DC-Balanced Improvement of Interlaken Protocol

by

Sarat Yoowattana

A Dissertation Submitted in Partial Fulfillment of the Requirements for the Degree of

Doctor of Engineering in Microelectronics and Embedded Systems

Examination Committee: Dr. Mongkol Ekpanyapong (Chairperson)

Prof. Matthew N. Dailey

Mr. Chumnan Panyasai

External Examiner: Dr. Ian Marsland

Department of Systems and Computer Engineering

Carleton University

Canada

Nationality: Thai

Previous Degree: Master of Engineering in Electronics

King Mongkut’s Institute of Technology

Ladkrabung, Thailand

Scholarship Donor: Royal Thai Government Fellowship

Asian Institute of Technology

School of Engineering and Technology

Thailand

May 2021

**AUTHOR’S DECLARATION**

I, Sarat Yoowattana, declare that the research work carried out for this dissertation was in accordance with the regulations of the Asian Institute of Technology. The work presented in it are my own and has been generated by me as the result of my own original research, and if external sources were used, such sources have been cited. It is original and has not been submitted to any other institution to obtain another degree or qualification. This is a true copy of the dissertation, including final revisions.

Date: 22 April 2021

Name (in printed letters): Sarat Yoowattana

Signature:

**ACKNOWLEDGMENTS**

Firstly, my thank is given to Dr. Mongkol Ekpanyapong who is my advisor. Since at the beginning of my study in AIT, he always gives me suggestions and directions of my study and dissertation. Dr. Matthew N. Dailey and Mr. Chumnan Panyasai who are my examination committee also give me interested and useful comments. ISE’s secretaries those who let me keep going in the right direction. Finally, with the support from Royal Thai Government, it makes me to have a chance to study doctoral degree in AIT.

# ABSTRACT

High-speed serial data communication is essential for connecting peripherals in high performance computing systems. The Interlaken is a high-speed serial data communication protocol that has been widely adopted in various applications because it can run on multiple medias such as PCBs, backplanes or over cables. The Interlaken uses 64b/67b line coding to maintain the run length (RL) and the running disparity (RD) with the advantage of an inversion bit that indicates whether the receiver must flip the data or not. By using the inversion bit, it increases 1 bit overhead to every data word. This thesis proposes 64b/i67b line coding for encoding and decoding, which is a new technique to improve the cumulative running disparity of 64b/67b without additional bit overhead. The results have been obtained from simulations that use random data and the Squash data set, and the proposed method reduces the maximum cumulative running disparity value up to 33%.

**CONTENTS**

**Page**

ACKNOWLEDGMENTS iii

[ABSTRACT iv](#_Toc51682283)

[LIST OF TABLES vii](#_Toc51682284)

[LIST OF FIGURES viii](#_Toc51682285)

[LIST OF ABBREVIATIONS x](#_Toc51682286)

[CHAPTER 1 INTRODUCTION 1](#_Toc51682287)

[1.1 Background of the Study 1](#_Toc51682288)

[1.2 Statement of the Problem 2](#_Toc51682289)

[1.3 Objectives of the Study 2](#_Toc51682291)

[1.4 Scope](#_Toc51682292) 3

[CHAPTER 2 LITERATURE REVIEW](#_Toc51682294) 4

[2.1 Run-Length 4](#_Toc51682295)

[2.2 Running Disparity](#_Toc51682296) 6

[2.3 Bit Error Rate](#_Toc51682299) 8

[2.4 Literature Review](#_Toc51682302) 9

[2.4.1 8b/10b Line Coding](#_Toc51682303) 9

[2.4.2 16b/18b Line Coding](#_Toc51682304) 11

[2.4.3 64b/66b Line Coding](#_Toc51682304) 12

[2.4.4 64b/67b Line Coding](#_Toc51682304) 13

[2.4.5 Another Line Coding](#_Toc51682304) 14

[2.5 Interlaken Protocol](#_Toc51682305) 16

[2.5.1 Overview of the Protocol.](#_Toc51682304) 16

[2.5.2 64b/67b Encoding in Interlaken Protocol](#_Toc51682304) 18

[2.5.3 Meta Frame](#_Toc51682304) 22

[2.5.4 Commercial Product with Interlaken Protocol](#_Toc51682304) 23

[2.6 Summary](#_Toc51682306) 25

[CHAPTER 3 METHODOLOGY](#_Toc51682294) 27

[3.1 The Proposing Technique Called 64b/i67b](#_Toc51682295) 27

[3.2 64b/67b Encoding and Decoding](#_Toc51682296) 29

[3.3 64b/i67b Encoding and Decoding](#_Toc51682299) 32

[3.4 Probability](#_Toc51682302) 40

[3.4.1 Probability of 64b/67b](#_Toc51682303) 40

[3.4.2](#_Toc51682304) Probability of 64b/i67b 41

[3.5 Implementation](#_Toc51682305) 44

[3.5.1 Simulation With MATLAB](#_Toc51682303) 44

[3.5.2](#_Toc51682304) HDL Design 47

[CHAPTER 4 RESULTS AND DISCUSSION](#_Toc51682294) 49

[4.1 Purpose 4](#_Toc51682295)9

[4.2 Random Simulation](#_Toc51682296) 51

[4.3 Using Squash Compression Benchmark Simulation](#_Toc51682299) 54

[4.4 Worst Case of 64b/i67b](#_Toc51682302) 59

[CHAPTER 5 CONCLUSION AND RECOMMENDATIONS](#_Toc51682294) 60

[5.1 Conclusion 60](#_Toc51682295)

[5.2 Recommendations](#_Toc51682296) 60

[REFERENCES 61](#_Toc51682307)

[APPENDIX 63](#_Toc51682308)

# LIST OF TABLES

**Tables Page**

[Table 2.1 Bit Stuffing Technique](#_Toc52113192) 6

[Table 2.2 Running Disparity Example](#_Toc52113193) 6

[Table 2.3 16b/18b Encoding](#_Toc52113192) 12

[Table 2.4 Inversion bit Interpretation](#_Toc52113193) 19

[Table 2.5 Sync bit Encoding](#_Toc52113192) 19

[Table 2.6 64b/67b Encoding Example](#_Toc52113193) 21

[Table 2.7 64b/67b Encoding Example](#_Toc52113192) 22

[Table 2.8 Interlaken Commercial Products](#_Toc52113192) 23

[Table 3.1 Example of Raw 64-bit Data](#_Toc52113192) 29

[Table 3.2 64b/67b Encoded Example](#_Toc52113193) 30

[Table 3.3 64b/67b Decoded Example](#_Toc52113192) 31

[Table 3.4 64b/i67b Encoding Example](#_Toc52113193) 33

[Table 3.5 Comparison Between CRD of 64b/67b and 64b/i67b](#_Toc52113192) 34

[Table 3.6 64b/i67b Decoding Example](#_Toc52113192) 35

[Table 3.7 Encoding Probability of 64b/67b](#_Toc52113192) 35

[Table 3.8 Encoding Probability of 64b/i67b](#_Toc52113192) 38

[Table 3.9 Decoding Probability of 64b/i67b](#_Toc52113192) 38

[Table 3.10 Example of Block A and Block B Combination](#_Toc52113192) 43

[Table 3.11 Encoder Resource Utilization](#_Toc52113192) 48

[Table 3.12 Decoder Resource Utilization](#_Toc52113192) 48

[Table 4.1 Simulation Result Comparison of 200K Random Data](#_Toc52113192) 50

[Table 4.2 Simulation Result Comparison of 400K Random Data](#_Toc52113192) 52

[Table 4.3 Simulation Result Comparison of 600K Random Data](#_Toc52113192) 53

[Table 4.4 Simulation Result When Using Squash Benchmark](#_Toc52113192) 55

[Table 4.5 Average CRD Value of Simulation Result From Squash Benchmark](#_Toc52113192) 57

[Table 4.6 Worst Case of 64b/i67b](#_Toc52113192) 59

# LIST OF FIGURES

# Figures Page

[Figure 2.1 Meghelli’s Phase Detector](#_Toc52113238) 4

[Figure 2.2 Phase Detector, Clock Lagging Data](#_Toc52113238) 5

[Figure 2.3 Phase Detector, Clock Leading Data](#_Toc52113238) 5

[Figure 2.4 Voltage Shifting Situation](#_Toc52113238) 7

[Figure 2.5 Manchester Encoding](#_Toc52113238) 8

[Figure 2.6 VNormal When DC-balanced and VNormal When DC-Unbalanced](#_Toc52113238) 9

[Figure 2.7 8b/10b Encoding](#_Toc52113238) 10

[Figure 2.8 8b/10b State Diagram](#_Toc52113238) 11

[Figure 2.9 16b/18b Line Coding](#_Toc52113238) 11

[Figure 2.10 64b/66b Line Coding](#_Toc52113238) 13

[Figure 2.11 64b/67b Line Coding](#_Toc52113238) 14

[Figure 2.12 128b/130b Line Coding](#_Toc52113238) 15

[Figure 2.13 128b/132b Line Coding](#_Toc52113238) 15

[Figure 2.14 Interlaken Word Format](#_Toc52113238) 16

[Figure 2.15 Transmit Interface State](#_Toc52113238) 17

[Figure 2.16 Receive per Lane State](#_Toc52113238) 18

[Figure 2.17 64b/67b Word Boundary Lock](#_Toc52113238) 20

[Figure 2.18 Interlaken Meta Frame](#_Toc52113238) 23

[Figure 2.19 64b/i67b, the Proposed Line Coding](#_Toc52113238) 25

[Figure 3.1 Example of 64-bit Payload](#_Toc52113238) 27

[Figure 3.2 Example of Separate 64-bit to two Blocks](#_Toc52113238) 28

[Figure 3.3 Pseudo Code of 64b/67b Encoding](#_Toc52113238) 36

[Figure 3.4 Pseudo Code of 64b/67b Decoding](#_Toc52113238) 37

[Figure 3.5 Pseudo Code of 64b/i67b Encoding](#_Toc52113238) 39

[Figure 3.6 Pseudo Code 3.4 64b/i67b Decoding](#_Toc52113238) 40

[Figure 3.7 Interlaken Communication Block](#_Toc52113238) 44

[Figure 3.8 MATLAB Encoding Flowchart](#_Toc52113238) 45

[Figure 3.9 MATLAB Decoding flowchart](#_Toc52113238) 46

[Figure 3.10 Encoder-Decoder Testbench Connection](#_Toc52113238) 47

[Figure 3.11 Simulation Wave From for 64b/67b](#_Toc52113238) 47

[Figure 3.12 Simulation Wave From for 64b/i67b](#_Toc52113238) 48

[Figure 4.1 Random Data at 200K Word](#_Toc52113238) 49

[Figure 4.2 Random Data at 400K Word](#_Toc52113238) 51

[Figure 4.3 Random Data at 600K Word](#_Toc52113238) 54

[Figure 4.4 Simulation Result When Using Squash Benchmark](#_Toc52113238) 56

[Figure 4.5 CRD Comparison of 64b/67b and 64b/i67b in “xargs1”](#_Toc52113238) 57

# LIST OF ABBREVIATIONS

|  |  |
| --- | --- |
| AC  BER  CDR  CRD  DC  EMI  GbE  HDL  PCIe  RD | = Alternate Current  = Bit Error Rate  = Clock and Data Recovery  = Cumulative Running Disparity  = Direct Current  = Electromagnetic Interference  = Gigabit-Ethernet  = Hardware Description Language  = Peripheral Component Interconnect express  = Running Disparity |
| RL  SNR | = Run Length  = Signal-to-Noise Ratio |
| SONET | = Synchronous Optical Networking |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

# INTRODUCTION

## Background of the Study

High-speed serial data communication is now very popular for connecting various resources in high-performance computing systems (Abhijit and Carl, 2005). In such high-speed serial links, a line coding is important to control the RL and RD, because a large run length causes insufficient transitions on data-links that makes it difficult to perform reliable CDR, and a bounded running disparity is needed for maintaining the DC-balance of data links. These requirements for a line coding, however, cause additional bits to be transmitted, which decreases the throughput of data transmission but still needed to maintain RL and RD.

One of the challenges to improving performance is controlling RL and RD in the communication line with giving lowest bit-overhead. There are many popular line coding has been using in various application such as USB, PCIe, IEEE 802.3ae (10GbE), IEEE 802.3ba (40GbE and 100GbE) and Interlaken. Those are the popular protocols which are RD, RL controlled but some not. This dissertation presents a new approach to controlling a running disparity of Interlaken Protocol without additional bit-overhead. The experimental results show that the proposed technique reduces the maximum of cumulative running disparity and average cumulative running disparity per word. Serial communication is widely used because it has higher performance, compared to parallel communication, in speed, pin counts, EMI, and so on (Abhijit Athavale).

In the general serial communication, parallel data is first serialized. After this step, a line coding is applied, and then the encoded data is actually transmitted through datalinks. A receiver receives the encoded serial data, decodes it, and then converts it to parallel data. This line coding is needed in order to avoid sending long sequences of “0” or “1” bit. If long sequences of the same bits are sent, it is difficult to perform reliable clock and data recovery (CDR), because the CDR circuit tries to periodically synchronize with the transmitted data by detecting transitions (rising or falling edges) on data-links. The number of same bits is called RL. RL increases when the current bit has the same value with the previous one, and it is reset to 1 when a different bit appears.

Another important role that line codings play is to control the DC-balance of AC-coupled data links. In AC-coupled data links, if the numbers of “0” and “1” that appear in the links are unbalanced, the voltage levels of the links are shifted, and it finally causes errors in samplers of the receiver side. The difference of numbers of “0” and “1” is called RD. RD initially takes 0, and is incremented (or decremented) when “1” (or “0”) appears in data. Line codings are used to bound the RL and RD values, and thus to improve the signal integrity of the transmitted serial data.

They, however, introduce additional bits to be transmitted, which decreases the throughput of data transmission. The ratio of the additional bits to the raw data bits is called bit overhead. It is desirable to obtain a line coding with lower bit-overhead for he given RL and RD bounds.

