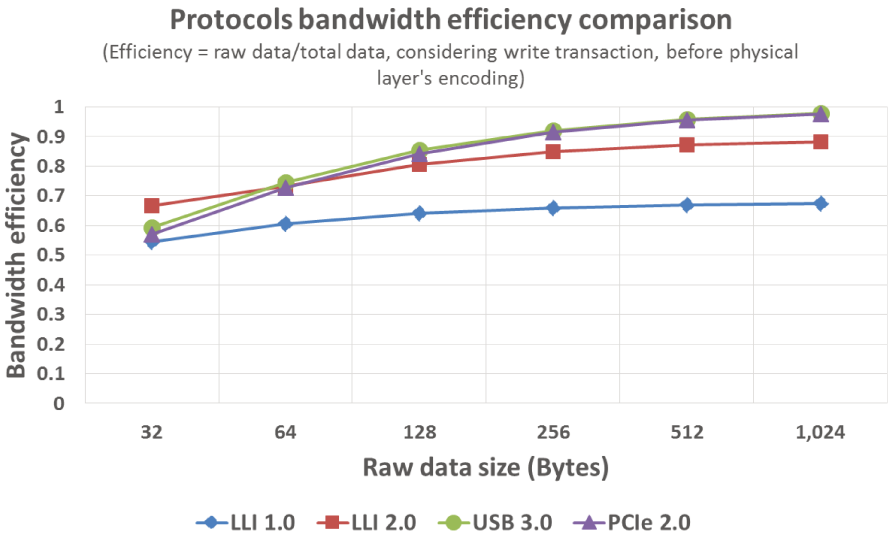


Figure 3.5 Throughput efficiency comparison for USB, PCIe and LLI for a write transaction and before line coding

LLI's packet partitioning was designed to provide very fast errors detection and retry mechanism, enabling the low latency feature. The Physical Layer of LLI ensures a BER of 10^{-10}. But with the 8-bits CRC for every 64 bits in the 1.0 version, LLI targets a BER of 10^{-20} to meet the requirements of a DDR and cache refill operations. This packets partitioning affects LLI's throughput efficiency as we can see in Figure 3.5. The maximum efficiency (raw data/total data) in LLI 1.0 is 0.67, LLI 2.0 introduced a new packet partitioning increasing the efficiency to 0.88. This comes at the cost of latency increase, but the LLI's 1.0 partitioning is still kept with higher priority level for latency-sensitive transactions.

USB and PCIe have the throughput as priority and the efficiency is about 0.98 ensuring high throughput beneficial to the transfer of huge amounts of data. However, they have a BER of 10^{-12} ensured by the Physical layer and their retry mechanism is slow compared to LLI.

More details about latency, throughput and others parameters comparison can be found in the overview we made in [25].

3.2.6 *Comparison's* Summary

Table 3.1 summarizes the overview.

Parameter	Protocols	Advantages	Consequences
Differential Swing = 800mV	USB PCIe	Long distances applications (cables)	High power consumption
Differential Swing = 400mV	LLI	Low power consumption	Short distances applications
Memory mapping	LLI PCIe	Direct access to data (memory sharing possibilities)	Occupying the CPU bus
No memory mapping	USB	Not occupying the CPU bus	No direct access to data (No memory sharing)
Multi-lane scalability	LLI PCIe	Multiplying throughput and decreasing latency	More power consumption and no external connectors possibility
No multi-lane scalability	USB	External connectors possibility	No throughput increasing possibility
Low latency error retry time	LLI	Cache refill operations possibility	Low data efficiency (throughput)
High latency error retry time	PCIe USB	High data efficiency (throughput)	No possibility for cache refill operations
Time Framing QoS	USB	All devices are served	High latency for interrupts
Priority based QoS	LLI PCIe	Low latency for interrupts and for high priority operations	Other devices or operations have to wait to be served

Table 3.1 Overview Table of some HSSLs

We conclude that USB with its intelligent software and hot plug feature allows easy Human Interface Device usage, and with its high throughput, it allows mass storage device usage. But with its high latency, high BER, and because USB is not memory mapped, it can allow neither memory sharing nor cache refill operations. PCIe with its intelligent NorthBrigde/ SouthBridge system design allows I/O connecting, and with its memory mapped instructions and its high throughput, even though it is latency-criticized [26], it allows data-

hungry devices (like graphics card) to share the system's main DDR when connected directly to the root complex and using up to 32 lanes to increase throughput and decrease latency. But using multi-lanes will increase power consumption which is an important issue in mobile applications.

To allow DDR chip-to-chip sharing and cache refill operations inside mobile phones, and in order to enable manufacturers to remove the modem's DDR and reduce the e-BoM, MIPI Alliance created the LLI featuring a low BER, low latency and low power consumption physical layer (the M-PHY), but at the cost of lower throughput efficiency.

3.3 *Line Coding's State of the Art*

3.3.1 Introduction

As mentioned in chapter 2, Line Coding is one of the biggest challenges in data transmission. That's why there is a big variety of coding methods that were proposed over time, and it is quite difficult to go through all of them.

As seen earlier in this chapter, HSSLs protocols add information to the data and decrease the efficiency before the PHY layer. Line coding must then be optimized as much as possible to not degrade the efficiency furthermore.

In this section, "line coding's state of the art", we will try to go through the most efficient line coding methods, and especially the ones implemented in HSSLs standards.

The next paragraphs will overview the following line coding methods: the Bit Stuffing, the 8b10b encoding, the Scrambling and the polarity-bit coding.

3.3.2 The Bit Stuffing (BS)

The Bit Stuffing is a method to limit the Run Length (RL). At the transmitter, the number of Consecutive Identical Bits/Digits (CIBs/CIDs) is counted. Whenever the RL reaches a specific value that we will call N, an inverted bit will be added to create a transition, even if the next data bit contains a transition.

On the other side, the receiver will benefit from this transition to recover the clock and the data without the risk of time sampling errors. And then the bit stuffing decoder will count the number of CIDs, and whenever it reaches N, the next bit will be removed. The data is then restored to its raw state.

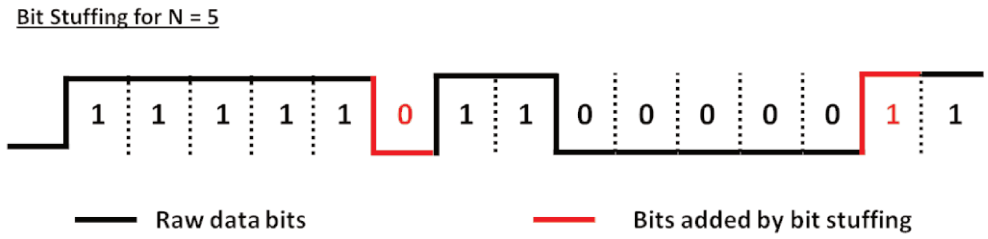


Figure 3.6 Bit Stuffing Example for Run length limitation of 5

Bit Stuffing's advantage is its simplicity but there is a major drawback. The number of added bits is data dependent. If the raw data contains enough transitions and does not exceed the limit fixed by the bit stuffing, the bit stuffing will not intervene and the overhead will be zero. The worst case is when the data is all ones or all zeroes, the bit stuffing's overhead is 1/N.

When a certain throughput is targeted, the frequency must be pushed to the worst case overhead to ensure the throughput. Let's consider the case of a link that should ensure 10 Gbits/s for video data transmission. This link is encoded with Bit Stuffing at N = 5. Even though the BS might add few percent overhead, the worst case should be considered which is 1/5 = 20%. The frequency at which the link should run to ensure 10 Gbits/s will be f_{link} = 10+10*20% = 12 GHz.

Bit Stuffing is used in protocols such as CAN (Controller Area Network) that uses the NRZ signaling and does the BS with N = 5. BS is also used by the USB 2.0 [27] that uses NRZI signaling and does the BS with N = 6 for consecutive 1's only, because a 0 already contains a transition in NRZI.

We note that Bit Stuffing does not help in reducing the EMI and in spreading the spectrum. Repetitive patterns will stay repetitive with bit stuffing. Bit stuffing also does not help in reducing the RD.

3.3.3 The 8b/10b encoding

The 8b/10b encoding [28] [29] was introduced back in 1983 and has gained success because of its excellent characteristics. 8b/10b encoding is made via 5b/6b and 3b/4b sub-block encoding for every byte to be transmitted. If we look at it in a different point of view, 8b/10b encoding transforms each data byte into a 10-bit symbol providing $2^{10} = 1024$ valid data words instead of $2^8 = 256$ valid data words necessary to transmit an 8-bit information. Only the "best" combinations out of 1024 are chosen to represent the data bytes, i.e. the ones ensuring a Run Length limited to 5, and a Running Disparity bounded to +/- 3. In addition, 8b/10b encoding provides control symbols from the remaining combinations. The rest will be non-valid combinations used for errors detection.

However, because of adding 2 bits to each byte, 8b/10b encoding has an overhead of 2/8 = 25%. With the increasing demand for bandwidth, 25% of overhead seems to be an important issue.

8b/10b encoding helps in reducing by a factor of 2 the repetition of some bytes, but not all of them. There is then a positive effect on EMI but this might not be enough.

3.3.4 Data Scrambling

Scrambling is a XOR (eXclusive OR, modulo-2 addition) operation between the raw data (the data to scramble) and the output pattern of an LFSR (Linear Feeback Shift Register). The output pattern of the LFSR is called PRBS (Pseudo-Random Binary Sequence). A simplified schematic of the scrambling operation is represented in Figure 3.7.

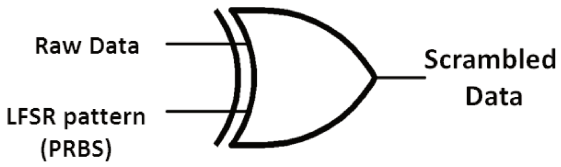


Figure 3. 7 Simplified representation of scrambling

The Linear Feedback Shift Register (LFSR):

An LFSR is a shift register that, when clocked, advances the signal through the register from one bit to the next most-significant bit. Some of the outputs are combined in eXclusive-OR configuration to form a feedback mechanism. An LFSR is represented by a polynomial: for example $X^{16} + X^5 + X^4 + X^3 + 1$ represents an LFSR formed by 16 shift registers for which the 3rd, the 4th and the 5th shift registers content are modified by a XOR between the output and the previous shift register as we can see in figure 3.8. The LFSR is initialized at a seed value.

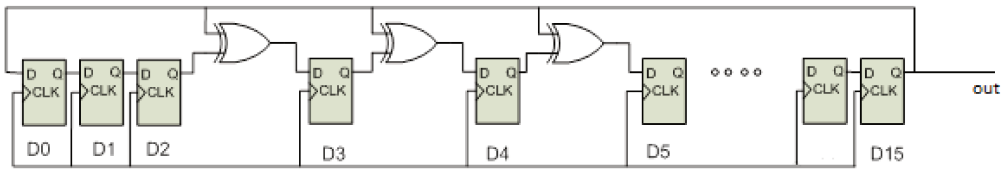


Figure 3.8 LFSR Galois representation of the polynomial: X16 + X5 + X4 + X3 + 1

We note that there are two representations of the LFSR, the Galois and Fibonacci. We won't go through the details and the differences because it is out of scope of this thesis. We also note that the polynomial's value (even when it

is from the same degree) should be carefully chosen to generate a good pseudo-random sequence. In the simulations in this thesis, we will use polynomials that were implemented in famous standards and have been proven to provide good characteristics.

The Pseudo-Random Binary Sequence (PRBS) characteristics:

An N-bit LFSR generates a repetitive PRBS of length 2^N-1 bits. The PRBS pattern ensures a Run Length bounded to N bits. The PRBS provides equal probability of 1's and 0's. The Running Disparity of the PRBS pattern varies from a polynomial to another. An example of the $X^{16} + X^5 + X^4 + X^3 + 1$ polynomial with FFFFh as seed value is represented in Figure 3.9.

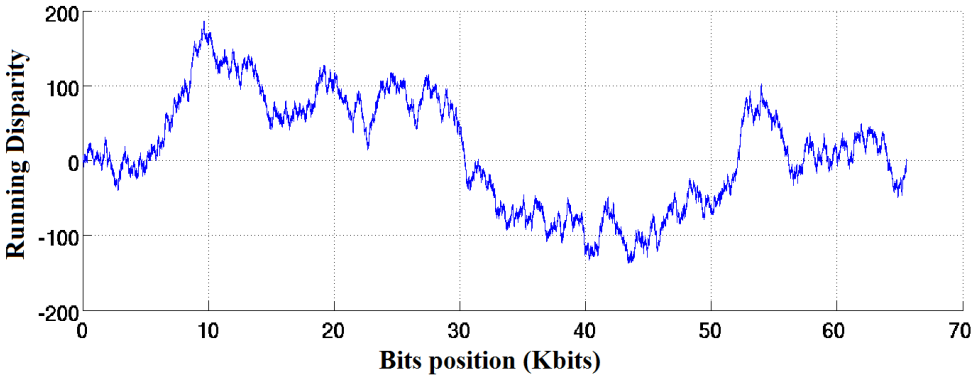


Figure 3.9 RD representation of the PRBS generated by the polynomial: X16 + X5 + X4 + X3 + 1, seed value 1FFFFh

Scrambled data's characteristics:

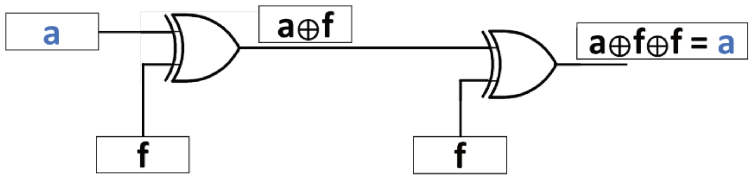
As mentioned before, scrambling is a XOR between the raw data and the PRBS sequence. The XOR operation was chosen because of its characteristics:

- Binary data with any probability distribution of 1's and 0's, once XORed with a sequence of equal distribution of 1's and 0's, results in data (scrambled data) with equal probability of 1's and 0's. This isn't the case

for other logic gates. In other terms, if we consider P as the probability of 1's:

- If P(PRBS) = 0.5 → P(scrambled data) = 0.5 if the data window is long enough, whatever P(raw data) is. This is explained in details in **Annex A** and a simulation in Figure 3.10.a shows the percentage of 1's after scrambling for frames with different percentages of 1's before scrambling.

- The raw data (initial data) can be reproduced on the receive side from the scrambled data by reproducing the PRBS (applying the same LFSR). It is a well know result that the XOR and NXOR functions are the only binary functions of two variables that feature this involution property.



- When the data is XORed with the PRBS, the scrambled data will be statistically randomized. This results in spreading the PSD spectrum of the V_{cm} eliminating the peaks, and reducing the EMI. This is illustrated in Figure 3.10.b.

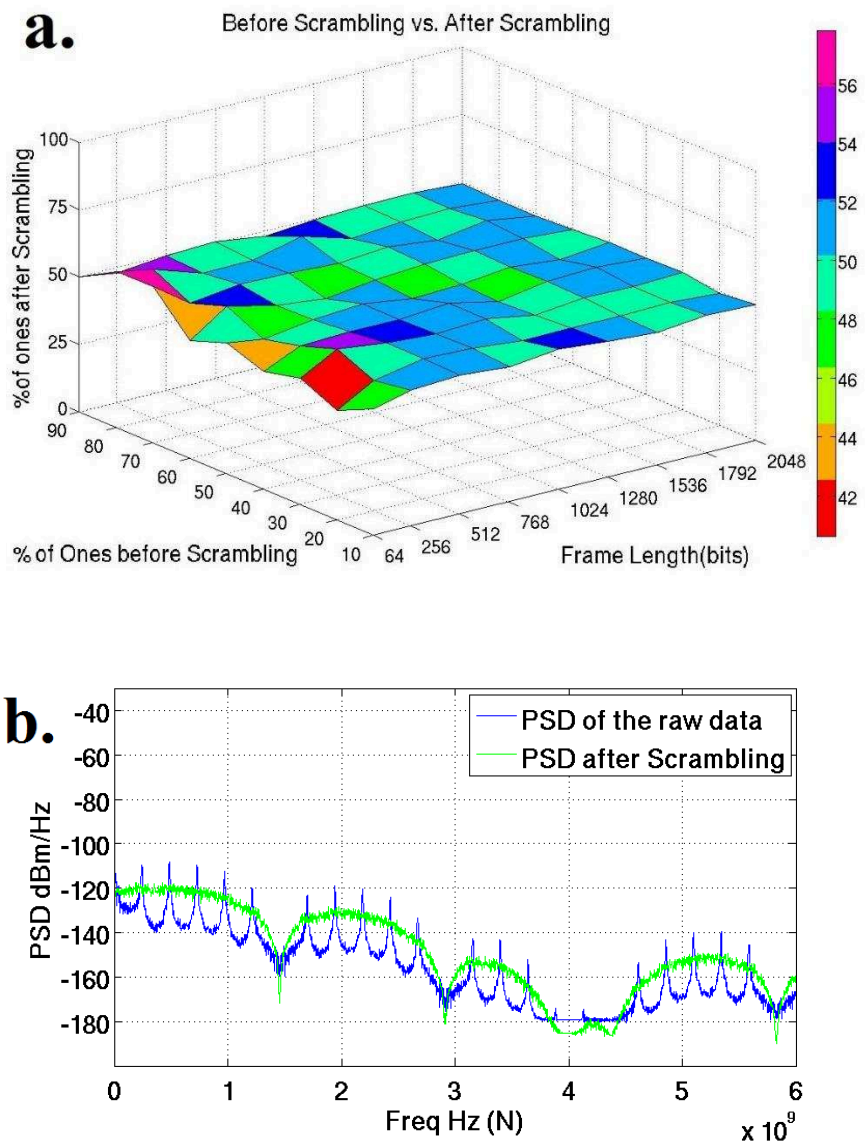


Figure 3.10 a. Percentage of 1's before and after scrambling b. spectrum of the V_{cm} of the data before and after scrambling

Balancing the number of 1's and 0's inside the data results in two major benefits:

1. Scrambled data has statistically more transitions than raw data before scrambling especially if the raw data is very unbalanced in terms of 1's and 0's. By using Markov Chains, we can get a theoretical estimation of the run length distribution. Table 3.2 summarizes the distribution from a RL of 5 to

a RL of 20. The values in Table 3.2 are deduced from the theoretical study in **Annex B**. We also made a simulation on long sequences of data and made a comparison.

Run Length	Occurs Theoretically in average (Bytes)	Occurs according to our simulation Min/Average/Max (Bytes)
5	4	1/8.45/26
6	8	1/17/49
7	16	2/35.6/100
10	128	9/302/748
14	2 K	128/6.34 K/19.3 K
18	32 K	5.42 K/64.6 K/240.3 K
20	128 K	5.7 K/262 K/784.3 K
:	:	:

Table 3.2 Run Length Distribution after scrambling

2. Scrambling statistically reduces the Running Disparity especially if the raw data is not balanced. Figure 3.11 shows an example.

