

DYNAMIC DATA ENCODING FOR PAGE-ORIENTED MEMORIES

by

Leo Selavo

Diploma in Mathematics, University of Latvia, 1995

M.S. in Computer Sciences, University of Latvia, 1996

Submitted to the Graduate Faculty of
the School of Arts and Sciences in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

University of Pittsburgh

2004

UNIVERSITY OF PITTSBURGH

FACULTY OF ARTS AND SCIENCES

This dissertation was presented

by

Leo Selavo

It was defended on

July 12, 2004

and approved by

Bruce Childers, Assistant Professor, Computer Science

Daniel Mosse, Associate Professor, Computer Science

Steven P. Levitan, Professor, Computer Engineering

Alexander A. Sawchuk, Professor, Electrical Engineering, USC

Dissertation Director: Donald M. Chiarulli, Professor, Computer Science

Copyright by Leo Selavo

2004

DYNAMIC DATA ENCODING FOR PAGE-ORIENTED MEMORIES

Leo Selavo, PhD

University of Pittsburgh, 2004

This dissertation presents a key portion of the system architecture for a high performance page-oriented memory. The focus of this research is the development of new dynamic encoding algorithms that provide high data reliability with code density that is higher than in the conventional static modulation schemes. It also presents an intelligent read/write head architecture capable of implementing the most promising of these algorithms in real-time.

Data encoding techniques for page-oriented mass storage devices are typically conservative in order to overcome the destructive effects of inter-symbol interference and noise due to the physical characteristics of the media. Therefore significantly more bits are required in an encoded version of data than in the original information. This penalty in the code density, usually referred to as *code rate*, keeps the utilization of the media relatively low, often less than 50% of the capacity of a maximally dense code. This is partially because encoding techniques are static and assume the worst case for the information surrounding the data block being encoded. However, in the context of page-oriented data transfers it is possible to evaluate the surrounding information for each code block location, and, thus, to apply a custom code set for each code block. Since evaluating each possible code during runtime leads to very high time complexity for encoding and decoding algorithms, we also present alternative algorithms that successfully trade time complexity for code density and are a strong competition to the traditional static modulation schemes.

In order to verify that the encoding algorithms are both efficient and applicable, they were analyzed using a two-photon optical memory model. The analysis focused on how well the algorithms performed as a trade off between complexity and code density. It resulted that a full enumeration of codes yielded code density as high as 83%, although the time complexity for the enumeration approach was exponential. In another study, a linear time algorithm was analyzed. The code density of this algorithm was just over 54% percent. Finally, a novel quasi-dynamic encoding algorithm was created, which yielded 76% code density and had constant time complexity.

Keywords: modulation encoding, page-oriented memory, two-photon memory.

TABLE OF CONTENTS

PREFACE	XIII
1.0 INTRODUCTION	1
2.0 MOTIVATION	4
2.1 DATA STORAGE GOALS AND TRADEOFFS	4
2.1.1 Capacity	4
2.1.2 Data Transfer Rate and Access Time	5
2.1.3 Integrity	6
2.2 DATA STORAGE ENCODING HIERARCHY	6
2.3 ENABLING TECHNOLOGY	8
2.3.1 Smart Read/Write Head Architecture	8
3.0 BACKGROUND AND PREVIOUS RESEARCH	10
3.1 DEFINITIONS	10
3.2 STORAGE MEDIA	11
3.2.1 Optical Storage	11
3.2.2 Magneto-Optical Storage	13
3.2.3 Advanced Optical Storage	13
3.2.3.1 Holographic storage systems	14
3.2.3.2 Spectral hole burning	16
3.2.3.3 Two-photon absorption effect storage systems	16
3.3 DATA MODULATION CODES	18
3.3.1 Technical Requirements for Data Modulation Encoding	19
3.3.1.1 FM, MFM and RLL	20
3.3.1.2 PRML	22
3.3.1.3 Block codes versus convolutional codes	22
3.3.1.4 Viterbi algorithm	23
3.3.1.5 Page-oriented memory codes	24
4.0 PROBLEM STATEMENT	25
5.0 APPROACH	29

6.0	EXPERIMENT DESIGN	32
6.1	MEMORY MODEL.....	32
6.2	PSF MATRIX SIZE SELECTION	33
6.3	BENCHMARK DATA SETS.....	35
6.3.1	Randomness Quality Analysis	35
6.3.1.1	Entropy.....	35
6.3.1.2	Chi-square Test.....	35
6.3.1.3	Arithmetic Mean.....	36
6.3.1.4	Monte-Carlo Value for Pi.....	36
6.3.1.5	Serial Correlation Coefficient	37
6.3.2	Test Data Sets.....	37
6.3.2.1	Random-Set.....	37
6.3.2.2	Work-Set	37
6.3.2.3	Training-Set.....	38
6.3.2.4	Usage of the test data sets	39
6.4	SIMULATION SOFTWARE	40
7.0	DYNAMIC ENCODING ALGORITHMS	43
7.1	FULL ENUMERATION ALGORITHM – EMPIRICAL UPPER BOUND	43
7.1.1	Full Enumeration Algorithm - Simulation Results.....	45
7.1.2	Variable vs. Constant Code Rate.....	50
7.1.3	Dynamic Encoding and Arithmetic Precision	52
7.2	LINEAR TIME “EITHER-NEITHER” ALGORITHM.....	54
7.2.1	Either-Neither Algorithm - Simulation Results.....	56
7.3	DYNAMIC ENCODING SUMMARY	58
8.0	QUASI-DYNAMIC ENCODING PARADIGM	59
8.1	“SECOND CHANCE RE-CODING” ALGORITHM	60
8.1.1	Encoding and Decoding.....	60
8.1.2	Construction of the Re-coding Table.....	62
8.2	RE-CODING TABLE GENERATION EXPERIMENTS	64
8.2.1	Training Set Selection for Re-coding Table Generation.....	64
8.2.2	Testing Re-coding Table Generation for Certain File Types	66

8.3	PAGE FILL STRATEGIES.....	68
8.3.1	Point Spread Function Matrix	68
8.3.2	Linear Fill	69
8.3.3	Uniform Fill	70
8.3.3.1	Z-Fill and X-Fill Strategies	70
8.3.3.2	Pseudo-Random Fill Strategy.....	72
8.3.4	Page Fill Simulation Results	72
8.4	SECOND CHANCE RE-CODING PERFORMANCE TESTS	74
8.5	RE-CODING PERFORMANCE IN THE PRESENCE OF ISI	75
8.6	RE-CODING PERFORMANCE IN THE PRESENCE OF JITTER.....	78
8.7	RE-CODING PERFORMANCE IN THE PRESENCE OF MEDIA NON- UNIFORMITIES	80
8.7.1	Adapting to Low Frequency Noise.....	82
9.0	ANALYSIS OF THE TWO-PHOTON MEMORY MODEL.....	84
9.1	VARIATIONS OF POINT SPREAD FUNCTIONS	84
9.2	DYNAMIC ENCODING FOR TWO-PHOTON MEMORY SYSTEM	86
10.0	SUMMARY, CONCLUSIONS, AND FUTURE DIRECTIONS.....	90
10.1	SUMMARY AND CONCLUSIONS.....	90
10.2	FUTURE DIRECTIONS	93
	APPENDIX A: PROGRAMMABLE OPTOELECTRONIC INPUT HEAD	94
A.1	ARCHITECTURE	95
A.2	CONFIGURABLE LOGIC BLOCK DESIGN.....	95
A.3	INTERCONNECT	96
A.4	OPTICAL RECEIVER ARRAY	96
A.5	CONFIGURATION MODES.....	97
A.6	PERFORMANCE	97
	APPENDIX B: PAGE ORIENTED STORAGE SYSTEM SIMULATION SOFTWARE.....	99
B.1:	SIMULATION SOFTWARE INTERFACE	100
B.2:	SIMULATION SOFTWARE IMPLEMENTATION	106

BIBLIOGRAPHY 107

LIST OF TABLES

Table 1: FM encoding.....	21
Table 2: MFM encoding	21
Table 3: RLL encoding.....	21
Table 4: Chi-square test measurement interpretation	36
Table 5: Random-Set analysis.....	37
Table 6: Work-Set analysis	38
Table 7: Training Set analysis.....	39
Table 8: Dynamic Full Enumeration encoding performance.....	50
Table 9: Variable code rate performance v.s. fixed code rate.....	51
Table 10: Second Chance Re-coding example	60
Table 11: Re-coding table generation experiment results	65
Table 12: Generating Re-coding tables for specific file types	67
Table 13: Fill order performance comparison.....	73
Table 14: Second Chance Recoding simulation results	77
Table 15: Areal density for two-photon memory system	89
Table 16: Dynamic encoding algorithm comparison: Time complexity vs. code density.	91
Table 17: Simulator command line parameters	100

LIST OF FIGURES

Figure 1: Storage areal density advancement.....	4
Figure 2: Encoding hierarchy	7
Figure 3: Smart read/write head architecture.....	9
Figure 4: Data storage channel.....	19
Figure 5: Example and comparison of FM, MFM and RLL modulation codes	22
Figure 6: Trellis diagram for Viterbi algorithm.....	23
Figure 7: Two-photon optical memory	25
Figure 8: Raw data images.....	26
Figure 9: Raw data histogram.....	27
Figure 10: Noise margins	27
Figure 11: Data page fragment	30
Figure 12: Dynamic encoding approach	30
Figure 13: PSF matrix size analysis	34
Figure 14: Simulation flow	40
Figure 15: Simulation software graphical user interface	41
Figure 16: Full Enumeration algorithm. The framed area generates a valid code set.	44
Figure 17: Full enumeration encoding results for 4x4 sized CBs.....	45
Figure 18: Full enumeration approach: Code density vs. CB size at 0.40 threshold	46
Figure 19: Encoded data images using full enumeration approach.....	47
Figure 20: Code density vs. threshold (NM=.03)	48
Figure 21: Code density vs. noise margin (T=.45)	48
Figure 22: Full enumeration approach: Code density vs. CB size at 0.45 threshold	48
Figure 23: Dynamic Full Enumeration encoding results.....	49

Figure 24: Full Enumeration encoding with variable (VarLen) and 7/9 constant (FixLen) code rates for various arithmetic precisions.	53
Figure 25: Linear time Either-Neither algorithm.....	55
Figure 26: Threshold and Noise Margin for Either-Neither and Full Enumeration encoding algorithms.....	57
Figure 27: Either-Neither encoding performance	57
Figure 28: Second Chance Re-coding algorithm	61
Figure 29: Re-coding codeword likelihood and encoding tables	63
Figure 30: Re-coding table generation experiment results.....	65
Figure 31: Generating Re-coding tables for specific file types	67
Figure 32: Pixel crosstalk and ISI in optical memory media.	69
Figure 33: Sequential fill of the data page.	70
Figure 34: Z-Fill pattern fill of a data page	71
Figure 35: X-Fill pattern fill of a data page.....	71
Figure 36: Fill order performance comparison	73
Figure 37: Page fill order code density distribution over Work-Set files.....	74
Figure 38: Quasi dynamic encoding algorithm verification test results	76
Figure 39: Quasi-dynamic encoding code density and time performance	76
Figure 40: Jitter measurement.....	78
Figure 41: Re-coding performance in the presence of jitter.....	79
Figure 42: Noise matrix examples.....	81
Figure 43: Re-coding performance in the presence of noise.....	82
Figure 44: Various Point Spread Functions	84
Figure 45: PSF recorded from a 2-photon memory system (provided by Call-Recall Inc., CA).	85
Figure 46: PSF 2D and 3D plots.....	86
Figure 47: Code density plot	87

Figure 48: Effective Areal density plot for $2\mu^2$ bit size.....	88
Figure 49: Modulation encoding techniques. Comparison chart. Code density.....	91
Figure 50: OFPGA logical and physical layouts.....	94
Figure 51: OFPGA configurable logic block	96
Figure 52: OFPGA interconnect	96
Figure 53: OFPGA waveforms	98
Figure 54: Simulation flow	99
Figure 55: Simulation software graphical user interface	106

PREFACE

I want to thank all the people who provided me with guidance, help, and support while I was working on this dissertation.

Most of all, I am indebted to my advisors, Dr. Donald M. Chiarulli and Dr. Steven P. Levitan, for their continuous assistance, motivating encouragement, and great patience. Dr. Chiarulli initially generated my interest in computer architecture through the courses he taught. Later, both he and Dr. Levitan promoted my ideas and instructed me on how to pursue them, fueled my motivations, taught how to clearly present my ideas and solutions and how to focus on the important problems. Without their guidance, this thesis would never have been completed.

I am also grateful for the interactions with the dissertation committee members. Dr. Bruce Childers sparked my interest in encoding through his low-power computation course. Dr. Daniel Mosse showed me how to consider things from systems perspective. Dr. Alexander Sawchuk proved to me the uncontested power of examples.

My scientific curiosity was greatly boosted by the teachers whose courses I took in graduate school. Strong motivations to pursue a teaching career came from the courses taught by Dr. Kirk Pruhs and Dr. Rajiv Gupta.

At many stages of its development, this dissertation has benefited from discussions with my colleagues and friends: Jason Bakos, Vahan Grigoryan, Dave Evans, and Rastislav Bodik. The memories of the first years in the graduate school will forever be connected with Hiroki Kobayashi, and Sylvain Lauzac. Very special thanks go to Dr. Majd Sakr and Jose Martinez for not just asking hard questions, but also for providing strong encouragement. Thank you all for your comments and critique!

Last, but not least, I want to thank my family for their love and support. I am especially grateful for the values they taught me and for the environment they provided for my work. They supported my numerous hobbies, which eventually lead to my current interest in computer science. Finally, I want to thank my wife Corina Petrescu for her care and support. Not only made her encouragement possible to finish this dissertation, but she also has been a great motivation and resource for me, and an excellent example at persistence and time-management.

1.0 INTRODUCTION

With the increasing popularity of multimedia and other storage intensive applications, storage systems require larger capacity and higher data transfer rates. In response to this demand, volumetric optical storage systems, such as holographic [1,2] and two-photon memories [3] are in development. Alternatively non-optical page-oriented data storage systems such as the *Millipede* project at Carnegie Mellon University are under investigation [4]. This thesis deals with issues of designing and implementing volumetric page-oriented storage systems (PODS), specifically data modulation coding algorithms and their implementation. As the target system we have chosen the optical two-photon storage system to design and evaluate our encoding techniques.

Conventional magnetic and optical storage systems contain data organized linearly, in tracks. A limited number of tracks can be read or written in parallel. This is the approach used in most magnetic and magneto-optical disks, CDs, and DVDs. Volumetric optical memories usually store data in blocks of two-dimensional pages. They outperform conventional data storage systems in two ways. First, they store information bits through the volume of the medium, thus increasing the capacity. Secondly, volumetric memories have parallel block-oriented readout, thus increasing the information throughput.

The most important requirement of any memory system is to maintain data integrity. Several effects may corrupt the data during the process of recording, storing and reading out from optical memory. These effects include electrical noise in the writing and readout system, media material defects or non-uniformity, optical noise from reflection and diffusion, and inter-pixel cross talk due to additive noise. Mechanical noise sources, such as misalignment and jitter, are also present due to the positioning of the readout mechanism or the optical media. Even varying environmental properties, such as humidity and temperature, may influence data integrity.

Data integrity is typically ensured in multiple levels by using a hierarchy of error correcting and media-specific data modulation codes. This research focuses on modulation codes for PODS that improve data integrity. A modulation code is a system of mapping from the source data to a sequence of symbols, which is compatible with the characteristics of the storage media. Modulation codes are designed to tolerate errors introduced by the problems

described above. This tolerance comes at the cost of code density. For example, if we want to encode four bits of data, we may actually need nine bits or symbols to reliably store them on the media. This leads to a code rate of $4/9 = 0.44(4)$, which means code density below 45%, in other words, the overhead of the information stored on the media is more than 55%. Several data encoding schemes proposed for page-oriented memories, such as parity array codes [5] and cross talk minimization [6,7] suffer from low code densities that do not exceed 50%.