## Statement of the Problem

Any line coding protocol need to add extra bit to maintain RD and RL. However, with trade-off between signal integrity and data throughput, increasing the number of additional bits can increase performance of signal integrity but decrease data throughput. The trend of high-speed communication technology is serial. Some protocols can’t control the parameters that is used to maintain signal integrity to reduce bit-overhead for increase data throughput, but it can be used in some application such as the application that has a small length of cable.

In this research, we propose to increase data throughput with controlling of signal integrity. To perform a line coding that has low bit-overhead and would be maintained signal integrity in communication.

## Objectives of the Study

The RD in high speed serial communication must be controlled to maintain signal integrity of communication.

1. Minimize the running disparity of existing line coding protocol named   
    64b/67b without additional bit-overhead.
2. Design and implement encoder and decoder in simulator and Verilog HDL.
3. Analyze obtained result whether it can reduce running disparity by comparing   
    to original line coding.

## Scope

Line coding is used to maintain signal integrity of communication. To maintain disparity from transmission data, the average CRD and maximum CRD must be controlled to be as minimized as possible. The original 64b/67b line coding is the baseline to the proposing technique. The result of proposing technique will be compared to the baseline with random data and files from compression benchmark.

# LITERATURE REVIEW

This chapter will focus on major parameters to be maintain in line coding and literature review. The RD and RL are the parameters that need to be controlled to maintain stability of serial communication. Also, the previous works are studied and analyzed to understand exiting techniques that will be used in this research.

## Run-Length

The clock and data recovery or CDR need transition of serial data to recovery clock at receiver. There are several techniques to recover clock at receiver. If the data always change, the CDR can recover clock precisely (Alexander, 1975). But, if not, the clock can be shifted, and it causes transmission errors. The Run-length or RL is the number of bit data without transition. If this number gets higher, it means the CDR does not have enough transition edges to adjust the up/down of clock phase. So, if the RL can be controlled as low as possible, the CDR will use this benefit to recover clock at receiver and this will result in better performance for CDR circuit.

**Figure 2.1**

*Meghelli’s Phase Detector*

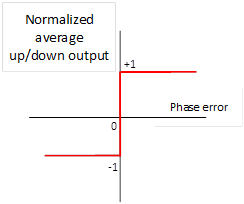
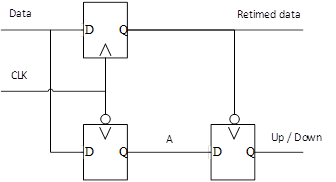
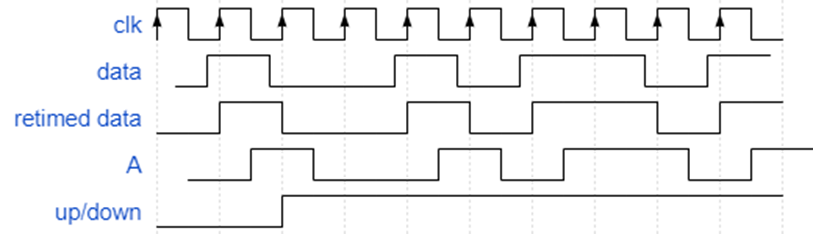


Figure 2.1 show a simple CDR circuit from (Meghelli et al, 2000). The data from transmitter is connected to Data input of flipflop and clocked by rising edge of receiver clock the output of this flipflop named retimed data, also the A signal is the flipflop output of Data but with negative clock. Then, the A signal and Retimed data signal are used to adjust receiver clock phase by the A signal is connect to another flipflop and the Retimed data is connected to negative clock input of the flipflop. If clock is lagging data, the falling edge of Retimed data will latch ‘1’ to up/down output but if the clock leading data it will latch ‘0’ to up/down output. The up/down output will be used to adjust receiver clock phase.

**Figure 2.2**

*Phase Detector, Clock Lagging Data*



**Figure 2.3**

*Phase Detector, Clock Leading Data*

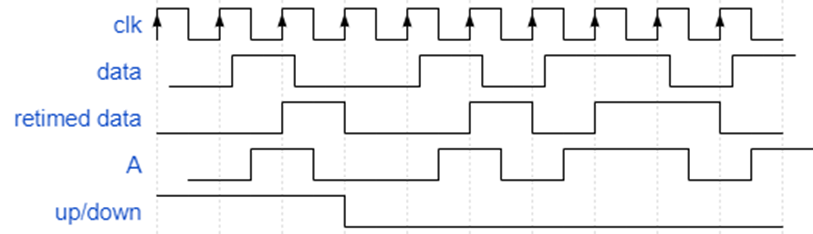


Table 2.1 shows an example of bit stuffing technique (Saadé et al, 2015) (BOSCH, 1991). To maintain RL when maximum allowed number of RL is 5. The encoder will count the number of same bits in raw data, and when it reaches to the maximum value, the invert of the last bit will be added into the encoded data. Here this additional bit increases bit-overhead.

**Table 2.1**

*Bit Stuffing Technique*

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Raw data | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 |  |
| Encoded data | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 0 |
| RL | 1 | 1 | 2 | 3 | 4 | 5 | 1 | 1 | 1 |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Raw data | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 0 |  |
| Encoded data | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 |
| RL | 1 | 1 | 2 | 3 | 4 | 5 | 1 | 2 | 3 |

## Running Disparity

The running disparity needs to be controlled to maintain DC-balance of transmission line otherwise the effect of DC-balance will be cause error in transmission. Table 2.2 shows an example of counting running disparity. It different from run-length that it would not reset disparity value when opposite bit has been transmitted. The running disparity start at zero and it will add ‘1’ if transmit ‘1’ and minus ‘1’ if transmit ‘0’. So, the running disparity value can be positive or negative instead of run length that can be only positive.

**Table 2.2**

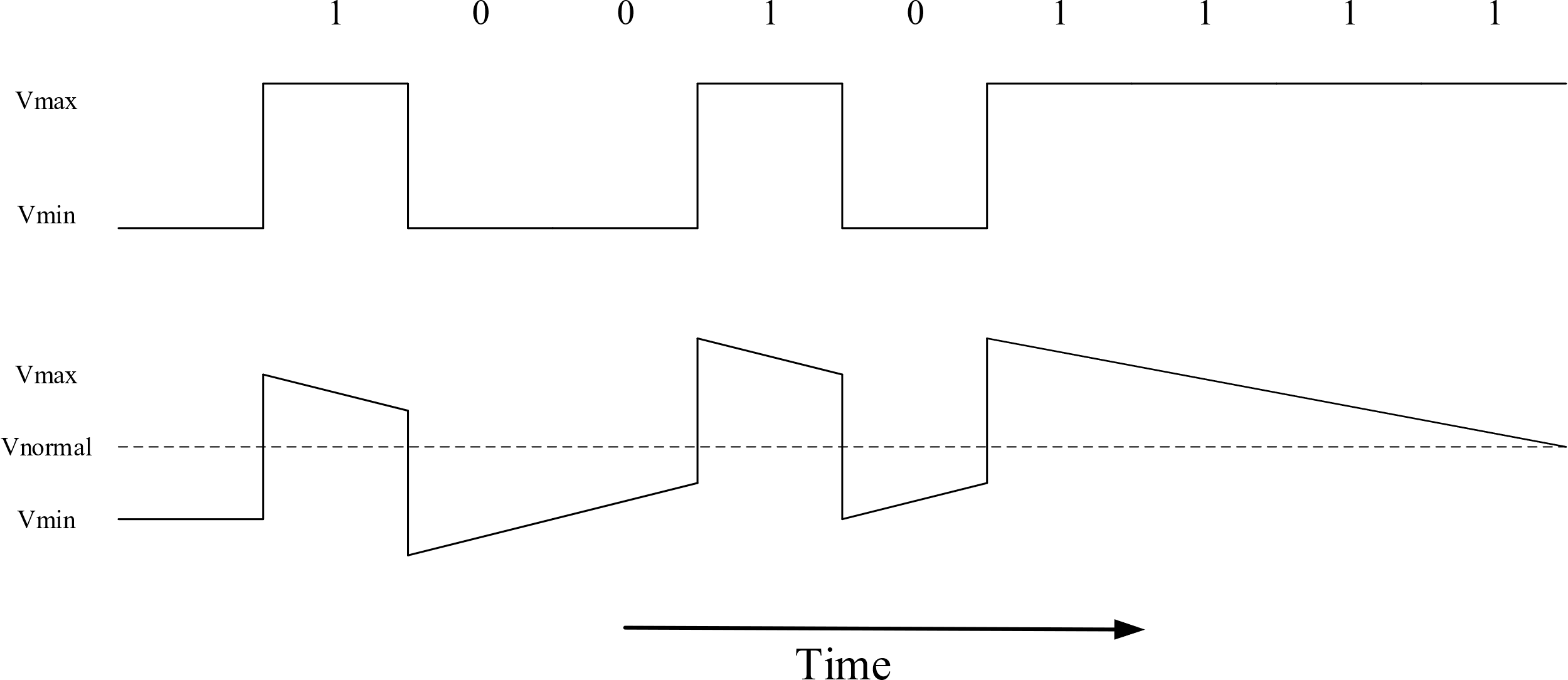
*Running Disparity Example*

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Raw data |  | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 1 |
| RL | 0 | 1 | 2 | 1 | 0 | -1 | -2 | -1 | -2 | -1 | 0 | 1 |

The 64b/67b encoding technique controls DC-balancing of serial data by keeping Running disparity (RD) not over ±96. This 64b/67b proposed for Interlaken (Interlaken, 2008) is a modification of the 64b/66b (Qinglun et al, 2006) (Mohapatra et al, 2017) (Mohapatra et al, 2017) used with 10 Gigabit Ethernet (IEEE 802.3ae) the serial transmission uses AC-couple to prevent the accumulation of charge across the coupling capacitor. Figure 2.4 shows the situation of the voltage shifting in the transmission line when a long period of transition appears. The signal will disappear from the center line and cause transmission error when a receiver receives shifted voltage level. For example, a 2-PAM (Bakir and Meindl, 2008) the logic ’1’ can be detected as ’0’ or logic ’0’ can be detected as ’1’. Another effect of voltage shifting by DC-balanced is the received signal-to-noise ratio or SNR will be reduced (Morris, 1983) and the bit error rate will be increased (Smith, 2003).

**Figure 2.4**

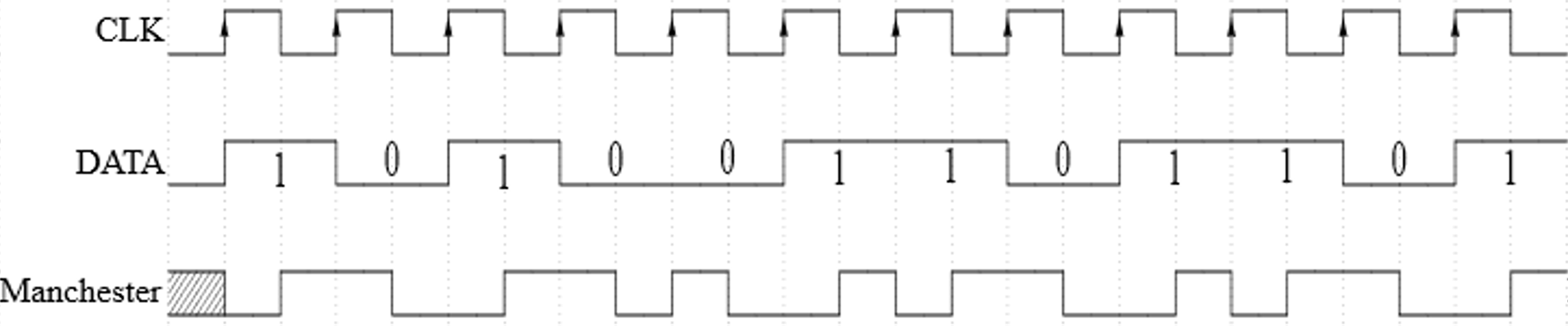
*Voltage Shifting Situation*



To avoid and resolve this problem, the maximum of running disparity must be controlled. For example, the most popular DC-balanced algorithm is Manchester encoding (Forster, 2000). It controls DC-balance every bit. The advantage of Manchester encoding (Figure 2.5) is to maintain DC-balance of a transmission line very easily because every single bit is converted to 2 bits by adding an invert bit. However, this causes a very high bit-overhead to 100% of raw data (Uday et al, 2018).

**Figure 2.5**

*Manchester Encoding*



## Bit Error Rate

The Bit Error Rate or BER is the ratio between number of error bits and total number of bits sent. Equation (2.1) denote the equation for BER. Where is number of error bits and N is total number of bits sent

**Equation 2.1**

Then, to measure a reliable of BER measurement, we need to transmit at least 100 million bits and it might take hours or days (Wang et al, 2007). Thus, the calculation method has been used to measure BER instead of comparing bits by applying Gaussian noise and calculate the probability of error and it can be expressed by Equation (2.2) and the relationship between BER and SNR can be expressed by Equation (2.3) (Fan and Zilic, 2008) (Horvath et al, 2016).

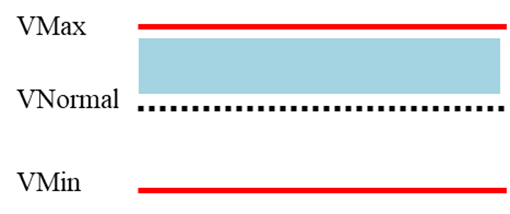
**Equation 2.2**

**Equation 2.3**

Here, the SNR will decrease if the running disparity get higher and it affects to BER that will be increased. Controlling RD or baseline wander will control SNR and that help to minimize the BER. Figure 2.6 (a) shows normal margin between VNormal and VMax when DC is balanced perfectly in transmission instead of Figure 2.6 (b) that shows VNormal is shifted close to VMax and this decrease signal to noise ratio.