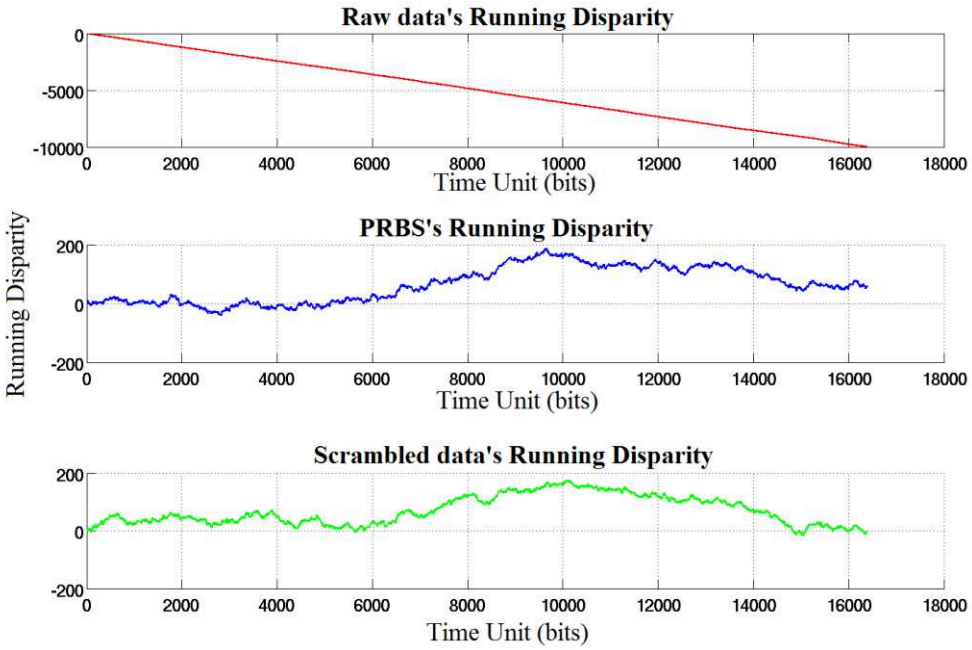


Figure 3.11 Raw data's disparity vs Scrambled data's disparity (raw data distribution 80% of 0's and 20% of 1's, polynomial: X16 + X5 + X4 + X3 + 1, seed value FFFFh)

***Scrambling's advantages*:**

To summarize, we can deduce the following advantages from scrambling:

1. Scrambling helps in reducing EMI by randomizing the data and eliminating redundant patterns.
2. Scrambling creates transitions by balancing the number of 1's and 0's. This is beneficial in clock and data recovery.
3. Scrambling reduces the Running Disparity, which means Baseline Wander reduction and Data Dependent Jitter reduction.
4. Scrambling has 0% overhead. No bits are added to the transmission

Scrambling's drawbacks:

Despite all of its advantages, scrambling has the following drawbacks:

1. Scrambling could produce repetitive patterns that will cause peaks in the V_{cm} spectrum, causing EMI. We will call them EMI Killer packets. Even though their probability of happening is low, they could still happen.
2. Scrambling creates transitions inside the data, but it does not ensure a guaranteed bound for the RL. Let's suppose a CDR that can handle a maximum run length of 9. According to table 3.2, a run length of 10 happens theoretically every 128 bytes. An error could then occur on the recovery every 128 bytes requiring a retry and degrading system performance. Even when the CDR can handle big values of RL, patterns could be designed (aligned with the PRBS) to create hundreds of consecutive Identical Bits [30] that are known as killer packets.

3. Scrambling reduces the RD but it does not guarantee a certain bound. The RD could still reach high values that can go more than +/- 1000. In addition to analog filters that could be added to correct the BLW, Protocols like PCIe 3.0 cut the transmission when the RD reaches high values and send special patterns to balance the RD. This also affects system performance and latency.

Standards using scrambling:

Many scrambling-based encodings have been implemented on HSSLs standards. The 64b/66b encoding used in 10G Ethernet uses scrambling and adds 2 bits "sync header" ('10' or '01') to every 64 bits to ensure a transition and indicate whether the frame is control or data. PCIe 3.0 uses 128b/130b encoding using the same principle. USB 3.1 uses 128b/132b encoding adding 4 bits sync header ('1010' or '0101') enabling a single error in the sync header to be corrected without going through a retry.

3.3.5 The Polarity Bit Coding

The polarity bit coding is one of the most overhead-optimized methods that bounds the Running Disparity. Over time, DC-balanced codes have been introduced. In 1986, Knuth proposed a method [31] to construct frames with equal number of 0's and 1's. Knuth proved that any binary sequence of a specific size, could be balanced by inverting, at a specific bit position, all the rest of the sequence. The drawback of this method is that this particular bit position must be sent with the frame (and should be balanced as well) for the receiver to know how to reconstruct the original frame. This will add a relatively important number of bits for small frames. For large frames, the number of added bits is less important, but the RD could reach high values inside the frame before going back to zero. Other Knuth-based methods were proposed, but as far as we know, they did not solve the high overhead issue.

The simplest and the lowest overhead method is the polarity-bit coding. It consist of systematically adding 1 bit to a frame of a specific size to indicate whether it is inverted or not depending on the Cumulated RD (CRD) and the RD of the frame itself; i.e. if the CRD is positive, and the RD of the frame is positive as well, all the bits inside the frame will be inverted and the polarity bit will transmit the info to the receiver.

The polarity bit coding is used by the 64b/67b encoding; 3 bits are added to the 64 bits of the frame: 2 bits (‘10b’ or ‘01b’) to ensure a transition and indicate whether the frame is raw data or control, and 1 polarity bit to indicate if the 64 bits (which are scrambled) are inverted or not. The CRD bound ensured by such coding could be deduced from the worst case scenario according to equation (3.1):

$$CRD_{bound} = +/- (FrameSize + FrameSize/2) \quad (3.1)$$

Which gives for the 64b/67b encoding CRD_{bound} = +/- 96 for FrameSize = 64. The overhead cost for the CRD bound is 1/64 = 1.56 %. The total overhead cost is 3/64 = 4.687 %.

3.3.6 Summary of some existing encoding methods

The table below summarizes the line coding’s state of the art.

Line Coding	Standards	Max RL	RD Bound	Overhead
Bit Stuffing	CAN USB 2.0	5 6	N/A N/A	0% to 20% 0% to 16.6%
8b/10b	PCIe 2.0, USB 3.0 …	5	+/- 3	25 %
Scrambling-Based codings				
64b/66b	10G Ethernet	64	N/A	3.125 %
128b/130b	PCIe 3.0	128	N/A	1.562 %
128b/132b	USB 3.1	128	N/A	3.125 %
Scrambling + polarity bit based coding				
64b/67b	Interlaken	64	+/- 96	4.687 %

Table 3.3 Overview on some existing encoding methods

3.4 *State of the Art's Conclusion*

In the first part of this chapter we overviewed three High Speed Serial Links and we showed the differences on system-level justifying the variety of HSSLs protocols.

In the Line Coding's state of the art, we overviewed many encoding methods used in today's standards. We showed how a line coding that bounds the RL and the RD to low values will have high overhead, and when releasing the constraints on RL and RD we can design a line coding with low overhead. Releasing the RL and RD constraints might result in more analog complexity.

One interesting line coding which has no overhead is the scrambling. Scrambling has 0% overhead while providing good characteristics, but it does not guarantee randomization, or RL bounds, or RD bounds.

In this thesis we propose methods that are able to benefit from scrambling's advantages while guaranteeing randomization, RL bounds and RD bounds with a very low overhead.

Chapter 4

Low EMI encoding method

4.1 *Chapter's* Introduction

Using Scrambling as a technique to reduce EMI is efficient. However, as we explained in chapter 3, scrambling could generate repetitive patterns that will end up increasing EMI. Repetitive patterns after scrambling could also be designed on purpose to break the system.

In this chapter, we propose a technique that eliminates the possibility of generating or designing a repetitive pattern.

4.2 Probability of a repetitive pattern

The probability of having a repetitive pattern after scrambling is considered low. In **Annex C** we calculate the probability for a pattern of length "L" bits to be repeated "M" times after scrambling. This probability is given in equation (4.1):

$$\mathbf{P}_{(L,\,M)} = \frac{2^{L}}{2^{L*M}} \qquad (4.1)$$

Where **L**: length of the pattern in bits

M: the number of repetition

Example:

Consider we want to calculate the probability of a byte to be repeated 5 times in a row:

$$P_{(8,\,5)} = \frac{2^{8}}{2^{8*5}} = \frac{2^{8}}{2^{40}} = 2.328 \times 10^{-10}$$

This is the probability of happening in a time unit of 40 bits. For this repetition to happen once, we can calculate after how many bits this could happen as follows:

$$P_{(L,\,M)} \rightarrow L*M = 40 \text{ bits}$$

$$1 \text{ occurrence} \rightarrow X \text{ bits?}$$

$X = 40 / P_{(L,\,M)} = 1.718 \times 10^{11}$ bits

This means that after scrambling, a byte can be repeated 5 times in a row once every 1.718×10^{11} bits. At 10 Gbits/s throughput, this will happen theoretically in average every 17 seconds (1.718×10^{11} bits/10×10^{9} bits/s).

As we saw in this section, the probability of a repetitive pattern is low, but it can happen rapidly depending on the link's frequency and could generate EMI, creating errors in RF components or neighboring lanes of the same link. It is then a question of time.

If the critical pattern length and repetition number that could cause errors shows to happen rarely, i.e. a pattern of length 8 bits will be repeated 8 times every 14 years at 10 Gbits/s after scrambling, then scrambling can be good enough.

With the increasing demand for bandwidth, repetitive patterns can happen more often, and the small number of repetitions could generate EMI. An error every few seconds or milliseconds can trigger the retry mechanism and degrade system performance. A protection from EMI killer packets (repetitive packets) after scrambling might then be a necessity.

Another reason why there might be a need to ensure the protection from repetitive patterns is that they might be designed easily for attack purpose; once the scrambling polynomial is known, the PRBS sequence is also known. Patterns could be designed such as once XORed with the PRBS sequence, they generate repetitive patterns that will be source of high EMI.

In the next section, we will present a method to eliminate the probability of a repetitive packet or the possibility of designing such packet.

4.3 Method to eliminate the probability of repetitive patterns

A good method to randomize a repetitive pattern is to scramble it. To randomize the repetitive packets after scrambling, we propose to scramble a second time. But should we re-scramble all the data after a first scrambling or should we re-scramble the repetitive packets only?

4.3.1 Re-scrambling all the data after the first scrambling

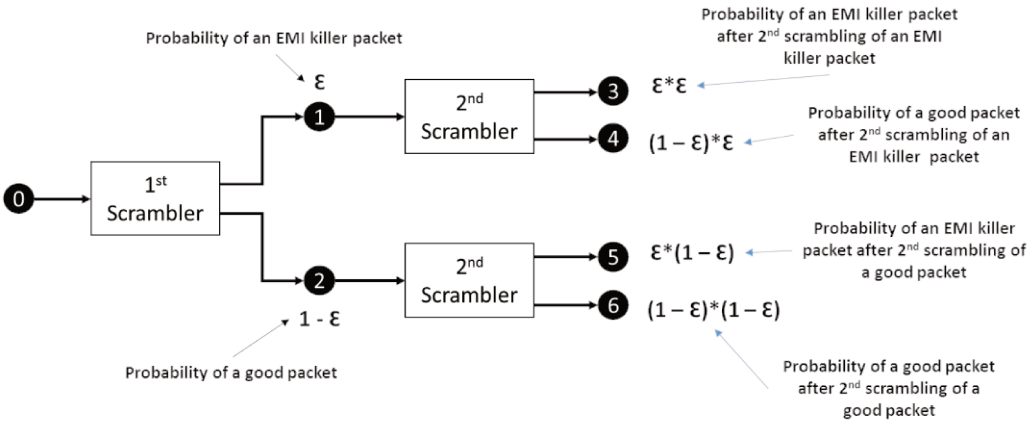


Figure 4.1 Probability of a repetitive pattern after a 2nd scrambling

The probability of having a repetitive packet after scrambling is low. In figure 4.1, its value is represented by epsilon "Ɛ", and this state is called state ① after the 1st scrambler. The probability of having a good (randomized) packet after scrambling is the probability of not having a repetitive packet, which is 1- Ɛ. In figure 4.1, it is represented by the state ② after the 1st scrambler.

When we automatically apply the 2nd scrambler after the first one, the 2nd scrambling is performed on both states ① and ②, this results in 2 states for each, which means 4 final states: ③, ④, ⑤ and ⑥:

- ③ is the state where a killer packet is generated after the scrambling of state ① (a killer packet resulting from the 1st scrambling). Its probability is: Ɛ*Prob(state 1) = Ɛ*Ɛ
- ④ is the state where a good packet is generated after the scrambling of state ① (a killer packet resulting from the 1st scrambling). Its probability is: (1- Ɛ)*Prob(state 1)= (1- Ɛ)*Ɛ

- ⑤ is the state where a killer packet is generated after the scrambling of state ② (a good packet resulting from the 1st scrambling). Its probability is: ε*Prob(state 2) = ε*(1- ε)
- ⑥ is the state where a good packet is generated after the scrambling of state ② (a good packet resulting from the 1st scrambling). Its probability is: (1- ε)*Prob(state 2) = (1- ε)*(1- ε)

To verify, the sum of the probabilities of states ③, ④, ⑤ and ⑥ is 1.

The probability of having a killer packet is the sum of the probabilities of states ③ and ⑤ which is:

$$Prob_{(Killer)} = \varepsilon*\varepsilon + \varepsilon*(1- \varepsilon)$$

$$\underline{Prob_{(Killer)} = \varepsilon}$$

The probability of having a good packet is the sum of the probabilities of states ④ and ⑥ which is:

$$Prob_{(good)} = (1- \varepsilon)*\varepsilon + (1- \varepsilon)*(1- \varepsilon)$$

$$\underline{Prob_{(good)} = (1- \varepsilon)}$$

Conclusion:

The probability of an EMI killer packet and the probability of a good packet after applying a 2nd scrambler for all the packets of the 1st scrambling, are exactly the same as the probabilities of states ① and ②. Therefore, there is no interest from applying a 2nd scrambling on all packets.

4.3.2 Re-scrambling with repetitive packets selection

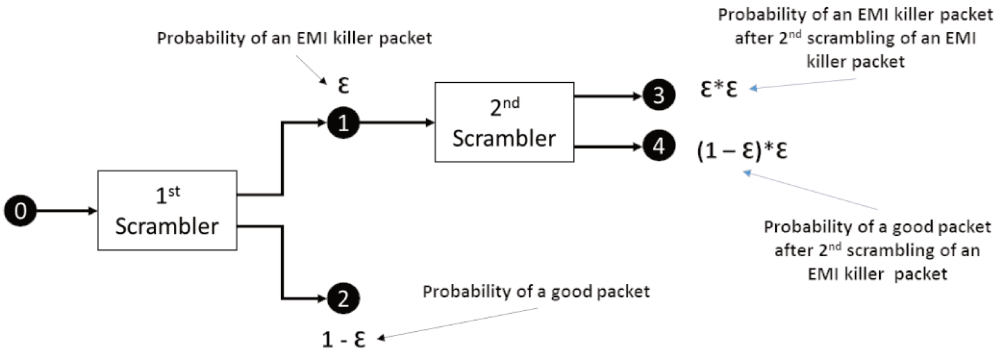


Figure 4.2 Probability of a repetitive pattern after a 2nd scrambling of repetitive packets only

In this case we consider the detection of killer packets, and we apply the 2nd scrambling only when state ① is detected. The final states are states ③, ④, and ②.

To verify, the sum of the probabilities of states ③, ④ and ②is 1.

The probability of having a killer packet is the probability of the state ③ which is:

$$\underline{Prob_{(Killer)} = Ɛ*Ɛ}$$

The probability of having a good packet is the sum of the probabilities of states ④ and ② which is:

$$Prob_{(good)} = Ɛ*(1- Ɛ) + (1- Ɛ)$$

$$\underline{Prob_{(good)} = 1 - Ɛ*Ɛ}$$

Conclusion:

In this case, the probability of a repetitive packet has been changed from 'Ɛ' after the first scrambling into 'Ɛ*Ɛ' after the 2nd scrambling. And because 'Ɛ' is a small fraction of 1, this means that:

$$Ɛ*Ɛ << Ɛ$$

The probability of a killer packet has been extremely reduced with repetitive packet selection.

The "Ɛ* Ɛ" case:

In **Annex D**, we analytically prove that a repetitive pattern can happen after the selection and re-scrambling of a repetitive pattern, only if the 2nd scrambler's PRBS sequence is repetitive. If the LFSR polynomial of the 2nd scrambler is well chosen, it will never include a repetitive sequence in its PRBS pattern. A repetitive sequence after scrambling the repetitive pattern can't happen and cannot even be designed.

4.3.3 Reduced EMI line-coding

According to the analysis we made, we propose the following method for a reduced EMI line coding:

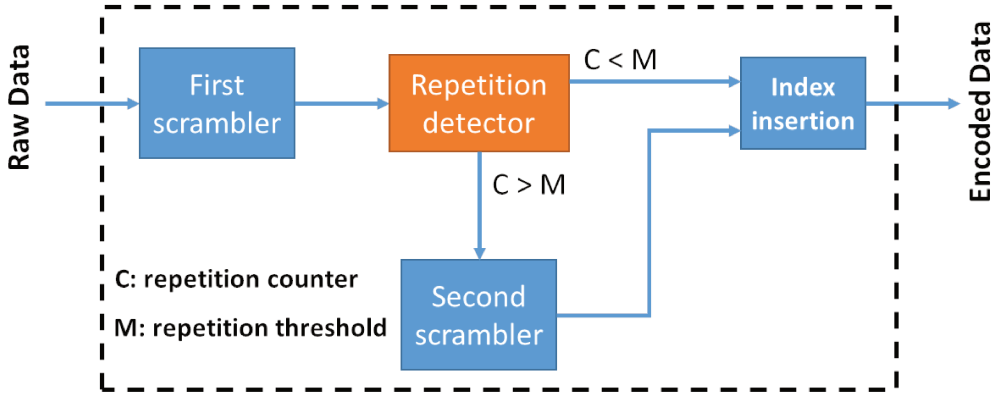


Figure 4.3 Proposal's block diagram for a reduced EMI line coding

The line coding we propose consists of scrambling all the raw data with a first LFSR polynomial. At the output of the first scrambling stage, a pattern repetition detector buffers a frame of a specific number of bits and counts repeated patterns. For every repetition, the repetition detector will increment its counter 'C' by 1 unit. If there is no repetition, or if 'C' do not exceed a certain

threshold “M” (that does not cause EMI problem), the buffered frame will be forwarded to the index insertion block that will add a bit, to indicate to the receiver that the frame is scrambled once (by adding a ‘0’ bit to the frame for example).

If C > M, the frame will be forwarded to the second scrambler which has a different LFSR polynomial than the first scrambler. The frame is then scrambled a second time and forwarded to the Index insertion block that adds a bit (a ‘1’ for example) to indicate that the frame has been scrambled twice. The encoded data and the framing are illustrated in figure 4.4.

We note that the 2nd scrambling polynomial should be, as mentioned, different from the first one, because re-scrambling the data with the same sequence means de-scrambling. The frame (that is scrambled twice with the same polynomial) in this case will be restored to its raw state and might be unbalanced (in the number of 0’s and 1’s) and might contain undesired long sequences of consecutive 1’s and/or 0’s.

Figure 4.4 Proposal’s framing example

The frame’s size must be chosen according to the maximum pattern repetition allowed. For example, if the maximum repetition is 4 identical bytes, the frame size can be 4 bytes. 1 bit is then inserted every 64 bits. The proposed method has then an overhead of 1/frame size. The repetition counter is maintained from a frame to another. i.e. if the maximum repetition is 4 and a pattern X has been repeated 2 times in frame 1, and 3 times in frame 2, frame 2 will be scrambled a second time so that the repetition of the pattern X stays 2.

4.4 *Chapter's conclusion*

Scrambling is an efficient method to eliminate redundancy and give a random aspect to the spectrum of the data, randomizing the V_{cm} spectrum which is responsible of EMI in differential signaling. But scrambling could generate EMI killer packets.

In this chapter, we introduced a new method to ensure reduced EMI. The proposed method consists of a first scrambler stage to scramble all the data. A repetition detection block forwards only the frames containing repetitive data to a second scrambling block. This block randomizes the repetitive data with a polynomial different than the first one.

When EMI is a main constraint, the presented method eliminates the possibility of having a repetitive pattern or designing an EMI killer packet. The cost of the proposal is 1 additional bit for each frame.

Chapter

5 Low overhead Run Length limited encoding method

5.1 *Chapter's Introduction*

As we saw in chapter 3, two of the most used methods to limit the Run Length (RL) have two major drawbacks; the 8b/10b encoding bounds the RL to 5 but has 25% overhead. The Bit Stuffing (BS) bounds the RL to the desired value (N), but the BS's Overhead (BSO) is not predictable because it is data dependent, and it can reach high values that goes to 20% for N = 5 for example.

In this chapter, we propose a line coding that can bound the Run Length with a very low overhead down to 8 times lower than 8b/10b's overhead and down to 6 times lower than Bit Stuffing overhead and for the same RL bounds.

5.2 Bit Stuffing overhead *vs. data's distribution*

5.2.1 The maximum Bit Stuffing Overhead (BSO)

The maximum BSO is reached when the raw data to be encoded is all 1's or all 0's, a bit will be added to every N bits, then $BSO_{Max} = 1/N$. This is illustrated in figure 5.1.