The code density of these encoding schemes is so low because they map the data to the code words using a static code table taking into account a worst-case analysis of error sources. In our approach, we assign codes dynamically, by taking advantage of the fact that information about the surrounding data in the memory page is available to us for encoding and decoding. Thus, we are able to consider the actual data rather than the worst-case behavior in each region of each page and compute the codes dynamically. We assign codes that maximize code density while satisfying the system constraints, such as allowing only code words that do not create destructive optical crosstalk. In order to compute each of these encoding schemes, we assume intelligence in the read/write head with ability to process information at the source in real time. The result is higher code densities and, therefore, higher media utilization while retaining the same reliability level of the data.

Initial analysis, as part of this research, has shown that it is possible to achieve code density up to 83% by using dynamic data modulation schemes [8]. Unfortunately, it comes at the cost of exponential time complexity with respect to the size of the code block. However, trading code density for speed yields algorithms that require less than exponential time and still have good code density. This dissertation presents several such algorithms.

Since the need for intelligent read/write heads for the dynamic modulation schemes was determined, we also present a smart read-write head architecture that defines a system capable of efficient data encoding and decoding using dynamic encoding algorithms. We have investigated optoelectronic CMOS logic devices that are important components of this architecture. The system must have a direct optical interface with the storage system in order to provide high data throughput, i.e. data, which is read optically, is directly projected onto the processing chip device with integrated photo detectors, processing fabric, and electrical I/O capabilities. A prototype for such a device, which was designed in our laboratory, is an optically reconfigurable gate array [9]. Demonstrating that such a device

can be built has proven that new architectures receiving data from the media and performing decoding on-chip in real time are feasible.

By addressing the problems described above our work has evolved into this dissertation with three major contributions. First, we investigated new dynamic and quasi-dynamic encoding algorithms to support modulation encoding of page-oriented optical memory. Secondly, we evaluated these algorithms including an empirical analysis of susceptibility to such noise sources as inter-symbol interference, jitter, and material non-uniformity. And finally, we have devised an architecture for an intelligent read/write head capable of executing such algorithms in real time.

We begin in the next chapter with a motivation for this research, an overview of conventional and volumetric data storage systems, and the state of the art data encoding techniques for them, followed by the problem statement. The subsequent section depicts our proposed approach to the problem and the data encoding and decoding algorithms. It is followed by a description of the memory model used for simulations. Then, the design of dynamic and quasi-dynamic encoding algorithms is discussed, followed by an evaluation of the performance of the prospective quasi-dynamic algorithm. The dissertation concludes with a summary of our research and future work projections.

2.0 MOTIVATION

The design of a data storage system must address competitive goals: capacity, transfer rate, access time, and integrity. By increasing capacity and transfer rate, it becomes ever more challenging to guarantee data integrity in real time. This thesis presents solutions to the integrity issues at the level of data modulation in the presence of the other goals for specific high performance page-oriented storage (PODS) systems. The following section 2.1 briefly discusses the three goals and how they are related, then illustrates the opportunity for research in data modulation coding schemes to improve effectiveness of using storage capacity and high transfer rates provided by PODS systems. Section 2.3 of this chapter describes an architecture enabling the coding schemes presented by this dissertation for PODS.

2.1 DATA STORAGE GOALS AND TRADEOFFS

2.1.1 Capacity

Increasing the number of bits that fit in storage media area increases mass storage capacity. This is known as areal density, which has been doubling every nine months, surpassing the predictions by the special case of the Moore's law for data storage [10]. Figure 1 shows graph in which years are plotted against the achieved areal densities, taking in account the recent commercially available storage devices. As more and more of storage space is requested by the modern multimedia and other data-hungry applications, the data storage companies attempt to increase areal density on magnetic drives. Currently, a commercial magnetic

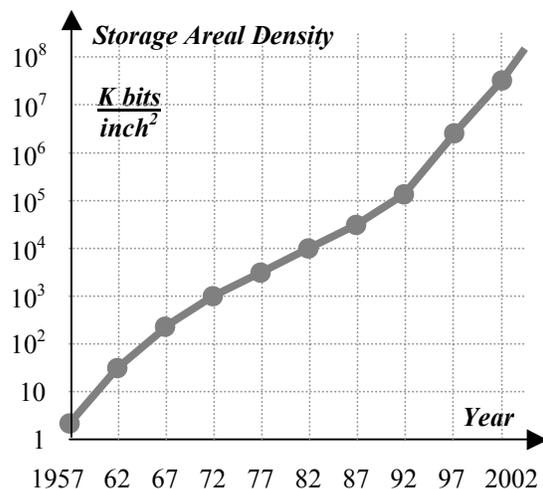


Figure 1: Storage areal density advancement

hard drives reach areal densities of 100 Gbits/inch² [11]. However, it is predicted that magnetic storage is approaching the theoretical limit for areal density set by superparamagnetic effect [10]. Therefore researchers look into alternative mass storage systems such as holographic and two-photon PODS memories [1,2,3] that promise better data densities, since they store data in several layers throughout the volume, approaching 1 Tbits/cm³ on the storage media [12].

2.1.2 Data Transfer Rate and Access Time

Another advantage of the PODS systems over magnetic disk storage is highly parallel data readout providing very high data transfer rates. Data is transferred in rectangular 2D data pages as opposed to the conventional storage systems transferring data to the storage media bit by bit. There have been attempts to increase the parallel input and output of the conventional mass storage by either RAID arrays or parallel channel readout heads. However, these solutions have much fewer number of channels than on a PODS data page, for example the DVD-ROM drive developed by Philips is capable of reading up to 9 tracks in parallel resulting in 9-bit word transfer [13]. A PODS system typically has 10³-10⁶ bits on each data page [14]. Thus, PODS transfer rates are on the order of 10⁵ higher than for the conventional storage, which makes up for data access time difference. A modern magnetic hard drive has access time between 7-9ms, while PODS are reported to have 100ms [3] and 10ms [4].

As increased areal density and transfer rates enable high performance mass storage, they also increase data susceptibility to cross talk. In this context the cross talk, also known as inter-symbol interference (ISI), occurs when data symbols are recorded so close on the storage media that the neighboring symbol values influence the detection of the value of the symbol being read. The closer symbols are located, the higher is the dependence of the read value on the values of the neighbors, and hence the higher is the probability of error during readout. In addition, at high data transfer rates the time to detect data symbols is very limited, which can lead to weak readout signals and hence even more of the data degradation. All this leads to challenging data integrity problems, discussed in the next section.

2.1.3 Integrity

Increasing storage capacity and speed come at a cost of data integrity. Packing encoded data symbols more densely brings about closer proximity and less separation boundaries between them and, thus, more destructive influence to each other, called inter-symbol interference (ISI). In addition, by increasing data recording, reading, and transfer rates lead to weakened or distorted data readout. The potential problem sources, just to name a few, are material non-uniformities across the media and jitter due to mechanical reasons. As a result typical storage media usually has raw bit error rates on the order of $10^{-3} - 10^{-4}$ [14]. An industrial mass storage system requires having a bit error rate at most 10^{-12} [14]. Thus, advanced data modulation and error correction techniques are necessary.

Conventional storage systems have both physical design and data encoding techniques developed to improve the bit error rates. For example, IBM has developed giant magneto-resistive heads (GMR) and used them in Microdrive devices [15,16,17]. Similarly, almost every commercial hard drive has built-in partial response maximum likelihood (PRML) encoding [18] to compensate for ISI and lower bit error rates. Volumetric PODS memories differ from conventional storage systems by using page-oriented input and output. Consequently, problems such as ISI are even harder to deal with because data is organized in two and even three dimensions. Although, several techniques have been proposed for PODS systems [5,6,7,14], they provide relatively low code density, often below 50%. Thus, the first goal, the increase of storage capacity, is in jeopardy due to the high overhead in terms of extra space for correction codes. This dissertation presents dynamic data modulation encoding as a better alternative that provides competitive code density up to 83%.

Modulation encoding is at the lowest level of encoding hierarchy for data storage systems, which is described in the next section.

2.2 DATA STORAGE ENCODING HIERARCHY

In order to preserve integrity data are encoded in two steps by following the encoding hierarchy shown in Figure 2. Prior to recording to a storage media the source data intended for storage are encoded using a high-level error

correction code (ECC), such as Reed-Solomon ECC [19]. The result is further encoded using modulation-encoding scheme, which is designed to tolerate noise and has inherent knowledge about the physical media properties. Finally the encoded data is written to the media. Some popular modulation schemes such as FM, MFM, RLL, PRML, and 4/9 encoding are discussed in detail in the “Data Modulation Codes” chapter 3.3 later in this dissertation.

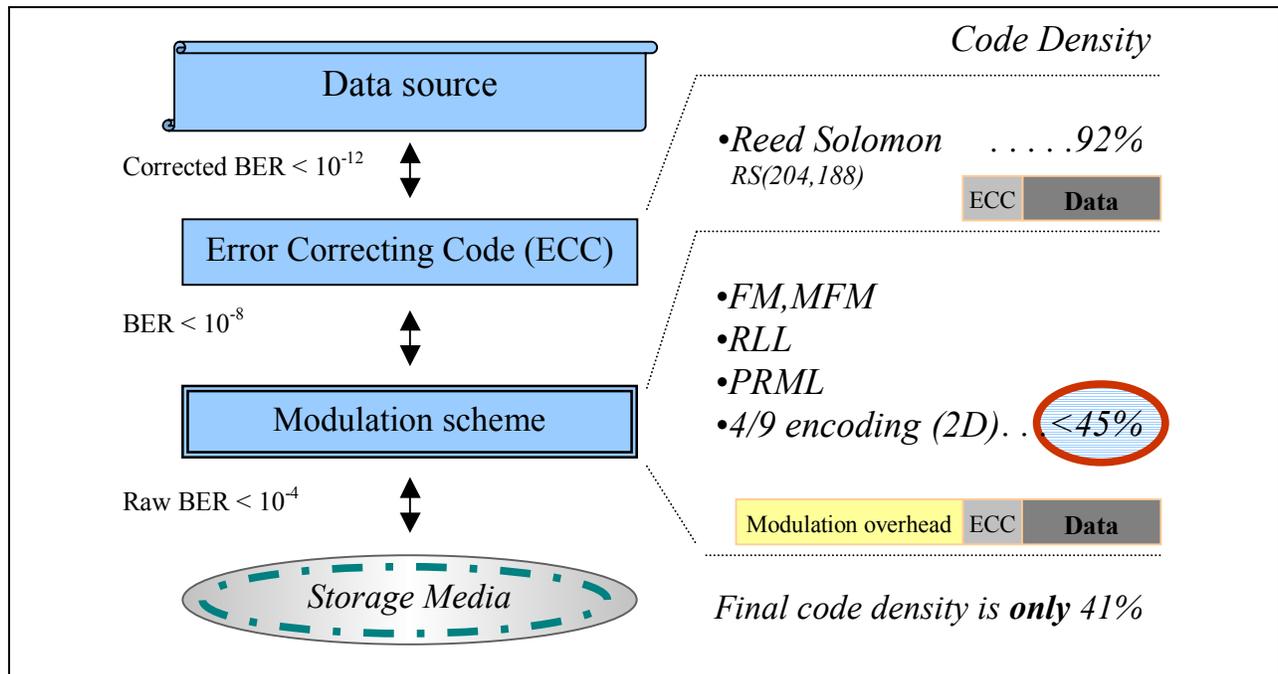


Figure 2: Encoding hierarchy

In order to read the data from the media it needs to be decoded in the reverse order. Typical raw bit error rate of storage media with no encoding present is between 10^{-3} - 10^{-4} [14]. The bit error rate is reduced to approximately 10^{-8} by applying the modulation scheme. Finally the higher level ECC is used to decode the data and to reduce the bit error rate to industrial standard levels of 10^{-12} - 10^{-13} [14].

Encoding has overhead in terms of encoded data size. For example, applying Reed-Solomon code RS(204,188) reduces the data density to 92%, i.e. only 92% of the whole data packet contains the original data, as can be seen on **Figure 2**. By further applying a modulation scheme, such as 4/9 encoding for PODS [6] that has code density below 45%, the overall code density is decreased to 41%. The research presented in this dissertation started with a question: “Can we do better?” As it was discovered, much higher code densities can be achieved by designing

modulation codes of higher efficiency that dynamically adjust to the surrounding data, which is the main focus of this research. The following section presents an architecture that enables such modulation encoding for PODS.

2.3 ENABLING TECHNOLOGY

Upon storage of data it needs to be encoded, and then recorded to the media. Similarly upon retrieval of the data it needs to be read from the media and then decoded. Since PODS assume highly parallel data readout, ideally the data is transferred to the decoding device in parallel, since serialization would limit data transfer rate significantly due to high data page sizes. Therefore it is preferable to process the data as close to the source as possible. In the case of optical PODS systems this means projecting data page image directly to a photo-detector array that is integrated in the processing fabric performing data decoding. The decoding process is performed in parallel. In the following chapter we present such architecture that enables dynamic modulation decoding.

2.3.1 Smart Read/Write Head Architecture

The dynamic encoding approach depends on a smart optical read/write head, which has to be capable of performing decoding on-chip, receiving parallel optical data, and transmitting the resulting data electronically. Ideally, it also should be able to encode data on-chip and transmit the data optically for the writing process.

As part of this dissertation we have devised the following architecture enabling implementation of dynamic encoding for page-oriented optical media. The optical page-oriented mass storage I/O system consists of the media, shown as an optical disk in the Figure 3, and a smart read/write head. The data page image from the media is projected onto an optical detector array and converted into a binary array. The array is transported locally to the processing fabric, which has a 1-bit processor for each pixel. The processors perform the encoding and decoding computations, collaborating with the neighboring processors using local interconnect. The global parameters are broadcast to all the processors using a global interconnect. The decoded data is transmitted through an electrical channel to a data consumer.

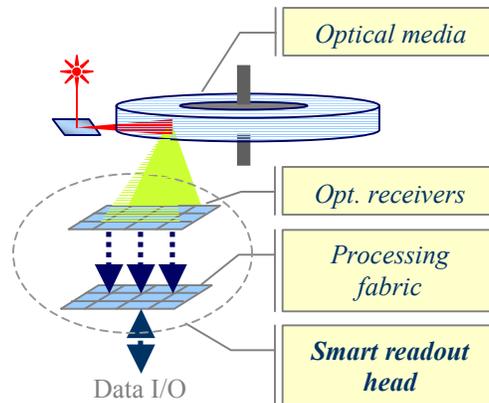


Figure 3: Smart read/write head architecture

Current research and technologies show that such mass storage I/O systems are possible. Two-photon volumetric page-oriented optical disk systems have been developed and tested at Call-Recall Inc. in San Diego, California [3]. Furthermore, we have developed an optically reconfigurable field programmable gate array (OFPGA) with integrated optical detectors as proof of a concept device [9]. The OFPGA device can be used as a prototype for the processing fabric with integrated photo-detector array for parallel optical data input. The OFPGA device design, implementation and capabilities are described in a separate chapter in the Appendix A.

3.0 BACKGROUND AND PREVIOUS RESEARCH

Properties of the media and the way in which data is stored on it inherently affect data modulation schemes. Therefore, this chapter will give an overview of magnetic and optical media and continue with an outline of conventional modulation schemes and current research in this area. We start by clarifying a few of commonly used terms in this research area.

3.1 DEFINITIONS

This research relates to data integrity issues for data storage. Data integrity is measured by using **Bit Error Rate** (BER), which is a ratio of the number of incorrectly read bits over the total number of bits read from the system. BER is written as a number times ten in a negative power. For example, modern hard drives guarantee BER of 10^{-13} or lower.

Raw Bit Error Rate is the BER of a storage system with no error correction scheme present. Raw BER is typically between $10^{-3} - 10^{-5}$ depending on the physical characteristics of the storage system [14]. At this point modulation codes are applied to improve BER to $10^{-8} - 10^{-9}$. On top of the modulation codes, typically stronger error correction codes are applied, such as Reed-Solomon codes [19], which further improve BER to $10^{-12} - 10^{-13}$.