**Figure 2.6**

*(a) VNormal When DC-balanced, (b) VNormal when DC-unbalanced*



(a)



(b)

## Literature Review

### 8b/10b Line Coding

In 1983 A. Widmer (Widmer and Franaszek, 1983) proposed 8b/10b line coding that is the most popular in serial communication and was implemented to PCIe 2.0, USB 3.0, and so on. The 8b/10b line coding has very high performance in controlling the RL and RD, but also has high bit-overhead. It can control maximum absolute RD to five and maximum RL to three. This encoding technique uses lookup-tables to transform 8-bit data into 10-bit data. It actually has two lookup-tables (i.e., two 10-bit data for each 8bit data), and one table is used depending on the sign (positive or negative) of the current RD. When the current RD is positive (negative, resp.), the negative (positive) set of 10-bit data is used to control the RD. This high performance (small RL and small absolute RD bounds) of this technique is obtained from the sacrifices of the high bit-overhead, which reaches 25%.

**Figure 2.7**

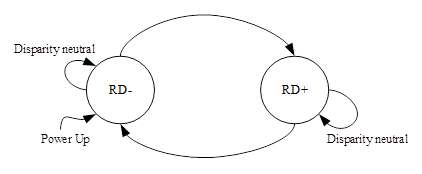
*8b/10b Encoding*



The encoding of 8b/10b can be mapped to 256 data characters (D\_(X.Y)) and 12 control characters (K\_(X.Y)). The 8-bit data will be grouped to two blocks as 3-bit and 5-bit and they are named HGF and EDCBA. Then, the 3-bit block will be encoded to 4-bit “fghj” and 5-bit block will be encoded to abcdei and those two blocks are combined to 10-bit data. Figure 2.8 show at start up state, the current running disparity initial as negative or RD- state, when the 8-bit data has been feed to an encoder. It will use negative disparity table for encoding. If the encoded 10-bit data has disparity neutral, the state would not change, then the next 8-bit data will use negative disparity table for encoding again. If the 10-bit encoded has positive disparity, the state will be changed to RD+ state and the next 8-bit data will use positive disparity table for encoding.

**Figure 2.8**

*8b/10b State Diagram*

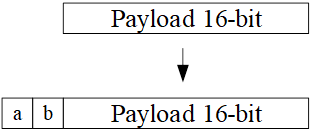


### 16b/18b Line Coding

The improvement of the 8b/10b algorithm was proposed in 1998 by A. Coles from HP (Coles and Cunningham, 1998)), which is called 16b/18b. Its bit-overhead is 12.5%, a half of the 8b/10b bit-overhead.

**Figure 2.9**

*16b/18b Line Coding*



This encoding technique adds two bits preamble to 16 bits data, and those two bits tell a receiver whether the transmitted frame should be inverted or not. That is, when the current RD is positive, and the data frame to be transmitted has positive RD, this frame is inverted in order to decrease the RD. The same handling is also performed when the current RD is negative, and the data frame has negative RD. The performance in term of bit-overhead is better compared to the 8b/10b, but the RL and RD bounds are worse than the 8b/10b. The maximum RL and maximum absolute RD is 42 and 26, respectively. Although it has the RL and absolute RD bounds higher than the 8b/10b, it is used in SONET (Anis et al, 2008). Figure 2.9 shows the encoded frame that a and b are inserted with 16-bit payload. The a and b bit are used to let the receiver know how to decode the data. Table 2.3 16b/18b encoding show encoding scheme of 16b/18b, the payload can be data or control. If control a and b will be only “10” if not, “01”, “00” and “11,” are possible. For example, the transmitting data is data payload 0x000F and the cumulative RD is +10 . the RD of this payload is -8 (), then the a and b will be ‘1’ and ‘1’ respectively.

**Table 2.3**

*16b/18b Encoding*

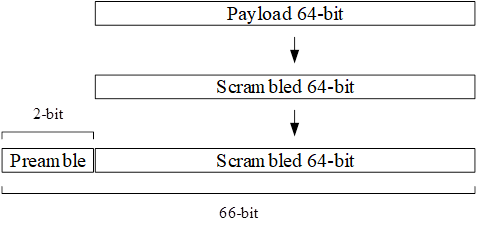
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Payload type | RD payload | RD | a | b | Invert? |
| Data | >0 | <0 | 1 | 1 | N |
| Data | >0 | >0 | 0 | 0 | Y |
| Data | <0 | <0 | 0 | 0 | Y |
| Data | <0 | >0 | 1 | 1 | N |
| Data | - | - | 0 | 1 | N |
| Control | 0 | - | 1 | 0 | N |

### 64b/66b Line Coding

Another popular line coding is 64b/66b (Mohapatra et al, 2017) (Balasubramanian et at, 2011) . It is used in IEEE P802.3ae standard for 10Gb Ethernet. This technique also adds two bits preamble to 64-bits scrambled data frame. The added two bits indicate whether the current frame is either a control frame or a data frame. The preamble can be “01” and “10”, and if the preamble is “00” or “11”, then it means errors occur in the current frame. The scramble polynomial that is used in this line coding is X^58+X^19+X^0 and initial value is not needed because it run continuously on all payload bit. The Figure 2.10 shows the step to encode 64b/66b.

**Figure 2.10**

*64b/66b Line Coding*



The bit-overhead of this line coding is only 3.125%. However, this technique can control only RL bounds. That is, the maximum RL is guaranteed to be 64, but the absolute RD can be infinite.

### 64b/67b Line Coding

Furthermore, 64b/67b is used in Interlaken protocol Figure 2.11 that was invented by Cisco Systems and Cortina Systems in 2006. The encoding scheme is almost the same as 64b/66b, but there is an additional 1-bit to indicate whether the 64-bit data must be inverted or not and the scrambling technique for 64b/66b is self-synchronous but the 64b/67b will send scramble state to receiver. This encoding makes it possible to bound the RD within +/-96, which cannot be done by the 64b/66b. However, to guarantee the RD bounds, it has larger bit-overhead 4.687%, 1.562% larger than the 64b/66b.

**Figure 2.11**

*64b/67b Line Coding*



### Another Line Coding

In 1983, Bosch (BOSCH, 1991) developed a serial communication protocol, called Controller Area Network or CAN Bus. This protocol uses a dynamic handling of communication frames. For example, suppose that the RL bound is five. If the transmitted data have the same bits whose length is longer or equal to five, then the inverted bit will be inserted and transmitted. This technique is called bit stuffing. The bit stuffing simply guarantees the RL bounds. In 2015 J. Saade (Saade et al, 2015) proposed a new line coding that can control RL and RD by using bit stuffing and inverting data with a bit indicator.

Their technique reduces the bit-overhead very effectively. For example, the bit overhead to achieve the same RL and RD bounds as 8b/10b is reduced to 17.4%.

128b/130b (Figure 2.12) is used in PCIe 3.0 and 4.0. The encoding scheme is as 64b/66b but doubling the payload from 64 to 128bit and use a different scrambling polynomial. However, the frame sync still has only two bits. Thus, the overhead from line coding of 128b/130b can be reduced from 3.125% of 64b/66b to 1.5625%.

**Figure 2.12**

*128b/130b Line Coding*



128b/132b is the modification of 128b/130b shown in Figure 2.13 and it is used in USB3.1. and USB3.2 the extra frame sync two bits have been added to reduce the undetected error. The first 4 bits is “1100” for control block and “0011” for data block. If the one of these 4 bits get flipped, the design will make it correct and it will not go to recovery mode. However, if there are two bits flipped, the receiver can detect the problem and go to recovery mode. The USB3.2 is using same protocol and line coding but they increase speed by using multi-lane technique.

**Figure 2.12**

*128b/130b Line Coding*



## Interlaken Protocol

### Overview of the Protocol.

Interlaken (Interlaken, 2008) (Shekhar, 2015) is a high-speed channelized point-to-point or chip-to-chip interface. The main features are listed below.

• Support for 256 communications channels, or up to 64K with channel extension

• A simple control word structure to delineate packets, similar in function to SPI4.2

• A continuous Meta Frame of programmable frequency to guarantee lane alignment, synchronize the scrambler, perform clock compensation, and indicate lane health

• Protocol independence from the number of SerDes lanes and SerDes rates

• Both out-of-band and in-band per-channel flow control options, with a simple Xon/Xoff semantic

• 64b/67b data encoding and scrambling

• Performance that scales with the number of lanes

**Figure 2.14**

*Interlaken Word Format*



The protocol has only two fundamental structures, the data transmission format, and the Meta Frame. Transmission data is segmented into bursts. Each burst is covered by two control words, before and after. The Meta Frame is used to supported transmission of the data. It contains with a set of four unique control words.

Figure 2.14 shows word formats of the Interlaken Protocol, there are Data Word and Burst/Idle Control Word. The Data Word contains with eight data bytes and three framing bits. The Burst/Idle Control Word is used to control transmission including with burst control, start-of-packet, end-of-packet, Flow control, Channel, etc. the framing bits will be used to indicate whether this word is Data Word or Burst/Idle Control Word and it should be flipped at the receiver or not. At initial state, the transmitter and receiver need to synchronize each other by the transmitter keep transmit an Idle Word to receiver.

Figure 2.15 shows transmitter operation flowchart. After reset, Transmitter cannot transmit data immediately, it needs to synchronize to receiver by transmit an Idle Words to receiver until it gets locked. Also, the transmitter will check flow control (XON) before transmit data.

**Figure 2.15**

*Transmit Interface State*



**Figure 2.16**

*Receive per Lane State*



Receiver operation flowchart shows in Figure 2.16, After reset, the transmitter will transmit Idle Words until CDR and 64b/67b word boundary are locked. Then, scramble state must be synchronized before receiver is ready to receive data from transmitter. In case of the receiver lose CDR, word boundary or scramble synchronization. The receiver will go to reset state and needed to re-synchronize CDR, Word Boundary and scramble again.

### 64b/67b Encoding in Interlaken Protocol

As mentioned earlier, the 64b/67b line coding has been adopted from 64b/66b line coding, that’s used in IEEE 802.3ae 10 Gigabit Ethernet. It uses scrambling technique to avoid DC-Balance problem and use two unscrambled bits (sync bit) to specify control or data word and guarantee the RL to 64 bits. The sync bit can be “01” for a data word and “10” for a control word. The combination “00” and “11” will cause an error in word format. When the receiver is searching for a valid pattern in data or control word, it will check on these bits and check until 64 correct matched then the word boundary lock will be true. If the receiver loses word boundary lock (the sync bits are not correct) the receiver will search for a valid pattern and check until 64 correct matched again. The 64b/67b also use this searching pattern scheme to lock word boundary but the inversion bit was added in front of those two sync bits.

**Table 2.4**

*Inversion bit Interpretation*

|  |  |
| --- | --- |
| Bit 66 | Interpretation |
| 0 | Bits [63:0] are not inverted; the receiver may process this word without modification |
| 1 | Bits [63:0] are inverted; the receiver must un-invert before processing this word |

**Table 2.5**

*Sync bit Encoding*

|  |  |
| --- | --- |
| Bit 66 | Interpretation |
| 001 | Data Word, no inversion |
| 010 | Control Word, no inversion |
| 101 | Data Word, bits [63:0] are inverted |
| 110 | Control Word, bits [63:0] are inverted |
| All others | Illegal states |

**Figure 2.17**

*64b/67b Word Boundary Lock*



The 64b/67b has 64 data bit and combination of inversion bit and sync bits that is three bits. In total is 67 bits. The combination of inversion and sync bits can be only 50% of the possible combinations (4 values) the all others are illegal to use. Table 2.5 Sync bit encoding show possible combinations’ meaning of inversion and sync bits. The example of encoded data words shows in Table 2.5 Sync bit encoding. Figure 2.17 shown the flow diagram of 64b/67b word boundary lock. After reset state, it will select 67-bit candidate and check the sync word is correct or not. If yes, the counter will be checked whether it was sync for 64 times and then the status will be change to Word Lock. Otherwise the counter will be reset to 0 and it needs to check for correct sync 64 times again.

**Table 2.6**

*64b/67b Encoding Example*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  | Encoded Data | | |  |
| I | Raw Data | Bit 66 | Bit [65:64] | Bit  [63:0] | CRD |
| 1 | 0x0000000000000000 | 0 | 01 | 0x0000000000000000 | -65 |
| 2 | 0x0000000000000000 | 1 | 01 | 0xFFFFFFFFFFFFFFFF | 0 |
| 3 | 0x000000000000FFFF | 0 | 01 | 0x000000000000FFFF | -33 |
| 4 | 0xFFFFFFFFFF000000 | 0 | 01 | 0xFFFFFFFFFF000000 | -18 |
| 5 | 0xFFFFFFFFFFFFFFFF | 0 | 01 | 0xFFFFFFFFFFFFFFFF | 45 |
| 6 | 0x0000000000000000 | 0 | 01 | 0x0000000000000000 | -20 |
| 7 | 0x00000000AAAAAAAA | 1 | 01 | 0xFFFFFFFF55555555 | 13 |

Example encoding and decoding of the 64b/67b show in Table 2.6 and Table 2.7, all data are assumed to be data word then the sync bit always “01”. The encoding step is followed from the Interlaken Protocol definition 5.4.2 (Interlaken, 2008) page 26 that is “if the new word and the existing disparity both have the same sign, the bits within the new word are inverted.

Then if the CRD of current state and the disparity of incoming data has the same sign, the inversion bit or bit 66 will be set to ‘1’ and the data must be flipped. The iteration 1 from Table 2.6 show the first raw data is 0x0000000000000000, the CRD before transmit data is zero then the inversion bit will not set and the CRD is -65. At Iteration 2, raw data is 0x0000000000000000 again but the current CRD value -65 and the disparity of data in iteration 2 is -64, then the inversion bit will be set to 1 and data will be flipped. The decoding is quite simple, the decoder just checks the inversion bit to decide whether data will be flipped or not. The decoder does not need to calculate CRD because it will not use CRD in decoding process.