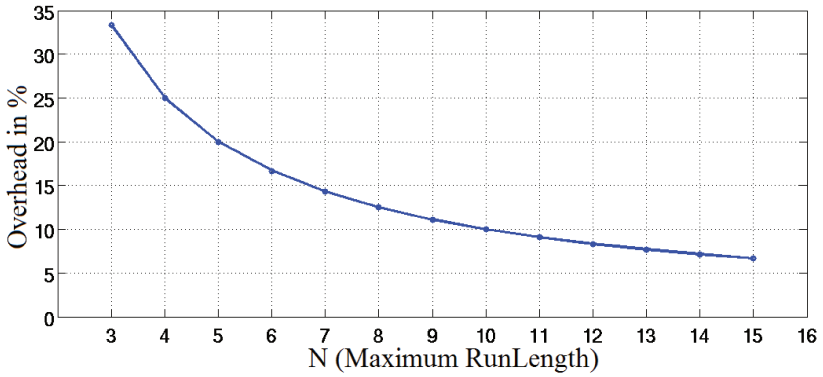


Figure 5.1 Bit Stuffing Maximum Overhead for different N

5.2.2 Theoretical overhead estimation for bit stuffing

In paragraph 5.2.1, we showed the worst case Bit Stuffing Overhead (BSO). In this paragraph, we're going to show the BSO for the different distribution of 1's and 0's inside the frame and deduce the best case. For this purpose, we draw the Markov chain representation for the possible states when doing bit stuffing in figure 5.2 as follows:

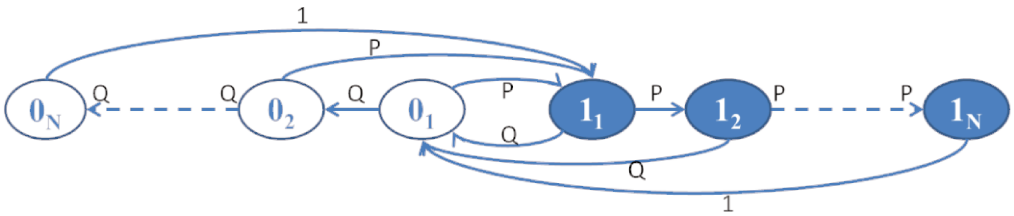


Figure 5.2 Markov Chain representation of Bit Stuffing for a maximum RL of N

We denote by P the probability of 1's, and Q the probability of 0's. The blue circles in figure 5.2 represents the state of 1's and the white ones represents 0's.

1_2 represents the state of 2 consecutive 1's, and 1_N represents the state of N consecutive 1's. Same for the 0's states. To go from a 1_i state to another 1_{i+1} state, or from 0_i state to 1_1 state, the probability is P (the probability of 1's). Conversely, to go from a 0_i state to another 0_{i+1} state, or from 1_i state to 0_1 state, the probability is Q (the probability of 0's).

If the bit stuffing is fixed to N, when we are on the state 1_N, the only possibility is to go to the state 0_1 with a probability of 1. Same when we are on the state 0_N, the only possibility is to go to the state 1_1 with the probability of 1 because bit stuffing is performed for N consecutive identical bits.

In **Annex E** we calculate from the above Markov chain the probability of having N consecutive identical bits, which is the sum of the probabilities of states 0_N and 1_N. This particular probability also represents the bit stuffing overhead, because a bit is added every time the states 0_N and 1_N are reached. The Bit Stuffing Overhead (BSO) for a Maximum Run Length of N can be given by equation (5.1):

$$BSO(N) = Q^{N-1}\Pi 0_1 + P^{N-1}\Pi 1_1 \qquad (5.1)$$

Where Q = Probability of 0's

P = Probability of 1's = 1-Q

$\Pi 0_1$ and $\Pi 1_1$= Probability of the states 0_1 and 1_1

In **Annex E** we also demonstrated that:

$$\Pi 0_1 = \Pi 1_1 = \frac{1}{A + B}$$

Where $A = \frac{1 - Q^N}{1-Q}$ and $B = \frac{1 - P^N}{1-P}$

The Bit Stuffing's overhead for a max RL of N can then be calculated depending on the data's probability distribution of 1's and 0's (P and Q). This is illustrated in figure 5.3.

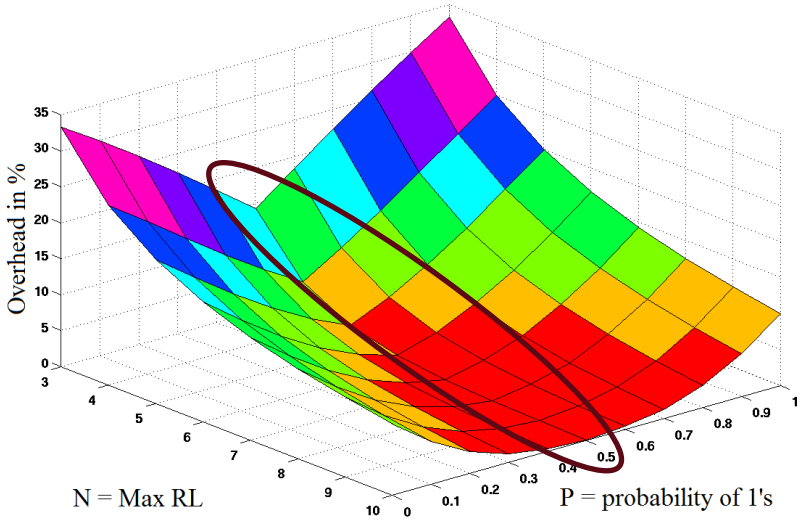


Figure 5.3 Theoretical Bit Stuffing Overhead estimation

5.2.3 The minimum Bit Stuffing Overhead

From figure 5.3, we can see that the BSO is on its minimum values when P = Q = 0.5. This is illustrated in figure 5.4 and compared to the maximum and average BSO values and we can see the huge difference.

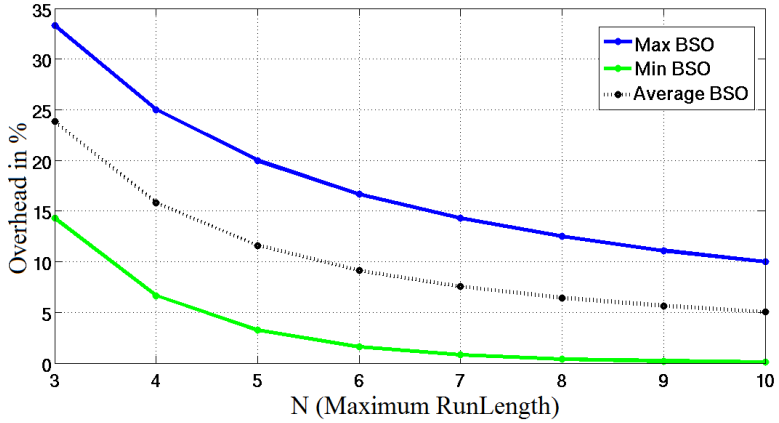


Figure 5.4 Bit Stuffing minimum vs. Maximum Overhead for different N

5.3 Proposal for a low overhead Run Length Limited Encoding

5.3.1 *Proposal's block diagram*

According to the analysis we made in the previous section, the BSO is at its minimal values when the probability of 1's and 0's of the data are equal. As we saw in the study in Chapter 3 and **Annex A**, scrambled data has equal probability of 1's and 0's. Our proposal for a low overhead Run Length limited Line coding is then the following:

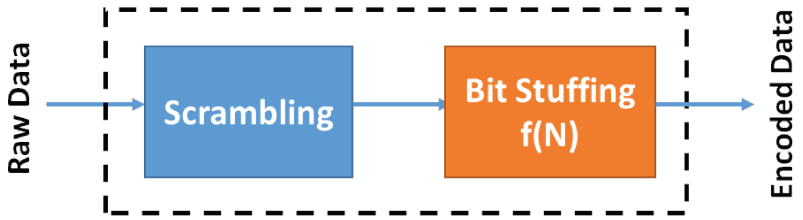


Figure 5.5 Proposal's block diagram for low overhead RL limited encoding

We propose to scramble the raw data at first to give it the characteristic of P = Q = 0.5. After scrambling, the data goes into a bit stuffing block that will insert bits when the RL reaches N.

The theoretical overhead for our RL limited line coding for different N is summarized in table 5.1 and corresponds to the minimum BSO. We also considered the case of encoding real image data so we can make a comparison with the theoretical study. The picture's dimensions are 512x512 pixels, with each pixel being a 24-bit RGB component. The total size of the picture is 786 Kbytes. We can see that the overhead numbers are very close in both cases.

N	3	4	5	6	7	8	9	10
Theory	14,29 %	6.67 %	3.23 %	1,59 %	0,79 %	0,39 %	0,20 %	0,10 %
Image	16.65 %	7.13 %	3.33 %	1.61 %	0.79 %	0.39 %	0.19 %	0.09 %

Table 5. 1 RL-limited encoding proposal's overhead

5.3.2 Power Spectral Density Aspects

To verify that the presented solution does not harm the randomization aspect given by scrambling, we plot the PSD of the V_{cm} generated by encoding the data according to our proposal in figure 5.6 and we compare it with scrambling-only. We can clearly see that the PSD plots are very similar. The presented RL-limited method does not eliminate the random aspect.

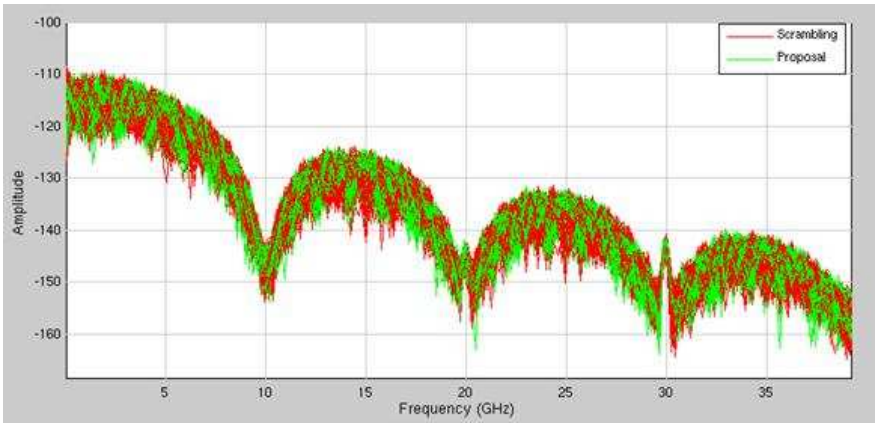


Figure 5.6 PSD of the proposed RL limited method vs. PSD of Scrambling-only at 10 GHz frequency

5.3.3 *Proposal's advantages*

The biggest advantage of the proposed line coding is its very low overhead. As we can see in table 5.1, to ensure the same RL bound as 8b10b encoding which is 5, our proposed method has an overhead of 3.23% whereas the 8b10b's overhead is 25%. If we release the constraints on the RL bound, we can also

lower the overhead down to less than 1%. Practically, Low overhead offers many advantages for the designers or the users as follows:

a. Improved bandwidth efficiency over 8b/10b encoding

A link running at a specific frequency will benefit from an obvious improvement in throughput. The raw throughput (Th) as a function of the link's frequency (LF) and the encoding's overhead (OH) could be given by the following equation:

$$Th = \frac{LF}{1+OH} \qquad (5.2)$$

An example of the raw throughput difference between 8b/10b encoding and the RL-limited encoding for N= 5 (OH considered 3.5 %, equivalent to 8b/10b encoding in RL bound) at different link frequencies are shown in figure 5.7.

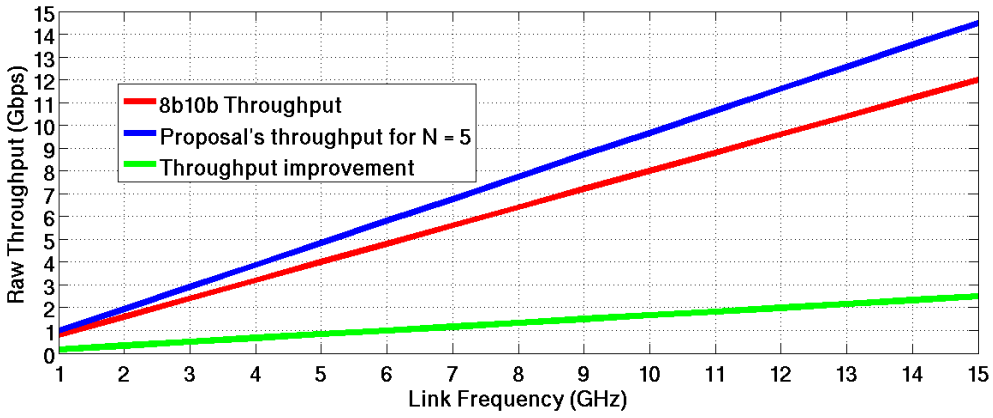


Figure 5.7 Raw Throughput comparison vs. Link frequency for data encoded with 8b/10b and the proposed RL-Limited encoding

As we can see from the above figure, we can improve the throughput to many Gigabits per second (Gbps) thanks to the proposed encoding while keeping the same RL bounds. At 6 GHz link frequency, the raw throughput using our line coding is 1 Gbps better than when using 8b/10b encoding. At 12 GHz, we can gain up to 2 Gbps throughput.

b. Power consumption reduction

One of the benefits from reducing the overhead is power consumption reduction. While the power consumption for the high speed links is generally given in mW/Gbps, one of the recent studies and implementations [32] estimates the power consumption per transmit/receive unit at 2.8 mW/Gbps. When the data is encoded, the power consumption (Pc) could be given by the following equation:

$$Pc(encoded\_data) = Pc(raw\_data) + OH*Pc(raw\_data) \quad (5.3)$$

If we consider we target a throughout of 10 Gbps, the power consumption compared to 8b/10b encoding could be given as follows:

Target Throughput	Power consumption per Gbps	8b/10b encoded data power consumption	Proposed encoding power consumption
10 Gbps	**2.8 mW**	**35 mW**	**28.98 mW**

We can see that we can save 6 mW per transmit/receive unit when using the line coding we propose in this chapter.

c. Lane Count reduction over 8b/10b encoding

Reducing the line coding's overhead can enable in many cases lane count reduction. Multi-lanes is the feature of many protocols because it allows throughput improvement and multiplication. However, throughput multiplication might not be the protocol's requirement because the protocol might need few Gpbs more to reach its target raw throughput. The proposed low overhead line coding might then enable lane count reduction. This is illustrated in figure 5.8 where we consider MIPI's M-PHY physical layer running at High-Speed Gear 4 (HSG4) which is 11.64 GHz. The figure illustrates the lanes

saving for different raw target throughput. We can see that we save up to 50 % of the Physical layer's complexity and power consumption thanks to our encoding.

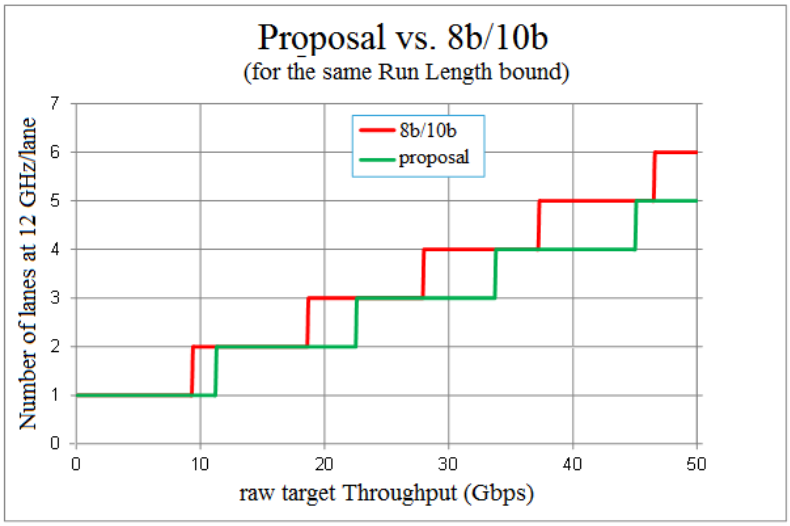


Figure 5.8 Lane-count reduction thanks to our proposed RL-limited encoding in the case of MIPI's M-PHY running at HSG4 (11.64 Gbps)

Table 5.2 shows real use cases where lanes reduction and power/area saving could be done.

					data rate [Mbps]			nb of lanes	
App.	use case	Pixels	fps/ rate	bits	raw	8b10	bit stuffing5	8b10	bit stuffing5
camera	33.2MPIX30P10B	33177600	30	10	9953	12442	10302	2	1
	4K2K120P10B	8294400	120	10	9953	12442	10302	2	1
DDR	DDR1066-32		1067	32	34133	42667	35328	4	4
	DDR1600-32		1600	32	51200	64000	52992	6	5
	DDR2133-32		2133	32	68267	85333	70656	8	7
	DDR3200-32		3200	32	102400	128000	105984	11	10

Table 5.2 Real use cases that can benefit from lanes reduction

d. *Reduce the CDR's analog complexity*

As highlighted in paragraph 2.3.3, the lack of transitions inside the data can push designer to integrate analog solutions that could increase the clock recovery's complexity up to twice. The proposed RL-limited solution enables hardware complexity reduction (which means area and power consumption) over encoding that are not RL-limited.

e. Early Errors Detection

Errors could be detected when the run length exceeds N (the maximum fixed by the proposed encoding) before forwarding the data to the upper layer (Data Link Layer) and CRC check.

f. Interoperability

This line coding also allows interoperability between CDR units having different RL requirements. i.e. a receiver can ask a transmitter to encode with bit stuffing for a specific N. This can happen at the link initialization process; an attribute can be allocated for this purpose.

5.4 *Chapter's conclusion*

In this chapter we proposed a low overhead run length bounded line coding which combines the benefits of scrambling and bit stuffing.

The proposed coding enables a run length bounded to 5 while having an overhead of 3.23% instead of 25% for 8b/10b for the same RL bound. This allows better throughput efficiency for the same link frequency, or reducing the frequency for a same target throughput. Throughput reduction can enable lane count saving up to 50%, which means 50% power consumption reduction of the physical layer which is the most power-hungry part of a High-Speed Serial link.

This line coding also allows reducing the CDR complexity, early errors detection and interoperability between CDR units having different run length requirements.

We note that the variable data length due to this proposal can be problematic to the PHY layer's framing, a proposal to variable length data is added in **Annex G.**

Chapter

6 Low overhead DC-balanced encoding method

6.1 *Chapter's Introduction*

The polarity-bit encoding is the most overhead optimized DC-balanced method as we saw in chapter 3. However, for small RD (Running Disparity) bounds, this method have a high overhead as illustrated in figure 6.1.

As we can see, this method is only advantageous for high RD bounds. For the same RD bounds ensured by 8b/10b encoding (+/- 3), the polarity bit method adds 50% overhead whereas 8b/10b has 25% overhead.

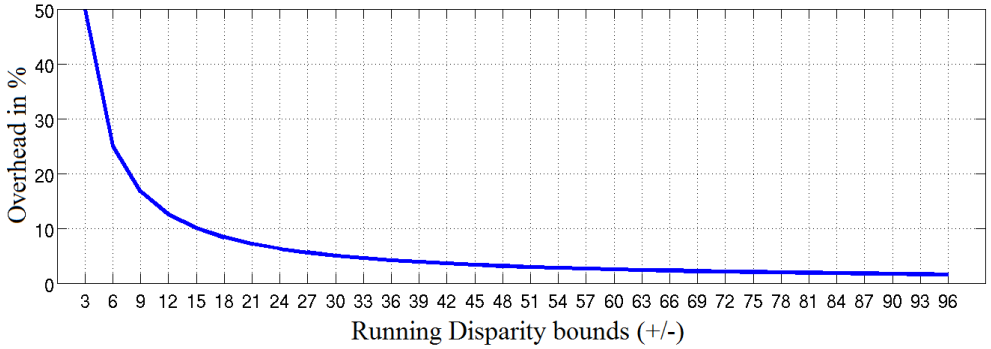


Figure 6.1 Polarity-bit encoding's overhead (deduced from equation 3.1)

In this chapter, we will introduce a novel method which bounds the Running Disparity with a much lower overhead than the polarity-bit encoding for small RD bounds as well as for high RD bounds. This method has also an overhead significantly lower than 8b/10b's overhead, for the same RD bounds.