Modulation and error correction codes have an overhead in terms of the size of data written to the media. This overhead is expressed as **Code Rate** [20], which is a ratio of source data size in bits over the encoded data size. For example, one of the modulation schemes for page-oriented memories developed at the University of Southern California [6] encodes every four source data bits to a 3 by 3 code block, i.e. nine bits. Therefore, the code rate is 4/9.

Some of the encoding techniques described in this work use variable code rates. Hence, another measurement is introduced: a **Code Density**, which is the average code rate [21]. Code density reflects the ratio of total size of

source data or file over the size of the encoded data. Commonly, this is expressed in percentage, and is occasionally called **Media Utilization** [8], because it reflects how much of the media would be occupied by the original source data if no encoding were applied.

Often mass storage media capacity is described by number of bits that fit in a certain area or volume. These metrics are called **Areal Density** and **Volumetric Density** respectively. Different information sources express these measurements in bits, kilobits, megabits, gigabits, or terabits per area or volume. The area for areal density is measured in cm^2 , mm^2 , μ^2 , or inch^2 . For example, current commercial mass storage devices are achieving $100\text{Gb}/\text{inch}^2$ [11]. In order to be consistent in this dissertation the areal density is converted to $\text{bits}/\text{inch}^2$, although referenced documents may have expressed it differently.

3.2 STORAGE MEDIA

3.2.1 Optical Storage

The two major optical data carriers today are compact discs (CD), introduced in 1984, and digital versatile discs (DVD), 1995 [22]. These optical storage media are composed of a protective layer, a reflective layer (aluminum, silver or gold), a digital data layer molded in polycarbonate, and a thick polycarbonate bottom layer. DVDs pack data more tightly on the media and may have one to two data layers on one or both sides, thus, providing higher capacity than CDs. Further is a brief overview to methods for increasing transfer rate and improving data integrity that enable access to high-capacity CDs and DVDs.

On CDs and DVDs, data is organized linearly, in tracks. The readout reliability depends on the addressing readout laser alignment, the media surface quality and cleanness, the reflection quality, etc. For that reason, a sophisticated data error correction code (ECC), such as Reed-Solomon encoding [19], is needed on top of modulation schemes, e.g. run-length-limited (RLL) [23], which is discussed in the data modulation section. In this case the price for reliability is paid by media surface utilization, and the encoded data may take twice the size of the original data.

Older CD drives use constant linear velocity (CLV) to ensure a constant data rate regardless of the data track location. This requires faster disk rotation when the data is read closer to the center of the disk. However, this sets a limit on data transfer rates, since it is difficult to change the rotation speed of a disk very fast, when moving from track to track. A more up to date approach is constant angular velocity (CAV), which provides faster access, since the system does not have to wait for the disk to speed up for data access that is closer to the center.

Newer CD and DVD drives attempt to increase the data transfer rate by increasing the rotation speed of the disk, yet, this leads to higher signal noise due to jitter and weaker readout signals, thus decreasing reliability. Although these effects are compensated to a certain degree with ECC, the disk rotation speed has to be limited, which limits the data transfer rate.

Another approach to increase the transfer rate has been to read several tracks in parallel. Zen Research developed this technique in 1994 [22, 24]. Thus, higher transfer rates can be achieved at a slower disk rotation speed and with less noise and jitter in the system. Hence, higher data reliability was possible. For instance, the Zen Multibeam drive of 40x CLV-rating rotates at about the same speed as a conventional 8x CLV-rated system. At the same time, there have been attempts to apply similar technology to DVD drives [13], in order to produce high performance mass storage. However, the number of parallel channels for the described CD and DVD systems do not exceed nine, which is inferior to thousands of channels for PODS systems.

There have been other attempts to produce highly competitive mass storage devices and media. Commercial companies have done research in order to fit up to 40 gigabytes on a conventional size CD. Some of the methods, which they have developed in order to achieve this goal, were:

- Blue lasers, leading to smaller feature size and thus higher data density [25];
- Advanced optical lens systems that deliver clearer readout signals [26];
- Advanced modulation schemes that allow denser data packing [23];
- Writing modes, e.g. modulation of the laser beam rather than just turning it on and off [27];
- Track grooves for data isolation and better synchronization [28];
- New CD media materials for higher density data storage [29].

All of the innovations listed above are promising, however they assume linear data organization in tracks, which limits data density and transfer rates.

A relatively new development is the HD-ROM created by Norsam Technologies [30]. HD-ROM uses a 50nm wide particle beam instead of a laser beam used in DVD (350nm wide beam) and CD (800nm wide beam) drives for recording and readout to achieve higher data density. HD-ROM has a capacity of up to 650GB, while DVDs have 4.7-17GB depending on the number of data layers and CDs have 0.65-0.8 GB on the same size media.

3.2.2 Magneto-Optical Storage

Magneto-optical data storage devices benefit from both magnetic and optical data storage technologies by using a laser to read the data and a combination of a laser and a magnetic field to write the data. The data bits are recorded by heating the magnetic layer with a focused laser beam in the presence of a magnetic field. Changes in the magnetic orientation along a track represent 0s and 1s. The layer also changes the polarization of reflected laser light depending on the stored bits due to the “Kerr Effect” [31]. The magnetic polarity is changed only at a temperature above the “Curie point” (about 200°C).

3.2.3 Advanced Optical Storage

Strong evidence suggests that planar or quasi-planar storage devices such as hard disks, CD and DVD are approaching fundamental limits in data storage density and capacity [32]. Therefore, alternative volumetric and page-oriented data storage (PODS) technologies such as holography [1,12,32-36], spectral hole burning [37-42], and two-photon absorption [3,43-48] have been considered in optical mass storage system research. Using such technologies systems with very high storage capacity exceeding 1 Tbits/cm³ (16 Tbits/inch³) and transfer rates at 1 Gbits/s have been envisioned [12,43,48]. Researchers from the University of Arizona claim that their volumetric bit-wise optical memory can achieve areal density up to 3 Tbits/inch² [49]. They also argue that due to the lower cost and an order of magnitude smaller size and mass makes their storage system a good candidate in comparison of other data storage for space applications.

Theoretically, even higher data densities can be conceived: a study has showed the recording density limit to be 10^{11} bits/cm² (>600 Tbits/inch²) for near-field optical memory that uses photo-chromic medium [50]. For non-near field recording the recording density is limited by the inverse of the recording and readout light source wavelength squared. The areal density could be further improved by using alternative bit layout patterns [51] and modulation codes allowing a tighter fit between bits. This shows that there is a great potential in optical data storage, if the theoretical limits can be approached.

Therefore, the following sections discuss three types of potential systems for high-density page-oriented mass storage: holographic, persistent hole burning, and two-photon absorption memories.

3.2.3.1 Holographic storage systems

In 1963, Pieter J. van Heerden from Polaroid proposed the idea to use holography for 3D data storage. He suggested to read and to write many data images (pages) by referencing a holographic media crystal at different angles and positions [1]. The theoretical limits for the storage density of this technique are tens of terabits per cubic centimeter [2].

The holographic technique records data in the form of binary 2-D array pages transmitted to the holographic media by appropriate modulation device, such as spatial light modulator (SLM). An optical system transfers the data array onto the storage media. The recording of the interference between the data signal beam and a plane-wave reference beam forms a hologram. The recorded page is read out by the illumination of the hologram, with the plane-wave reference used for recording. The reconstruction is projected onto a photo-detector array.

If the angle of the reference beam is rotated with respect to the vector of the signal beam, then the hologram becomes Bragg mismatched, and the reconstruction disappears almost completely [36]. Therefore more holograms can be recorded at the same location by changing the reference beam angle.

In 1994, H.-Y. S. Li and D. Psaltis [34] described an angle-multiplexed holographic disk based storage system. The authors also proposed wavelength multiplexing, which was another method for storing multiple data at the same

location of the holographic media. The authors estimated storage capacity for the 3D holographic disk to be 120 bits/ μ^2 , which was two orders more than for the magnetic storage devices at the time.

In the same year J. F. Heanue, M. C. Bashaw, and L. Hesselink demonstrated volume holographic storage of digital data using a 2x1x1 cm LiNbO crystal as the storage media [12]. They used a differential encoding technique in order to improve bit error rate, which is measured at 10^{-3} - 10^{-4} . In order to improve the data integrity, a Hamming error code having 66% code density was used. Thus BER is improved to 10^{-6} . The raw capacity of the system was 245KB organized as four 308 page stacks.

Two years later A. Pu and D. Psaltis [36] reported using a Dupont's photopolymer as the holographic recording material for the implementation of 3D holographic disk described above [34,35]. The photopolymer was placed on a glass substrate, which made a disk shaped storage media. Areal density of 10 bits/ μ^2 (6.5 GBits/inch²) was demonstrated with 32 holograms superimposed at the same media location. The authors used 100 μ thick polymer, but they argued that much higher areal densities of 100 bits/ μ^2 could be achieved using a 1mm thick polymer.

Eventually, it was noticed that using photosensitive polymers for volumetric storage systems is troublesome due to their shrinkage over time, which significantly distorted the recorded data. Therefore new polymer materials, such as bacteriorhodopsin (bR), are currently being investigated [52, 53].

Researchers at IBM in [54] described holography as a volumetric technique, making its density proportional to one over the third power of the wavelength as opposed to inverse of a square of the wavelength for planar optical storage systems. Additionally, laser beams could be moved with no mechanical components, allowing access times of the order of 10 microseconds. They also explored potentials of holography as content addressable data storage with complex query search capabilities using optical correlation techniques. Such optical search engine could search databases of 100,000 records while keeping the probability of missing even a single record at 10^{-12} .

A high quality optical system is needed for holographic data storage systems in order to obtain reliable data readout. This makes reliable holographic memory expensive and rather space consuming, unless smart data modulation schemes are introduced to prevent errors, thus relieving the design requirements for the optical system.

3.2.3.2 Spectral hole burning

The idea behind persistent spectral hole burning (PHB) is that local effects can shift resonant frequencies of atoms that are otherwise identical [37,41]. This effect can be used to change the readout spectrum of medium locations and observe “holes” in the spectrum. It can be translated into a binary string with a “hole present/not present” rule and, thus, be used for very dense (less than $1 \mu\text{m}^2$ per location) information storage systems. However, the environmental requirements for spectral hole burning (such as near-absolute-zero temperatures) are very hard and expensive to maintain.

In 1986, F. M. Schellenberg et al. [42] showed that spectral hole burning permits using optical energy to store digital information, albeit at cryogenic temperatures of $\sim 4\text{K}$. The optical data access to the storage medium was provided through high quality optical windows in a cryostat. The researchers were able to focus to a 12μ diameter spot using diode laser. The results showed 20-30 Mbit/s transfer rates possible for the PHB data storage system. Such system ultimately has an areal density limit of $10^{11} - 10^{12}$ bits/cm² (0.65-6.5 Tbits/inch²).

In 1993, B. Kohler et al. [39] reported recording of 2000 image holograms in a single spectral hole burning medium. The holograms are recorded at different frequencies and different values of the applied electric field. A hologram is recorded by changing the frequency of a laser over a narrow interval while changing the hologram phase by 2π . Two years later, E. S. Maniloff et al. [40] reported recording of 6000 holograms by use of spectral hole burning. However, the recording and retrieving process required cryogenic temperatures, hence PHB is not considered a practical technique in near future.

3.2.3.3 Two-photon absorption effect storage systems

The two-photon absorption data storage technique records and retrieves data by propagating light beams through a data storage media polymer [3]. During the data recording operation, the polymer molecules affected by a laser beam are excited and they bind with “dye” molecules, thus writing a symbol ‘1’. Typically, the recording location in

the storage media is referenced by an address beam and information beam. The recording occurs in the crossing point of both beams, where sufficient photon energies are reached [44,46]. For the data readout operation the addressing light beam propagates through the media and excites the locations that were recorded as '1'. The excited locations emit light, thus becoming detectable by photo sensors.

A two-photon absorption PODS system is constructed by recording whole data pages (2D arrays of bits) at different depths of the media. The addressing readout laser references one data page at a time, thus creating a data page image, which is optically projected to a photo detector array. The readout can be performed with a laser that is optically shaped to a sheet of light, thus bringing out the whole page of data and providing high data transfer rate. However, this technique is very susceptible to noise and ISI because the excited '1'-bright bits may interfere with their neighboring '0'-dark bits in all three dimensions by adding light intensity. A few researchers have addressed this problem by providing modulation schemes, which prevent the ISI [14,55,56]. However, the proposed solutions involve high overhead in terms of the recorded data size. In order to minimize this overhead and maximize the data capacity for PODS systems, this dissertation proposes the application of a dynamic modulation encoding.

In 1989, D.A. Parthenopoulos and P.M. Rentzepis [47] described a novel 3D optical memory device based on two-photon effect. The proposed memory system wrote data on a photo-chromic material embedded in a polymer matrix. Experimental absorption and emission data showed that two-photon writing and reading is feasible. It was discussed that the theoretical areal density for such devices is $3.5 * 10^8$ bits/cm² (2.1 Gbits/inch²) for recording with 532nm laser. The volumetric density limits were estimated to be $6.5 * 10^{12}$ bits/cm³ (100 Tbits/inch³). Writing was achieved by addressing a location with two laser beams while readout required only one beam. In 1990, two-photon absorption memory system was recognized as a potential competitor to magnetic storage due to high potential capacity and high bandwidth through parallel output [45].

In 1997, M.M. Wang et al. [48] demonstrated reading of 100 planes representing memory pages in volumetric two-photon memory. Signal to noise ratio and bit error rates were characterized. The bit error rate was measured to be $5.13 * 10^{-4}$ for the recorded 100 planes of bits of size 30x30x80 microns each.

In 1998, J.L. Kann [55] showed that both surface and bulk scattering play significant role in two-photon data storage system design through numerical simulations. The perturbations introduced by the system affected the quality of readout data and hence could have led to data corruption. The authors suggested physical solutions to solve the scattering problem, but they also stressed that good modulation codes have to be designed ensuring that the number of ones and zeroes written in every spatial region be roughly equal.

In 2001, researchers at Call-Recall Inc. proposed to use only one laser for the recording process [3]. The recording would occur in the narrowest waist of the laser beam. This eliminated the need for two lasers, however the bit shape becomes elongated in one of the three dimensions, potentially limiting the data density on the medium. The reported bit size was $5 \times 5 \times 20$ microns. The bit size was further reduced to $0.7 \times 0.7 \times 4.5$ microns with projected decrease to $0.3 \times 0.3 \times 2$ microns [43]. The researchers at Call-Recall Inc. also reported a two-photon memory system incorporating 3.5-inch 5mm thick storage media disk featuring readout of 64 channels and delivering throughput of 64Mb/s.

The following section provides a brief overview of data modulation schemes previously proposed for conventional and PODS systems described above.

3.3 DATA MODULATION CODES

Data reliability is a very important issue for data storage systems. Acceptable error rates are achieved with a hierarchy of encoding schemes. First, the source data is encoded using an error correction scheme, such as Reed-Solomon encoding [19], then, the result is further encoded using some modulation scheme that is dependent on the media, such as run-length-limited (RLL) encoding.

A data storage system can be modeled as a data channel as shown in Figure 4. Source data is encoded first by using error correction code (ECC) and then by modulation encoding code, transmitted to the storage media, where noise

and distortions are added, and then the data is decoded and read back. Thus, results from the data channel theory can be applied. The focus of this research is on modulation encoding and decoding components of this scheme.