**Table 2.7**

*64b/67b Encoding Example*

|  |  |  |  |
| --- | --- | --- | --- |
| Encoded Data | | | Decoded Data |
| Bit 66 | Bit [65:64] | Bit [63:0] | Bit [63:0] |
| 0 | 01 | 0x0000000000000000 | 0x0000000000000000 |
| 1 | 01 | 0xFFFFFFFFFFFFFFFF | 0x0000000000000000 |
| 0 | 01 | 0x000000000000FFFF | 0x000000000000FFFF |
| 0 | 01 | 0xFFFFFFFFFF000000 | 0xFFFFFFFFFF000000 |
| 0 | 01 | 0xFFFFFFFFFFFFFFFF | 0xFFFFFFFFFFFFFFFF |
| 0 | 01 | 0x0000000000000000 | 0x0000000000000000 |
| 1 | 01 | 0xFFFFFFFF55555555 | 0x00000000AAAAAAAA |

### Meta Frame

The Interlaken introduces the concept of a Meta Frame (Figure 2.18). It is defined as the per-lane set of several control word those are Synchronization, Scrambler State, Skip and Diagnostic words, along with the payload data carried on each lane. The Meta Frame is a combination of data payload and one set of Synchronization, Scramble, Skip and Diagnostic words. Each has its own role to maintain data communication for Interlaken protocol.

However, with Meta Frame concept, there are additional overhead to data payload from Synchronization, Scramble, Skip and Diagnostic words. But those word are not sent frequently then; they consume a minimal of interface bandwidth and it depends on MetaFrameLength. For example, if the MetaFrameLength is 2K words the worst-case overhead is only 0.20%.

**Figure 2.18**

*Interlaken Meta Frame*



* SY = Synchronization: Used to align the lanes of the bundle
* SS = Scramble state: Used to synchronize the scrambler
* SK = Skip: Used for clock compensation in a repeater
* Payload = Control and Data transported by the interface
* DI = Diagnostic: Provides per-lane error check and optional status message

### Commercial Product With Interlaken Protocol

The commercial products from leading companies are show in Table 2.8 with several kind of product such as an IP core and network processors

**Table 2.8**

*Interlaken Commercial Products*

|  |  |
| --- | --- |
| Company | Product description |
| EZ-CHIP | * NP-4 Network Processor – 100G NPU with TM |
| Intel | * Intel FPGA- Interlaken IP Bandwidth 25G to 300G |
|  |  |
| Company | Product description |
| MoreThanIP | * IL256: 20Gbps to 70/200Gbps Interlaken IP Core for ASIC and Altera FPGA * XLIL408: Programmable Quad 10Geth / 40Geth Bridge with IEEE1588 support |
| Microsemi | * PM5990 DIGI-G4 Multi-service OTN Processor for 200G / 400G line cards * PM5992 META-240G 24x10G / 6x40G / 2x100G OTN Wrapper and Ethernet PHY * PM5420 HyPHY 20G High-Capacity Single-Chip Multi-Rate Multi-Protocol PHY * PM5422 META 20G Dual Port Single-Chip PHY with Integrated OTN/10GE * PM5450 HyPHY 20Gflex High-Capacity Single-Chip Multi-Rate Multi-Protocol PHY with Flexible ODUk Support * PM5440 DIGI 120G High-Capacity 12x10G / 3x40G / 100G Multi-Service OTN Processor * PM5441 DIGI 60G High-Capacity 6x10G / 1x40G Multi-Service OTN Processor * PM5442 META 120G High-Capacity 12x10G / 3x40G / 100G OTN Wrapper and Ethernet PHY |
| Open-Silicon | * Interlaken IP Core supporting up to 1.2 Tbps bandwidth |
| Tabula | * ABAX 3PLD Product Family |
| Xilinx | * Hard and Soft Interlaken IP cores for FPGAs. Supported FPGAs include: UltraScale+, UltraScale and 7-series. * Interface bandwidth: 5G to 600G |
| Company | Product description |
| Achronix | * Speedster 22i FPGA with Hard Interlaken IP core |
| Bay microsystems | * Network Processor and Traffic Management Solutions |
| Cavium Networks | * Embedded Multicore MIPS64 Processors for networking, wireless and storage applications |
| Inphi | * CS1999 – 40Gb/s OC768/STM256 Framer * CS3477 – Quad 10G Ethernet MAC with SFP+ * CS3472 – 24 Port Gigabit Ethernet MAC * CS605x – 40G and 100G Optical Transport Processors |
| IDT | * Next generation packet header processing solutions |
| Lattice | * Interlaken FPGAs |
| Metronime | * NFP-3200 family of network flow processors |

## Summary

The RD and RL is major parameters that effect to signal integrity of transmission line. Several line coding can control and guarantee both, but some cannot such as 64b/66b that could not guarantee RD.

**Figure 2.19**

*64b/i67b, the Proposed Line Coding*



The proposed technique 64b/i67b shows in the rest of this paper will further improve its maximum number of running disparity without increasing bit-overhead and give a better result than 64b/67b. The RL value of 64b/67b is guaranteed by two sync bit that indicate data or control word (“01” or “10”) thus the 64b/i67b is also using the same two sync bit and the RL value are controlled too. The Figure 2.19 shows an example for encoding of proposed technique that we separate the 64-bit payload to two 32-bit blocks.

# METHODOLOGY

The 64b/67b uses inversion bit for a single word, if the new word and the existing disparity both have the same sign, the new word will be flipped, and the inversion bit is set. So, the disparity for each word will be combined with the next word to minimize CRD. However, our proposed change the meaning of inversion bit to indicate only one in two blocks of word to let the disparity within each word can be combined to minimize the CRD in each word instead of waiting for the next word.

## The Proposing Technique Called 64b/i67b

A new technique to decrease the maximum number of running disparity is 64b/i67b, it separates the payload [63:0] into two blocks [63:32] and [31:0] and check the running disparity of the two blocks. Data can be grouped the two possibilities for the running disparity of those two blocks as below.

* The RD of first block higher than second block or the RD of first block lower than second block.
* The RD of first block and second block is equal

**Figure 3.1**

*Example of 64-bit Payload*



Figure 3.1 shows an example of 64-bit payload that the number of ‘0’ and ‘1’ is 25 and 39 respectively. The disparity for this payload is (-25) + 39 = 14. Then, Figure 3.2 shows an example when separate the 64-bit payload to two 32-bit blocks. The disparity for those blocks is shown below

* Block A
  + Number of ‘0’ is 17
  + Number of ‘1’ is 15
  + CRD is (-17) + 15 = -2
* Block B
  + Number of ‘0’ is 8
  + Number of ‘1’ is 24
  + (-8) + 24 = 16

**Figure 3.2**

*Example of Separate 64-bit to two Blocks*



The Block A’s disparity is -2 and Block B’s disparity is 16. Then, the disparity for this 64-bit payload is 14. The total for 64-bit payload when we separate to two blocks is as same as when we did not separate it.

After getting the disparity for each block, the result of will be calculated based on highest running disparity block. Equation 3.1 has two terms of those are and , it was calculated by using , disparity and inversion bit. The inversion bit has been fixed to (-1) when the inversion bit is not set and +1 if the inversion bit is set. The Figure 3.2 shows block B has highest absolute value of running disparity for this payload.

**Equation 3.1**

Then, if the || higher then || the inversion bit will be set to ‘1’ but if not, it will be set to ‘0’. That means the inversion bit is used to indicate the highest block whether it is flipped or not and the lower block will not be changed. For the decoder, it is also separated the data word to two blocks and the inversion bit will be used to indicate a highest block to be flipped. If the absolute value of block A and block B is equal, the inversion bit will be fixed to indicate block B. For example, let us assume the sequence for data word as Table 3.1. There are 8 example iterations in Table 3.1 and every iteration has same 64-bit data payload those are 0x000000000000000F and the for each payload is (-60) + 4 = -56. So, if we transmit these data without encoding the will get increasing and it could not control.

**Table 3.1**

*Example of Raw 64-bit Data*

|  |  |  |
| --- | --- | --- |
| Iteration | 64-bit data | CRD after transmitted |
| 1 | 0x000000000000000F | -56 |
| 2 | 0x000000000000000F | -112 |
| 3 | 0x000000000000000F | -168 |
| 4 | 0xFFFFFFFFFFF0000F | -136 |
| 5 | 0xFFFF00000000000F | -160 |
| 6 | 0x00000000FFFFFFFF | -160 |
| 7 | 0xFFFFFFFFFFFFFFFF | -96 |
| 8 | 0x0000000000000000 | -160 |

Table 3.2 shows an 64b/67b encoding example by using data sequence from Table 3.1, each encoding word contains with three major parts.

1. Inversion bit
2. Sync bit (2-bit)
3. Data (64-bit)

## 64b/67b Encoding and Decoding

At iteration 1 from Table 3.2, the CRD start from 0 then after transmitting iteration 1, the CRD will be

The inversion bit of iteration 1 will be set to 0 because the CRD is start from 0 and there is no sign (+, -). The Sync bit is always 0 because the protocol uses only “01” and “10” for both control and data word, the others are inhibited and will not be used, we will not mention about sync bit in our calculation. The Data is 64-bit data that related to inversion bit, in this iteration the inversion bit is not set, the 64-bit data will not be flipped then the original data will be used to calculate CRD for this iteration. The 64-bit data in iteration 1 has 60-bit of ‘0’ and 4-bit of ‘1’. So, the total CRD after transmit iteration 1 is (-57). Then, the iteration 2, current CRD is (-57) and CRD of iteration 2’s 64-bit data is

In this situation, from the Interlaken Protocol definition “if the new word and CRD both has the same sign, the bit within the new word are inverted”. Then, the current CRD is (-57) and 64-bit data CRD is (-56) since both has same sign, the inversion bit will be set to ‘1’ and 64-bit data will be flipped. The iteration 2 in Table 3.2 shows inversion bit has been set to ‘1’ and 64-bit data was flipped. Then at the end of iteration 2’s transmission, the CRD will be 0, as shown below

**Table 3.2**

*64b/67b Encoded Example*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Iteration | 66 | 65:64 | 63:0 | CRD |
| 1 | 0 | XX | 0x000000000000000F | -57 |
| 2 | 1 | XX | 0xFFFFFFFFFFFFFFF0 | 0 |
| 3 | 0 | XX | 0x000000000000000F | -57 |
| 4 | 0 | XX | 0xFFFFFFFFFFF0000F | -26 |
| 5 | 1 | XX | 0x0000FFFFFFFFFFF0 | -1 |
| 6 | 0 | XX | 0x00000000FFFFFFFF | -2 |
| 7 | 0 | XX | 0xFFFFFFFFFFFFFFFF | 61 |
| 8 | 0 | XX | 0x0000000000000000 | -4 |

At I3 the 64-bit data are same as I1 and I2, the encoding process is the same as the I1 but in I4 the 64-bit data change to 0xFFFFFFFFFFF0000F and the I4 64-bit data disparity is

So, the CRD and I4’s disparity has different sign the inversion bit for I4 will not be set and the 64-bit data wouldn’t be flipped. After this stage, the CRD is

Let us keep continue to I5, with 64-bit data is 0xFFFF00000000000F the disparity is (-24). Here the CRD and the new word disparity has the same sign, the inversion bit will be set to ‘1’ and the 64-bit data must be flipped. Then, the CRD after transmitting the I5 is

**Table 3.3**

*64b/67b Decoded Example*

|  |  |  |  |
| --- | --- | --- | --- |
| Iteration | INV bit | Encoded data | Decoded data |
| 1 | 0 | 0x000000000000000F | 0x000000000000000F |
| 2 | 1 | 0xFFFFFFFFFFFFFFF0 | 0x000000000000000F |
| 3 | 0 | 0x000000000000000F | 0x000000000000000F |
| 4 | 0 | 0xFFFFFFFFFFF0000F | 0xFFFFFFFFFFF0000F |
| 5 | 1 | 0x0000FFFFFFFFFFF0 | 0xFFFF00000000000F |
| 6 | 0 | 0x00000000FFFFFFFF | 0x00000000FFFFFFFF |
| 7 | 0 | 0xFFFFFFFFFFFFFFFF | 0xFFFFFFFFFFFFFFFF |
| 8 | 0 | 0x0000000000000000 | 0x0000000000000000 |

For the left iteration (6,7 and 8), the encoding results are show in Table 3.2. This is an example for encoding the original 64b/67b. To decoding the 64b/67b, the process of receiver is very easy, the receiver just check the inversion bit, if it set to ‘1’ the 64-bit data will be flipped, if it ‘0’, there is nothing to do with the 64-bit data. Table 3.3 shows the encoded data from Table 3.2 and decoded data in next column by using an inversion bit (INV bit) from Table 3.2. The decoded data are correct to original data in Table 3.1.

## 64b/i67b Encoding and Decoding

We are proposing a new technique called 64b/i67b that is a modification from the original 64b/67b, but it will improve a dc-balanced performance without additional bit overhead. Table 3.4 shows an encoding example by using raw data from 64-bit data from Table 3.1. At I1 stage, the 64-bit data is 0x000000000000000F and we separate the data into two blocks that are Block A and Block B.

Then, we calculate a disparity for both blocks. Those are -32 and -24 for Block A and Block B respectively, in this example. Absolute value of Block A and Block B are 32 and 24. So, the inversion bit for this encoding word will be used to indicate the highest disparity block that is Block A with = 32.

The Equation 3.1 will be used to calculate the best case for . At this state, all parameters are shown below



From the calculation the has better result than , meaning the flipped 32-bit data in Block A give a better result than not flip. So, the encoding result of I1 will be 0xFFFFFFFF0000000F. The inversion bit will be set to ‘1’ because the Block A (highest disparity) must be flipped to maintain the best result in .