6.2 A Novel DC-balanced Line Coding

6.2.1 Introducing the method

Inverting bits is an efficient method to reduce the RD, but systematically inverting means systematically adding a polarity bit to indicate to the receiver if the frame has been inverted or not, which as we saw is not beneficial for small RD bounds.

The method we propose consists of bits inversion using aperiodic frames. The RD of the transmitted data that we denote by CRD (Cumulated Running Disparity) is counted bit-by-bit on the transmitter's side, and when the CRD reaches a certain threshold T, the RD of the next packet of Size 'S' bits is checked to see if the packet should be inverted, or not. A bit will be inserted after the S bits to indicate if they were inverted or not. Only when RD(S) = 0, there will be no bit added. In other words, the programming should be done according to the following logic:

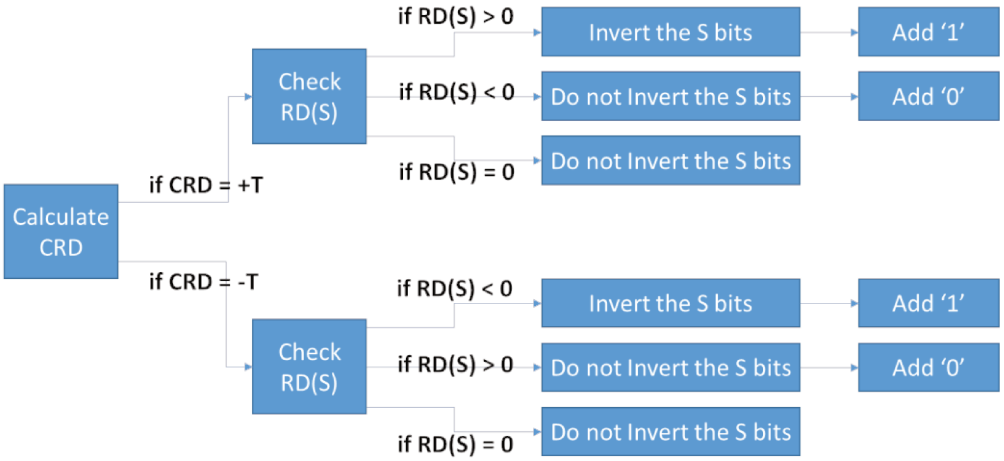


Figure 6.2 Organization chart of the proposed balancing method

• If CRD = +T and RD(S) > 0, the S bits will be inverted and a '1' bit will be inserted to indicate it to the receiver.

• If CRD = -T and RD(S) > 0, the S bits will not be inverted and a '0' bit will indicate it to the receiver.

• If CRD = +T and RD(S) < 0, the S bits will not be inverted and a '0' bit will indicate it to the receiver.

• If CRD = -T and RD(S) < 0, the S bits will be inverted and a '1' bit will be inserted to indicate it to the receiver.

• If CRD = +/-T and RD(S) = 0, the S bits will not be inverted and **no** bit will be inserted. The receiver will know when RD(S) = 0 that the bits were not inverted by default.

Figure 6.3 shows an example of data being coded by the proposed method and figure 6.4 shows and example of the behavior of the CRD of scrambled data, before and after applying the proposed encoding.

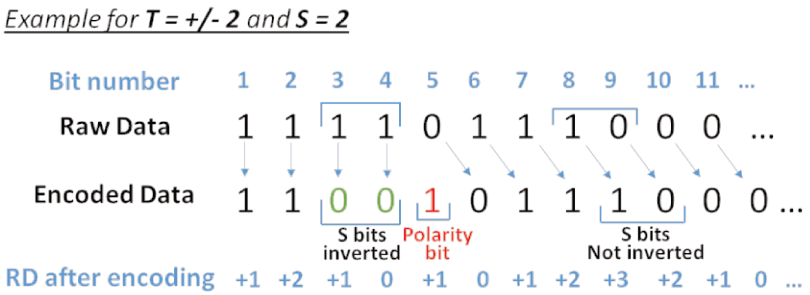


Figure 6.3 Example of data coded with our proposed method

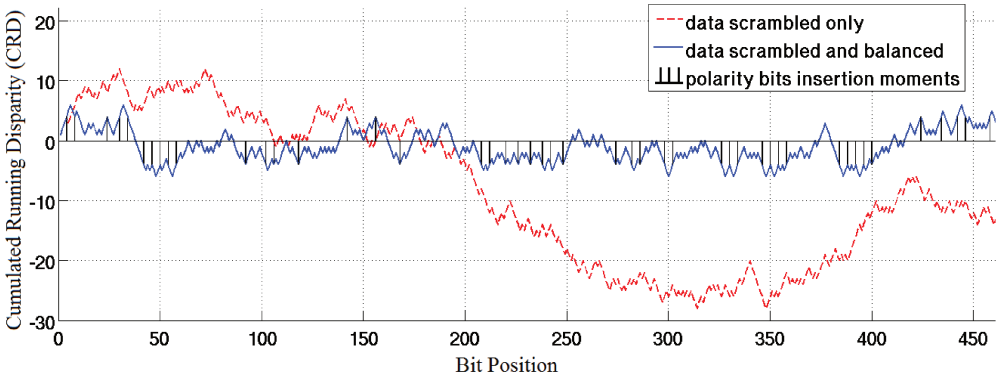


Figure 6.4 Example of the CRD of scrambled before and after balancing with our proposal for T=5 and S=2 (scrambling polynomial: X23 + X21 + X16 + X8 + X5 + X2 + 1 with seed value FFFFFFh)

This logic allows the receiver to recover the data. Figure 6.5 illustrates the Transmitter and the Receiver's block diagram. S bits should always be buffered and RD(S) is calculated permanently. The values of T and S should be agreed on by the transmitter and the receiver before the start of transmission. This could be done at the initialization of the link, and attributes could be allocated for this purpose. The values of T and S could be also pre-fixed by the protocol that is using this coding.

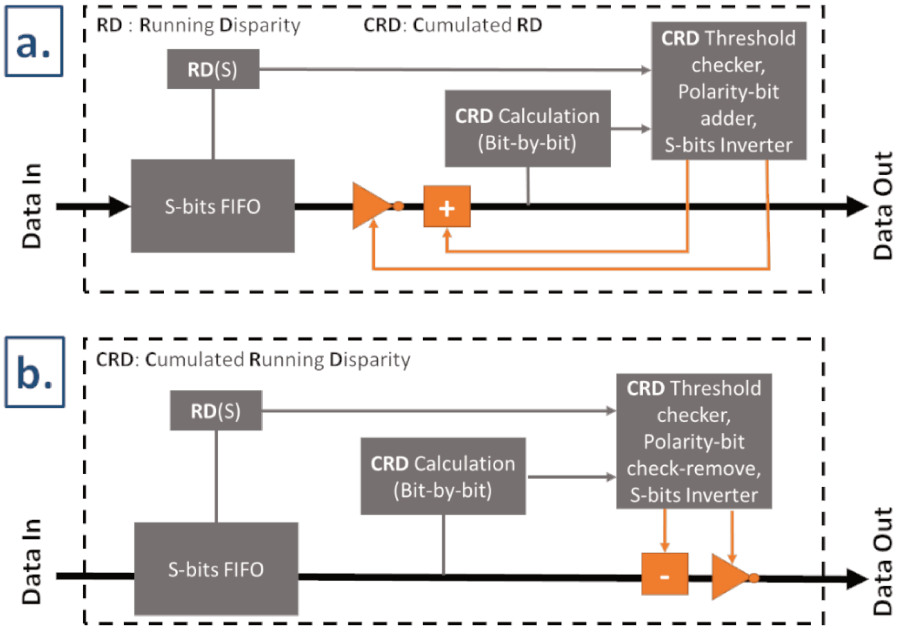


Figure 6.5 Proposed DC-balancer's block diagram a. Transmitter b. Receiver

6.2.2 Ensured Running Disparity Bounds

The worst case scenario that will push the running disparity to its maximum/minimum value is to have a CRD of +T (or –T) and then have an S packet which will be half 1's followed by a next half of 0's (or half 0's followed by half 1's). In this case RD(S) equals zero. The S bits won't be inverted so the CRD will reach it worst case value. The proposed DC-balancer ensures then a Running Disparity bounded to:

$$CRD_{bounds} = +/- (T + S/2) \qquad (6.1)$$

6.2.3 Ensured Run Length Bounds

The fact of bounding the Running Disparity (RD) offers Run Length (RL) bounds. Let's suppose we encode to bound the CRD to +/-3, the worst case RL

will be when going from a CRD of –3 to a CRD of +3 with a RL of 6 ones, or inversely. The RL bounds could be given by the following equation:

$$RL_{bounds} = 2*CRD_{bounds} = 2*(T + S/2) \qquad (6.2)$$

6.2.4 Conditions required

To ensure the bounds mentioned in equation (6.1), condition 1 should be respected:

Condition 1: **T > S/2**

If T <= S/2, the S bits can push the RD out of the limits as follows:

e.g. if T = 2 and S = 6 the CRD should be bounded to +/- 5. But suppose at a certain time we have CRD = +2 and RD(S) = -6. In this case the S bits won't be inverted because they allow us to reduce the CRD. The CRD will the go down to -4, and with the polarity bit inserted (which will be 0) the CRD is now at -5. We should check then the next S bits again. Suppose the next bits are at "000111", RD(S) = 0, the bits are not inverted and the CRD will then reach -8 violating the +/- 5 bounds. If T > S/2, this cannot happen.

The following conditions, 2, 3 and 4, should be respected in order to optimize the overhead as much as possible:

Condition 2: **S is even**

It is the only case where RD(S) could be equal to 0, enabling the encoding to not add a polarity bit and reducing the overhead.

Condition 3: **insertion of the polarity-bit after the S bits**

Inserting the polarity-bit at first will increase the overhead because it should be inserted also for the case where RD(S) = 0, whereas polarity-bit insertion after the S bits will allow the receiver to check the S bits first and know that

once RD(S) = 0, no polarity-bit has been inserted by the transmitter and overhead will be saved.

Condition 4: **Apply Scrambling before the proposed line coding**

This condition is optional but scrambling the data before applying the proposed DC-balancing will reduce the RD of the raw data. The proposed DC-balancer will then intervene less adding less bits. A second reason to use scrambling is that it allows the overhead to be independent from the raw data's distribution.

6.2.5 Power Spectral Density Aspect

To verify that the presented solution does not harm the randomization aspect given by scrambling, we plot the PSD of the V_{cm} generated by encoding the data according to our proposal in figure 6.6 and we compare it with scrambling-only. We can clearly see that the PSD plots are very similar. The proposed DC-balancer does not eliminate the random aspect.

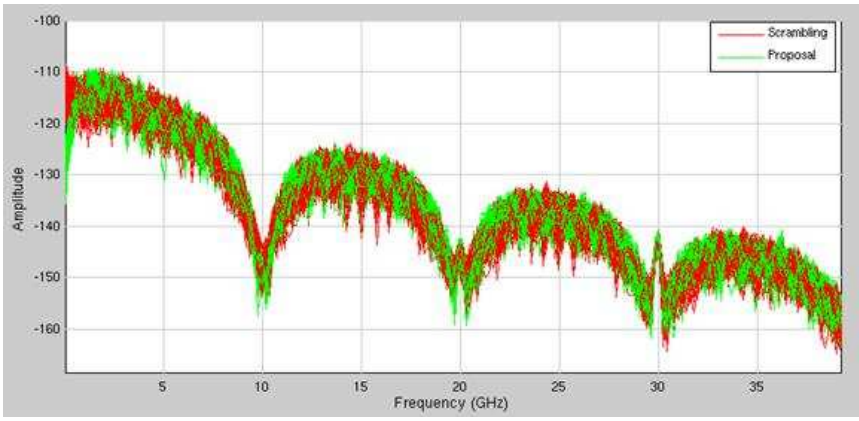


Figure 6.6 PSD of the V_{cm} of our proposed method vs. Scrambling's PSD at 10 GHz frequency

6.3 Overhead Estimation

6.3.1 Simulation-Based Overhead Estimation

On Matlab, we generate 200 random frames of 400 Kbits each, and then apply the line coding we propose on scrambled frames. We then make the average of the overhead of the 200 frames. The results of the overhead is given in table 6.1. We also made a theoretical overhead estimation study in **Annex F** for some overhead values and the results are also shown in table 6.1.

T	S	CRD bounds	Simulation Overhead	Theoretical Average Overhead
2	2	**+/- 3**	**14.27 %**	**16.67 %**
3	2	**+/- 4**	**9.05 %**	**10.00 %**
4	2	**+/- 5**	**6.60 %**	**7.14 %**
5	2	**+/- 6**	**5.32 %**	**5.56 %**
5	4	**+/- 7**	**4.32 %**	**5.21 %**
9	6	**+/- 12**	**2.05 %**	**--**
16	16	**+/- 24**	**0.80 %**	**--**
32	32	**+/- 48**	**0.31 %**	**--**
64	64	**+/- 96**	**0.11 %**	**--**

Table 6. 1 Proposed DC-balancer's overhead

An overhead comparison is given in figure 6.7

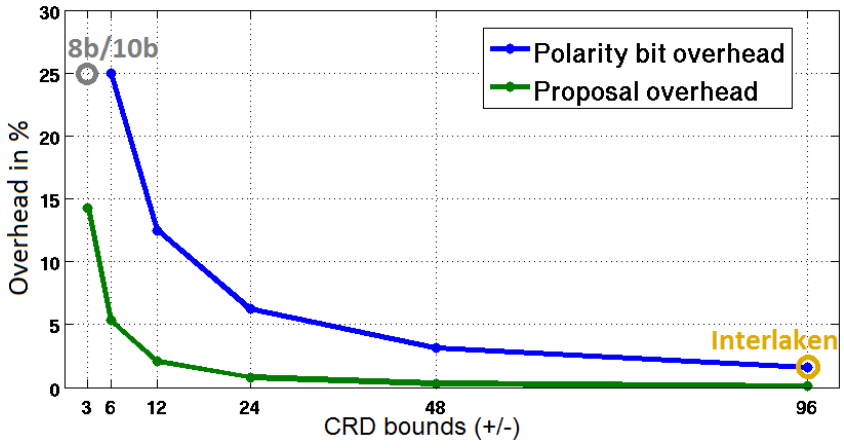


Figure 6.7 Proposal's overhead (green) compared to the polarity-bit encoding (blue), 8b/10b encoding and Interlaken's protocol

As we can see, the overhead due to our proposal is very advantageous when compared to the polarity-bit encoding and other encodings. For a CRD bounded to +/- 3, we have more than 10% overhead reduction compared to 8b/10b encoding. If we release the constraints of the RD, we can still bound the CRD to low values with only few percent of overhead or even less than 1%. When compared to the Interlaken's protocol which adds 1.56% to bound the CRD to +/- 96, we can obtain the same bound with only 0.11% overhead which is more than 10x lower.

We note that the values of T and its corresponding S were chosen based on the combinations that give the lowest overhead according to simulation.

6.3.2 *Deducing the Overhead's equation*

To allow designers to have an estimation of the overhead for specific RD bounds without referring to simulation themselves, we deduced the equation of the proposal's overhead by plotting on Microsoft Excel a chart from the simulation values in table 6.1. We then draw the "Trendline" and use the "Display equation on chart" option. The equation is then generated as we can see in figure 6.8.

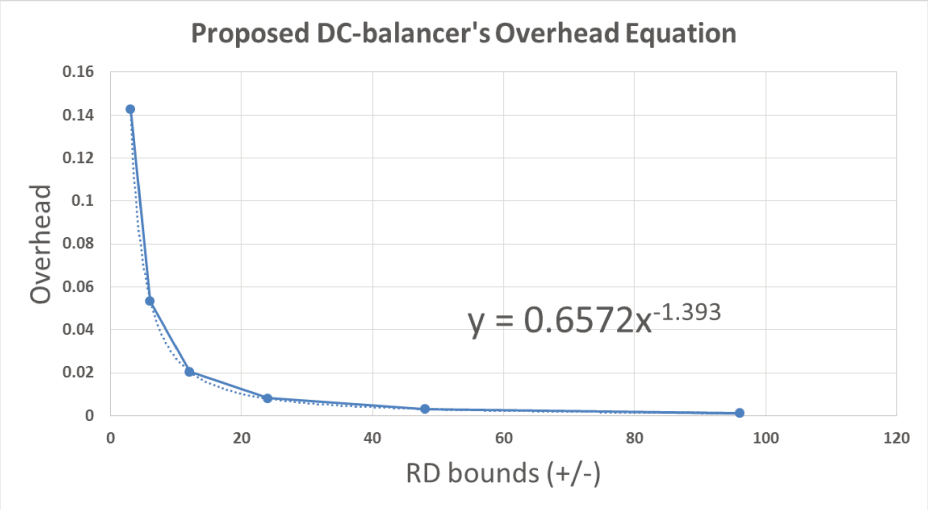


Figure 6.8 Excel representation of the overhead and equation generation

The relation between the RD bounds and its corresponding Overhead (OH) is displayed in figure 6.8. In other terms, it could be written as follows:

$$OH \approx 0.66*|RD_{bounds}|^{-1.39} \qquad (6.3)$$

An important condition for equation 6.3 to work properly is that T and S values must be chosen to provide the lowest overhead. As mentioned earlier, this could be done by simulation.

6.4 *Chapter's Conclusion*

Polarity-bit coding is a low overhead method which bounds the Running Disparity. However for small RD bounds, this method has a very high overhead that exceeds 8b/10b encoding's overhead.

In this chapter, we proposed a novel line coding which is able to bound the RD with low overhead even for small RD bounds. The presented method is based on aperiodic frames inversion, when necessary. The overhead simulations and the theoretical overhead have shown to be very low when compared to other existing line coding methods which bound the Running Disparity.

As we saw in chapter 5, low overhead could enable lane count reduction (up to 50% saving in power, area and complexity) or bandwidth increase for better performance.

Other advantages are the feature of the proposed DC-balanced encoding:

- Scalability: the RD bounds could be chosen according to the application's requirements
- Early errors detection: an error could be detected whenever the RD exceeds the bounds

- Reduce the analog complexity: no (or less) filters will be needed to correct the baseline wander

We shall note again that the Run Length is automatically bounded with our solution, but the RL bound depends on the RD bounds and is not scalable. In the next chapter we propose a scalable solution.

We note as well that the variable data length due to this proposal can be problematic to the PHY layer's framing, a proposal to variable length data is added in **Annex G.**

Chapter

7 DC-balanced and Run Length Limited Line Coding

7.1 *Chapter's Introduction*

In chapter 5, we proposed a low overhead method which bounds the Run Length (RL) to the desired value. In chapter 6 we proposed a low overhead method which bounds the Running Disparity (RD) to the desired value. As we showed earlier, chapter 6 method bounds the RL as well, but the RL bound depends on the RD bound which might not be enough. Bounding the RD to +/- 10 for example will bound the RL to 20 which could be considered a high value.

This chapter's purpose is to propose a line coding that enables choosing the desired bounds for the RD as well as for the RL by merging both methods (of chapter 5 and 6) together with some modification.

7.2 Merging possibilities

7.2.1 Reminder of the methods of chapters 5 and 6

The methods are summarized in the block diagrams in figure 7.1.

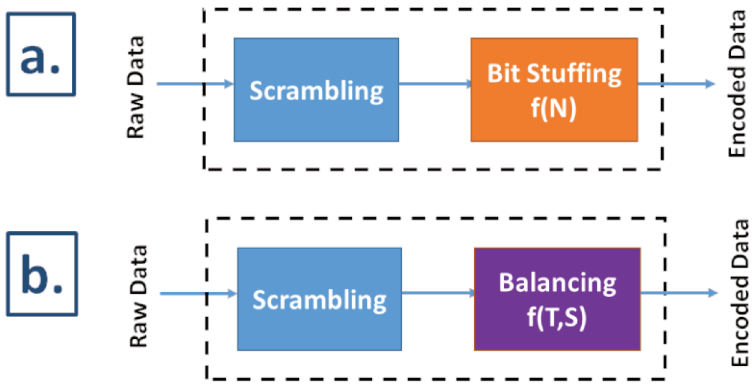


Figure 7.1 Block diagrams of the methods presented in a. chapter 5, and b. chapter 6

7.2.2 Merging Possibilities

Analysis show that if the two methods are put together, one will disrupt the other; i.e. if the balancing is applied after BS (scrambling→ BS → Balancing), the BS will be disrupted due to bit inversion and the result will be more consecutive bits than the BS has ensured. If BS is applied after balancing (scrambling→ Balancing → BS), balancing will be disrupted by the BS. For example, if the BS adds more 1's than 0's after the balancing, the RD will end up diverging. To merge the two methods, a modification should be done.