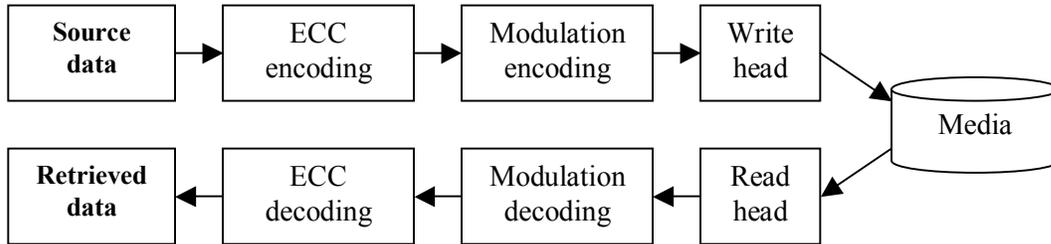


Figure 4: Data storage channel

3.3.1 Technical Requirements for Data Modulation Encoding

There are three coupled goals for data modulation systems: high data density, high transfer rates, and high data integrity. These goals must be met in the presence of noise and inter-symbol interference. This section describes encoding systems that are used to reach these goals with minimal errors in the output. The following are parameters of the data modulation encoding for a storage system. The examples of encoding incorporating these parameters are listed in the next section.

- **Amplitude encoding vs. phase change encoding.** The information is written to the media as a sequence of symbols. In some cases it is easier to read symbol changes (phase change) rather than detect the symbols themselves (precise amplitude), as it is in the case of magnetic storage. Magnetic flux reversals occur when the magnetic polarity on the media changes from NS to SN and vice versa. If absolute magnetic field strength were measured for data reading purposes, its intensity would differ significantly based on the surrounding fields, which contribute their intensity to their neighbors in close proximity. For example, partial response maximum likelihood (PRML) codes are designed to detect these changes in the presence of very high data density, as described in the next section.
- **Synchronization.** It is necessary to mark the beginning and the end of an information fragment. For instance, if there is a single '1' bit placed somewhere in the middle of a string of 10000 zeroes, a

mechanism is required to determine where exactly this “one” bit is. Therefore, some modulation schemes, such as RLL, encode synchronizing clock signals within data.

- **Field separation.** Once information is put on the media very densely, the neighboring symbols start to interfere. This is typically referred to as inter-symbol interference (ISI): e.g. consider an optical memory, where “zero” bit (dark) is surrounded with “one” bits (light). The “zero” bit now has become too bright due to its bright neighbors, and thus it is impossible to be distinguished correctly from “one” bits. The dynamic encoding presented by this dissertation encodes data to minimize ISI while keeping the same field separation distance. The dynamic encoding is described in more detail in the Approach chapter 5.0 below.
- **Noise.** There are several other factors that may influence the quality of the readout information, and, hence, the integrity of the data: e.g. dirt on the media, jitter of mechanical parts, misalignment, and material non-uniformity. All these destructive effects must be tolerated by a data modulation encoding. Dynamic encoding is able to tolerate certain amount of such noise by providing a noise margin as described in the Approach chapter.

Typically, data integrity is measured by bit error rate (BER), which is the number of bits read incorrectly from the media over the total number of bits read. Raw BER represents BER for data read straight from a media without any encoding present. Corrected BER is obtained after a correction scheme is applied. Various media types may have different BER, which may range from 10^{-3} to 10^{-8} . A typical requirement for corrected BER of OEM devices is that it should be below 10^{-12} after all error recovery schemes have been applied [5]. The following sections provide a brief overview to such modulation coding schemes.

3.3.1.1 FM, MFM and RLL

Frequency modulation (FM) was the first commonly used encoding system to store digital data on magnetic mass storage media [18]. The idea underlining it worked as follows: every bit started with a flux reversal (R), thus providing synchronization, and was followed by a reversal in case of ‘1’ and no reversal (N) in case of ‘0’. Thus, there are an average of 1.5 reversals for every data bit (Table 1). This encoding solved the synchronization problem by never having a flux reversal more than 1 symbol apart.

Table 1: FM encoding

Bit pattern	Encoding	Flux reversals per bit	Commonality in a random bit string
0	RN	1	50%
1	RR	2	50%
Weighted average		1.5	100%

Modified frequency modulation (MFM) is an improvement over FM [18]. It reduces the number of flux reversals by inserting a reversal only between consecutive zeroes. Thus, there are an average of 0.75 reversals per bit (Table 2).

MFM doubles the data density over the FM. Today this technique is still used for floppy disks.

Table 2: MFM encoding

Bit pattern	Encoding	Flux reversals per bit	Commonality in a random bit string
0 preceded by 0	RN	1	25%
0 preceded by 1	NN	0	25%
1	NR	1	50%
Weighted average		0.75	100%

Run length limited (RLL) encoding is an improvement over MFM [18]. It has two parameters: run length, which is the minimum distance between the flux reversals, and run limit, which is the maximum distance between the reversals. For instance (2,7) RLL has run length of 2 and limit of 7. In this case, there is an average of 0.4635 flux reversals per bit (Table 3).

Table 3: RLL encoding

Bit pattern	Encoding	Flux reversals per bit	Commonality in a random bit string
11	RNNN	1/2	25%
10	NRNN	1/2	25%
011	NNRNNN	1/3	12.5%
010	RNNRNN	2/3	12.5%
000	NNNRNN	1/3	12.5%
0010	NNRNNRNN	2/4	6.25%
0011	NNNNRNNN	1/4	6.25%
Weighted average		0.4635	100%

An example of encoding a binary string “10001111” is shown in Figure 5. It illustrates how RLL encoding improves over MFM and FM by reducing the amount of space required by the same data bits to one third of the one required for regular FM encoding.

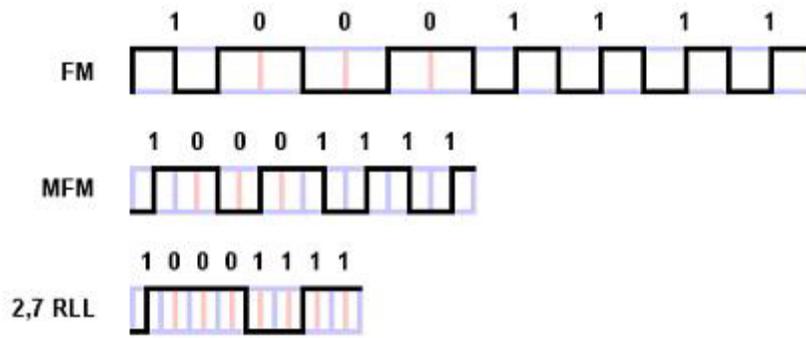


Figure 5: Example and comparison of FM, MFM and RLL modulation codes

3.3.1.2 PRML

Increasing information density on media weakens the intensity of the readout signal to a point where it becomes difficult to detect the flux reversal locations precisely. Therefore, a technology, called *partial response maximum likelihood* (PRML), was introduced [18]. It uses digital signal sampling and processing technologies to manipulate the analog data coming from the media (partial response) in order to determine the most likely sequence of symbols (maximum likelihood). This allows increasing the code density by 30%-40% compared to standard detection schemes. PRML modulation codes are usually decoded with the Viterbi algorithm, which is described below.

The PRML approach requires ability to sample the analog output at each stored symbol, because every symbol is represented as a different amplitude at its location. This may become technically challenging for parallel page-oriented readouts due to the large number of pixels per page and the limited area on the detector device for analog to digital conversion for each pixel. Consequently, the application of the PRML approach for page-oriented memories is problematic.

3.3.1.3 Block codes versus convolutional codes

Block codes take k input bits and produce n output bits, where k and n are large. For block codes, there is no data dependency between blocks. This makes them useful for data communications and prevents error propagation. Conversely, convolutional codes take a small number of input bits and produce a small number of output bits each time period. Data passes through convolutional codes in a continuous stream and may be dependent on the

preceding data. These codes are useful for low-latency communications. One of the most effective methods for decoding convolutional codes is the Viterbi algorithm.

3.3.1.4 Viterbi algorithm

The Viterbi algorithm was proposed in 1967 by Viterbi [58,59] and is a very popular decoding method for information channels that use convolutional codes, including data storage systems. The Viterbi algorithm reads a message, i.e. a string of values, each representing a symbol from media. It works similarly to the traveling salesman problem, where one must find the least expensive path from node A to node B in a graph, where the vertex A is the beginning and the vertex B the end of the message. Each symbol of the message is represented with a set of nodes (arranged in columns) in the trellis diagram as shown in Figure 6, where each node represents a state of the decoder at this symbol position (arranged in rows in the graph). The decoder changes states as it reads the message. The message is encoded prior to writing by using a finite state machine. As a result, only some states are likely to follow each state while reading the message. The states that are likely to be adjacent on the media are linked with edges on the graph, thus, creating a trellis diagram. Based on the readout symbols each edge is assigned a probability that the linked symbols followed each other during encoding process. Then the graph of the probabilities is analyzed. The most likely string of symbols is determined.

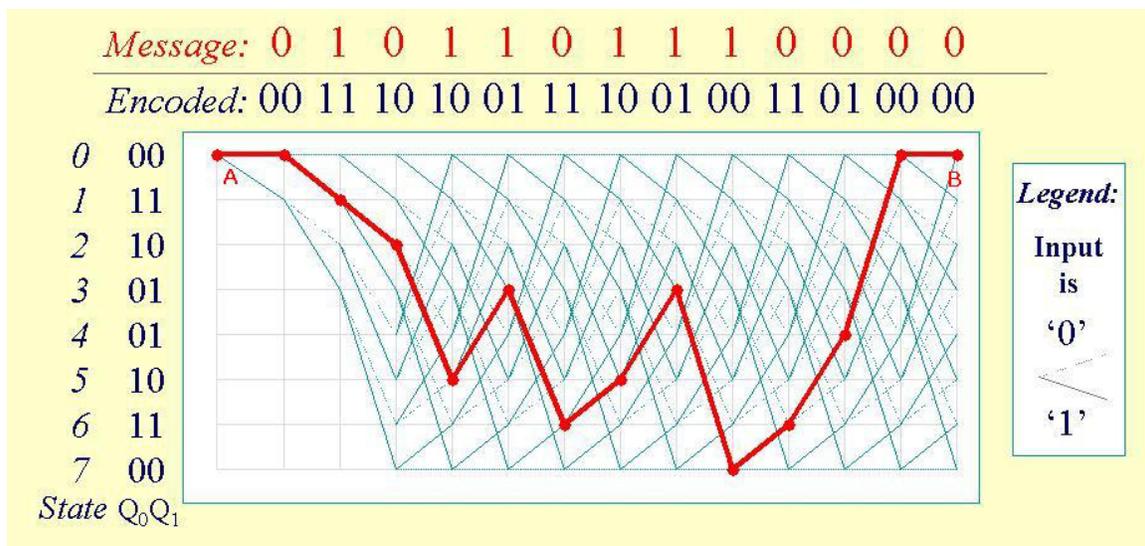


Figure 6: Trellis diagram for Viterbi algorithm

3.3.1.5 *Page-oriented memory codes*

The simpler array codes for 2D memory pages are essentially the parity checking codes in two dimensions [5]. The information in a bit array of a certain size is evaluated, and, then, the parity bits are added to the array for each line and each column. Thus, the code can detect one error per each line or column and correct one error per array. This is not a very strong correcting code, but one could add more error checking bits and achieve higher reliability at the cost of a larger code block size.

Researchers at the University of Southern California have devised an encoding procedure for 2D memory pages called 4/9 encoding [6,7]. This encoding scheme uses a static encoding table and is based on the idea of encoding four source bits in a 3x3 bit array. The information bits are located on the edges, the corner bits are always 0, and the center bit is chosen in such a way that it minimizes the interference with the information bits. As a result, the ISI is minimized. However, the code density is only 45%. In a later work code density was improved to 50% by using variable length two-dimensional modulation codes [60].

Another proposed method for data reliability improvement is the 2D4 algorithm [61]. The idea is to apply a two-dimensional distributed data detection (2D4) algorithm to the problem of binary valued image restoration. The image can be restored in ten iterations for each pixel, where each iteration takes 10,000 clock cycles. This leads to high computational complexity and, thus, slow data access speed.

Researchers at the University of Southern California proposed yet another interesting approach for decoding, in which the data decoding is done in several iterations [51]. Every iteration determines data bits that are unambiguously of a certain value, with a very small probability that ISI is present and marks them as decoded. Every next iteration reevaluates the remaining un-decoded pixel values based on the already decoded information and decodes a few more bits. Through this approach, the number of incorrectly decoded bits is minimized, however not eliminated completely.

4.0 PROBLEM STATEMENT

Figure 1 shows a typical model for a two-photon optical memory [3], in which the data is read from memory in pages. The page addressing laser beam is shaped into a sheet of light and targeted to the memory page we want to read. The storage media is transparent to the laser light. The pixels in the 3D space (voxels), which represent value '1', are excited by the laser light and fluoresce. The optical readout system projects the image to the detector array and decoding system. Each pixel in the 2D page with optical intensity above a specific threshold is read as a '1'.

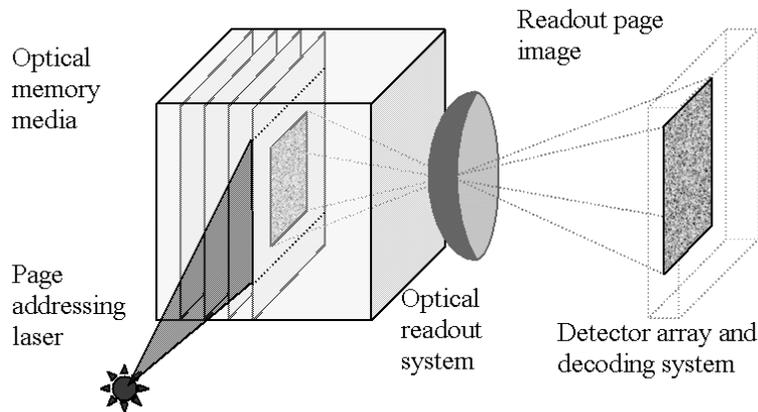


Figure 7: Two-photon optical memory

Ideally, optical power at each pixel on the detector would be relatively uniform for all of the voxels that represent zeros and, similarly, uniform for all of the voxels that represent ones. However, this is generally not the case since a number of effects can either reduce or contribute optical intensity to a pixel. In our research, we have focused on the three most common and most significant effects, with an emphasis on the inter-symbol interference:

- *Inter Symbol Interference.* The light output for each pixel usually contributes some optical power to neighboring pixels on the detector array. This type of crosstalk should be minimized by the optical system; however a cluster of "1" pixels near a "0" pixel can produce enough crosstalk power so that "0" is detected in error as "1".
- *Misalignment and jitter.* The readout head, laser, or media may not be aligned perfectly due to the mechanical nature of an addressing page. Similarly, other kinds of jitter may be introduced due to moving

parts of the storage system, such as spinning storage media disk. Therefore, the image may be slightly shifted and pixels may lack some intensity or contribute some intensity to their neighbors.

- *Non-uniformity of the media.* The material impurities and physical characteristics of the media may have localized effects on the stored data making readout signals “weaker” or “shifted” in some areas due to lowered transparency or refraction.