The I2 has as same 64-bit data as the I1 state, the disparity of both blocks will be same to the I1 but different in because after I1 state the is 9. So, the candidate to be indicated by inversion bit is Block A. The and will be

The will give the best result for this iteration. The inversion bit for this data word will be set to ‘1’ again. I3 also has the same payload data to I1 and I2, the and are

Next state is I4, the 64-bit data is 0xFFFFFFFFFFF0000F and the disparity for Block A and Block B are 32 and 0 respectively. The value 32 affect to Block A that will be indicated by an inversion bit. The and of I4 are

The value (-4) is better than 58 then the will be used, and the invasion bit must be set to ‘1’. The I5 has a different from I1 to I4 that is the data is 0xFFFF00000000000F the disparity for Block A and Block B are 0 and (-24) respectively. Here the Block B will be indicated by inversion bit. The and of I5 are

The I6 is an example when the disparity of Block A and Block B are the same. In this case, there is no highest disparity value to be selected because both are equal. The Block B will be fixed to be chosen in this situation. Then, the and of I6 are

**Table 3.4**

*64b/i67b Encoding Example*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Iteration | 66 | 65:64 | 63:32 | 31:0 | CRD |
| 1 | 1 | XX | 0xFFFFFFFF | 0x0000000F | 9 |
| 2 | 1 | XX | 0xFFFFFFFF | 0x0000000F | 18 |
| 3 | 1 | XX | 0xFFFFFFFF | 0x0000000F | 27 |
| 4 | 1 | XX | 0x00000000 | 0x000FFFF0 | -4 |
| 5 | 1 | XX | 0xFFFF0000 | 0xFFFFFFF0 | 21 |
| 6 | 0 | XX | 0x00000000 | 0xFFFFFFFF | 20 |
| 7 | 1 | XX | 0xFFFFFFFF | 0x00000000 | 21 |
| 8 | 1 | XX | 0x00000000 | 0xFFFFFFFF | 22 |

The give the best result then the inversion will not be set to ‘1’ the 64-bit data is not flipped. For the left of iterations, it has been shows in Table 3.4 and the comparison of after transmitting each iteration between 64b/67b and 64b/i67b has been show in Table 3.5.

**Table 3.5**

*Comparison Between CRD of 64b/67b and 64b/i67b*

|  |  |  |
| --- | --- | --- |
| Iteration | 64b/67b’s CRD | 64b/i67b’s CRD |
| 1 | -57 | 9 |
| 2 | 0 | 18 |
| 3 | -57 | 27 |
| 4 | -26 | -4 |
| 5 | -1 | 21 |
| 6 | -2 | 20 |
| 7 | 61 | 21 |
| 8 | -4 | 22 |

Decoding the 64b/i67b is a bit different from 64b/67b that the 64b/67b just checking the inversion bit to decide whether the receiver should flip the 64-bit data or not. However, the 64b/i67b check the inversion bit. If it ‘0’ the data will not be flip and the receiver will not find disparity for the both blocks but if it set to ‘1’ it needs to find the highest disparity of Block A and Block B and flip the block that has a highest absolute value of disparity. The decoded of 64b/i67b from encoding data in Table 3.4 is show in Table 3.6

The Figure 3.3 and Figure 3.4 show pseudo code for normal 64b/67b encoding and decoding. The encoder will calculate disparity for 64-bit data (Line 2: Data’s Disparity) of the transmitting data and use it to compare with current CRD. If the flipped data has better result for CRD (minimized CRD) the inversion bit or IVB will be set and Data will be flipped. It means, if the current CRD has the same sign with Data’s disparity, the inversion bit (IVB) will be set and data must be flipped. Otherwise if there are different sign of CRD and Data’s disparity.

**Table 3.6**

*64b/i67b Decoding Example*

|  |  |  |  |
| --- | --- | --- | --- |
| Iteration | INV bit | Encoded data | Decoded data |
| 1 | 1 | 0xFFFFFFFF0000000F | 0x000000000000000F |
| 2 | 1 | 0xFFFFFFFF0000000F | 0x000000000000000F |
| 3 | 1 | 0xFFFFFFFF0000000F | 0x000000000000000F |
| 4 | 1 | 0x00000000000FFFF0 | 0xFFFFFFFFFFF0000F |
| 5 | 1 | 0xFFFF0000FFFFFFF0 | 0xFFFF00000000000F |
| 6 | 0 | 0x00000000FFFFFFFF | 0x00000000FFFFFFFF |
| 7 | 1 | 0xFFFFFFFF00000000 | 0xFFFFFFFFFFFFFFFF |
| 8 | 1 | 0x00000000FFFFFFFF | 0x0000000000000000 |

**Table 3.7**

*Encoding Probability of 64b/67b*

|  |  |  |
| --- | --- | --- |
| CRD | Data’s Disparity | IVB |
| ~ | ~ | 1 |
| + | ~ | 0 |
| - | ~ | 1 |
| ~ | - | 0 |
| + | - | 0 |
| ~ | + | 1 |
| + | + | 1 |
| - | - | 1 |

The IVB will be set to ‘0’ and Data would not be flipped. From Figure 3.3 line 2 to line 9, it will check that in case of the Data’s Disparity is zero, means there is no sign for Data’s disparity. Then, if the CRD has + sign. The encoder will set IVB to ‘0’. This will help to insert (-1) to CRD and decrease CRD absolute value. Line 10 to 16 is the case when Data’s Disparity has plus or minus sign, this will describe in Table 3.7.

Table 3.7 show the 64b/67b encoding probability between CRD and Data’s Disparity with different sign, the ~ means there is no sign, it is zero disparity. When the CRD or Data’s disparity has zero value, the IVB will force to make an invert sign to disparity. For example, if CRD is + and Data’s Disparity is ~ the IVB will be ‘0’ or (-1) to decrease CRD value. Other cases are following the (Interlaken, 2008) rule. Line 18, in Pseudo code 3.1 in the pseudo function that send the encoded data included with Inversion bit (IVB), Sync bit and data. Line 19 shows after transmit encoded data, the CRD will be updated with function UpdateCRD().

To decode the 64b/67b, it shows in Figure 3.2. Line 2 to 4 is extracting parameter from input data. The input data of decoding process is 67-bit and this include with IVB[ bit 66], Sync[bit 65:64] and Data[bit 63:0]. Then, Line 5, the IVB is checked whether if it set to ‘1’ the Data will be flipped. If not, there is nothing to do. Line 8 is Store(Sync, Data), this function used to pass decoded data to another process.

**Figure 3.3**

*Pseudo Code of 64b/67b Encoding*



The Figure 3.5 is encoder pseudo code of our proposed technique 64b/i67b. Each block's has been calculated by function GetsDisparity() in lines 2 and 3 and keep in x and y parameters. Both disparity values will be compared to decide whether which block must be flipped or not. The m and n parameters have been calculated by combining value with x and y values. The Inversion bit (IVB) has been calculated by comparing the m and n values whether which one can give a lowest . Line 4 is comparing absolute value of x and y parameter.

The x is disparity of Data[31:0] or Block B and y is disparity of Data[63:32] or Block A.. Line 5 and 6 is the calculation to get both value of when Block B has highest disparity value stored in m and n parameters. Line 7 is copy data in Block A from Data to iData that will be use later. Also with Line 8, but the Block B is flipped before store in iData. Line 10 to 13, it is the case when absolute value of y has higher disparity value than x or Block A has highest disparity value. This will do like the pseudo code Line 5 to 8 but instead of flipping x, it will be flipping y value to find m and n parameters. And the Block B will be flipped and store in iData. Line 15, here the m and n are compared to find minimize absolute . If m has lower or equal, then n (not flip give lower ) the IVB bit will set to ‘0’. If not, the IVB will be set to ‘1’ and data in iData is moved to Data.

The probability of and Block A and Block B disparity values show in Table 3.8.

**Figure 3.4**

*Pseudo Code of 64b/67b Decoding*



Also, the Figure 3.6 extract receiving 67-bit data to IVB, Sync and Data. If the IVB is '0' the Data will be stored to memory. If it is not, the x and y will be calculated to get disparity for the both Blocks (A and B). Line 6 is x calculation to get disparity value of block B and Line 7 for Block A. Line 8 is comparing of absolute value of x and y, if x greater or equal than y, the Block B will be flipped otherwise the Block A is flipped.

**Table 3.8**

*Encoding Probability of 64b/i67b*

|  |  |  |  |
| --- | --- | --- | --- |
| x:y | m:n | IVB | Block to be inverted |
| = | = | 0 | No |
| = | > | 1 | B |
| = | < | 0 | No |
| > | = | 0 | No |
| > | > | 1 | B |
| > | < | 0 | No |
| < | = | 0 | No |
| < | > | 1 | A |
| < | < | 0 | No |

**Table 3.9**

*Decoding Probability of 64b/i67b*

|  |  |  |
| --- | --- | --- |
| IVB | x:y | Block to be inverted |
| 0 | = | No |
| 0 | > | No |
| 0 | < | No |
| 1 | = | B |
| 1 | > | B |
| 1 | < | A |

**Figure 3.5**

*Pseudo Code of 64b/i67b Encoding*



**Figure 3.6**

*Pseudo Code 3.4 64b/i67b Decoding*



## Probability

### Probability of 64b/67b

The best value for the transmission is zero. Then, the chance to get the best when the total number of ‘1’ bit equal to ‘0’ bit. The probability for the 64b/67b can be finding by using probability equation as Equation 3.2.

**Equation 3.2**

The “Number of ways it can happen” or is the disparity value in the data word which related to number of ‘0’ and ‘1’ within the word. So, the ways it can happen can be calculated from combination equation shows in Equation 3.3. The is “Total number of outcomes” and it is which our data word is 64 bits.

**Equation 3.3**

is 64 because the word has 64bits and is number of ‘1’ in word that can be sum to ‘0’ and give result equal to . For example, if is 0 the number of ‘1’ in the word that will give best value is 32 (number of ‘1’ is 32 and number of ’0’ is 32, the total disparity is (32)+(-32) = 0). Here, is 32. Then, we combine the Equation 3.2 and 3.3. The probability of the best () of 64b/67b shows in Equation 3.4.

**Equation 3.4**

So, the probability of word’s disparity that the value will be equal to 0, it will be calculated by using equation 3.4.

### Probability of 64b/i67b

Instead of waiting for next data word to be compensate . The 64b/i67b can be self-compensate to reduce to be zero. As 64b/67b the possibility to obtain is when number of ‘1’ and ‘0’ in 64-bits are equal (+32-32 = 0). For the 64b/i67b, this technique increase chance to get to be zero by flipping one block. For example, if of block A and block B are 32, the block A can be flip to be -32 and the total will be zero. The 64 bits will be separated to two blocks with 32 bits each. The probability can be calculated by adding probability when with probability when block A and block B are equal. From equation 3.2 the calculation will be changed to

**Equation 3.5**

Where is “Number of ways it can happen in block A”, is “Number of ways it can happen in block B” and . Then, we are finding the same number in block A and block B whatever positive or negative value. So, the equation for one combination value will be.

Where

The is disparity of block A and block B. It starts from 32 to -32. Thus, the can be calculated by

Then, the equation to calculate one probability combination of block A and block B is

**Equation 3.6**

The Equation 3.6 shows probability for only one combination of Block A and Block B. However, the inversion bit allows the value of combination can be more than one value to be zero as show in Table 3.10. It can be more than one combination from 32 to -32. So, the possibility combination can be calculated by

**Table 3.10**

*Example of Block A and Block B Combination*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Block A’s RD | Block B’s RD | CRD | Block A’s iRD | Block B’s RD | CRD |
| 32 | 32 | 64 | -32 | 32 | 0 |
| 30 | 30 | 60 | -30 | 30 | 0 |
| - | - | - | - | - | - |
| - | - | - | - | - | - |
| -30 | -30 | -60 | 30 | -30 | - |
| -32 | -32 | -64 | 32 | -32 | 0 |

The total probability can be calculated by summation of probability of all possibility case. Then, the final equation for 64b/i67b shows in equation 3.8

**Equation 3.8**

The probability for the 64b/i67b technique to give the best result has more probability because the inversion bit helps to invert data within the data word and give more chance than 64b/67b that can be zero when the number of ‘1’ equal to number of ‘0’.

## Implementation

The proposed technique has been implemented into simulator and HDL. The simulator was written in MATLAB to obtain result for every transmitting bit. In addition, HDL is written in Verilog HDL to analyze hardware resource usage to comparing between the 64b/67b and 64b/i67b. Since, this dissertation is proposing the technique to minimize 64b/67b encoding, the 64b/67b is in Layer 1 on OSI model (Li et al, 2011). Then, the implementation of 64b/67b and 64b/i67b will be focused only encode-decode in Layer 1 because the other layers are same. Figure 3.7 shows several states for Interlaken protocol encode-decode state.

The implementation for MATLAB simulation will implement only Scramble, Encoder, Descrambler and Decoder these show in Figure 3.7 in the green and light-orange blocks. For the hardware design, the scrambler and descrambler are not needed. As mentioned earlier, the different between our proposing is encoder and decoder. Then, the hardware design was implemented only the green blocks as shown in Figure 3.7 (Andrea, 2018).

### Simulation With MATLAB

MATLAB code for 64b/66b, 64b/67b and 64b/i67b was implemented for encoder and decoder to investigate results . To get the maximum value from simulation, implementing the encoder alone is enough, but the decoder is needed for checking the correctness of encode-decode process by decoding the encoded data and use decoded data to compare with the raw data. Figure 3.8 and Figure 3.9 show the flowchart of simulation process that was implemented and the output from decoding process will be compared to original files that feeds to encoding process.

**Figure 3.7**

*Interlaken Communication Block*



**Figure 3.8**

*MATLAB Encoding Flowchart*



**Figure 3.9**

*MATLAB decoding Flowchart*



### HDL Design

The Encoder and Decoder circuit of 64b/67b and 64b/i67b were designed using Verilog code by reference from Xilinx Kintex-7 series, with 326,080 Logic Cells. XC7K325TFFG900-2 FPGA. The hardware logic for all circuits (64b/67b encoder-decoder and 64b/i67b encoder-decoder) show in Appendix section.