7.3 Proposal for a DC-balanced and RL limited encoding

7.3.1 *Proposal's block diagram*

To make both methods work together, the solution we propose is to scramble, do the balancing presented in chapter 6, then apply a **Modified** Bit Stuffing (MBS). A block diagram is shown in Figure 7.2.

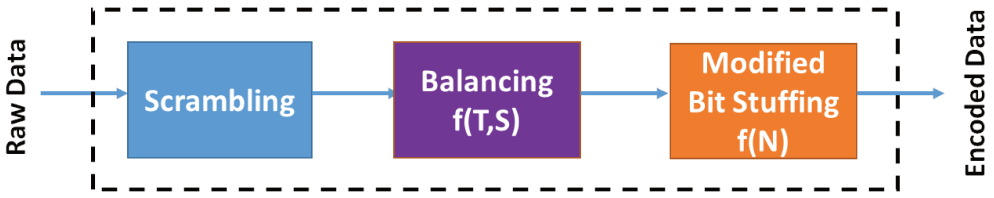


Figure 7.2 DC-balancer and RL limiter's block diagram

7.3.2 The Modified Bit Stuffing

Instead of adding a '0' bit after a run of N consecutive 1's or a '1' bit after N consecutive 0's, we will add '01' after N consecutive 1's and '10' after N consecutive 0's. The balancing will not be disrupted because the added RD is zero.

If we consider a pattern that has been scrambled and then balanced to a bound of +/-6, when we apply the modified bit stuffing for N = 5, the CRD bound won't change, not even by 1 unit. To exceed the limit of the CRD to +7 for example, we should have a CRD to +6 and then do the MBS by adding '10' so that the CRD can go to +7 when we add the '1' bit. However, to have a CRD to +6, and MBS by '10', it should happen with run of 5 consecutive 0's. In this case, it is impossible to have a CRD at +6. Thereby, the MBS will not change the CRD bounds of the balancing.

We note that the modified bit stuffing we propose can be applied on any balanced data to ensure transitions and without disrupting the Running Disparity (it can be added for example after a standard polarity-bit coding).

7.3.3 *Proposal's Overhead*

The Modified Bit Stuffing (MBS) adds two bits instead of one for the standard bit stuffing procedure. The MBS Overhead (MBSO) should normally, if applied immediately after scrambling, be twice the overhead of the standard bit stuffing presented in table 5.1.

However, the MBS is applied after balancing the data, and balancing the data creates transitions and bounds the RL to a value that is RD_{bounds} dependent as we saw in chapter 6. The MBSO depends then also on the RD_{bounds} ensured by the balancing block. Some examples are illustrated in table 7.1 below and more overhead results will be presented in chapter 8.

Balancing			BO (Balancing's Overhead)	**MBS RL Bound (Modified Bit Stuffing)**	MBSO (Modified Bit Stuffing's Overhead)	**TO (Total Overhead)**
T	**S**	**RD Bounds**				
2	2	**+/- 3**	14.27 %	**5**	3.13 %	**17.4 % ***
3	2	**+/-4**	9.05 %	**6**	1.65 %	**10.7 %**
5	2	**+/- 6**	5.32 %	**5**	5.43 %	**10.75 %**
7	6	**+/- 10**	2.66 %	**10**	0.11 %	**2.77 %**
15	10	**+/-20**	1.03 %	**8**	0.71 %	**1.75 %**
64	64	**+/- 96**	0.11 %	**7**	1.56 %	**1.67 %**

(*) equivalent to 8b10b in RL and RD bounds

Table 7.1 DC-balanced and RL-limited line coding's overhead examples

As we can see in table 7.1, to have the same equivalent of 8b10b encoding in RL and RD bounds, we get an overhead of 17.4% whereas 8b10b encoding has 25% overhead, which is more than 7% overhead reduction.

If we release the constraints on the RL and/or the RD bounds, we can have a much lower overhead.

7.3.4 Power Spectral Density Aspect

To verify that the presented solution does not harm the randomization aspect given by scrambling, we plot the PSD of the V_{cm} generated by encoding the data according to our proposal in figure 7.3 and we compare it with scrambling-only. We can clearly see that the PSD plots are very similar. The proposed DC-balanced and RL-limited line coding does not eliminate the random aspect.

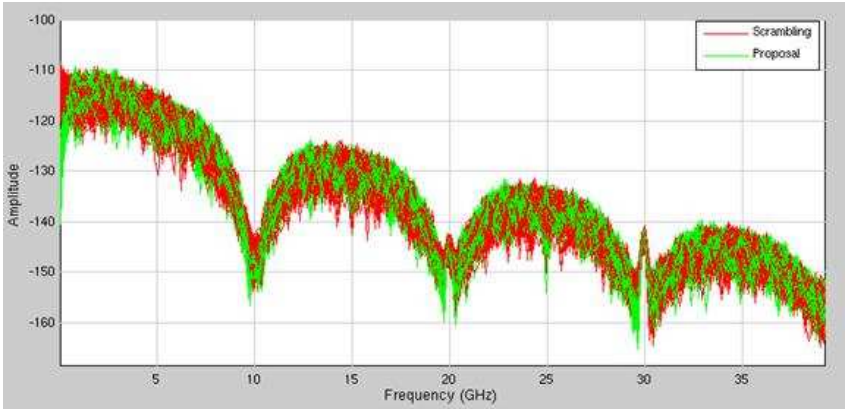


Figure 7.3 PSD of the V_{cm} of the proposed solution vs. scrambling's PSD at 10 GHz frequency

7.4 *Chapter's Conclusion*

In this chapter we presented a novel Low Overhead, Run Length Limited and DC-balanced line coding methodology.

The presented line coding has 7% less overhead than 8b/10b encoding's overhead for the same RD and RL bounds. If we release the constraints on the RD and the RD bounds, the overhead of the proposed encoding drastically decreases.

In addition to its low overhead characteristic, the presented method offers scalability; the RD and RL bounds are completely programmable and adaptive. A transmitter can encode the data according to the receiver's RL and RD requirements.

We note that the variable data length due to this proposal can be problematic to the PHY layer's framing, a proposal to variable length data is added in **Annex G.**

Chapter

8 Experimental Results

8.1 *Chapter's Introduction*

This chapter's purpose is to show the simulation results of the four methods we presented in chapters 4, 5, 6 and 7 which are summarized in table 8.1.

Method 1: Double Scrambling ($Max_{repetition}$)
Purpose: eliminate data repetition for low EMI
Method 2: Scrambling + Bit Stuffing(RL_{bound})
Purpose: bound the run length for clock and data recovery
Method 3: Scrambling + Balancing(RD_{bounds})
Purpose: bound the running disparity to reduce baseline wander
Method 4: Scrambling+Balancing(RD_{bounds})+Modified Bit Stuffing(RL_{bound})
Purpose: bound both the run length and the running disparity

Table 8.1 Summary of the encoding methods presented in this thesis

The rest of this chapter is organized as follows:

In paragraph 8.2, we highlight the peaks reduction in the Power Spectral Density (PSD) of the common mode voltage (V_{cm}) thanks to double scrambling (method 1).

In paragraph 8.3, we give more overhead simulation results for the "scrambling + bit stuffing" (method 2) and for "scrambling + balancing + bit stuffing" (method 4).

In paragraph 8.4, we give gate count hardware estimation of the proposed methods based on a VHDL model we designed.

In paragraph 8.5 we show the different eye diagrams for methods 2, 3 and 4 based on Matlab/Simulink simulation using the S-parameters of a DC-coupled channel and an AC-coupled channel. We then highlight the efficiency of the proposed methods.

Paragraph 8.6 summarizes this chapter.

We note that every simulation in this chapter that includes scrambling is done with the following LFSR polynomial:

$G(X) = X^{23} + X^{21} + X^{16} + X^8 + X^5 + X^2 + 1$ with seed value 1D-BFBCh.

The 2nd scrambling polynomial used for the simulations of the proposed low EMI method is:

$G'(X) = X^{16} + X^5 + X^4 + X^3 + 1$ with seed value 1FFFFh.

8.2 Double scrambling (method 1) PSD simulation

As we saw in paragraph 2.3.2, redundancy and repetitive patterns have a direct impact on the Power Spectral Density (PSD) of the V_{cm}, which is a

primary contributor to EMI. There are other factors that affect the PSD of the V_{cm}, such as the rise time, the fall time, the timing mismatch between the differential positive (D_p) and the differential negative (D_n), and the voltage mismatch between D_p and D_n.

For the simulations in this paragraph, we will focus on the repetitions effect only. The rise and fall time, the timing and voltage mismatch are the same for all the simulations. The window function used is the Hamming Window.

1st Simulation

We consider a packet of length 56 bytes in which a byte (F0h) is being repeated 16 times in a row (EMI killer packet) as we can see in figure 8.1.a. After applying the "double scrambling" (method 1) for a repetition threshold M = 8 and for a symbol of length L = 8 bits (byte), we get the pattern in figure 8.1.b. In figure 8.2 we plot the PSD of pattern a. and pattern b.

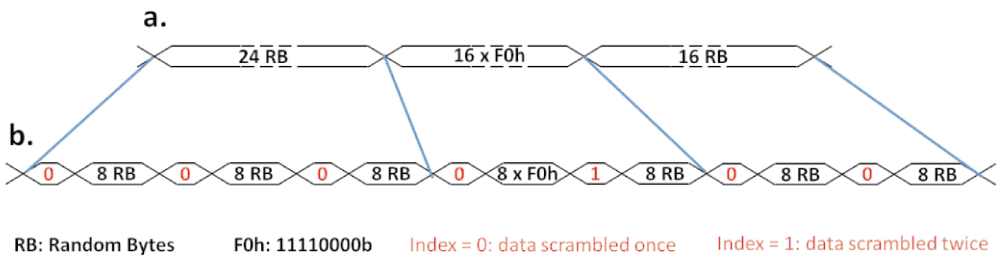


Figure 8.1 EMI killer packet before and after applying the "double scrambling" method

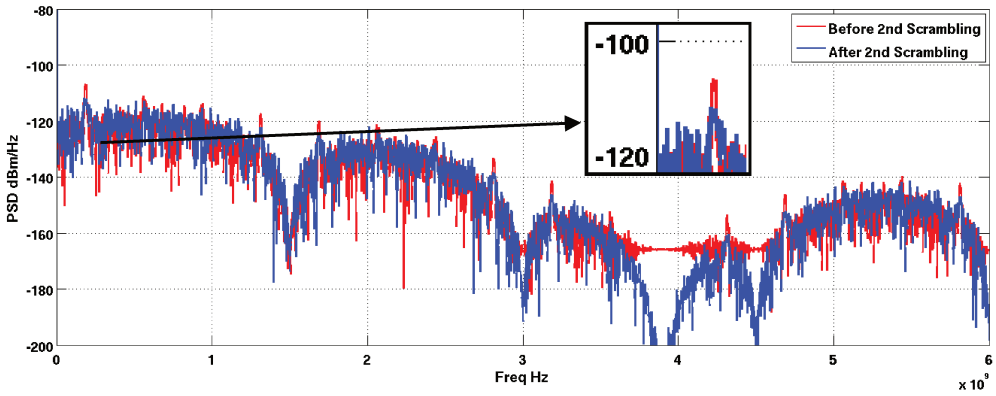


Figure 8.2 PSD of an EMI killer packet before and after applying the "double scrambling" method (slew rate = 50% of UI, time shift = 3% of UI, voltage mismatch between D_p and D_n 5% of swing)

In figure 8.2, we can see that the peaks (in red) before applying the proposed "double scrambling" method (method 1) have been reduced by almost 5 dBm/Hz after applying the proposed method (PSD in blue).

2nd Simulation

We consider a packet of length 56 bytes in which a byte (F0h) is being repeated 32 times in a row (EMI killer packet) as we can see in figure 8.3.a. After applying the "double scrambling" method (method 1) for a repetition threshold M = 8 and for a symbol of length L = 8 bits (byte), we get the pattern in figure 8.3.b.

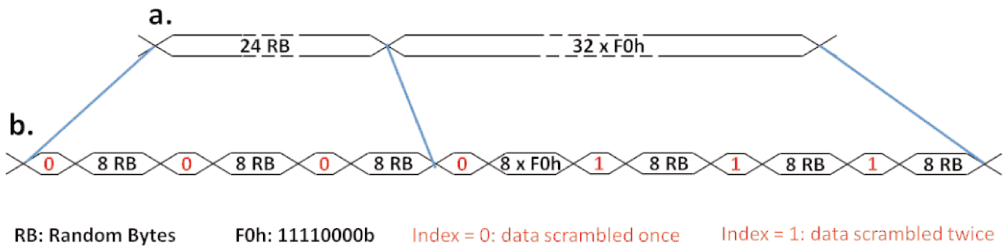


Figure 8.3 EMI killer packet before and after applying the "double scrambling method"

In figure 8.4 we plot the PSD of pattern a. and pattern b.

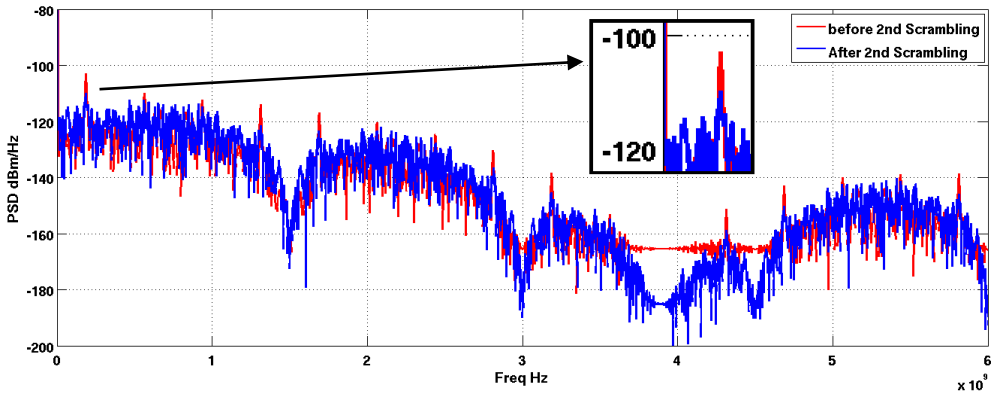


Figure 8.4 PSD of an EMI killer packet before and after applying the "double scrambling" method (slew rate = 50% of UI, time shift = 3% of UI, voltage mismatch between D_p and D_n 5% of swing)

In figure 8.4, we can see that the peaks (in red) before applying the "double scrambling" method (method 1) have been reduced by almost 10 dBm/Hz after applying the proposed method (PSD in blue).

Conclusion

Reducing the repetitions has an obvious positive effect on the power spectral density of the common mode voltage. Thanks to the "double scrambling" method (method 1), we can reduce the peaks of an EMI killer packet by about 10 dBm/Hz.

8.3 More overhead simulation results

8.3.1 Scrambling + bit stuffing (method 2) overhead simulation

In chapter 5 we presented the “scrambling + bit stuffing” method (method 2), we calculated the theoretical overhead and compared it with a simulation on picture data. The picture’s data had a specific distribution of 1’s and 0’s and we wish to make a simulation on different data distribution.

On Matlab, we generate frames with different distribution of 1’s and 0’s using the “rand” function. For each distribution, 200 frames of 2048 bits each are generated. We encode the generated frames using bit-stuffing only and then using the “scrambling + bit stuffing” method (method 2) we proposed, we calculate the overhead for each case and averaging is then made. Figure 8.5.a. shows the overhead of the bit-stuffing only and figure 8.5.b shows the overhead of our proposal (bit stuffing after scrambling).

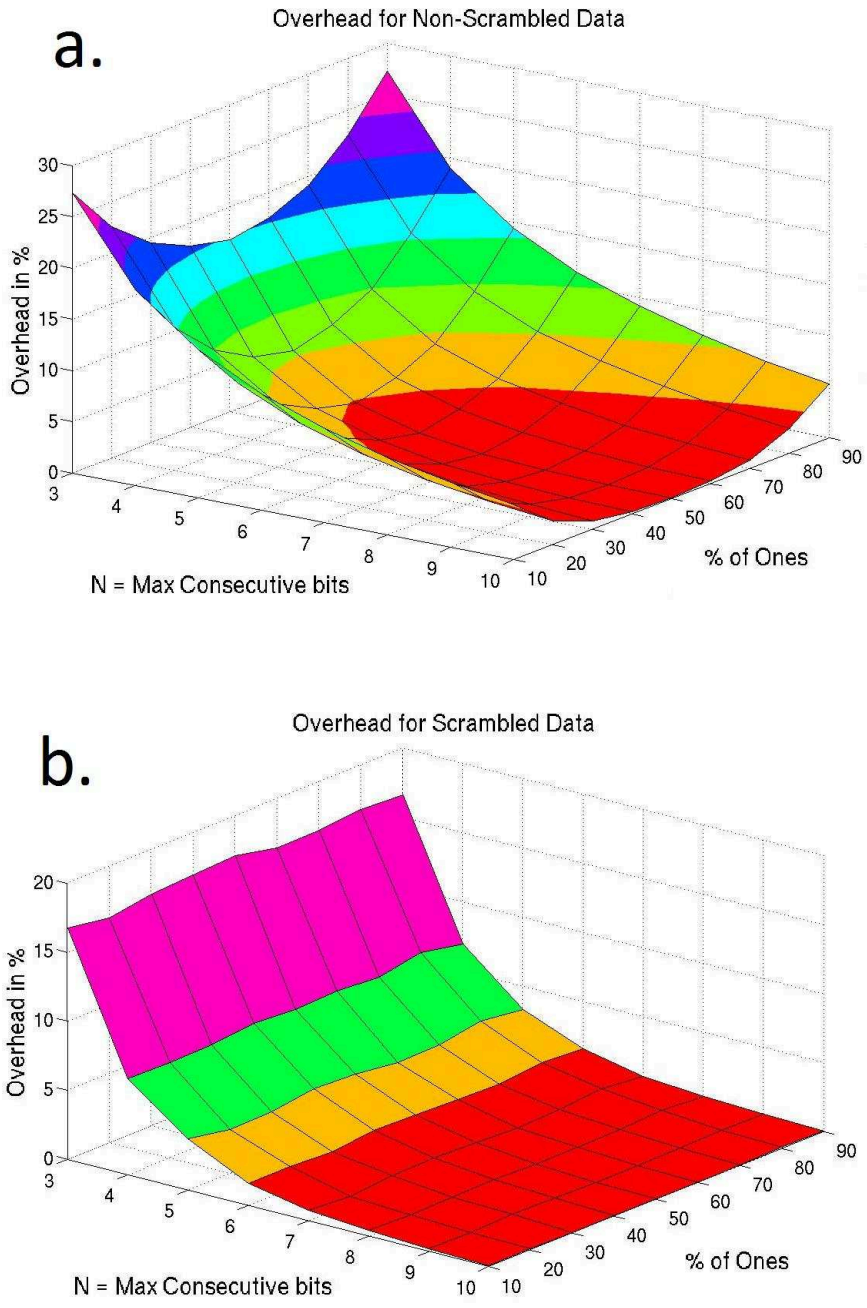


Figure 8.5 Bit Stuffing Overhead for: a. Non-Scrambled data / b. Scrambled data

In figure 8.5.a, we can see that the overhead is distribution-dependent and very similar to the theoretical graph in figure 5.3. When the data is scrambled, the bit stuffing's overhead is independent from the data's 1's and 0's

distribution and is very low as we can see in figure 8.5.b. The exact values are added to the ones in table 5.1 and are merged in table 8.2 as follows:

N	3	4	5	6	7	8	9	10
Theory	14,29 %	6.67 %	3.23 %	1,59 %	0,79 %	0,39 %	0,20 %	0,10 %
Image	16.65 %	7.13 %	3.33 %	1.61 %	0.79 %	0.39 %	0.19 %	0.09 %
Random	17.11 %	7.06 %	3.49 %	1.67 %	0.76 %	0.31 %	0.10 %	0.05 %

Table 8. 2 "scrambling + bit stuffing" method theoretical, image and random data's overhead

8.3.2 Scrambling + balancing + modified bit stuffing (method 4) overhead simulation

The "scrambling + balancing + modified bit stuffing" method (method 4) is constituted of 2 blocks which adds overhead: the Balancing block and the Modified Bit Stuffing (MBS) block. The Total Overhead (TO) could then be written as follows:

$$TO = BO + MBSO \qquad (8.1)$$

Where BO is the Balancing block's overhead

And MBSO is the MBS block's overhead

The values of the BO where presented in table 6.1 (not exhaustive) and some values of the MBSO and the TO were presented in table 7.1. The MBSO is RD_{bounds}-dependent (because of the balancing's block) and of course, RL_{bounds} -dependent (the N value at which the modified bit stuffing is executed). More detailed MBSO values as a function of the RD_{bounds} and RL_{bounds} are presented in table 8.3.