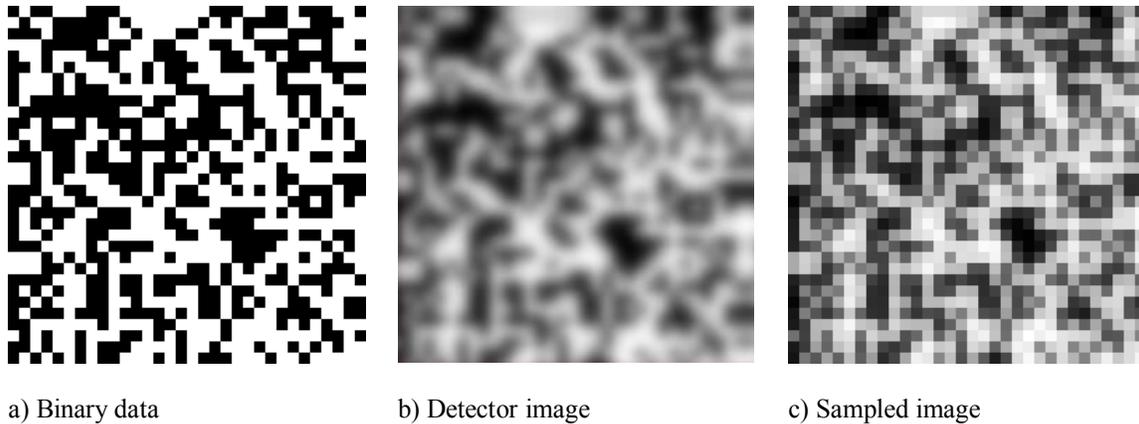


Figure 8: Raw data images

We will refer to the combined effect of each of the three mechanisms as noise in the optical system. Figure 8 illustrates this with three versions of 32x32 bit image: a) the raw, intensity-encoded binary data written to memory; b) the image as the detector array sees it, and c) the image as the detector array samples it. In order to detect correctly ones and zeroes from the sampled image the relative intensity of pixels that encode ‘1’s and ‘0’s must be clearly distinguishable. In other words, each pixel must have an intensity that is clearly above or below a specific threshold in order to interpret it correctly as a “1” or a ”0”, respectively. One method to measure how well a particular page meets this criterion is to generate at each intensity a histogram of the number of pixels. Figure 9 shows an example of one such histogram. The two regions in this picture, represented by the dotted and solid lines, respectively, are the pixels that encode either “zero” or “one”-bits. Thus, the intensity marked “A” represents the lowest intensity “one”-bit and the intensity marked “B” represents the highest intensity “zero”-bit. Since in this example intensity A is less than intensity B, the two regions partly overlap. If a pixel is read at intensity between A and B then it is impossible to unambiguously determine whether the pixel was originally recorded as “zero” or “one” bit.

A reliable data readout requires a histogram such as the one shown in Figure 10. Such a case guarantees that there is no destructive ISI, and the “zero” and “one” bits can be clearly distinguished. It also allows the storage system to tolerate certain amount of noise generated by the effects that have not been compensated for prior to recording, such as jitter and media non-uniformity.

Since system noise cannot be completely prevented, a “noise margin” is introduced as a parameter for encoding. In this context, the noise margin is a relative intensity above and below our detection threshold. The encoding will provide that no pixels will occur between intensities A and B in the histogram in Figure 10. The encoding should manipulate data so that the intensity values for ‘0’ and ‘1’ are sufficiently below or above our detection threshold T. This is achieved by ensuring that there is sufficient spatial separation between ones and zeroes on a 2D code block to limit noise effects.

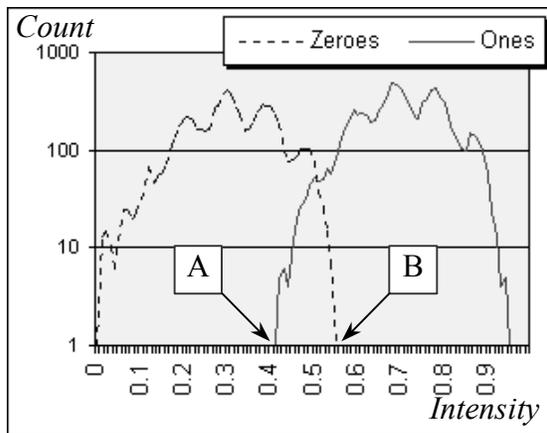


Figure 9: Raw data histogram

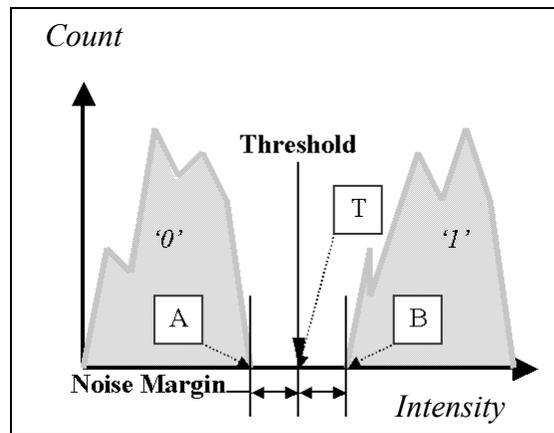


Figure 10: Noise margins

The fundamental problem for a PODS system is to compensate for ISI, jitter, and media non-uniformities in order to preserve integrity of the stored data. The problem is addressed by this research by devising an encoding that produces a histogram according to Figure 10 that achieves maximal code density and has a reasonable time

complexity. In the following chapter we show how to do this by designing an encoding scheme for predetermined threshold and noise margin. The encoding compensates for ISI by spatially arranging bits on a data page so that no destructive ISI is created and the selected threshold can be safely used for data decoding. Jitter and media non-uniformities are tolerated by creating a noise margin, which is built in the code.

5.0 APPROACH

In this chapter we present our approach for data encoding that guaranties a histogram shown in Figure 10. The encoding will prevent destructive ISI and provide a noise margin for other types of noise. First, the main parameters used by the encoding are discussed, followed by a general outline of the dynamic encoding algorithm.

As an example of the noise, *ISI* can cause a ‘0’ pixel that has many neighboring ‘1’ pixels to become “too bright” and, thus, read above the threshold. Similarly, if zeroes surround a weak ‘1’ pixel, it may be read as ‘0’ when a high detection threshold is selected, as our experiments confirmed. Thus, each forward error correcting modulation code must take into account not just the data to be written, but also the information in the surrounding pixels. In this section, we describe our mechanism for generating dynamic codes in real time that are able to maintain a specific noise margin. Among the parameters we use to create these codes are the following:

- Threshold – determines a light intensity below or above which every read data bit is considered “0” or “1”, respectively.
- Noise margin – determines an interval of intensities below and above the threshold in which there will be no data present, if there is no other noise than ISI in the system. Each code is designed to meet the specified noise margin criteria. Even though in realistic optical memory system noise may have topological variations, the encoding assumes the worst-case behavior of the media.
- Code-block size – dimensions of a 2-dimensional bit-array of encoded data. The memory page will be composed of such code blocks (CB). Each new code block will be designed and added to the data page, so that the noise margin criteria is met by the code block, as well as by the data surrounding it.

Figure 11 shows a section of a data page during the encoding operation. In this example, the light gray CB in the middle is being encoded. The dark gray CBs above have been encoded previously and the white blocks are assumed to be all zeroes (i.e., no light sources yet), since they will be subsequently encoded. Even though this particular example suggests a line-by-line encoding of the data page, our approach is not limited to a linear filling. Other page fill orders are possible and were investigated to determine their advantages and disadvantages (see section 0).

11001	01011	10111
00101	11010	10011
01010	11011	11010
10101	?????	00000
10001	?????	00000
11000	?????	00000
00000	00000	00000
00000	00000	00000
00000	00000	00000

Figure 11: Data page fragment

Data encoding and decoding processes consist of two steps: (1) determining the set of valid code blocks (VCS) and (2) mapping the source data using the computed set to the CB (

Figure 12). The approach is based on the ability to estimate the light intensity values for each pixel on a data page taking into account noise effects such as ISI. Thus, an image is created as seen by the photo-detector array of the readout head. Then valid codes for the CB are enumerated, thus creating the valid code set VCS. A code word is considered valid if the light energy it adds to the data page does not create destructive ISI for any of the data pixels surrounding it and within itself, i.e., the threshold and noise margin limits are not exceeded on the data page by

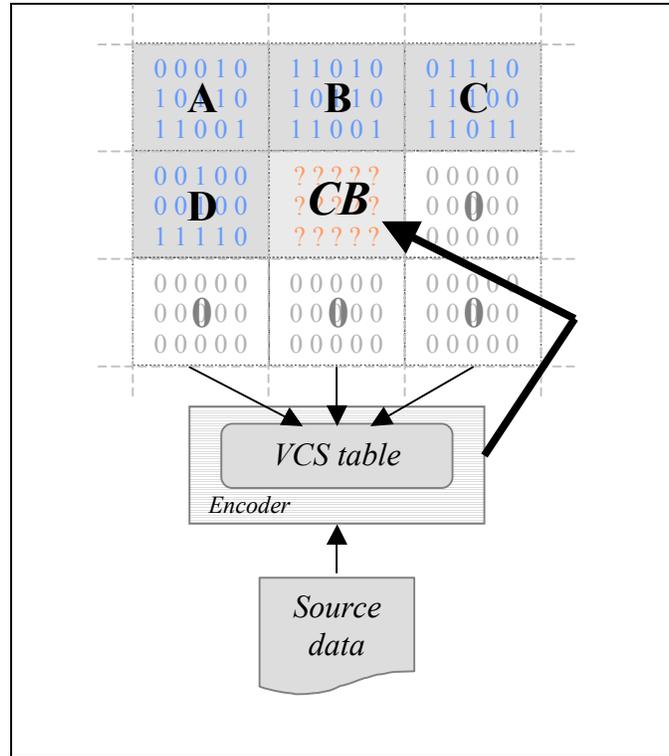


Figure 12: Dynamic encoding approach

placing this particular CB in this location. Thus, VCS is composed from valid code words that are tested using the previously encoded data areas A, B, C, and D as shown in the figure, in other words, VCS is a function of the data in areas (A, B, C, D). Once the valid code set is ready, it is used as a lookup table with a source data word as an entry address in it. The lookup table has the valid code words in the order specified by the VCS generation algorithm, for

example, the order can be ascending. Thus, the source data word is mapped to the appropriate code block by the lookup table and is written onto the optical data page.

In order to decode the memory page, first, the optical data page is read, then each pixel is decoded by the detector array using the threshold, and thus a binary data array is created, which represents the digital memory page. The page is transformed into the analogue array of light intensities by computing the intensity for each data bit and by taking into account contributions from surrounding pixels due to ISI. A valid code set is found for each CB location using the same method and parameters A, B, C, and D from the digital memory page (Figure 12) that was used for the encoding. Before code words are added to the VCS they are tested against the analogue array to determine their validity. Therefore the code will be the same code set as the one used for the encoding. Consequently, for each CB and its respective valid code set, the CB code is found in the set and, thus, the encoded data is uniquely determined in reverse lookup fashion. The uniqueness is guaranteed because the both lookup tables for encoding and decoding are generated using the same parameters – the data above and to the left of the encoded data block.

The valid code set can be generated in various ways by trading encoding speed for code density. For instance, only some code words can be considered, and heuristics can be used to choose code words that are likely to be valid first. However, the most effective method from the point of view of the code density is to enumerate all possible codes that do not violate noise margins. Unfortunately, the worst time complexity for a full enumeration approach is of the order of 2^{N-1} , where N is the number of bits in a CB.

Thus, the focus of this research can be rephrased as to devise modulation encoding algorithms and heuristics that exploit topological properties of data layout on a PODS page and that are both efficient in terms of code density and computation complexity, and well suited to parallel processing architectures, such as the smart readout head described in section 2.3. These algorithms are evaluated using PODS system simulator, which is described in the following chapter.

6.0 EXPERIMENT DESIGN

We start with the definition of a memory model that is used to evaluate the investigated data modulation algorithms. This is a model that represents a PODS memory system taking into account the optical noise generated by inter-symbol interference. The other noise sources, such as jitter and media non-uniformities, can be configured and added to the simulated data storage model. We have chosen the two-photon volumetric memory as media, but the model can be extended to other types of media and page oriented memory systems.

6.1 MEMORY MODEL

The memory model is organized in 2D pages, where each page represents a data page of storage. The page is essentially a 2D array of stored bits, where each bit can assume a value of “1” or ”0”. The symbol “1” corresponds to the presence of a light source at the location. Thus, when writing to a memory, the information is stored to a binary array. If the information for a bit has not been written yet, it is assumed to be “0”, i.e. “dark”. When reading from a memory, a whole data page is generated just as it would be seen by a photo-detector array, one bit per detector. Here each bit is represented with the light intensity computed for each pixel. The intensity depends on the binary value of the pixel as well as on its neighbors. Thus, the memory model incorporates inter-symbol interference. Additional noise can be added to the light intensity depending on the noise source modeled. Hence, we can model effects such as jitter and non-uniformities of the media material. The final binary information of the readout is determined by applying a threshold decision for each pixel on the detector image.

A light distribution from each light source is modeled using the continuous point-spread function (PSF) assuming square aperture [6]:

$$H(x, y) = 1/\sigma^2 \text{sinc}^2(x/\sigma, y/\sigma)$$

Where σ is a parameter that models the degree of blur in the system, x and y are the distances from the source, and

$$\text{sinc}(x, y) = \sin(\pi x)/\pi x * \sin(\pi y)/\pi y$$

The optical memory page image is generated using the 2D convolution operator \bullet with the stored binary data page and the PSF function for each location $[i, j]$ of the data page.

$$r[i, j] = a[i, j] \bullet h[i, j] + \text{noise}[i, j]$$

where $r[i, j]$ is the resulting intensity, $a[i, j]$ is the source binary data, $\text{noise}[i, j]$ is the additive white Gaussian noise or other noise as desired, and $h[i, j]$ is the effective discrete PSF matrix, which is a matrix representing ISI contribution to each neighboring cell $[i, j]$ with the light source in the center of the matrix. The PSF matrix is computed as follows:

$$h[i, j] = \iint H(x, y) dx dy \quad \text{for each square } [i, j] \text{ of PSF matrix.}$$

Ideally x and y are integrated over the whole plane. However, after a certain distance the light contribution is negligible. The memory model approximates the integral to 11×11 pixels with the light source at the center, each pixel approximated with discrete over-sampling method using 64×64 samples per pixel. The results are pre-computed and stored in an 11×11 discrete PSF matrix. The choice of this matrix size is discussed below. The degree of blur represents the quality of the optical system and is set to 1.4. The simulator supports arbitrary data page configurations, such as 64×64 , 128×128 , 256×256 , and 512×512 , 1024×1024 , 2048×2048 and 4096×4096 bit sized pages. In the following sections we discuss the PSF matrix size selection and benchmark data set used for encoding experiments.

6.2 PSF MATRIX SIZE SELECTION

Calculating ISI between all pairs of a data page would create very high overhead for the simulator performance. Besides, such precision is unnecessary, since even cumulative cross talk of distant pixels can be negligible. Therefore calculations were performed to determine how much unaccounted ISI is present if the PSF matrix is limited to a certain size. Symmetry of PSF with respect to any two locations on a data page is used for this purpose.

Note that the PSF is symmetric with respect to any two locations on a data page. If the PSF matrix was calculated with a pixel A as a light source in its center, then at the pixel (matrix cell) B it will determine the light energy fraction that A contributes to B. Now assume that B is also a light source. Then B will contribute exactly the same amount of light energy to A. Therefore a PSF matrix can also be interpreted as a matrix with values that show the

light energy contributed to the pixel in its center by other cells, if those cells have a light source in them. We use this property to determine the unaccounted ISI for a PSF matrix of limited size.

The unaccounted ISI is calculated in the following fashion. We assume one light source, and assume that the total light energy generated by it is one unit. Then we calculate the total ISI generated by this light source for a N by N pixel array, with the light source at the center. The unaccounted ISI is obtained by subtracting the calculated ISI for the NxN matrix from one. Due to the symmetry of PSF, the resulting number is also the worst case unaccounted light energy received by the pixel, assuming that all pixels outside the PSF matrix are light sources.

We performed calculations for PSF matrix sizes ranging from 3x3 to 33x33 cells, and plotted the resulting unaccounted ISI noise in Figure 13. We also decided to design dynamic modulation encoding schemes with noise margin at least 3%. As the plot shows, the closest PSF matrix size that has 3% or less unaccounted noise is 11x11 cells. This size was chosen to be PSF matrix size for the PODS simulator. Note that 3% is the worst case unaccounted ISI for the case when all pixels outside the matrix are light sources. This will hardly be the case, therefore the unaccounted noise will be less, and noise margin will be able to tolerate more of other noise sources, such as jitter and media non-uniformities.