The hardware core has been checked by connecting the output from transceiver to input of receiver, and the output from decoder core will be compared to input data to encoder core as Figure 3.10. The Figure 3.11 and Figure 3.12 show simulation result from Verilog HDL implantation

**Figure 3.10**

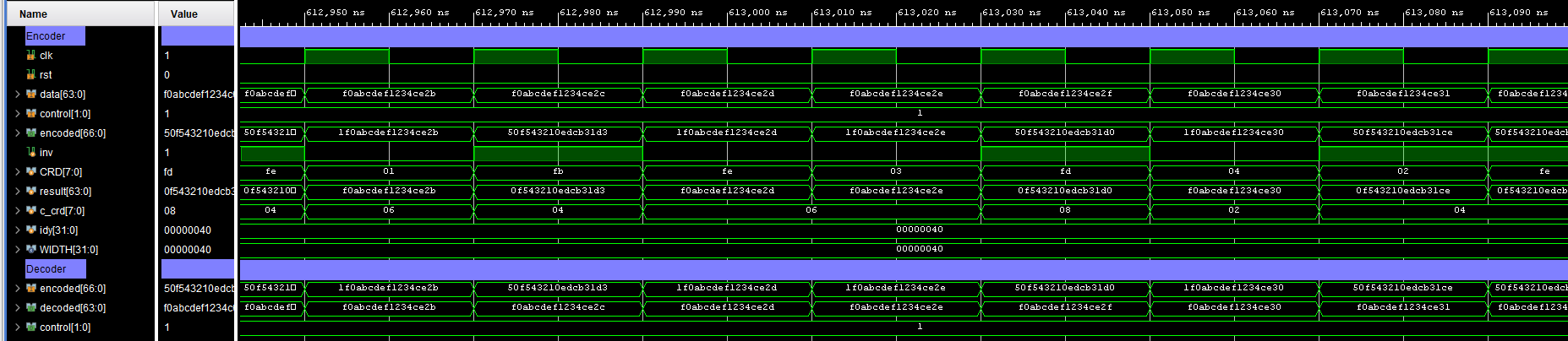
*Encoder-Decoder Testbench Connection*



The resource utilization of the encoder between 64b/67b and 64b/i67b are 176 and 194, that is 0.09% and 0.10% respectively. The utilization of the both decoders are 32 and 159 respectively, that is 0.02% and 0.08%. The utilization of the both encoders are not much different, the 64b/i67b adding a new counter to separate between bit [63:32] and [31:0] instead of [63:0] on 64b/i67b. On the other hand, the decoder is quite different due to the 64b/67b decoder has no counter, it just detects the inversion bit to decide the incoming word must be flipped or not, but the 64b/i67b needs to add counter to indicate which part of the word will be controlled by the invasion bit. In this reason, the resource utilization of the decoder is quite high. Table 3.11 and Table 3.12 show the Vivado synthesis’s resource utilization of the encoder and decoder. The modification will flip only half of word and this will be able to minimize the of the transmission. The synthesized circuits are show in Appendix A section.

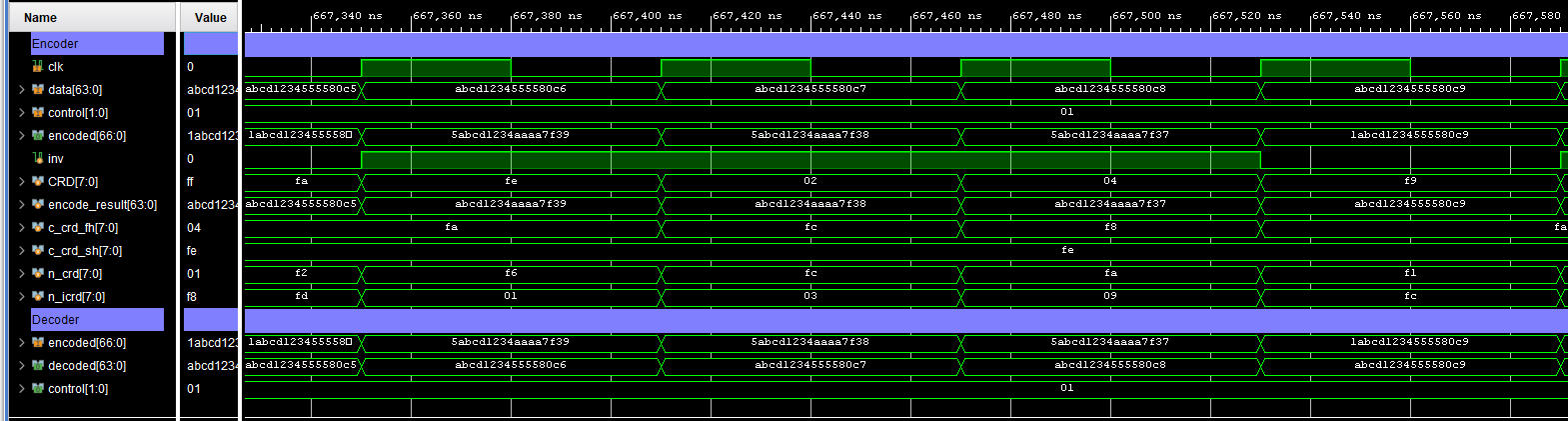
**Figure 3.11**

*Simulation Wave From for 64b/67b*



**Figure 3.12**

*Simulation Wave From for 64b/i67b*



**Table 3.11**

*Encoder Resource Utilization*

|  |  |  |
| --- | --- | --- |
| Resource name | 64b/67b | 64b/i67b |
| LUT | 176(0.09%) | 194(0.10%) |
| FF | 73(0.02%) | 73(0.02%) |

**Table 3.12**

*Decoder Resource Utilization*

|  |  |  |
| --- | --- | --- |
| Resource name | 64b/67b | 64b/i67b |
| LUT | 32(0.02%) | 159(0.08%) |

# RESULTS AND DISCUSSION

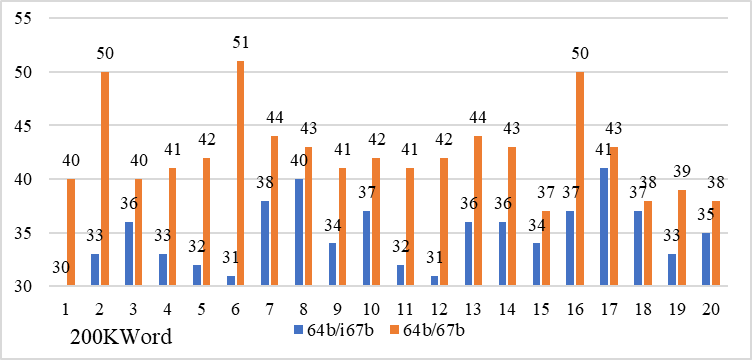
## Purpose

The Experimental Results has been collected from implementation of encoder and decoder with simulation and hardware synthesis by using MATLAB and Xilinx’s Vivado. The FPGA that was using in this experiment is Kintex-7-part number XC7K325TFFG900-2.

The 64b/67b, 64b/66b and the 64b/i67b encoder and decoder’s simulation in MATLAB have been implemented and using random data and Squash compression benchmark to run the testing. The random data uses three different word counts 200K, 400K and 600K word, each type will be simulated for 20 times to find an average. The Squash compression benchmark (Nemerson, 2015) contains with several benchmarks those are Canterbury Corpus, Silesia Corpus, Large Text Compression Benchmark and Snappy. Each benchmark contains with several file types such as PDF, English text, HTML Source, C Source, JPEG Image, A set is composed of Protocol Buffer data, Excel Spreadsheet, Tarred executables of Mozilla, XML and so on. For simulation with random data, every transmitting bit will be setup randomly and pass through the encoder and decoder the results are collected.

**Figure 4.1**

*Random Data at 200K Word*



For simulation with the squash, simulation will start to fetch binary data from file, pass through a scramble circuit and through the encoder and decoder, then the results are collected. We collected running-disparity data in every transmission bit for 64b/67b and the 64b/i67b. For the 64b/66b we did not collect the RD in every transmission bit because we will not use the data from 64b/66b to analyze because this line coding cannot guarantee the RD and it can reach to over 1000 unbalanced-DC. However, the maximum RD value of 64b/66b, 64b/67b and the proposed have been collected from the simulation to compare the performance of RD controlling in line coding technique.

**Table 4.1**

*Simulation Result Comparison of 200K Random Data*

|  |  |  |  |
| --- | --- | --- | --- |
| 200K | 64b/67b | 64b/i67b | Reduce % |
| 1 | 30 | 40 | 25 |
| 2 | 33 | 50 | 34 |
| 3 | 36 | 40 | 10 |
| 4 | 33 | 41 | 19.51 |
| 5 | 32 | 42 | 23.81 |
| 6 | 31 | 51 | 39.22 |
| 7 | 38 | 44 | 13.64 |
| 8 | 40 | 43 | 6.98 |
| 9 | 34 | 41 | 17.07 |
| 10 | 37 | 42 | 11.90 |
| 11 | 32 | 41 | 21.95 |
| 12 | 31 | 42 | 26.19 |
| 13 | 36 | 44 | 18.18 |
| 14 | 36 | 43 | 16.28 |
| 15 | 34 | 37 | 8.11 |
| 16 | 37 | 50 | 26 |
| 17 | 41 | 43 | 4.65 |
| 18 | 37 | 38 | 2.63 |
| 19 | 33 | 39 | 15.38 |
| 20 | 35 | 38 | 7.89 |

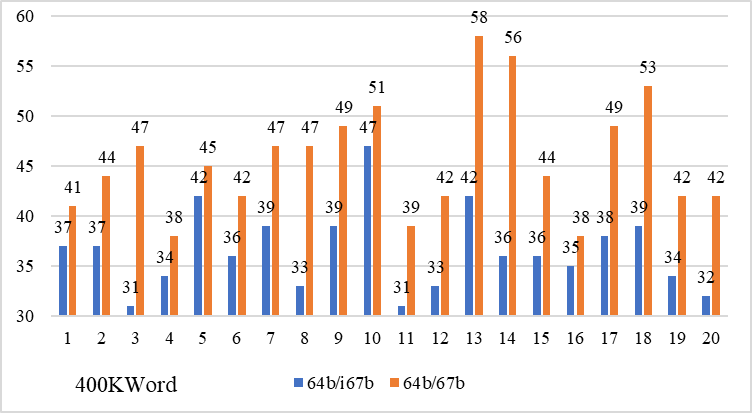
## Random Simulation

Figure 4.1, Figure 4.2, and Figure 4.3 show simulation results for random data at 200K, 400K and 600K word and each will be run at 20 times. The results show the 64b/i67b can reduce the maximum cumulative running disparity from 64b/67b. At 200K word, the maximum running disparity of 64b/67b is 51 and the 64b/i67b is 41, at 400K word, 64b/67b is 58 and the modification is 47 and at 600K word, 64b/67b is 53 and the 64b/i67b is 45. Table 4.1, Table 4.2 and Table 4.3 show the comparison and different percentage between 64b/67b and 64b/i67b.

The average cumulative running disparity within data word are collected and the value are 6.464 for 64b/67b and 4.9437 for 64b/i67b. Thus, from the result, the 64b/i67b can control the to the minimum value better than 64b/67b. Also, with an average value within data word, the average value of 64b/i67b has lower than 64b/i67b.

**Figure 4.2**

*Random Data at 400K Word*



**Table 4.2**

*Simulation Result Comparison of 400K Random Data*

|  |  |  |  |
| --- | --- | --- | --- |
| 400K | 64b/67b | 64b/i67b | Reduce % |
| 1 | 41 | 37 | 9.76 |
| 2 | 44 | 37 | 15.91 |
| 3 | 47 | 31 | 34.04 |
| 4 | 38 | 34 | 10.53 |
| 5 | 45 | 42 | 6.67 |
| 6 | 42 | 36 | 14.29 |
| 7 | 47 | 39 | 17.02 |
| 8 | 47 | 33 | 29.79 |
| 9 | 49 | 39 | 20.41 |
| 10 | 51 | 47 | 7.84 |
| 11 | 39 | 31 | 20.51 |
| 12 | 42 | 33 | 21.43 |
| 13 | 58 | 42 | 27.59 |
| 14 | 56 | 36 | 35.71 |
| 15 | 44 | 36 | 18.18 |
| 16 | 38 | 35 | 7.89 |
| 17 | 49 | 38 | 22.45 |
| 18 | 53 | 39 | 26.42 |
| 19 | 42 | 34 | 19.05 |
| 20 | 42 | 32 | 23.81 |

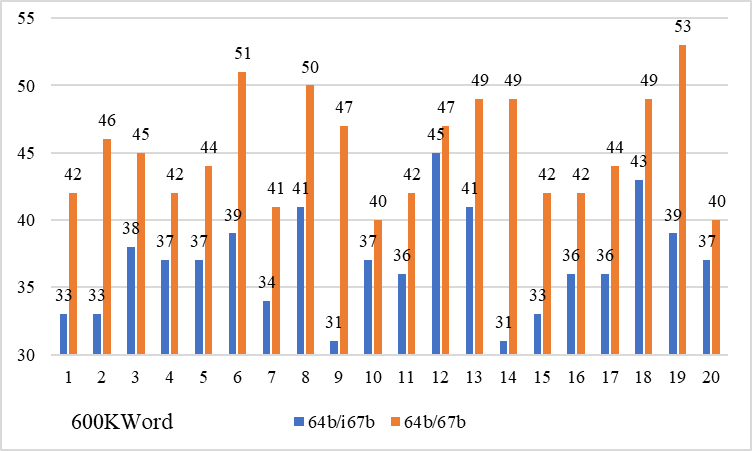
**Table 4.3**

*Simulation Result Comparison of 600K Random Data*

|  |  |  |  |
| --- | --- | --- | --- |
| 600K | 64b/67b | 64b/i67b | Reduce % |
| 1 | 42 | 33 | 21.43 |
| 2 | 46 | 33 | 28.26 |
| 3 | 45 | 38 | 15.56 |
| 4 | 42 | 37 | 11.90 |
| 5 | 44 | 37 | 15.91 |
| 6 | 51 | 39 | 23.53 |
| 7 | 41 | 34 | 17.07 |
| 8 | 50 | 41 | 18.00 |
| 9 | 47 | 31 | 34.04 |
| 10 | 40 | 37 | 7.50 |
| 11 | 42 | 36 | 14.29 |
| 12 | 47 | 45 | 4.26 |
| 13 | 49 | 41 | 16.33 |
| 14 | 49 | 31 | 36.73 |
| 15 | 42 | 33 | 21.43 |
| 16 | 42 | 36 | 14.29 |
| 17 | 44 | 36 | 18.18 |
| 18 | 49 | 43 | 12.24 |
| 19 | 53 | 39 | 26.42 |
| 20 | 40 | 37 | 7.50 |

**Figure 4.3**

*Random Data at 600K Word*



## Using Squash Compression Benchmark Simulation

All files in the benchmark are used to simulate the maximum for 64b/66b, 64b/67b and 64b/i67b. Table 4.4 shows the comparison by detailed in file type, file size and for each line coding, the result sorted by file size from 3.72Kbyte to 3,815Mbyte. Figure 4.4 shows result in bar chart but did not show the 64b/66b because the value of is very high. The result shows almost every 64b/i67b has a better result in minimizing the value. There is only “xargs1” that the proposed technique has higher value than 64b/67b, they are 21 and 26 which file sizes is 4.33KB. The maximum value of 64b/67b from this data set is 52 in WIN10 iso file while the proposed 64b/i67b is 42 and the 64b/66b is 271,119.