The Total overhead as a function of the RD_{bounds} and RL_{bounds} are presented in table 8.4.

RL bound / RD bound	3	4	5	6	7	8	9	10
+/- 3	31.6	10.75	3.10	0.43	0	0	0	0
+/- 4	32.35	11.85	4.57	1.71	0.49	0.07	0	0
+/- 5	32.54	12.39	5.08	2.07	0.79	0.28	0.07	0.08
+/- 6	32.66	12.85	5.41	2.31	1.01	0.41	0.16	0.05
+/- 7	33.45	13.58	5.89	2.53	1.05	0.45	0.17	0.05
+/- 8	33.41	13.74	6.00	2.65	1.17	0.52	0.22	0.09
+/- 9	33.42	13.84	6.10	2.73	1.22	0.55	0.24	0.09
+/- 10	33.56	14.04	6.24	2.82	1.27	0.58	0.26	0.11
+/- 15	33.56	14.23	6.52	3.00	1.42	0.68	0.32	0.13
+/- 20	33.47	14.23	6.56	3.08	1.47	0.71	0.34	0.15
+/- 40	33.39	14.32	6.64	3.167	1.55	0.77	0.38	0.18
+/- 60	33.37	14.31	6.65	3.17	1.56	0.77	0.38	0.18
+/- 96	33.35	14.30	6.65	3.18	1.57	0.79	0.39	0.19

Table 8.3 Modified Bit Stuffing Overhead (MBSO) in % for different RD and RL bounds / MBSO = f(RDbound, RLbound)

RL bound / RD bound	3	4	5	6	7	8	9	10
+/- 3	45.87	25.02	17.37	14.70	14.27	14.27	14.27	14.27
+/- 4	41.40	20.90	13.62	10.77	9.55	9.12	9.05	9.05
+/- 5	39.14	18.99	11.68	8.67	7.39	6.88	6.68	6.68
+/- 6	37.98	18.18	10.73	7.64	6.33	5.73	5.48	5.37
+/- 7	37.77	17.91	10.21	6.85	5.37	4.77	4.49	4.38
+/- 8	37.05	17.38	9.65	6.30	4.81	4.16	3.86	3.73
+/- 9	36.46	16.89	9.15	5.78	4.27	3.60	3.28	3.14
+/- 10	36.22	16.70	8.90	5.49	3.94	3.25	2.92	2.77
+/- 15	35.07	15.74	8.02	4.51	2.93	2.18	1.83	1.64
+/- 20	34.47	15.24	7.56	4.08	2.48	1.72	1.35	1.15
+/- 40	33.80	14.72	7.05	3.57	1.96	1.18	0.78	0.59
+/- 60	33.57	14.52	6.86	3.38	1.77	0.98	0.58	0.38
+/- 96	33.46	14.42	6.77	3.30	1.69	0.90	0.51	0.30

Table 8.4 Total Overhead in % for different RL and RD bounds / TO = f(RDbound, RLbound)

8.4 VHDL model and gate-count estimation

To make an estimation of the hardware complexity of "scrambling + bit stuffing" method (method 2), we make a VHDL (VHSIC Hardware Description Language) model of the bit stuffing. We consider scrambling's cost is excluded because it is becoming more and more mandatory in standards for EMI issues.

An example of the implementation is shown in figure 8.6.

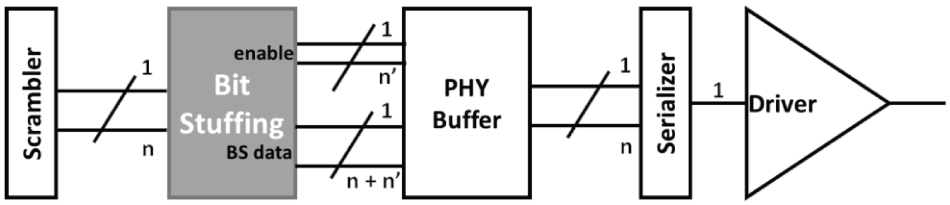


Figure 8.6 Bit Stuffing PHY Hardware implementation example

The data comes in parallel at the input of the bit stuffing block. The parallel data bus width is n, which is generally 8, 16 or 32 bits. At the output of the bit stuffing block, a "BS data" bus corresponds to the bit-stuffed data. The "BS data" bus width is n+n' and n' corresponds to the maximum number of bits that could be added by the bit stuffing. i.e. for n = 8 and bit stuffing of 5 consecutive identical bits (N=5), a maximum of 2 bits could be added to the 8 bits. n' = 2 in this case. At the output of the bit stuffing we add an "enable" bus of width n' to indicate to the PHY buffer which bits between the n' added bits are valid. According to this information, the PHY buffer will buffer the valid bits and then forwards them to the serializer.

According to our model, the gate count for different bus widths are the following:

Bus width	Gate count
8 bits	340 Gates
16 bits	880 Gates
32 bits	3000 Gates

Table 8.5 Gate count estimation of the bit stuffing block for different bus width

We can see the small gate count of the proposed solution. With the increased hardware complexity of today's chips, few hundreds of gates are considered negligible.

"Scrambling + balancing" (method 3) and "scrambling + balancing + bit stuffing" (method 4) are estimated to have a hardware complexity of the same order of magnitude as "scrambling + bit stuffing" (method 2).

8.5 Eye diagrams results and comparison

8.5.1 Eye diagrams on DC-coupled channel

In this section, we simulate on Matlab/Simulink using the S-parameters of a DC-coupled PCB (Printed Circuit Board) channel, data being encoded with different encoding methods. The data distribution used for this simulation is 80% of 0's and 20% of 1's. At first, we show in figure 8.7 the eye diagram of non-encoded data vs. 8b10b encoded data's eye at 10 GHz. The non-encoded data's eye is completely shifted from the baseline because of the non-balanced data distribution. It is considered closed.

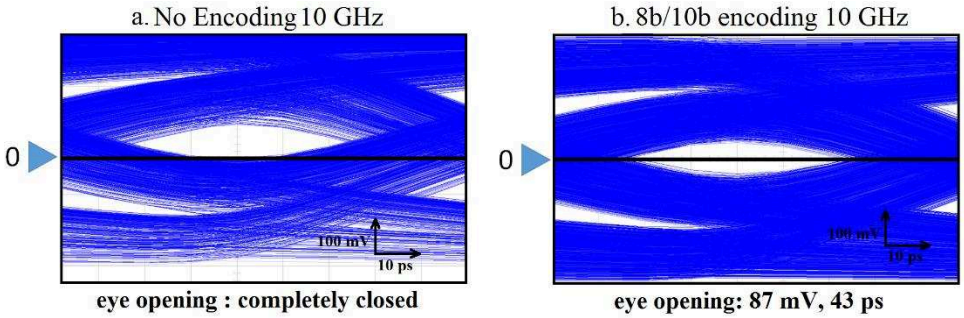


Figure 8.7 Eye diagrams on the receiver's side for a simulation of 10 Kbits on a DC-coupled channel without equalization, 800 mV transmitter swing for: a. data non-encoded at 10GHz / b. data 8b/10b encoded at 10 GHz

From figure 8.7, we can see the interest of line coding on the eye diagram.

Now we wish to plot the eye diagrams for the "scambling + bit stuffing" (method 2) for $RL_{bound} = 5$ (or N = 5, same bound ensured by 8b/10b encoding) and compare it with 8b/10b encoding. For this purpose, we make two comparisons:

- **Comparison 1**: eye diagrams comparison for a **same link frequency** of 10 GHz. In this case, 8b/10b throughput is 8 Gbps (using equation 5.2) whereas "scrambling + bit Stuffing" (method 2) throughput is 9.66 GHz (corresponds to 3.5% overhead for N = 5)
- **Comparison 2**: eye diagrams comparison for the **same target throughput** of 8 Gbps. In this case, the link's frequency when using 8b/10b encoding should be 10 GHz whereas when using "scrambling + bit Stuffing" (method 2) for $RL_{bound} = 5$, the frequency of the link should be 8.28 GHz

The eye diagrams are illustrated in figure 8.8 as follows:

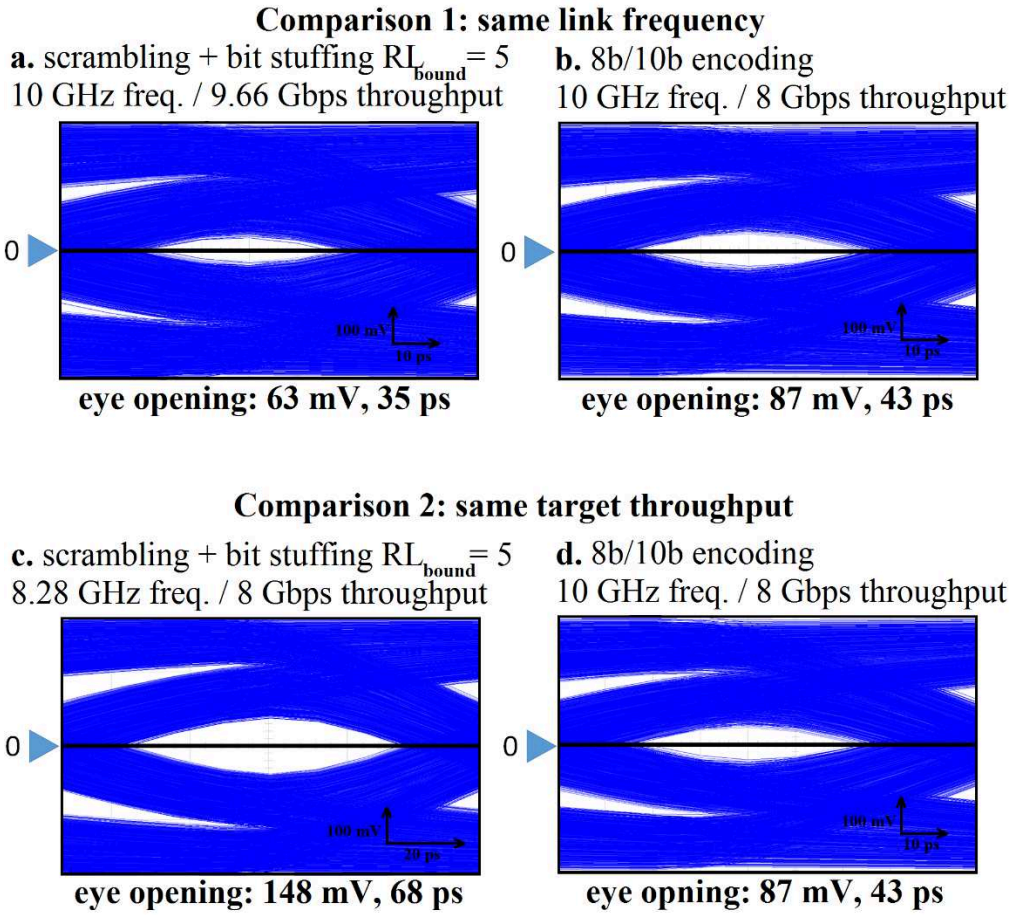


Figure 8.8 Eye diagrams on the receiver's side for a simulation of 10 Kbits on a DC-coupled channel without equalization, 800 mV transmitter swing for: a. data encoded with method 2 at 10GHz / b. data 8b/10b encoded at 10 GHz / c. data encoded with method 2 at 8.28 GHz / d. data 8b/10b encoded at 10 GHz

From figure 8.8, we can see that "scrambling + bit Stuffing" (method 2) gives an eye opening centered at the baseline (due to scrambling's effect) but it is less opened than 8b/10b encoded data's eye at the same frequency. In this case, 8b/10b's better eye comes at the cost of lower throughout (1.66 Gbps less than method 2 throughout). For the same target throughout, method 2 gives the best eye opening.

Conclusion: for DC-coupled channels, using "scrambling + bit Stuffing" (method 2) could be better than using 8b/10b encoding.

8.5.2 Eye diagrams on AC-coupled channel

In this section, we simulate on Matlab/Simulink using the S-parameters of an AC-coupled PCB (Printed Circuit Board) channel having a coupling capacitor of 5 pF, data being encoded with different encoding methods. The data distribution used for this simulation is 80% of 0's and 20% of 1's.

We make 3 comparisons:

- **Comparison 1: "scrambling + bit Stuffing" (method 2)** for RL_{bound} = 5 (same RL bound as 8b/10b encoding) **vs. 8b/10b** encoding at the **same target throughput** of 8 Gbps. Method 2 runs at 8.28 GHz (using equation 5.2) and 8b/10b runs at 10 GHz (using equation 5.2).
- **Comparison 2: "scrambling + balancing + modified bit Stuffing" (method 4)** for RD_{bounds} = +/-3 and RL_{bound} = 5 (same RD and RL bounds ensured by 8b/10b encoding) **vs. 8b/10b** encoding at the **same frequency** of 10 GHz. Method 4 throughput is 8.5 Gbps (corresponding to 17.4% overhead and using equation 5.2) whereas 8b/10b throughput is 8 Gbps (corresponding to 25% overhead).
- **Comparison 3: "scrambling + balancing + modified bit Stuffing" (method 4) for RD_{bounds} = +/-3 and RL_{bound} = 5** (same RD and RL bounds ensured by 8b/10b encoding) **vs. 8b/10b** encoding at the **same target throughput** of 8 Gbps. Method 4 runs at 9.3 GHz (corresponding to 17.4% overhead and using equation 5.2) whereas 8b/10b runs at 10 GHz (corresponding to 25% overhead).

The eye diagram results of this comparison are illustrated in figure 8.9 as follows:

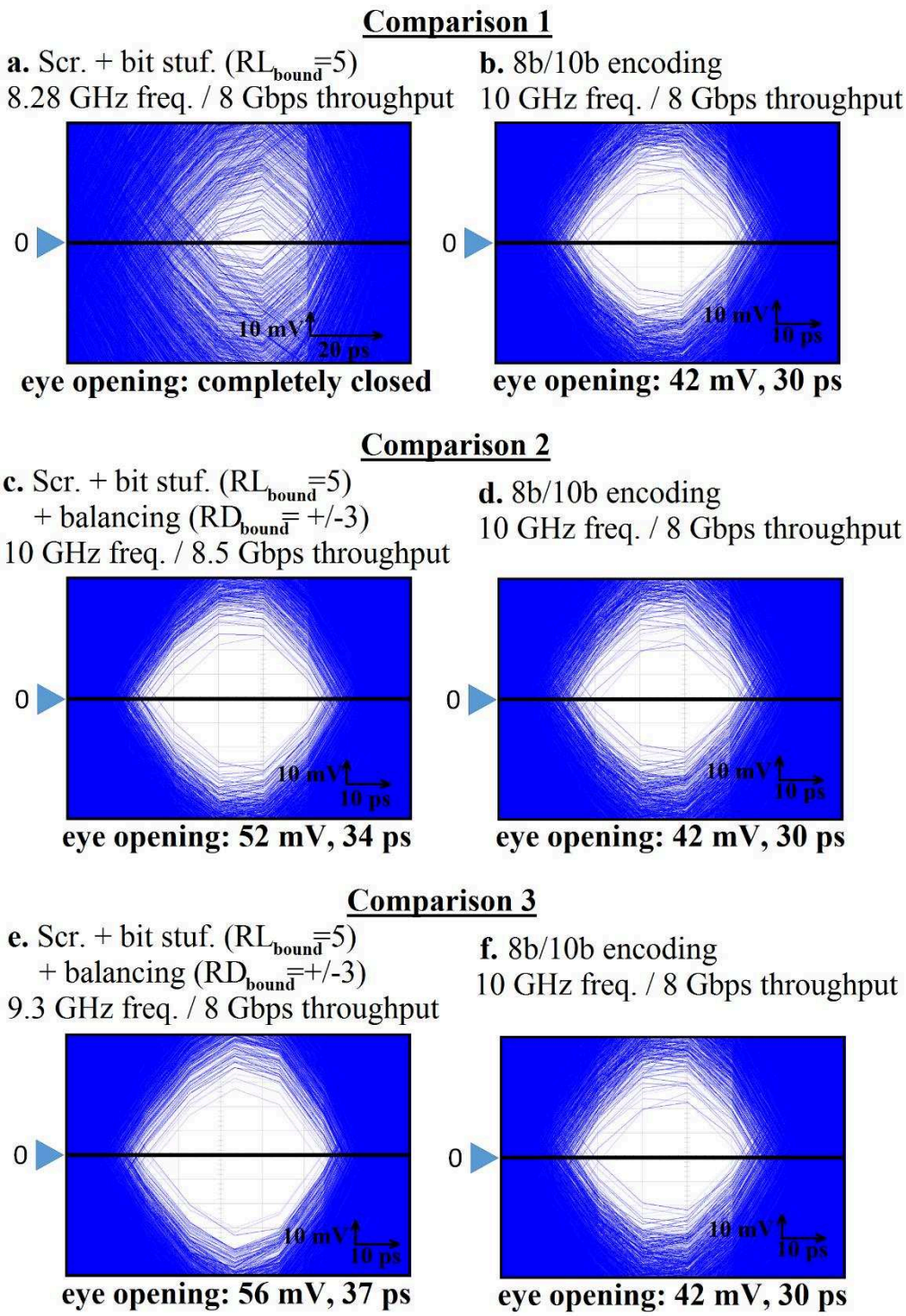


Figure 8.9 Eye diagrams on the receiver's side for a simulation of 400 Kbits on a AC-coupled channel (C = 5pF and R = 50 Ω), 800 mV transmitter swing for: a. data encoded with method 2 at 8.28GHz / b. data 8b/10b encoded at 10 GHz / c. data encoded with method 4 at 10 GHz / d. data 8b/10b encoded at 10 GHz / e. data encoded with method 4 at 9.3 GHz / f. data 8b/10b encoded at 10 GHz

From figure 8.9, we can see from comparison 1 that “scrambling + bit Stuffing” (method 2) might not be enough when using an AC-coupled channel because the Running Disparity for this proposal is not bounded. “Scrambling + balancing + modified bit Stuffing” (method 4), with RD bounded to +/-3 and RL to 5 has almost the same eye opening as 8b/10b encoded data at the same frequency and with a better throughput. For the same target throughout, “Scrambling + balancing + modified bit Stuffing” (method 4) offers the best eye opening.

8.6 *Chapter’s conclusion*

In this chapter we showed the positive effect of the “double scrambling encoding” presented in chapter 4 on the PSD of the common mode voltage, which means EMI reduction.

We also presented more overhead simulation results for the “scrambling + bit stuffing” line coding and the “scrambling + balancing + bit stuffing” line coding presented in chapters 5, 6 and 7.

We made a VHDL model for the “scrambling + bit stuffing” line coding and showed the low hardware overhead and complexity of the presented solution. The “Scrambling + balancing” (method 3) and “scrambling + balancing + bit stuffing” (method 4) are estimated to have a hardware complexity of the same order of magnitude as “scrambling + bit stuffing” (method 2).

We made eye-diagrams simulations on DC-coupled and AC-coupled channels and made a comparison with 8b/10b encoding and verified that the solutions we presented performed well and meet our expectations in terms of eye diagram opening.

Chapter

9 Conclusion

High Speed Serial Links (HSSLs) are major actors in mobile devices and networking, and their bandwidth is still facing an exponential increase to satisfy the users' requirements. Line coding is a very important step when designing a HSSL because it has a direct impact on the bandwidth efficiency and on the data transmission over the link as we showed in the problem statement chapter. The line coding must help in reducing EMI, the Run Length (RL) and the Running Disparity (RD) while having the lowest possible bandwidth overhead.