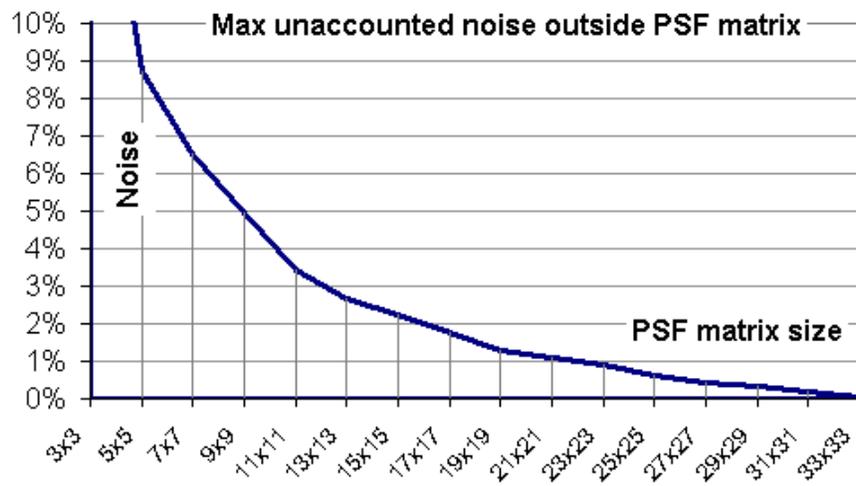


Figure 13: PSF matrix size analysis

6.3 BENCHMARK DATA SETS

The encoding and decoding simulations require incoming data, which will be encoded and stored to the simulated PODS system. Also, some data sets are necessary for the training purposes of certain encoding algorithms described in chapter 8.0. Therefore three data sets were made: random data set, a large set of typical files from a computer workstation, and a set of selected files of popular file types for training purposes. The data sets are described in detail in section 6.3.2 below, but first we discuss randomness quality analysis methods that were used to characterize these data sets. All data sets were analyzed for information density and randomness of the data sequence within the files using the following criteria: Entropy, Chi-square test, Arithmetic mean, Monte-Carlo value for Pi, and Serial correlation coefficient analysis. Although all these tests were implemented in the PODS simulation software used in this dissertation and described in section 6.4 below, an excellent software tool called “ENT” can perform all these tests for single data files independently [62]. The randomness quality tests are described in detail below.

6.3.1 Randomness Quality Analysis

6.3.1.1 Entropy

Entropy is a measure of information density, which expresses how much the information can be compacted [63, 64]. In our analysis we express entropy in percentage by how much a file could be compressed. Lower percentage means higher entropy. A good sequence of random numbers will have a high level of entropy. However, a data set with high level of entropy does not guarantee randomness. For example data compressor software such as *Gzip* or *WinZip* compress data to files resulting in high entropy, but the data is structured and therefore not random. Hence, analyzing only entropy of the test data is not enough when a good quality random sequence is desired.

6.3.1.2 Chi-square Test

The chi-square test is often used for testing the randomness of data [62, 63]. The test is very sensitive to quality in pseudorandom number sequence generators. The chi-square distribution is calculated for the stream of bytes in the file. It is expressed as an absolute number and a percentage that indicates how frequently a truly random sequence would exceed the calculated number value. The obtained percentage is interpreted as the degree to which the sequence tested is suspected of being non-random. The interpretation of Chi-square test is given in the Table 4. For

example, Chi square distribution for 500000 samples of lower 8 bits of the standard Unix *rand()* function is 0.01, and randomly would exceed this value 99.99 percent of the times [62]. This tells that the *rand()* function is not a good source for sequences close to truly random numbers. Therefore we used Mitchel-Moore’s random sequence generator [63], which yielded between 25 and 50 percent in Chi-square randomness tests.

Table 4: Chi-square test measurement interpretation

Chi-square test comparison against a truly random sequence in percents	Interpretation
Below 1% and above 99%	Sequence is almost certainly not random.
Between 1% and 5%, or between 95% and 99%	Sequence is suspect not to be random.
Between 1% and 5%, or between 95% and 99%	Sequence is “almost suspect”.
Between 10% and 90%	Sequence is very likely to be random.

6.3.1.3 Arithmetic Mean

For a truly random number sequence the mean should be right in the middle between the minimal and the maximal numbers to be generated. In our analysis we assume 8-bit numbers, therefore the mean should be close to 127.5, since the numbers in the sequence are integers between 0 and 255.

6.3.1.4 Monte-Carlo Value for Pi

Another evaluation for random sequence of numbers is to use the sequence to calculate Pi value using Monte-Carlo method and then compare to the actual Pi value [62]. A truly random sequence should generate the approximated Pi value that is equal to the correct Pi value. The method of calculating Pi value is as follows: each successive sequence of six bytes is used as a 24-bit X and Y coordinates in a square. If the coordinate is within a circle that fits in the square, this is considered a “hit”, otherwise it is a “miss”. A percentage of the hits is used to calculate the value of Pi. Finally the obtained value is compared to the correct Pi value, and the error is calculated in percents. The lower the error, the better the random sequence.

6.3.1.5 Serial Correlation Coefficient

The serial correlation coefficient is a quantity that measures the extent to which each byte depends on the previous byte in the sequence [63]. For random sequences, this value should be close to zero. A non-random sequence will yield values between 0.5 and 1.

6.3.2 Test Data Sets

6.3.2.1 Random-Set

The input data set referenced as “Random-Set” contained data files of three sizes generated with Mitchell-Moore random number generator [63] in order to obtain a set of uniformly distributed high entropy bit streams. These files were used as the first test set for all algorithms to verify that their performance is valid, competitive, and worth further analysis. Every data file was tested for quality of randomness. Table 5 shows the test results.

Table 5: Random-Set analysis

File	Mean	Inverse Entropy (%)	Chi-square (%)	Monte-Carlo Pi value error (%)	Serial correlation coefficient	Comments
rand_7bit_4MB.bin	127.49	0.0006	50	0.14	0.00015	4MB random sequence
rand_7bit_8MB.bin	127.51	0.0003	25	0.02	-0.00038	8MB random sequence
rand_7bit_16MB.bin	127.49	0.0001	25	0.06	0.00038	16MB random sequence

It can be observed from the table that the generated sequences indeed have high entropy, i.e. the compressibility percentage is very low. All other measures also show that the sequences have good randomness quality – mean is very close to 127.5, Chi-square test results are well within the interval between 10% and 90%, Monte Carlo values have very low error, and the serial correlation coefficients are close to zero.

6.3.2.2 Work-Set

Second input test set referenced as “Work-Set” in this dissertation had 1.3 GBytes of 7673 data files, which were selected from a storage device of a computer workstation with Microsoft Windows 2000 operating system installed. This collection represented a large set of data typically stored on a computer workstation. Six most popular data file

types were selected for the experiments that tested the algorithm performance for specific data types. The selected file types were as follows: TXT – plain text, PDF – portable document format, JPG – compressed image files, MP3 – audio files, ZIP – archive files, and EXE – binary executable files. These subsets were intended for experiments that tried to determine if encoding algorithms perform better on some file types than others. Table 6 shows the analysis for the whole set and for the subsets of selected file types.

Table 6: Work-Set analysis

File set type	Total set size (MB)	Average file size (KB)	Number of files	Mean	Inverse Entropy (%)	Chi-square (%)	Monte-Carlo Pi value error (%)	Serial correlation coefficient
<i>all files</i>	1347.9	180	7673	97.6	20.51	0.41	14.2	0.28275
TXT	3.4	19	182	74.3	41.82	0.01	26.8	0.40098
PDF	152.1	444	351	110.6	6.15	0.01	9.6	0.15822
JPG	414.4	387	1096	122.9	3.26	0.01	3.8	0.19029
MP3	276.6	4425	64	125.9	0.73	0.01	2.6	0.12740
ZIP	58.0	772	77	124.9	1.05	0.01	2.4	0.04942
EXE	78.5	765	105	90.3	23.32	0.01	9.3	0.27859

As can be seen from the table, no data set can be regarded as purely random due to Chi-square test results. However MP3, JPG, and ZIP files show high entropy as well as rather good randomness according to Monte-Carlo Pi value and arithmetic mean tests.

6.3.2.3 Training-Set

Finally, six files were selected for a “Training-Set” as representatives of the most common file types from the Work-Set: TXT, PDF, MP3, JPG, ZIP, and EXE. In addition, one of the randomly generated data files was added to the set. The file characteristics are shown in the Table 7. These files were selected as training data for the encoding table generation for encoding algorithms that are tuned to a specific file type.

Table 7: Training Set analysis

File type and size	Mean	Inverse Entropy (%)	Chi-square (%)	Monte-Carlo Pi value error (%)	Serial correlation coefficient	File Contents
TXT, 4.6MB	82.29	45.0	0.01	27.32	0.23410	Plain ASCII text file. King James Bible.
PDF, 5.4MB	92.27	11.3	0.01	14.10	0.31917	Adobe PDF file. Specification of the PDF file structure.
JPG, 3.1MB	130.69	0.3	0.01	1.62	0.00759	JPG image file.
MP3, 8.5MB	121.94	0.3	0.01	3.04	0.06638	MP3 audio file.
ZIP, 13.0MB	141.43	2.1	0.01	13.31	-0.07463	ZIP file compressed with WinZip software, normal compression ratio.
EXE, 8.4MB	100.50	20.6	0.01	0.12	0.32511	EXE executable binary. Microsoft Word executable.

The Chi-square test results shown in table suggest that none of the tested files is a good candidate for a random data sequence. However, entropy and Monte-Carlo Pi value suggest that JPG and MP3 could be considered semi-random. Interesting to note that, although EXE file has the smallest Pi value error, it is marked as non-random by arithmetic mean, entropy, and serial correlation coefficient.

6.3.2.4 Usage of the test data sets

All algorithms were tested using the Work-Set to evaluate their performance with respect to a large data set of various file types. They were also tested using the Random-Set to observe how they perform for random data streams. The Training-Set files were used for quasi-dynamic algorithm training purposes to analyze whether the algorithm is sensitive to different file types. The data file sets were provided as input data for the simulation software, which is described in the next section.

6.4 SIMULATION SOFTWARE

The optical memory simulation software (OMSS) was created to test various modulation encoding and decoding algorithms for optical page-oriented data storage (PODS) systems. It simulates data recording and retrieval processes for a PODS memory using point spread function (PSF), which describes each bit storage profile. The PSF together with the distance between stored bits determine inter-symbol interference (ISI), which is calculated by the simulation software. Additional noise effects such as jitter and material non-uniformities can be added to the simulation process.

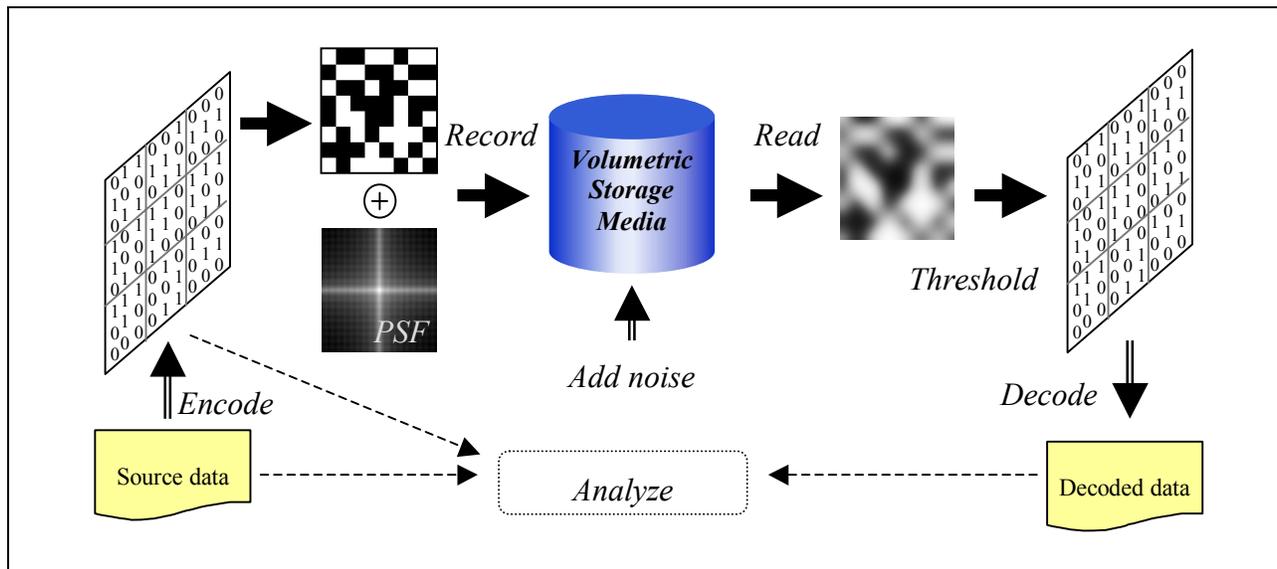


Figure 14: Simulation flow

The simulation flow is shown in Figure 14. Data to be encoded is read from a source file, then encoded using the selected encoding algorithm. The result is a binary 2D array representing a data page in the PODS. Convoluting the PSF that defines recording process profiles for each bit with the bit array simulates the recording process. The result is an array of analog intensity values for each bit. Jitter is simulated by appropriately displacing the PSF matrix before convoluting it with each bit. This analog array can be further manipulated by adding noise, described in detail later in this chapter. Applying a threshold to each bit intensity value simulates the readout of the data page. Thus every bit with intensity below the threshold value is detected as ‘0’ and every bit with intensity value above the threshold is detected as ‘1’. The detected bits make a binary 2D array that is decoded using the pre-selected modulation-encoding algorithm. The decoded data is compared to the original data to measure the bit error rate.

Besides the simulation capabilities the software includes other features that were extensively used for data generation and analysis. Such features include a random data file generation using Mitchell-Moore algorithm [63], File or folder randomness analysis tools, 2D array plotter for visual representations of a memory page and point spread functions, and PSF matrix generation from imported PSF images.

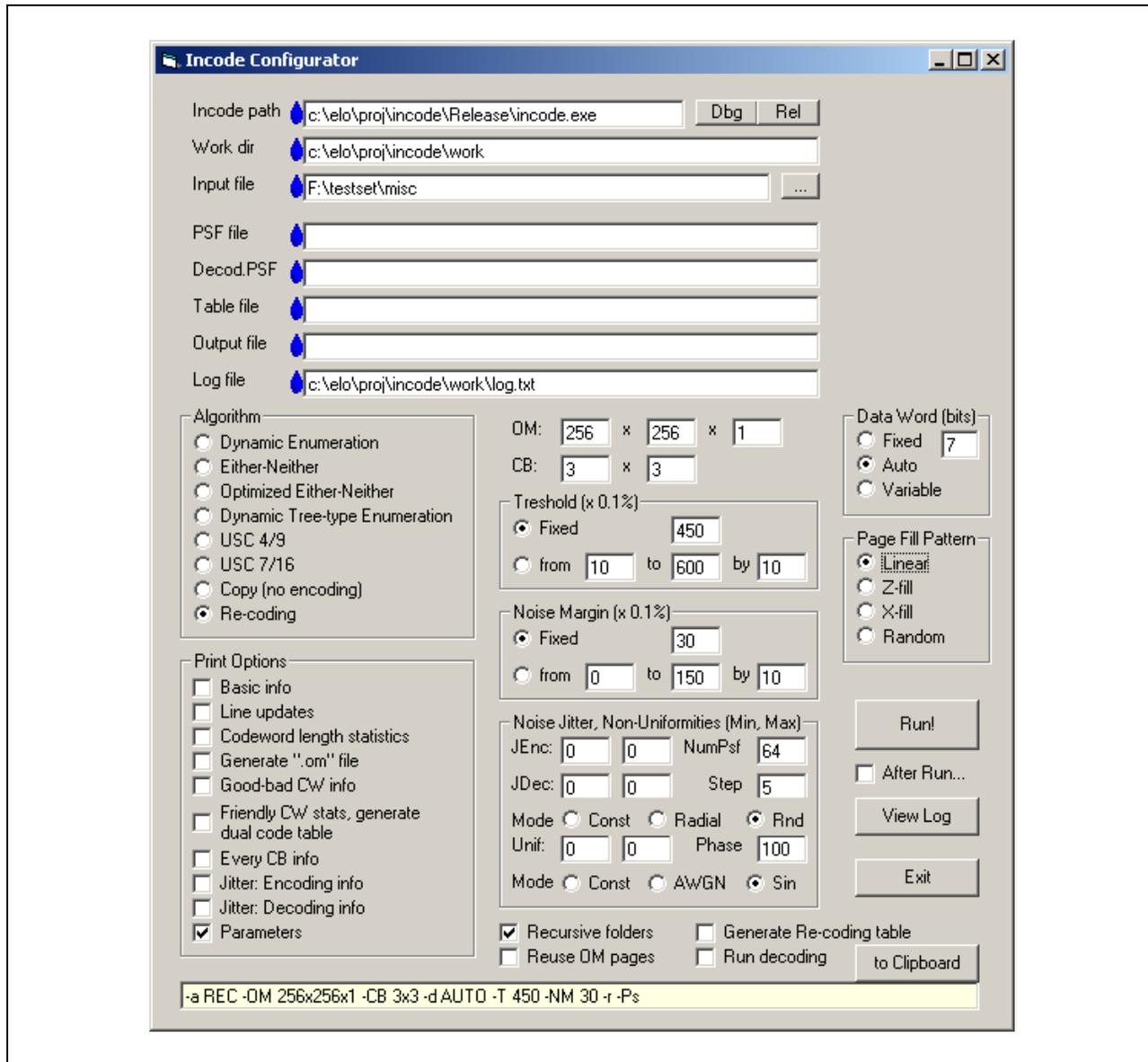


Figure 15: Simulation software graphical user interface

The OMSS provides command line interface for simulation configuration settings. There is also a front-end graphical user interface developed for convenient configuration of the experiments. Figure 15 shows a screenshot of the OMSS front-end dialogue box that allows for configuration and execution of a simulation run. The software features and interface are described in detail in the Appendix B. In the next chapter we describe dynamic encoding algorithms that are implemented within OMSS software.