However, it is not only maximum RD value that our technique can control and give better results than traditional 64b/67b, but it also controls the RD to be closed to zero running disparity. This gives good result to DC-Balance controlling. Including with “xargs1”, even the maximum RD value of the proposed has higher value than normal 64b/67b but in detail the proposed is controlled the RD to nearly zero running disparity better than normal 64b/67b.

Figure 4.5 shows the comparison simulation result of 64b/67b and the 64b/i67b for every bit of “xargs1” that the total number of bits is 36,381 bits. The maximum RD value of 64b/67b is 21 and 64b/i67b is 26 but we analyze in detail that the value of both protocols is different. The value of 64b/67b has higher number than 64b/i67b when the value are 6 to 20. On the other hand, the value of 64b/i67b has majority between 0 to 5. The 64b/i67b value has better controlling performance to control the to nearly zero than 64b/67b.

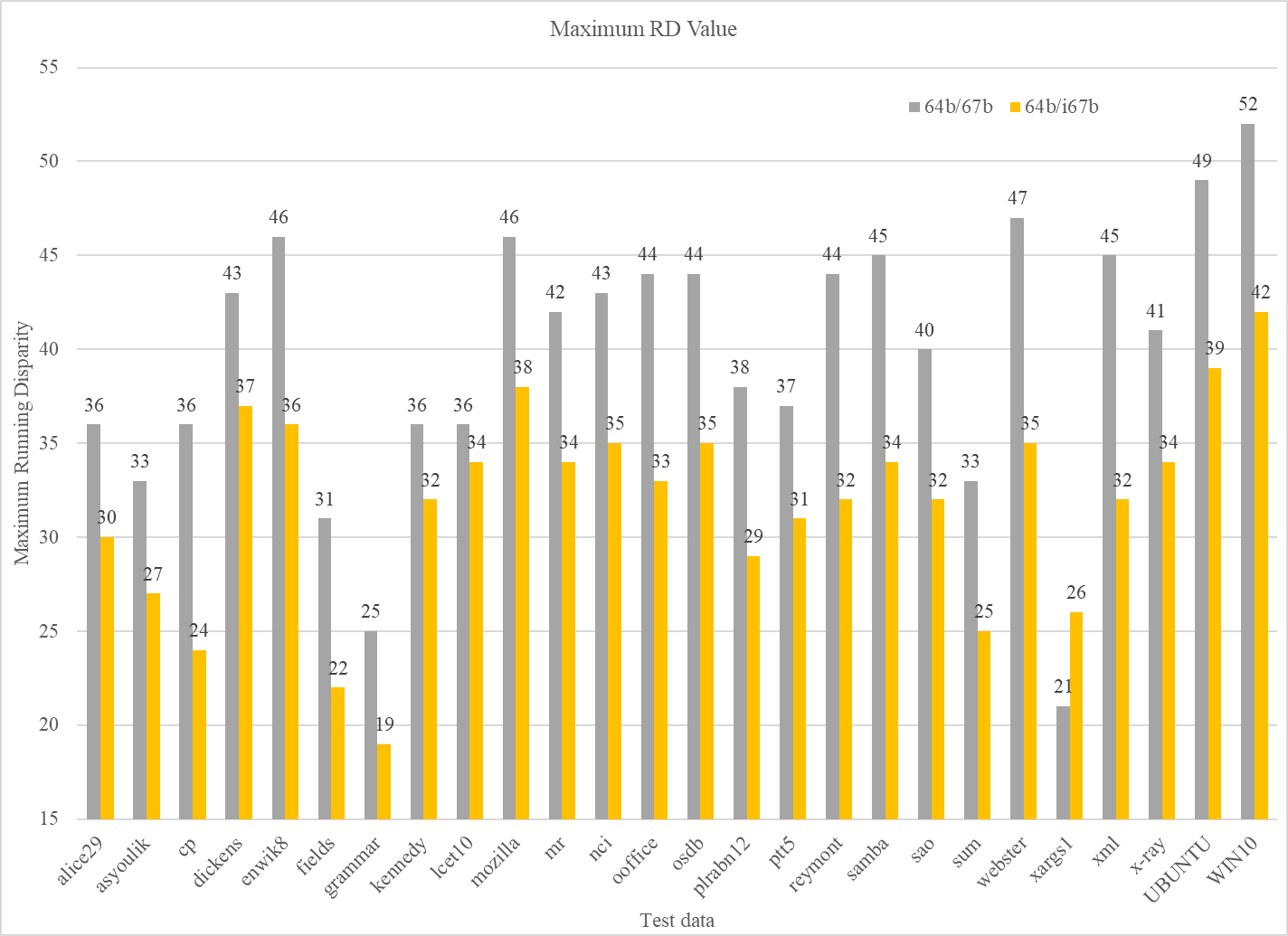
**Table 4.4**

*Simulation Result When Using Squash Benchmark*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  | CRD | | |
| Name | Type | Size | 64b/67b | 64b/i67b | 64b/66b |
| Grammar | LISP | 3.72K | 25 | 19 | 233 |
| Xargs1 | GNU Manual | 4.33K | 21 | 26 | 249 |
| Fields | C Source | 12K | 31 | 22 | 222 |
| Cp | HTML | 25K | 36 | 24 | 724 |
| Sum | SPARC exe | 38K | 33 | 25 | 1408 |
| Asyoulik | Text | 127K | 33 | 27 | 748 |
| Alice29 | Text | 149K | 36 | 30 | 1,337 |
| Lcet10 | Text | 417K | 36 | 34 | 2,589 |
| Plrabn12 | Poetry | 471K | 38 | 29 | 2,137 |
| Ptt5 | CCITT Test set | 502K | 37 | 31 | 1,924 |
| Kennedy | Excel | 1006K | 36 | 32 | 1,749 |
| Xml | XML | 5.10M | 45 | 32 | 6,567 |
| Office | DLL of open office | 5.87M | 44 | 33 | 7,722 |
| Reymont | Text | 6.32M | 44 | 32 | 7,861 |
| Sao | The SAO star catalog | 6.92M | 40 | 32 | 5,958 |
| x-ray | X-Ray Picture | 8.08M | 41 | 34 | 9,982 |
| Mr | Medical image | 9.51M | 42 | 34 | 7,723 |
| Osdb | MySQL | 9.62M | 44 | 35 | 9,678 |
| Dickens | Text | 9.72M | 43 | 37 | 6,908 |
| Samba | Tarred source code | 20.61M | 45 | 34 | 13,098 |
| Name | Type | Size | 64b/67b | 64b/i67b | 64b/66b |
| Nci | Chemical database | 32.80M | 43 | 35 | 23,819 |
| Webster | Dictionary | 39.54M | 47 | 35 | 16,757 |
| Mozilla | Mozilla’s Tarred exe | 48.85M | 46 | 38 | 14,376 |
| Enwik8 | Text | 96.44M | 46 | 36 | 98,758 |
| UBUNTU | ISO | 868M | 49 | 39 | 62,995 |
| WIN10 | ISO | 3815M | 52 | 42 | 271,119 |

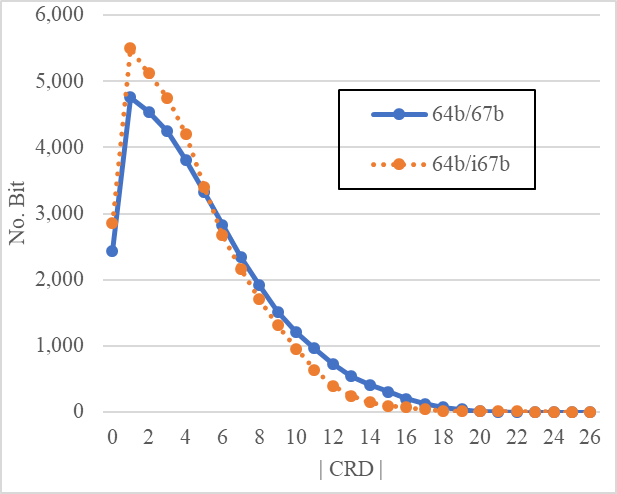
**Figure 4.4**

*Simulation Result When Using Squash Benchmark*



**Figure 4.5**

*CRD Comparison of 64b/67b and 64b/i67b in “xargs1”*



The comparison of average value for 64b/67b and 64b/i67b has been show in Table 4.5. It shows all 64b/i67b has better result in average of .

**Table 4.5**

*Average CRD Value of Simulation Result From Squash Benchmark*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  | Average CRD | |
| Name | Type | Size | 64b/67b | 64b/i67b |
| Grammar | LISP | 3.72K | 5.053 | 4.169 |
| Xargs1 | GNU Manual | 4.33K | 4.891 | 4.190 |
| Fields | C Source | 12K | 5.127 | 4.336 |
| Cp | HTML | 25K | 5.206 | 4.278 |
| Name | Type | Size | 64b/67b | 64b/i67b |
| Sum | SPARC exe | 38K | 5.131 | 4.413 |
| Asyoulik | Text | 127K | 5.210 | 4.384 |
| Alice29 | Text | 149K | 5.180 | 4.418 |
| Lcet10 | Text | 417K | 5.169 | 4.384 |
| Plrabn12 | Poetry | 471K | 5.191 | 4.380 |
| Ptt5 | CCITT Test set | 502K | 5.098 | 4.377 |
| Kennedy | Excel | 1006K | 5.159 | 4.387 |
| Xml | XML | 5.10M | 5.179 | 4.376 |
| Office | DLL of open office | 5.87M | 5.183 | 4.383 |
| Reymont | Text | 6.32M | 5.171 | 4.386 |
| Sao | The SAO star catalog | 6.92M | 5.172 | 4.381 |
| x-ray | X-Ray Picture | 8.08M | 5.174 | 4.385 |
| Mr | Medical image | 9.51M | 5.146 | 4.383 |
| Osdb | MySQL | 9.62M | 5.178 | 4.381 |
| Dickens | Text | 9.72M | 5.180 | 4.381 |
| Samba | Tarred source code | 20.61M | 5.169 | 4.381 |
| Nci | Chemical database | 32.80M | 5.150 | 4.379 |
| Webster | Dictionary | 39.54M | 5.175 | 4.384 |
| Mozilla | Mozilla’s Tarred exe | 48.85M | 5.172 | 4.381 |
| Enwik8 | Text | 96.44M | 5.173 | 4.382 |
| UBUNTU | ISO | 868M | 5.173 | 4.383 |
| WIN10 | ISO | 3815M | 5.174 | 4.383 |

## Worst Case of 64b/i67b

The 64b/i67b has a lower an average and maximum than 64b/67b. However, the 64b/i67b could not guarantee the maximum because if there is a worst case in transmission, the will get higher and keep continue . The worst case is happen when the input data 0xFFFFFFFFFFFFFFFF feed to the encoder. The inversion bit is set to ‘1‘ to flip Block B from 0xFFFFFFFF to 0x00000000 to have the best . On the other hand, the encoded data has disparity equal to 0 but the inversion bit is the one who increase disparity for each iteration.

Let us assume that if this data (0xFFFFFFFFFFFFFFFF) keeps repeating the will be increased to infinite. However, this case can rarely happen in transmission because with scrambler circuit, even though the data has a same value, the scramble circuit will scram data before feed it to the encoder circuit and that will help to avoid worst case of 64b/i67b to be occurred in transmission.

**Table 4.6**

*Worst Case of 64b/i67b*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Iteration | 66 | 65:64 | 63:32 | 31:0 |
| 1 | 1 | XX | 0xFFFFFFFF | 0x00000000 |
| 2 | 1 | XX | 0xFFFFFFFF | 0x00000000 |
| 3 | 1 | XX | 0xFFFFFFFF | 0x00000000 |
| - | - | XX | - | - |
| 100 | 1 | XX | 0xFFFFFFFF | 0x00000000 |
| 101 | 1 | XX | 0xFFFFFFFF | 0x00000000 |
| 102 | 1 | XX | 0xFFFFFFFF | 0x00000000 |
| 103 | 1 | XX | 0xFFFFFFFF | 0x00000000 |

# CONCLUSION AND RECOMMENDATIONS

## Conclusion

The 64b/i67b technique shows a better result than 64b/67b that cause good performance in transmission line that the maximum of has been reduced without additional bit overhead. The simulation results show the improvement can be reached up to 33%. The HDL implementation is comparing between the normal 64b/67b and 64b/i67b. The 64b/i67b encoder uses more resources than 64b/67b about 18 LUT of Xilinx xc7k325tffg900. However, for the decoder, the 64b/i67b has more complex circuit because they need to calculate of both blocks instead of normal 64b/67b that just check inversion bit and flip the data. That is the reason why the decoder uses 132 LUT over than the 64b/67b decoder.