In the state of the art's chapter, we overviewed the bit stuffing which is one of the most optimized RL-limited line coding methods and we showed its drawbacks. Bit stuffing's overhead is data-dependent and can reach high values when the data has a specific distribution. We then overviewed the 8b/10b encoding which is a widely used data coding because it ensures a RL bounded to 5 and a RD bounded to +/- 3, however, at the cost of 25% bandwidth overhead. We also showed that data scrambling has good characteristics in randomizing, creating transitions and reducing the RD of the raw data, but scrambling does not ensure any bounds for both the RL and the RD, nor randomization. We finally showed that the polarity-bit coding can offer a bounded RD at a low overhead cost. However, for small RD bounds, the polarity-bit coding's overhead is very high and becomes less competitive compared to 8b/10b encoding for example.

In this thesis, we proposed 4 novel encoding methods.

In chapter 4, we proposed a reduced EMI method (**double scrambling**, method **1**) that ensures the elimination of repetitive sequences (that are the cause of data-dependent EMI) by re-scrambling repetitive packets after the first scrambling block. The repetitive packets selection is mandatory to ensure the good functioning of the method.

In chapter 5, we showed that scrambling before bit stuffing can reduce the bit stuffing overhead to its minimum value and make the overhead predictable, independent of the raw data's distribution. So we proposed a low overhead RL-limited line coding (**Scrambling + bit stuffing**, method **2**) that has a low overhead down to 3.5% for a maximum RL of 5, the same as 8b/10b encoding's RL bound which comes at the cost of 25% bandwidth overhead. The proposed line coding offers scalability; the RL-bound can be programmable based on the CDR (Clock and Data Recovery) unit requirements. This can allow more overhead reduction, down to 0.1% for a maximum RL of 10.

In chapter 6, we proposed a low overhead DC-balanced line coding (**scrambling + balancing**, method **3**) that can bound the RD to low values, with a low overhead. This encoding is based on aperiodic frames polarity inversion after scrambling (but scrambling is not mandatory). Thanks to aperiodic frames, this method allows significant overhead reduction over the existing methods; 14.3% is the overhead necessary to limit the RD to +/- 3, whereas with 8b/10b the cost is 25% for the same RD bound. To limit the RD to +/- 96, the proposed method has an overhead of 0.11%, whereas the polarity-bit coding has an overhead of 1.56% for the same bounds. Scalability is also a feature of this method and allows choosing the desired RD limit.

The method we proposed in chapter 7 merges the methods proposed in chapters 5 and 6 to build a programmable low overhead, Run Length limited and DC-balanced line coding (**scrambling + balancing + modified bit stuffing**, method **4**). Scrambling is advised to be applied to the data first, the

balancing method of chapter 6 is then applied on the scrambled data, and finally a modified bit stuffing is applied as a final stage. The modified bit stuffing scheme was proposed to not disrupt the RD of the balanced data. This method is also programmable to the desired RD and RL bounds. For example, to limit the RL to 5 and the RD to +/- 3 which are the same equivalent of the 8b/10b encoding, the overhead is 17.4%, whereas the 8b/10b cost is 25%. If the RL and RD bounds constraints are released, we can still have decent bounds with a very low overhead.

With the multitude of the existing High Speed Serial Links (HSSLs) and the large domain of applications, the line coding presented in this thesis is perfectly adaptable to every case. And with the increasing demand for throughput, the line coding methods presented in this thesis can allow bandwidth increase for a specific link frequency. Reducing the frequency for a same target throughput could be another clever choice to make which enables reducing the power consumption, the complexity of the design, the noise etc…

Bibliography

[1] '4 milliards de smartphones et tablettes dans le monde en 2017' www.frenchweb.fr

[2] Credo Announces First 56G SerDes Technology Based on Conventional NRZ Modulation www.design-reuse.com

[3] MIPI Alliance www.mipi.org

[4] The OSI Model's Seven Layers Defined and Functions Explained https://support.microsoft.com/kb/103884

[5] J. Chandrasekhar, E. Engin, M. Swaminathan, K. Uriu and T. Yamada, "Noise Induced Jitter in Differential Signaling", 58th Electronic Components and Technology Conference (ECTC), 2008.

[6] C. Wang and J.L. Drewniak, "Quantifying the Effects on EMI and SI of Source Imbalances in Differential Signaling", IEEE International Symposium on Electromagnetic Compatibility, 2003.

[7] E. McCune and P. Lefkin, "Manage EMI from high-speed digital interfaces", January 17, 2014, www.edn.com

[8] R. Imran and M. Islam, "Industrial Modified Digital Scrambler & Descrambler System", HCTL Open Science and Technology Letters, June 2013.

[9] J. Redouté and M. Steyaert, "A CMOS Source-Buffered Differential Input Stage with High EMI Suppression", 34th European Solid-State Circuits Conference (ESSCIRC), 2008.

[10] H. Lee, "An Estimation Approach to Clock and Data Recovery", thesis dissertation, November 2006.

[11] Silicon Labs, "Jitter Attenuation – choosing the right Phase-locked Loop Bandwidth", 2010

[12] R. Leonowich, "Phase-Locked Loop System With Compensation For Data-Transition-Dependent Variations In Loop Gain", US. Patent 5,315,270, May 24, 1994.

[13] L. Devito, J. Newton, R. Croughwell, J. Bulzacchelli, F. Benkley, "A 52 MHz and 155 MHz Clock-Recovery PLL", IEEE International Solid-State Circuits Conference, 1991.

[14] T. Lee and J.F. Bulzacchelli, "A 155-MHz Clock Recovery Delay- and Phase-Locked Loop", IEEE journal of solid-state circuits, vol 27, December 1992.

[15] M. Hsieh and G.E. Sobelman, "Architectures for Multi-Gigabit Wire-Linked Clock and Data Recovery", IEEE circuits and Systems Magazine, fourth quarter 2008.

[16] H. Johnson, "When to use AC Coupling", High-Speed Digital Design Online Newsletter: Vol. 4 Issue 15, 2001.

[17] R. Lavoie, "Understanding the blocking capacitor effect on the HD/SD pathological signals", Brioconcept Application Note: AN-01(rev 0.1), 2008.

[18] "Choosing AC-Coupling Capacitors", Maxim Integrated Application Note: HFAN-1.1, rev. 1, April 2008.

[19] Y. Dong et al. "AC-Coupling Strategy for High-Speed Transceivers of 10Gpbs and Beyond", IFIP International Conference on Very Large Scale Integration, VLSI - SoC 2007.

[20] MIPI® Alliance Specification for Low Latency Interface (LLI), Revision 1.0.

[21] MIPI® Alliance Specification for Low Latency Interface (LLI), Revision 2.0.

[22] PCI Express Base Specification, Revision 2.1.

[23] PCI Express Base Specification, Revision 3.0.

[24] Universal Serial Bus 3.0 Specification, Revision 1.0.

[25] J. Saadé, F. Pétrot, A. Picco, J. Huloux, A. Goulahsen, "A System-level Overview and Comparison of Three High-Speed Serial Links: USB 3.0, PCI Express 2.0 and LLI 1.0", IEEE 16th symposium on Design and Diagnostic of Electronic Circuits and Systems, DDECS 2013.

[26] D. Miller, P. Watts, A. Moore, “Motivating future interconnects: a differential measurement analysis of PCI latency”, Proceedings of the 5th ACM/IEEE Symposium on Architectures for Networking and Communications Systems, 2009.

[27] Universal Serial Bus Specification, Revision 2.0

[28] P.A. Franaszek and A.X. Widmer, “Byte oriented DC balanced 8B/10B partitioned block transmission code”, U.S. Patent 4 486 739, December 4, 1984.

[29] P.A. Franszek and A.X. Widmer, “A DC-Balanced, Partitioned-Block, 8B/10B Transmission Code”, IBM Journal of research and development, Volume 27, Number 5, September 1983.

[30] H. Johnson, “Killer Packet”, High-Speed Digital Design Online Newsletter: Vol. 5 Issue 7, 2002.

[31] D.E. Knuth, “Efficient Balanced Codes”, IEEE transactions on Information Theory, vol it-32, no.1, January 1986.

[32] A. Nazemi et al., “A 2.8 mW/Gb/s Quad-Channel 8.5-11.4 Gb/s Quasi-Digital Transceiver in 28 nm CMOS”, Symposium on VLSI Circuits, 2013.

[33] W. Qian, M.D. Riedel, H. Zhou and J. Bruck, “Transforming Probabilities with Combinational Logic”, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2011.

[34] “Geometric Series”
http://classes.yale.edu/fractals/CalcTutorials/PowerSer/GeomSer/GeomSer.pdf

[35] H. Shi, V. Echevarria, W. T. Beyene and X. Yuan, “EMI Evaluation of a Differential Signaling Interconnect at 3.2 Gbps”, IEEE 14th Topical Meeting on Electrical Performance of Electronic Packaging, 2005.

[36] X. Duan, B. Archambeault, H. Bruens and C. Schuster, “EM emission of differential signals across connected printed circuit boards in the GHz range”, IEEE International Symposium on Electromagnetic Compatibility (EMC), 2009.

[37] T Matsushima, T. Watanabe, Y. Toyota, R. Koga and O. Wada, "Prediction of EMI from two-channel differential signaling system based on imbalance difference model", IEEE International Symposium on Electromagnetic Compatibility (EMC), 2010.

[38] G. Pitner et al., "EMI Sources from Mode Conversion in a Telco System High-Speed SERDES", Proceedings 60th Electronic Components and Technology Conference (ECTC), 2010.

[39] T. Koo, H. Kang; J. Ha, E. Koh and J Yook, "Signal integrity enhancement of high-speed digital interconnect with discontinuous and asymmetric structures for mobile applications", IEEE International Symposium on Electromagnetic Compatibility (EMC), 2013.

[40] M. Pajovic, J. Savic, A. Bhobe and Z. Xiaoxia, "The gigahertz two-band common-mode filter for 10-Gbit/s differential signal lines", IEEE International Symposium on Electromagnetic Compatibility (EMC), 2013.

[41] F. Michel, M. Steyaert, "Differential input topologies with immunity to electromagnetic interference", Proceedings of the 37th Solid-State Circuits Conference (ESSCIRC), 2011.

[42] S. Connor, B. Archambeault and M. Mondal, "The impact of common mode currents on signal integrity and EMI in high-speed differential data links", IEEE International Symposium on Electromagnetic Compatibility (EMC) 2008.

[43] M.H. Alser and M.M. Assaad, "Design and modeling of low-power clockless serial link for data communication systems", National Postgraduate Conference (NPC), 2011.

[44] D.G. Kam et al., "Is 25 Gb/s On-Board Signaling Viable?", IEEE Transactions on Advanced Packaging, Vol. 32, No.2, may 2009.

[45] B. Hong, C. Shin and D. Ko, "Emulation Based High-Accuracy Throughput Estimation for High-Speed Connectivities: Case Study of USB2.0", 48th Design Automation Conference (DAC), 2011.

[46] B.E. Boos, "High Speed Digital Signal Compensation on Printed Circuit Boards", thesis dissertation, January 2004.

[47] P.V.Y. Jayasree, J.C. Priya, G.R. Poojita and and G. Kameshwari, “EMI Filter Design for Reducing Common-Mode and Differential-Mode Noise in Conducted Interference”, International Journal of Electronics and Communication Engineering, Vol. 5, No. 3, 2012.

[48] M. Mansuri, “Low-Power Low-Jitter On-Chip Clock Generation” thesis dissertation, 2003.

[49] P. Koopman and T. Chakravarty, “Cyclic Redundancy Code (CRC) Polynomial Selection For Embedded Networks”, Preprint: The International Conference on Dependable Systems and Networks (DSN), 2004.

[50] C.H. Heymann, H.C. Ferreira and J.H. Weber, “A Knuth-based RDS-minimizing multi-mode code”, IEEE Information Theory Workshop (ITW), 2011.

[51] A. Al-Rababa'a, D. Dube and J.-Y. Chouinard, “Using bit recycling to reduce Knuth's balanced codes redundancy”, 13th Canadian Workshop on Information Theory (CWIT), 2013.

[52] V. Skachek and K.A.S. Immink, “Constant Weight Codes: An Approach Based on Knuth's Balancing Method”, IEEE Journal on Selected Areas in Communications, Vol. 32, 2014.

Annex A

How does scrambling balance the data

Scrambling is a XOR (eXclusive OR) operation between the raw data (the data to scramble) and the output of an LFSR (Linear Feeback Shift Register) also called PRBS (Pseudo-Random Binary Sequence).

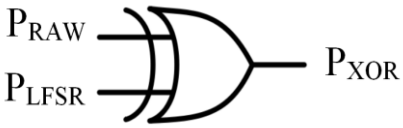


***Fig A.1. Scrambling's* representation**

The raw data is considered to be unknown, so the distribution of "ones" and "zeroes" cannot be determined and their respective probabilities are considered to be random.

But on the other side, the output of an LFSR is known to be uniformly distributed, and the probability of 1's is equal to the probability of 0's.

Now the question is: **What is the probability distribution of 1's and 0's after the XOR operation?**

We denote by P the probability of 1's and Q the probability of 0's.

As mentioned before, the LFSR generates patterns with the probabilities $P_{LFSR} = Q_{LFSR} = 0.5$.

From [33], the probability after a XOR operation could be calculated from the truth table of the XOR operation. Table A.1 shows the truth table with the different probabilities and Figure A.1 illustrates a XOR operation between the LFSR's pattern having $P_{LFSR} = 0.5$ and Raw Data pattern with unknown

probability of ones P_{RAW}. The probability to be determined is the probability of ones after the XOR operation denoted by P_{XOR}.

x	y	z	Probability
0	0	0	$(1-p_x)(1-p_y)$
0	1	1	$(1-p_x)p_y$
1	0	1	$p_x(1-p_y)$
1	1	0	$p_x p_y$

Table A.1. XOR truth table [33]

The probabilities indicated in Table A.1 are calculated through the following logic:

The probability of having a 0 after a XOR could be obtained by multiplying the probabilities of having both the inputs x and y at 0. The corresponding probability is $q_x.q_y$ which is $(1 - p_x)(1 - p_y)$. Same is for the rest.

Now we want to determine P_{XOR} while having the inputs with probabilities P_{RAW} and P_{LFSR}. From the truth table, the probability of 1's after the XOR could be given by the following equation:

$$P_{XOR} = (1 - P_{RAW})\, P_{LFSR} + P_{RAW}(1 - P_{LFSR})$$

$$P_{XOR} = P_{LFSR} + P_{RAW} - 2*P_{RAW}*P_{LFSR}$$

For $P_{LFSR} = 0.5$ this gives $\mathbf{P_{XOR} = P_{LFSR} = 0.5}$ and is independent of the P_{RAW}

Thereby, the probability distribution that comes from XORing any raw data with a uniformly distributed LFSR pattern is $P_{XOR} = Q_{XOR} = 0.5$.

We shall note that even though the scrambling balances the data, it does not guarantee any running disparity bounds.

Annex B

Data's Run Length distribution after scrambling

The data states after scrambling can be represented using Markov Chain as follows:

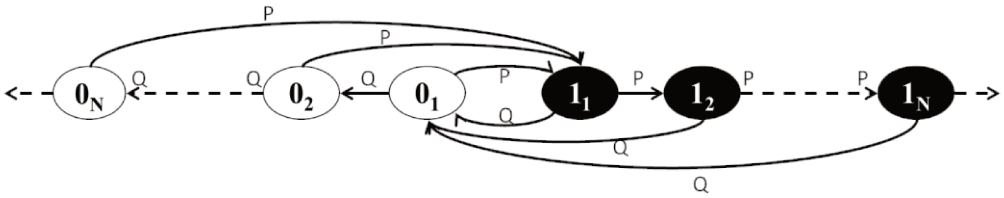


Fig B.1. Consecutive 1's and 0's Markov chain states representation after Scrambling, where P = Q = 0.5

In Figure B.1, P is the probability of 1's and Q is the probability of 0's. According to Annex A, **P = Q = 0.5** after scrambling.

We denote by Π, the probability of a specific state. i.e. $\Pi 1_2$ is the probability of the state 2 consecutive identical 1's. From Fig B.1 we can write an equation for each state. The probability of a specific state corresponds to the entering arrows. For example, $\Pi 1_2 = P*\Pi 1_1$. From this, we will try to deduce a generalized formula for each state:

$\Pi 1_2 = P*\Pi 1_1$
$\Pi 1_3 = P*\Pi 1_2 = P^2*\Pi 1_1$
|
|
$\Pi 1_i = P^{i-1}*\Pi 1_1$ ①

$\Pi 0_2 = Q*\Pi 0_1$
$\Pi 0_3 = Q*\Pi 0_2 = Q^2*\Pi 0_1$
|
|
$\Pi 0i = Q^{i-1}*\Pi 01$ or $\Pi 0_i = P^{i-1}*\Pi 0_1$ ②

To determine $\Pi 1_1$ and $\Pi 0_1$, we know that the sum of all the states equals 1:

$$\sum_{i=1}^{\infty} \Pi 1_i + \sum_{i=1}^{\infty} \Pi 0_i = 1 \quad ③$$

From ①, ② and ③:

➔ $\Pi 1_1 + P*\Pi 1_1 + P^2*\Pi 1_1 + \ldots + \Pi 0_1 + P*\Pi 0_1 + P^2*\Pi 0_1 + \ldots = 1$

➔ $\Pi 1_1(1 + P + P^2 + \ldots) + \Pi 0_1(1 + P + P^2 + \ldots) = 1$

➔ $(1 + P + P^2 + \ldots)(\Pi 1_1 + \Pi 0_1) = 1$ ④

According to Geometric Series [34], for any number r, if |r |<1:

$$\sum_{n=0}^{\infty} r^n = \frac{1}{1-r}$$

Thereby, from ④:

$\frac{1}{1-P}(\Pi 1_1 + \Pi 0_1) = 1 \quad \rightarrow \Pi 1_1 + \Pi 0_1 = 1 - P$ ⑤

According to Fig B.1:

$\Pi 1_1 = P*\Pi 0_1 + P*\Pi 0_2 + P*\Pi 0_3 + \ldots P*\Pi 0_i + \ldots = P* \Sigma \Pi 0_i$
$\Pi 1_1 = P*(1 + P + P^2 + \ldots)*\Pi 0_1$
$\Pi 1_1 = \frac{P}{1-P} \Pi 0_1$ if we preplace this in ⑤:

$\frac{P}{1-P} \Pi 0_1 + \Pi 0_1 = 1 - P \quad \rightarrow \quad \Pi 0_1 = (1-P)^2 = 0.25$
and symmetrically, $\Pi 1_1 = (1-P)^2 = 0.25$

Now Π01 and $\Pi 1_1$ are known, we can calculate from ① and ② the probability of each state and each run length as follows:

Probability of a run length of 5 consecutive identical bits:

$P_{RL}(5) = \Pi 0_5 + \Pi 1_5 = P^4* \Pi 1_1 + P^4* \Pi 0_1 = 0.0312$

0.0312 is the probability of happening in 1 unit. To calculate in how many bits this will happen, we use the following rule:

0.0312 → 1 unit
1 time → X bits?

X = 1/0.0312 = 32.0513 bits or around 8 bytes

We can deduce that a run length of 5 consecutive identical bits will happen theoretically in average after scrambling once every 8 bytes.

The same calculation is done for the rest of the run lengths according to the following formula:

$$P_{RL}(i) = \Pi 0_i + \Pi 1_i$$

The results are illustrated in table B.1 as follows:

Table B.1 Run length theoretical average occurrence after scrambling

Run Length	Occurs in Theoretical average (Bytes)
5	4
6	8
7	16
10	128
14	2 K
18	32 K
20 : :	128 K : :

Annex C

Calculating the probability of a repetitive pattern

In this annex we consider we want to calculate the probability of a pattern of length L bits, to be repeated M times in a row, **after scrambling**.

For this purpose, we consider L = 2 and M = 2, which is one of the easiest cases.

There are 16 possible states in a window of 2*2 (the repetition window M*L) as follows and the repeated states are highlighted:

00 00
00 01
00 10
00 11
01 00
01 01
01 10
01 11
10 00
10 01
10 10
10 11
11 00
11 01
11 10
11 11

We consider, after scrambling, that all the 16 states have equal probability (because P = Q = 0.5). The probability of a repetitive pattern to happen for L = 2 and M = 2 is 4/16.