7.0 DYNAMIC ENCODING ALGORITHMS

In this chapter we introduce the dynamic encoding algorithms. The “Full Enumeration” algorithm is discussed to present the empirical upper bound of the dynamic encoding algorithm. It is followed by a description of the “Either-Neither” algorithm, which trades code density for linear time complexity. The next section presents analysis for different page layout strategies. Finally the dynamic algorithm performance results are summarized, and the need for a better solution expressed.

7.1 FULL ENUMERATION ALGORITHM – EMPIRICAL UPPER BOUND

In this section, we discuss the full enumeration algorithm. Our intention is to measure the improvement in code density between conventional static encoding and our proposed dynamic encoding systems. We compute light intensity values and then build a valid code set (VCS) for code block (CB), which minimizes cross talk and complies with predefined noise margins.

Figure 16 shows the encoding and decoding algorithms in detail. For each code block location a valid code set VCS is constructed. The VCS construction is outlined with a frame around the algorithm part responsible for testing code word validity and their enumeration. This VCS generation part is identical for both encoding and decoding. The difference of the algorithms is that the encoding phase takes bits from the input stream and encodes them by looking up a corresponding code word in the VCS table. The input bit string is used as an index and thus the lookup has constant time complexity. Similarly, for the decoding phase code word is retrieved from CB and then used for reverse lookup to determine the original source word. This, again, can be done within constant time complexity, if the VCS table is generated in a reverse fashion, i.e., since the code words are unique, they can be used as addresses of the lookup table, and the source words as the values stored in the table. If the decoding VCS is organized in this fashion, then the reverse lookup takes constant time complexity, because the code word now can be used as an index of the table.

Phase 1: Encoding

For each code block location **CB**.

Empty valid code set **VCS**
For each possible code word **CW**
If **CW** does not create destructive ISI at the location **CB**
Add **CW** to the **VCS**
End For (go to next bit location)
 $K = \text{Log}_2$ (Number of code words in **VCS**)

Read K bits from the source data stream, use the bits as source word **SW**

Lookup code word **CW** at the position **SW** in the **VCS**

Write **CW** to the **CB**

End For (go to next **CB** location)

Phase 2: Decoding

For each code block location **CB**.

Empty valid code set **VCS**
For each possible code word **CW**
If **CW** does not create destructive ISI at the location **CB**
Add **CW** to the **VCS**
End For (go to next bit location)
 $K = \text{Log}_2$ (Number of code words in **VCS**)

Read the original code word **CW** stored at **CB**

Find **CW** in the **VCS**, remember the **CW** order number **SW**

Write **SW** to the output data stream

End For (go to next **CB** location)

Figure 16: Full Enumeration algorithm. The framed area generates a valid code set.

The straightforward approach for **VCS** generation is to try all possible patterns for each code block **CB** and *enumerate* only the valid ones, thus creating a valid code set **VCS**. Then a bit-string from the source **SW** can be encoded using the enumeration as a lookup table for a maximally dense code in this particular **CB** location. This takes $O(2^N)$ computations per **CB**, where N is the number of pixels in a **CB**. One could pre-compute the lookup table to speed up the process; however, when increasing N , the table size grows exponentially. Even though this approach is computationally very intensive, it gives us the empirical upper bound for the maximum code density, which we could expect with the dynamic encoding approach.

7.1.1 Full Enumeration Algorithm - Simulation Results

We tested the Full Enumeration algorithm for different sizes of CB, starting from 1x1 up to 6x6 bits. Some of the larger code block cases were not tested, such as 5x6, 6x5, and 6x6 bit CBs because the search space for VCS is equal or greater than $2^{5*6} = 2^{30}$, which resulted in prohibitively long simulations. Such large space for searching code words would also be impractical for real time encoding even if the yielded code densities were extremely high.

Before performing the encoding and decoding experiments we needed to decide on threshold and noise margin. Therefore we copied random data to a memory page without any encoding, and obtained the raw data histogram shown earlier on the Figure 9. The histogram shows the midpoint between zero-bit and one-bit lines to be approximately at 0.40 of the maximal light intensity. Therefore, this value was chosen as a threshold for the first set of experiments. The noise margin was selected to be 3% of the maximal light intensity on both sides of the threshold. This value allowed to compensate for the unaccounted ISI due to the limited PSF matrix size, as discussed in section 6.2 above. Once these parameters were decided, experiments for various code block sizes were performed. The page size was 512 by 512 bits and no additional noise sources were enabled.

The Figure 17 shows encoding experiment results plotted to a histogram. The histogram verifies that the noise margins are maintained and there is no destructive ISI after the encoding was performed. The table in Figure 18 shows code density vs. various code block sizes and shapes. The top row and the left column define the dimensions of the code block, while the cells inside the table show the respective code density achieved for this code block. The table data is plotted to a surface graph shown in the lower half of the figure.

The code density for the full enumeration algorithm was 59-80% (as shown in Figure 18) with higher code density for larger CBs. For the same number of bits in CB, equal-sided (square) rectangles yielded better results than non-square. It

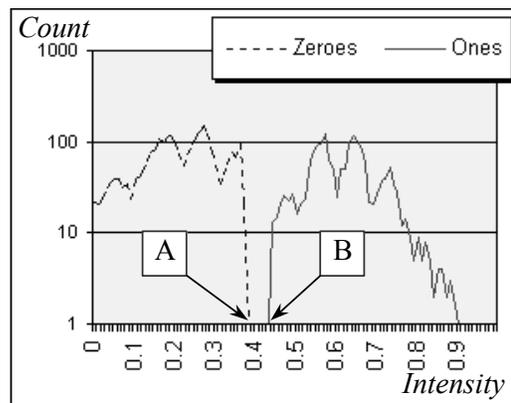


Figure 17: Full enumeration encoding results for 4x4 sized CBs

Code density for various CB sizes. Threshold: 0.40, Noise margin: 0.03

Y\X	1	2	3	4	5	6
1	0	59.27	63.65	64.23	65.28	64.81
2	62.07	68.34	71.68	70.65	69.97	71.94
3	65.11	71.16	74.31	75.24	76.12	76.06
4	66.89	72.81	76.88	75.31	78.19	79.50
5	68.12	74.64	76.79	78.69	80.81	
6	69.36	75.71	78.69	79.74		

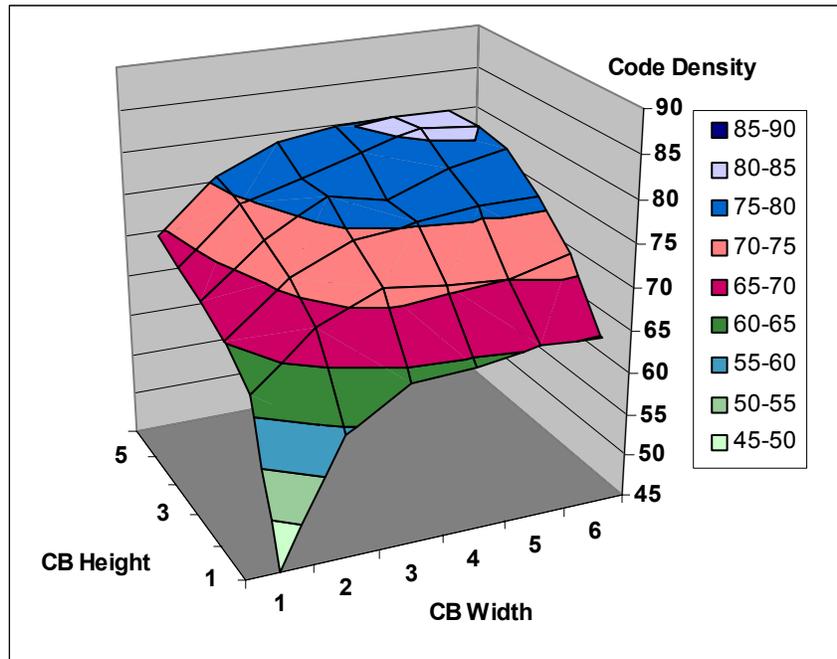


Figure 18: Full enumeration approach: Code density vs. CB size at 0.40 threshold

was also noticeable that “tall” rectangle CBs did not perform as well as “wide” ones. Intuitively, symmetric performance was expected for non-square code blocks, i.e. code block of size 2x4 bits was expected to deliver the same code density as 4x2 code block. However, the code density depended on the code block filling sequence, which is directionally sensitive, since the encoding is performed in left-to-right top-to-bottom raster fashion. Also, the asymmetry came from the fact that “flat” code blocks have much larger area below them filled with zeroes during encoding than “tall” code blocks. Consequently, since none of these zeroes can accept extreme ISI from the CB to be encoded, the CB has more restrictions on code word choice, and smaller number of valid code blocks.

When looking at the surface plot, two interesting features are revealed. First, the best results for code density are over 80%. This is much higher than reported results for a static encoding, namely 45% for 4/9-modulation scheme [6,7]. This suggested that there are modulation algorithms that give good code density and relatively low time complexity, as was confirmed later. The second feature is the trend of the surface to flatten out at CB sizes larger than 4x4. This suggests that it is unlikely that large CB sizes will achieve significantly better code density results; better time performance can be achieved for encoding schemes with smaller code blocks.

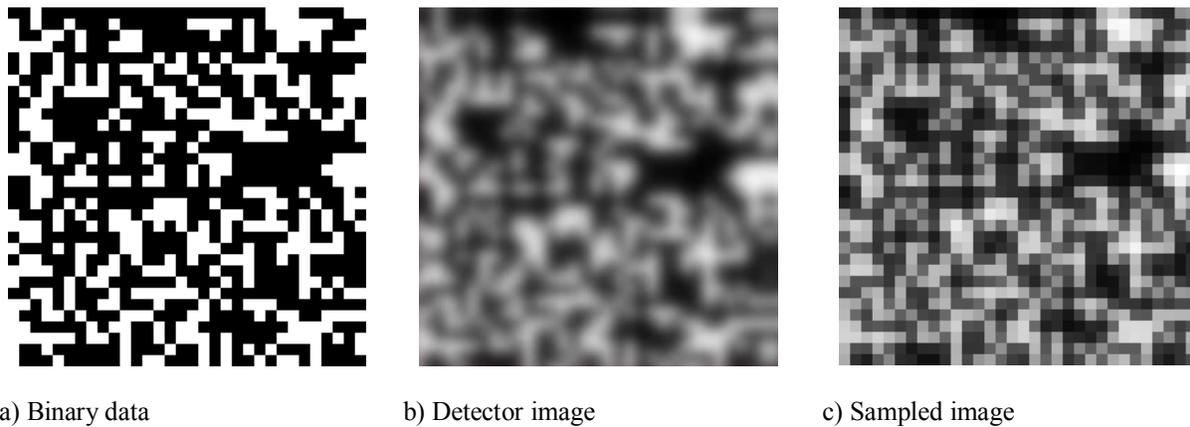


Figure 19: Encoded data images using full enumeration approach

Figure 19 presents a) binary, b) detector, and c) sampled images of memory page fragment after using 4x4 encoding with full enumeration. There are no single isolated dark pixels surrounded by bright light as an effect of encoding, and, thus, the crosstalk is minimized. Even the detector image b) seems to have a better contrast and more clearly defined patterns than the raw data image in Figure 8 b) presented in the problem statement chapter 4.0.

After performing experiments for a fixed threshold we decided to test how the dynamic encoding responds when the threshold or noise margins are modified. The experiments for the same input data set at different thresholds for 2x2 and 3x3 size code blocks generated the results in Figure 20, where some threshold values yield better code density than others. The best code density was 83% for the threshold value of 0.45 units of normalized light intensity. Even relatively small 2x2 blocks gave good code density of 70%.

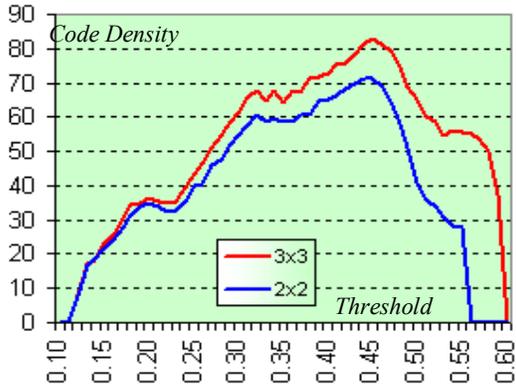


Figure 20: Code density vs. threshold (NM=.03)

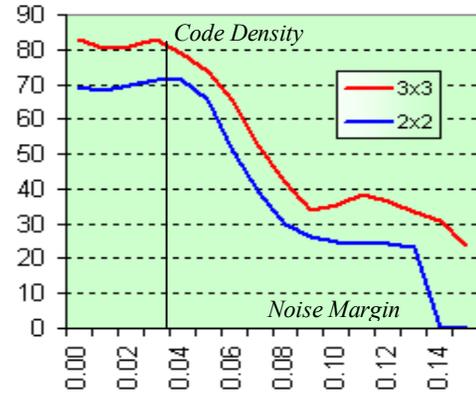


Figure 21: Code density vs. noise margin (T=.45)

Code density for various CB sizes. Threshold: 0.45, Noise margin: 0.03

Y\X	1	2	3	4	5	6
1	0	0	68.95	72.19	73.09	74.51
2	0	71.14	77.83	80.42	81.21	81.56
3	69.44	78.73	82.57	81.14	83.02	82.97
4	72.64	82.06	83.10	83.40	85.52	87.03
5	74.09	80.81	83.63	84.98	85.66	
6	74.91	80.85	84.64	86.57		

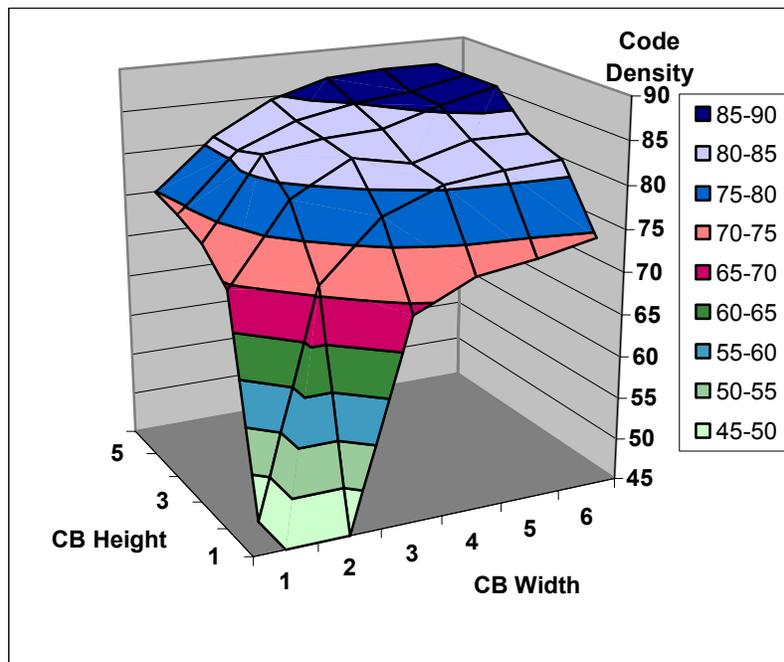


Figure 22: Full enumeration approach: Code density vs. CB size at 0.45 threshold

More experiments for 2x2 and 3x3 size code blocks showed how well the encoding performs for different noise margins. Ideally, a maximally wide noise margin is desirable. The experiments in Figure 21 show that the code density drops quite steeply when the noise margin is extended past 4% of normalized light intensity.