Although, the proposed technique cannot guarantee the because in the worst case scenario, it may cause infinite value but that case means the output data from scramble circuit must have a same disparity for both blocks (Block A and B) and not only one data, it must keep having a same disparity for both blocks continuously, each data word will increase 1 disparity (absolute value). However, the solution to avoid this problem and guarantee the to 96 as the original 64b/67b has been found. It is when the disparity of both blocks are the same. The fixed block is changed from Block A or Block B to all 64-bit data, the inversion bit will indicate to all 64-bit data (as 64b/67b). However, with the simulation results the worst case is difficult to happen and to add this solution to guarantee the worst case will not be implemented to minimize hardware size.

## Recommendations

The 64b/i67b shows the improvement in an average and maximum value that cause signal integrity in transmission line. This dissertation proposes encoding and decoding technique with simulation results that would be show the improvement in theory. The implementation in hardware design of the fully transceiver hardware design is the interesting to be learnt and investigate the signal integrity.

.

# REFERENCES

Abhijit Athavale, Carl Christensen, "High-speed serial io made simple, a designers’ guide, with FPGA applications. *Connectivity Solution*, Edition 1.0," Xilinx. 2005.

J.D.H. Alexander, "Clock recovery from random binary signals doi: 10.1049/el:19750415," *ELECTRONICS LETTERS*, vol. 11, no. 22, pp. 541 - 542, 30 October 1975.

Mounir. Meghelli, B. Parker, Herschel Ainspan and M. Soyuer, "SiGe BiCMOS 3.3 V clock and data recovery circuits for 10 Gb/s serial transmission systems," in IEEE International Solid-State Circuits Conference, USA, 2000.

Julien Saadé, Abdelaziz Goulahsen, André Picco, Joel Huloux and Frédéric Pétrot, "Low overhead, DC-Balanced and run length limited Line Coding," in 2015 *IEEE 19th Workshop on Signal and Power Integrity* (SPI), Berlin, 2015.

"BOSCH. CAN specification version 2.0", 1991.

Interlaken Protocol Definition, Interlaken Alliance, 2008.

Zhou Qinglun, Li Chunyan and Wang Yong, "Hardware Implementation of 64B/66B Encoder/Decoder for 10-Gigabit Ethernet," in 2006 *International Conference on Communication Technology*, Guilin, 2006.

Satyajit Mohapatra, Hari Shanker Gupta, Jatindeep Singh and Nihar Ranjan Mohapatra, "A 64b/66b Line Encoding for High Speed Serializers," in 30th *International Conference on VLSI Design and 2017 16th International Conference on Embedded Systems (VLSID)*, Hyderabad, 2017.

Karthik Balasubramanian, Segid Agili and Aldo Morales, "Investigating the new 64b/66b encoding scheme's power spectral density," in *International Conference on Consumer Electronics* (ICCE), Las Vegas, 2011.

Muhanned S. Bakir and James D. Meindl, Integrated Interconnect Technologies, Artech House Publishers, 2008.

David J. Morris, Pulse Code Formats for Fiber Optical Data Communication, CRC Press, 1983.

David R. Smith, Digital Transmission Systems, Springer, Nov 2003.

R. Forster, "Manchester encoding: opposing definitions resolved," *Engineering Science and Education Journal*, vol. 9, no. 6, pp. 278-280, December 2000.

Thummaluri Uday, Abhinav Kumar and Lakshmi Prasad Natarajan, "Generation of Perfectly DC Balanced Codes for Visible Light Communications," in 24th *National Conference on Communications*, Feb 2018.

L. Wang, C. Stroud and N. Touba, System-on-Chip Test Architec-tures, *Morgan Kaufmann*, 2007.

Albert X. Widmer and Peter. A. Franaszek, "A DC-Balanced, Partitioned-Block, 8B/10B Transmission Code," in *IBM Journal of Research and Development*, 1983.

Alistair Coles and David Cunningham, "Low Overhead Block Coding for Multi-Gb/s Links.," [*http://www.hpl.hp.com/techreports/98/hpl-98-168.pdf*](http://www.hpl.hp.com/techreports/98/hpl-98-168.pdf)*. ,*1998.

Muhammad Irfan Anis, M. Waqas, M. Sadiq, M. Javaid, Shafiq Ur Rahman and M. Arsalan, "Implementation of Dynamic Channelization on SONET," in *International Conference on Advanced Communication Technology*, Gangwon-Do, 2008.

Gaidhani Dhananjay Shekhar, "A Survey on Interlaken Protocol for Network Applications," *International Journal on Recent and Innovation Trends in Computing and Communication*, 2015.

Yadong Li, Danlan. Li, Wenqiang Cui and Rui Zhang, "Research based on OSI model," in *International Conference on Communication Software and Networks*, Xi'an, 2011.

Y. Fan, Z. Zilic, "BER Testing of Communication Interfaces", IEEE Transactions on Instrumentation and Measurement,Vol.57,Issue.5,pp.897-906, May 2008.

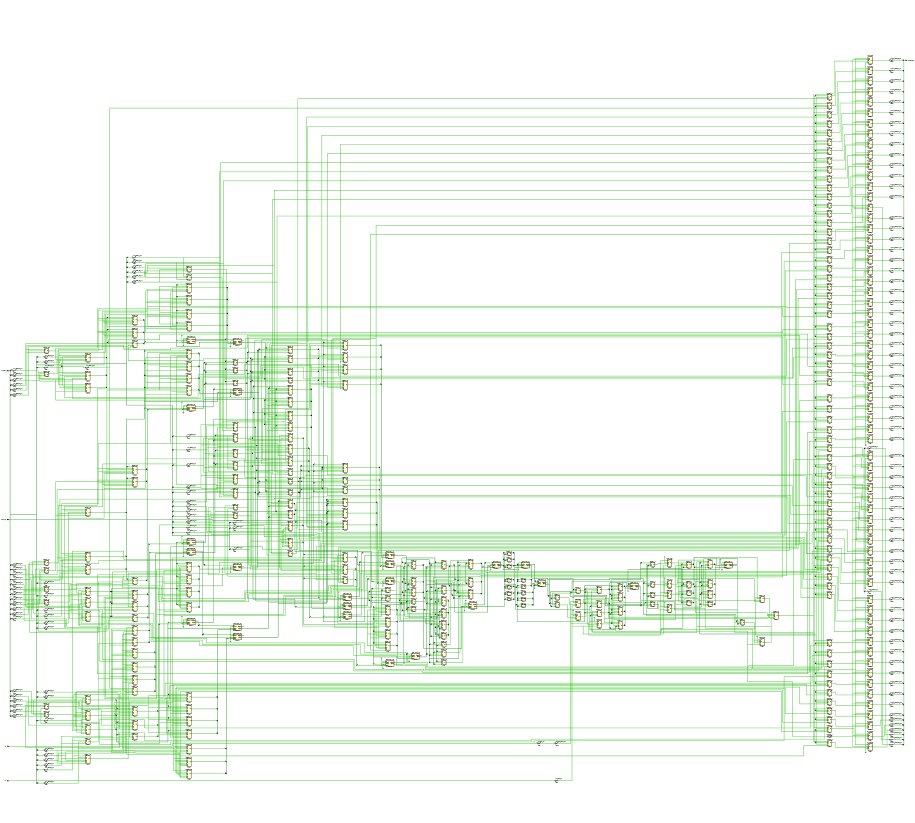
T. Horvath, R. Fujdiak and M. Cucka, "Comparison of Bit Error Rate of line Codes in NG-PON2", IJATES2, vol.5, no.2, pp.95-100, 2016.

Borga Andrea, "https://opencores.org," 19 March 2018. [Online]. [*https://opencores.org/projects/core1990\_interlaken*](https://opencores.org/projects/core1990_interlaken)*.*

Evan. Nemerson, "Squash," 2015. [Online]. Available: <https://quixdb.github.io/squash-benchmark/>.

# APPENDIX

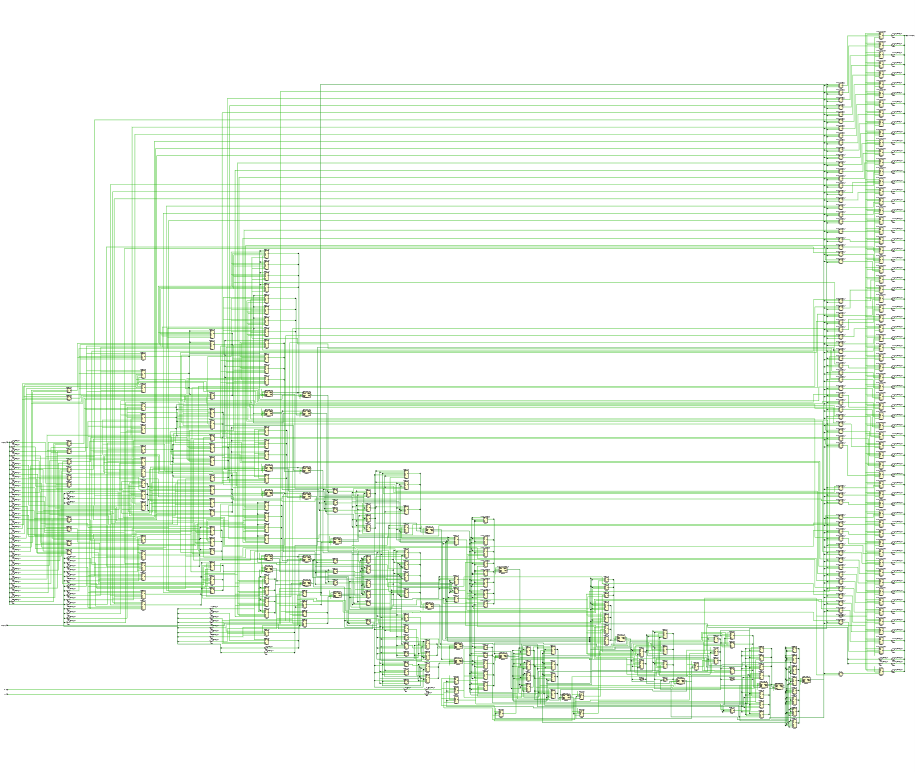
1. 64b/67b Encoder

****

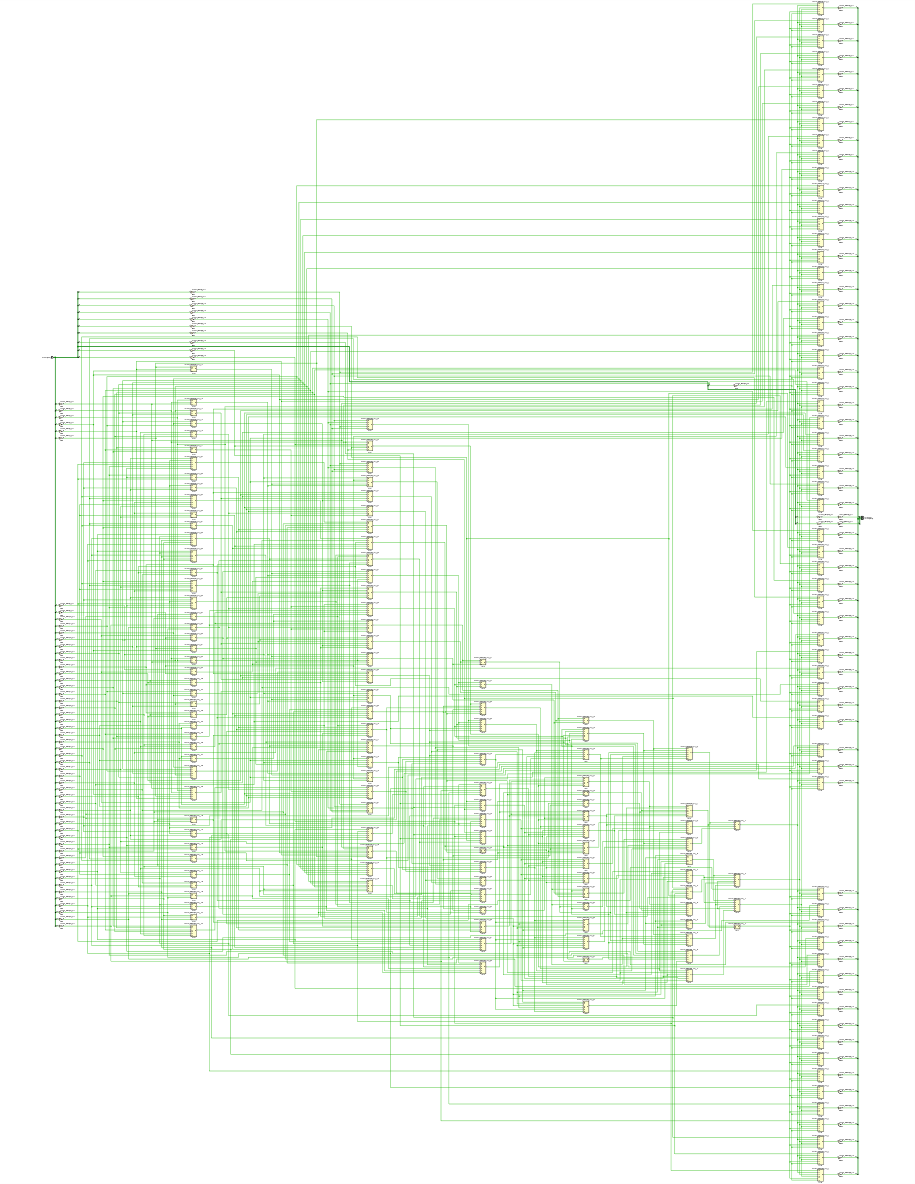
1. 64/67b Decoder

****

1. 64b/i67b Encoder

****

1. 64b/i67b Decoder

****