4 corresponds to all the possible states that can be formed by a pattern of length 2 (00, 01, 10 or 11) which is 2^2 or more precisely 2^L.

16 corresponds to all the cases that can be formed by a window of length 2x2 (or M*L) which is 2^{2x2} or more specifically 2^{L*M}.

Finally, the probability of a pattern of length L to be repeated M times (EMI Killer Packet) can then be written as follows:

$$\mathbf{P_{(L,\,M)}} = \frac{2^L}{2^{L*M}}$$

Annex D

Re-Scrambling of a selected repetitive packet

As we saw in chapter 4, the probability of a repetitive packet (EMI Killer Packet) after scrambling is low and was calculated in Annex C. this probability is considered as Ɛ, which is a small fraction of 1.

However, we consider that after re-scrambling the repetitive packet a second time, the probability of having a repetitive packet again is Ɛ*Ɛ. In this annex, we determine this particular Ɛ*Ɛ case.

We consider the data after the 1st scrambling stage generates the following data: 10 10, we consider this as a pattern of length 2 repeated 2 times (small values for the sake of simplicity) and we re-scramble this pattern a 2nd time (according to the method we proposed in chapter 4) with a polynomial and we look at the pattern after the 2nd scrambling stage.

All the possible 2nd scrambling patterns (PRBS) and all the possible data after 2nd scrambling's (10 10 XORed with the PRBS) results are cited as follows:

Data	PRBS	After 2nd Scrambling (Data XOR PRBS)
10 10	00 00	10 10
	00 01	10 11
	00 10	10 00
	00 11	10 01
	01 00	11 10
	01 01	11 11
	01 10	11 00
	01 11	11 01
	10 00	00 10
	10 01	00 11
	10 10	00 00
	10 11	00 01
	11 00	01 10
	11 01	01 11
	11 10	01 00
	11 11	01 01

The repetitive patterns after scrambling happen according to the above table only when PRBS pattern is repetitive.

The PRBS pattern can be repetitive for small L and M values, but for higher pattern lengths (i.e. a pattern of 8 bits) the repetition cannot exist if the PRBS is well chosen.

Conclusion:

The probability of a repetitive pattern after a second scrambling stage is 0 for relatively large pattern lengths, and they cannot even be designed if the Pseudo Random Binary Sequence (generated by the Linear Feedback Shift Register) is well chosen.

Annex E

Bit Stuffing Overhead after Scrambling

This chapter's purpose is to calculate the theoretical bit stuffing's overhead after scrambling. Bit Stuffing for N consecutive identical bits could be represented in the below Markov chain and is explained in section 5.2.2.

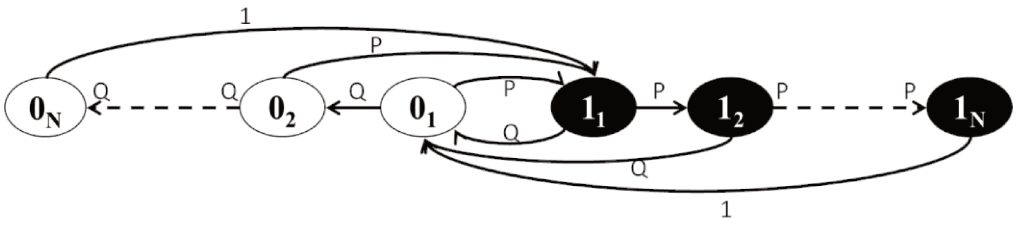


Fig E.1. Bit Stuffing Markov chain states representation after Scrambling, where P = Q = 0.5

The only difference between fig E.1 and fig B.1 is that when state N consecutive identical bits is reached (1_N or 0_N), the only possibility is to go back to the initial states (1_1 or 0_1) with the probability of 1 due to bit stuffing.

From fig E.1 we can write the following, for i < N:

$\Pi 1_2 = P*\Pi 1_1$
$\Pi 1_3 = P*\Pi 1_2 = P^2*\Pi 1_1$
|
|
$\Pi 1_i = P*\Pi 1_{i-1} = P^{i-1}*\Pi 1_1$ ①

$\Pi 0_2 = Q*\Pi 0_1$
$\Pi 0_3 = Q*\Pi 0_2 = P^2*\Pi 0_1$
|
|
$\Pi 0i = Q*\Pi 0_{i-1} = Q^{i-1}*\Pi 0_1$ ②

We can also write:

$\Pi 0_1 = Q*\Pi 1_1 + Q*\Pi 1_2 + ... + Q*\Pi 1_{N-1} + \Pi 1_N$
$= Q*\Pi 1_1 + Q*\Pi 1_2 + ... + Q*\Pi 1_{N-1} + P*\Pi 1_{N-1}$ (according to ①)

We realize that $Q*\Pi 1_{N-1} + P*\Pi 1_{N-1} = (Q+P)*\Pi 1_{N-1} = \Pi 1_{N-1}$
If we continue the simplification of the equation of $\Pi 0_1$ above, we will have at the end:

$\Pi 0_1 = Q*\Pi 1_1 + \Pi 1_2$
$\Pi 0_1 = Q*\Pi 1_1 + P*\Pi 1_1$ (according to ①)

➔ $\Pi 0_1 = \Pi 1_1$ ③

On the other side, we know that:

$$\sum_{i=1}^{N} \Pi 0i + \sum_{i=1}^{N} \Pi 1i = 1$$

➔ $\Pi 0_1*(1+ Q + Q^2 + \ldots + Q^{N-1}) + \Pi 1_1*(1+ P + P^2 + \ldots + P^{N-1}) = 1$

➔ $\Pi 0_1*A+ \Pi 1_1*B = 1$ Where $\Pi 0_1 = \Pi 1_1$

➔ $\Pi 0_1*(A+B) = 1$ or $\Pi 1_1*(A+B) = 1$

➔ $\Pi 0_1 = \Pi 1_1 = \frac{1}{A+B}$

From geometric series [34], for any number r :

$$1 + r + r^2 + r^3 + \cdots + r^{n-1} = \frac{1 - r^n}{1 - r} \quad ④$$

④ gives $A = \frac{1- Q^N}{1-Q}$ and $B = \frac{1- P^N}{1-P}$ where P = Q = 0.5 after scrambling

The probability of the state N corresponds to the overhead of the bit stuffing because a bit is added when the state N is reached. The probability of the state N could be written as follows:

$Prob(N) = \Pi 0_N + \Pi 1_N$
$Prob(N) = Q^{N-1}\Pi 0_1 + P^{N-1}\Pi 1_1$

And finally, the Bit Stuffing Overhead BSO could be written as follows:

$$BSO(N) = Q^{N-1}\Pi 0_1 + P^{N-1}\Pi 1_1$$

Where $\Pi 0_1 = \Pi 1_1 = \frac{1}{A+B}$, $A = \frac{1 - Q^N}{1-Q}$ and $B = \frac{1 - P^N}{1-P}$

Ex for N = 5, A= B= 1.937 / $\Pi 0_1 = \Pi 1_1$= 0.258 / BSO(5) = 0.0322 = 3.22 %

Annex F

Balancing Overhead after Scrambling

In this Annex, we will give a theoretical overhead estimation for the DC-balancing solution we provided in chapter 6. Markov Chains will be used for this purpose, but this time the different states will represent the Running Disparity (RD) instead of 1's and 0's states used in the previous annexes.

In this annex, for the sake of simplicity, we will use Markov chains transition matrices to calculate the different probabilities and we will provide 5 overhead calculation examples, and correlate the results with the simulation overhead provided in chapter 6.

Example 1: T = 2 and S = 2 (RD bounded to +/- 3)

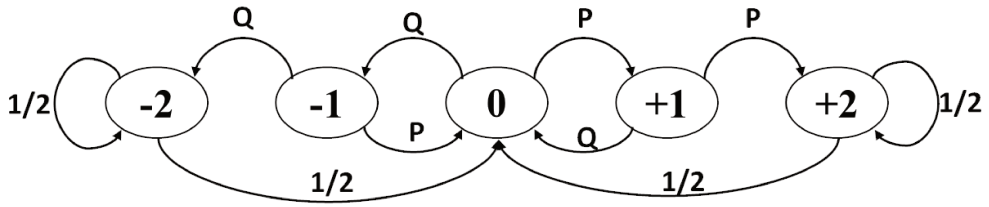


Fig F.1. Proposed DC-balancer Markov chain states representation after Scrambling, where P = Q = 0.5

Figure explanation:

In figure F.1, we represent the RD states: we can go from a RD of 0 to an RD of 1 with a probability P (probability of 1's which equals ½ after scrambling) and from an RD of 0 to an RD of -1 with a probability of Q (probability of 0's which equals ½ after scrambling)

When we are on state +2 (or -2), the RD of the packet S = 2 will be studied. In this case RD(S) can be either 0, +2, or -2. If RD(S) is +2, it will be inverted so it will be RD(S) = -2. The possible RD(S) can finally be either 0 or -2.

The packet S can be a 00, 01, 10 or 11. If S is 00 or 11, if we are on state '+2', RD(S) will be equal to -2 and we will transit to state '0' with a probability of ½ (2 states out of 4 possible for the packet S) . If S is 01 or 10, RD(S) = 0, if

we are on the state '+2', we will stay on state +2 with a probability of ½ (2 states out of 4 possible for the packet S).

Deducing the overhead equation:

The overhead due to T = 2 and S = 2 comes from the added polarity-bit. The polarity-bit will be added when we are on the states +2 or -2 with RD(S) = +/-2. There will be not bit added when RD(S) = 0 so this probability should be subtracted. The balancing overhead for T = 2 and S = 2 can be written then as follows:

$$BO(2,2) = \Pi_{+2} + \Pi_{-2} - ½ * \Pi_{+2} - ½ * \Pi_{-2}$$

Calculating the probabilities:

The probabilities could be calculated using the Markov chain transition matrix as follows:

	-2	-1	0	1	2
-2	1/2	0	1/2	0	0
-1	1/2	0	1/2	0	0
0	0	1/2	0	1/2	0
1	0	0	1/2	0	1/2
2	0	0	1/2	0	1/2

The columns and rows correspond to the states, and the crossing of each column and row corresponds to the probability of transition from the specific state to the other. The matrix could be written as follows and we will call it Y.

$$Y = \begin{pmatrix} 1/2 & 0 & 1/2 & 0 & 0 \\ 1/2 & 0 & 1/2 & 0 & 0 \\ 0 & 1/2 & 0 & 1/2 & 0 \\ 0 & 0 & 1/2 & 0 & 1/2 \\ 0 & 0 & 1/2 & 0 & 1/2 \end{pmatrix}$$

The different states could be written in a matrix of one row:

(-2 -1 0 +1 +2)

To find the probability of state '-2", we will do the following matrix multiplication:

$\Pi_{-2} = (1 \ 0 \ 0 \ 0 \ 0) * Y^X$ where X is a sufficiently big value that makes Π_{-2} stable after a specific X value.

$\Pi_{-2} = (1 \ 0 \ 0 \ 0 \ 0) * Y^{100} = 0.1667$ (calculation done on Matlab)

$\Pi_{+2} = (0 \;\; 0 \;\; 0 \;\; 0 \;\; 1) * Y^{100} = 0.1667$

➔ $BO(2,2) = \Pi+2 + \Pi-2 - ½ * \Pi+2 - ½ * \Pi-2$

➔ $BO(2,2) = 0.1667 = 16.67\ \%$

According to theory, the overhead due to the proposed balancing method for T = 2 and S = 2 is 16.67 %. The simulation results gave 14.27 %.

Example 2: T = 3 and S = 2 (RD bounded to +/- 4)

$$Y = \begin{pmatrix} 1/2 & 0 & 1/2 & 0 & 0 & 0 & 0 \\ 1/2 & 0 & 1/2 & 0 & 0 & 0 & 0 \\ 0 & 1/2 & 0 & 1/2 & 0 & 0 & 0 \\ 0 & 0 & 1/2 & 0 & 1/2 & 0 & 0 \\ 0 & 0 & 0 & 1/2 & 0 & 1/2 & 0 \\ 0 & 0 & 0 & 0 & 1/2 & 0 & 1/2 \\ 0 & 0 & 0 & 0 & 1/2 & 0 & 1/2 \end{pmatrix}$$

$\Pi_{-3} = (1 \;\; 0 \;\; 0 \;\; 0 \;\; 0 \;\; 0 \;\; 0) * Y^{100} = 0.10$

$BO(3,2) = \Pi_{+3} + \Pi_{-3} - ½ * \Pi_{+3} - ½ * \Pi_{-3} = 0.10 = 10\%$

Example 3: T = 4 and S = 2 (RD bounded to +/- 5)

$$Y = \begin{pmatrix} 1/2 & 0 & 1/2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1/2 & 0 & 1/2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1/2 & 0 & 1/2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1/2 & 0 & 1/2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1/2 & 0 & 1/2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1/2 & 0 & 1/2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1/2 & 0 & 1/2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1/2 & 0 & 1/2 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1/2 & 0 & 1/2 \end{pmatrix}$$

$\Pi_{-4} = (1 \;\; 0 \;\; 0 \;\; 0 \;\; 0 \;\; 0 \;\; 0 \;\; 0 \;\; 0) * Y^{100} = 0.0714$

$BO(4,2) = \Pi_{+4} + \Pi_{-4} - ½ * \Pi_{+4} - ½ * \Pi_{-4} = 0.0714 = 7.14\ \%$

Example 4: T = 5 and S = 2 (RD bounded to +/- 6)

$$Y = \begin{pmatrix} 1/2 & 0 & 1/2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1/2 & 0 & 1/2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1/2 & 0 & 1/2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1/2 & 0 & 1/2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1/2 & 0 & 1/2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1/2 & 0 & 1/2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1/2 & 0 & 1/2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1/2 & 0 & 1/2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1/2 & 0 & 1/2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1/2 & 0 & 1/2 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1/2 & 0 & 1/2 \end{pmatrix}$$

$\Pi_{-5} = (1 \ 0 \ \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0) * Y^{100} = 0.0556$

$BO(5,2) = \Pi_{+5} + \ \Pi_{-5} - ½ * \Pi_{+5} \ - ½ * \Pi_{-5} = 0.0556 =$ 5.66 %

Example 5: T = 5 and S = 4 (RD bounded to +/- 7)

$$Y = \begin{pmatrix} 3/8 & 0 & 1/2 & 0 & 1/8 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1/2 & 0 & 1/2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1/2 & 0 & 1/2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1/2 & 0 & 1/2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1/2 & 0 & 1/2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1/2 & 0 & 1/2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1/2 & 0 & 1/2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1/2 & 0 & 1/2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1/2 & 0 & 1/2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1/2 & 0 & 1/2 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1/8 & 0 & 1/2 & 0 & 3/8 \end{pmatrix}$$

$\Pi_{-5} = (1 \ 0 \ \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0) * Y^{100} = 0.0417$

$BO(5,4) = \Pi_{+5} + \ \Pi_{-5} - 3/8 * \Pi_{+5} \ - 3/8 * \Pi_{-5} = 0.0521 =$ 5.21 %

The comparison between the theoretical estimation and the simulation is illustrated in figure F.2 as follows:

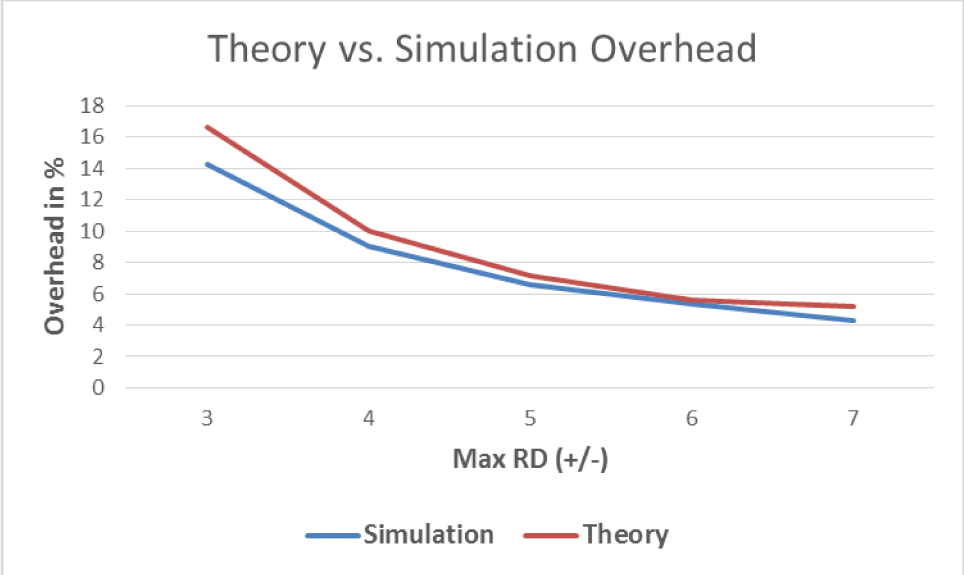


Fig F.1. Theoretical vs. Simulation Overhead

The exact values are also mentioned in the following table:

T	S	CRD bounds	Simulation Overhead	Theoretical Average Overhead
2	2	+/- 3	14.27 %	16.67 %
3	2	+/- 4	9.05 %	10.00 %
4	2	+/- 5	6.60 %	7.14 %
5	2	+/- 6	5.32 %	5.56 %
5	4	+/- 7	4.32 %	5.21 %

Annex G

Physical Layer Framing Proposal for variable length data

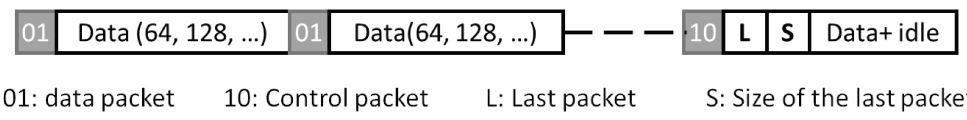
The Physical Layer (PHY) needs to distinguish control packets from data packets and other information such as the start of Frame (SoF) and the end of frame (EoF). For this reason, a mechanism should be defined, as it is the case of 64b/66b and 128b/130b encodings for example. The packet size is fixed to 64 or 128 so the PHY knows when a packet starts and ends, and 2 bits (10b or 01b) are added to each packet so the PHY can know if the packet is a data packet or a control packet.

In the proposals we made in chapter 5 and 6, even though the overhead is predictable, it will change the data size to a non-predictable exact size. Another framing mechanism should be defined for this type of encodings and in this annex we show how.

In order to optimize the overhead as much as possible, we distinguish big data bursts from small bursts and we propose a framing for each type.

Framing proposal for big data bursts (more than 4 to 5 packets of 64 or 128 bits)

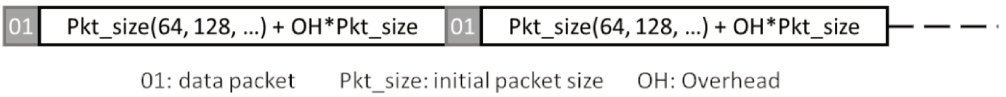
Because the data size is not fixed, the number of valid bits in the last received packet (of 64, 128, 256…) must be known. For this purpose, we propose the following framing:



2 bits, 10b or 01b indicate to the receiver a data packet or a control packet. In most protocols, the control packet includes a "control type field". We propose that this type field (that could be a byte at the beginning of a control packet) allocates a last packet indicator. It will indicate the last data packet followed by the size of the valid bits and the last bits themselves. The remaining bits (to finish the 64, 128 …) could be filled with idle bits (10101010…) to not create RL or RD problems.

Framing proposal for small data bursts (less than 4 to 5 packets of 64 or 128 bits)

For small data bursts, a whole packet can be added to few packets because of the overhead, a big increase in overhead can be then introduced eliminating the low overhead advantage. The previous framing can then be an issue when transmitting small amounts of data and to solve this issue we propose the following framing:



With a very high probability, all the data will stay within a data packet indicated by 01b at the start of each packet. The packet size will be the size of the initial packet (64, 128 bits…) plus the overhead bits that are supposed to be added due to the proposed encodings (RL and RD bounds dependent). The packet size will be then fixed to a specific value. If the data packet plus its overhead do not fit into the fixed packet, the previous framing can stay a feature to help. But in most cases the overhead is predictable and there will be no need to an additional "last packet" and the burst will end normally with a control frame to indicate the end of frame.