Once the best threshold and acceptable noise margin limits were determined, we performed another set of experiments for different code block sizes, only this time selecting the threshold to be 0.45 of the normalized light intensity. As results in Figure 22 shows, the same trends are observed on the surface plot as it was for the 0.40 threshold case. However, the code density improved, reaching up to 87% for the larger code blocks.

Finally, two more series of experiments were executed by performing Full Enumeration encoding using 2x2 and 3x3 sized code blocks on *Work-Set* files. The threshold was set to 0.45, and the noise margin was set to 3%. The results are shown in the Figure 23 and Table 8.

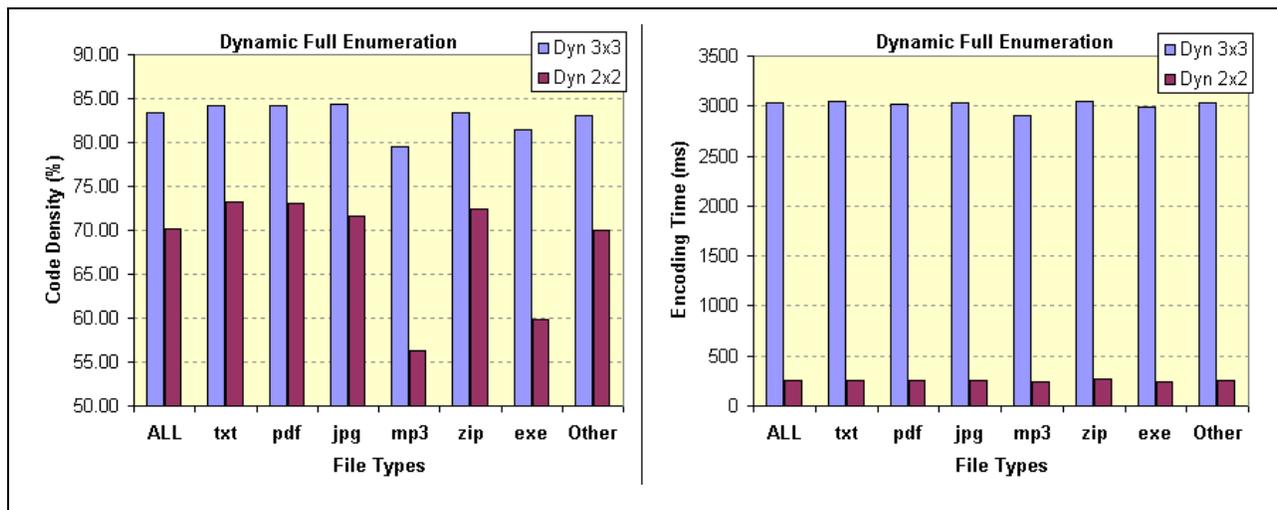


Figure 23: Dynamic Full Enumeration encoding results

The chart shows performance of 3x3 and 2x2 versions of the full enumeration encoding for various file types. It can be seen that both versions follow the same trend in terms of average code density. Interesting to note, that encoding MP3 and EXE file subsets yielded somewhat lower code density than for the rest of the file types, although above 78% in the case of 3x3 version, which is still competitive with 50% code density limit for current static encoding schemes.

The time chart shows average time the simulator required to encode a 512x512 memory page using the selected algorithm and file type. Note, that the time is only for algorithm and file type comparison purposes and the absolute time values are meaningless. However, it is clear that 2x2 version encoding takes noticeably less time than 3x3 version. Table 8 gives a better insight on the execution time difference: 2x2 version takes less than 10% of time that it takes for the 3x3 version to encode a data page.

Table 8: Dynamic Full Enumeration encoding performance

File types	Average Code Density		Average Encoding Simulation Time	
	3x3 CB size	2x2 CB size	3x3 CB size	2x2 CB size
<i>All files</i>	83.31	70.19	3029.74	254.31
txt	84.17	73.18	3051.43	250.90
pdf	84.12	73.11	3020.76	253.86
jpg	84.40	71.58	3034.18	254.25
mp3	79.53	56.35	2901.75	242.75
zip	83.40	72.38	3052.88	272.62
exe	81.38	59.87	2992.91	241.07
<i>Other types</i>	83.10	69.96	3030.54	254.58
MIN value	74.33	50.00	2720	160
MAX value	87.22	74.90	7056	1280
Standard deviation	1.87	4.86	404.9	90.9

7.1.2 Variable vs. Constant Code Rate

Note, that the full enumeration algorithm in Figure 16 generates the maximally dense code for every code block location. Therefore, it has a variable code rate, since the number of bits encoded at each CB location can vary

depending on the total number of valid code words. Modifying the algorithm as follows can enforce constant code rate: choose the constant code word size R . If $\text{Log}(\text{size}(VCS)) \geq R$ or, in other words, if there are enough valid code words in the VCS to encode any R -bit number, then encode or decode the data. Otherwise, write (or skip in the case of decoding) a blank code block that creates no destructive ISI and move to the next code block location.

A blank code block is all zeroes, i.e. all dark, no light sources, hence no ISI generated. However more kinds of blank code blocks are possible, for example a code block with one light source in the center, being far enough for the surrounding information to create destructive ISI. The advantages of such code block are that it can be used for synchronization and clock extraction purposes, and also creating a small contribution to near one-bits (bright bits) and thus improving their signal, while being weak enough and far enough not to create destructive ISI for the zero-bits (dark-bits). If blank codes with light sources in them are used, they should be tested for validity at the location, i.e. they should not be allowed if they create destructive ISI, in which case a completely dark code block should be stored.

Variable code rate approach has a potential to create higher code densities, because it will always use the maximal possible code rate for each code block location. However, a fixed code rate N/M approach may lose a code block location, if there are not enough code words to make up an encoding table that can map any source word of N bits to a code word. A variable code rate approach may be able to save this location by reducing the code rate. In practice, our experiments confirmed that variable code rate yields better code densities for dynamic encoding than fixed code rate, as shown in Table 9.

Table 9: Variable code rate performance v.s. fixed code rate

Encoding Algorithm	Code density (%)
Full enumeration encoding, variable code rate, 3x3 code block, Threshold=0.45, Noise margin=0.03	82.57
Full enumeration encoding, fixed code rate, 7 bits encoded using 9 bits in a 3x3 code block, Threshold=0.45, Noise margin=0.03	73.79

Although a variable code rate creates denser codes, it can contribute to error propagation. Consider a case of variable code rate encoding when due to an uncorrectable error on a memory page a source word is decoded as one

bit short. In this case the whole stream will be shifted by one bit, thus the error will propagate to the subsequent data. Thus, in order to create a stable dynamic encoding a constant code rate is favorable, although it creates a less dense code. If an error occurs within a source word encoded with a fixed code rate, only this source word will be corrupted and likely corrected with an error correction scheme at a higher data abstraction level.

Both variable and fixed code rate versions of full enumeration algorithm were tested. The fixed code rate gave less code density than the variable version, yet the results were impressive. For example, encoding 3x3 code blocks using the variable code rate at the threshold of 0.45 and noise margin of 3% achieved 68% - 87% code density depending on the tested source data file. In the case of fixed code rate $7/9$, where every 7 bits were encoded in a 3x3 block, the code density was between 72% and 77%, which was well above the static encoding.

7.1.3 Dynamic Encoding and Arithmetic Precision

We performed a set of experiments to observe how the dynamic encoding is affected by the arithmetic precision. This would be interesting when implementing the algorithms in hardware, because decisions must be made about the type and precision of the arithmetic. Faster performance and lower overhead in terms of hardware/chip real estate is achieved using integer arithmetic with possibly small word size. On the other hand, better precision and hence better performance in terms of code density is achieved with floating point arithmetic. In order to analyze these tradeoffs and also to display the difference between variable and fixed code rates the simulator permits experiments for different word sizes with integer arithmetic and floating-point arithmetic.

The experiments used 3x3 code blocks for dynamic full enumeration encoding. The simulation software limited the precision of arithmetic operations and numbers, such as ISI coefficients stored in the PSF matrix. We run two experiments for each precision, starting from 8-bits to 20 bits. Another pair was performed using floating-point arithmetic, which uses 64 bits: 1 for sign, 11 for the exponent, and 52 for the mantissa. Each pair of experiments had one encoding session running at a variable code rate and the other at a constant $7/9$ code rate. The input file in this case was from the Random-Set, and the same file was used for all experiments so that the encoding performance results could be meaningfully compared with respect to arithmetic precision and fixed v.s. variable code rate encoding.

The results of these simulations are depicted in Figure 24. It is shown that using 20-bit integer arithmetic costs less than 1% of the code density than using floating point FP arithmetic. This suggests that the implementation of the algorithms can afford to use integer arithmetic without significantly reducing code density. On the other hand the word size should not be below 12 bits, or the performance will suffer significantly. The experiments also illustrate that the fixed code rate approach follows the same rules regarding the arithmetic and precision, and that it reduces the code density to approximately 74%, which is still competitive with the static encoding.

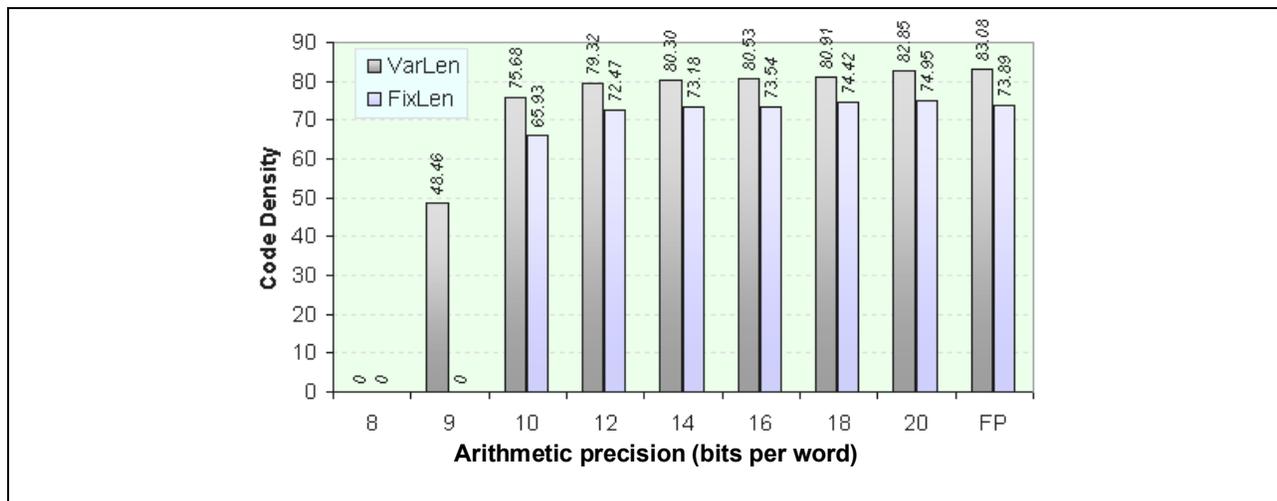


Figure 24: Full Enumeration encoding with variable (VarLen) and 7/9 constant (FixLen) code rates for various arithmetic precisions.

Dynamic full enumeration encoding has exponential time complexity in terms of code block size. This is because the number of code words that need to be tested for validity is exponential. Practical applications would require better time complexity. Therefore we explored other algorithms that trade time complexity for code density. One such algorithm is described in the next section.

7.2 LINEAR TIME “EITHER-NEITHER” ALGORITHM

After Full Enumeration approach proved to be very competitive with respect to code density we started looking into reduced complexity algorithms, such as the linear time complexity algorithm described here. Alternatively to full enumeration the valid code for every CB location can be generated by evaluating the CB *bit by bit* and deciding whether it can assume both ‘0’ or ‘1’ values without creating destructive ISI, as it is done by the Either-Neither algorithm (Figure 25). First, a bit location on a data page is tested in the following fashion: a zero bit is written, then it and surrounding bits within the PSF matrix borders are checked for destructive ISI. Then a one-bit is written to the same location, and again the surrounding area is checked for destructive ISI. If both tests passed, the location is marked valid for one bit encoding since the location can assume either bit value without creating destructive ISI. Then the algorithm takes one bit from the source data, copies it to this location, and moves to the next bit location in the CB. If the bit location was not marked valid, i.e. one or both of the tests described above failed, the algorithm writes zero (nothing) to this pixel and moves to the next bit.

The decoding is done in a similar fashion. First, the detected bit is saved to a temporary memory. Then the bit location is investigated just as before for acceptability of either zero or one bits. If both tests pass, the saved bit value is appended to the output bit stream. Otherwise, this is considered a bad location and the algorithm moves on to the next bit location ignoring this one.

This encoding method is much faster than the full enumeration algorithm and it runs in linear time. However, the resulting code density is much lower than in the full enumeration case, as shown in the following section.

Phase 1: Encoding

*For each code block location **CB**.*

*For each bit **B** in **CB***

*Test **B** by assigning ‘0’ and ‘1’ to it and measuring ISI for its neighbors*

*If **B** can assume value ‘1’ without creating or receiving destructive ISI*

*If **B** can assume value ‘0’ without creating or receiving destructive ISI*

*Read source bit **BS** and write it to the bit **B** location in the **CB***

*Else write blank bit 0 to the **B** location*

*Else write blank bit 0 to the **B** location*

End For (go to next bit location)

*End For (go to next **CB** location)*

Phase 2: Decoding

*For each code block location **CB**.*

*For each bit **B** in **CB***

*If **B** can assume value ‘1’ without creating or receiving destructive ISI*

*If **B** can assume value ‘0’ without creating or receiving destructive ISI*

*Output **B** to the decoded stream of bits*

End For (go to next bit location)

*End For (go to next **CB** location)*

Figure 25: Linear time Either-Neither algorithm

We devised two flavors of Either-Neither encoding. They differ with respect to how they treat the bit location that has deemed to be non-usable due to too much ISI at this location. When a location has too many neighboring pixels in close proximity, the cumulative ISI generated by the neighbors may add up over the threshold. Therefore at this location it is impossible to store zero-bit (dark-bit) and later detect it as zero. Therefore Either Neither algorithm will skip such locations. However, although this location is not used for storage, there still can be a choice whether to write zero or one-bit there. The simple version of the algorithm always write zero. The balanced version will attempt to balance the number of ones and zeroes on the page, thus creating a uniform light energy spread across the whole page. Thus, if there have been more zeroes written so far, then the algorithm will attempt to write one-bit in the blind pixel area. If no destructive ISI is created, then the blank bit location has one-bit now. If, however, by writing one-bit destructive ISI is created, then the algorithm will write zero-bit. We called this “Balanced Either-Neither” algorithm.

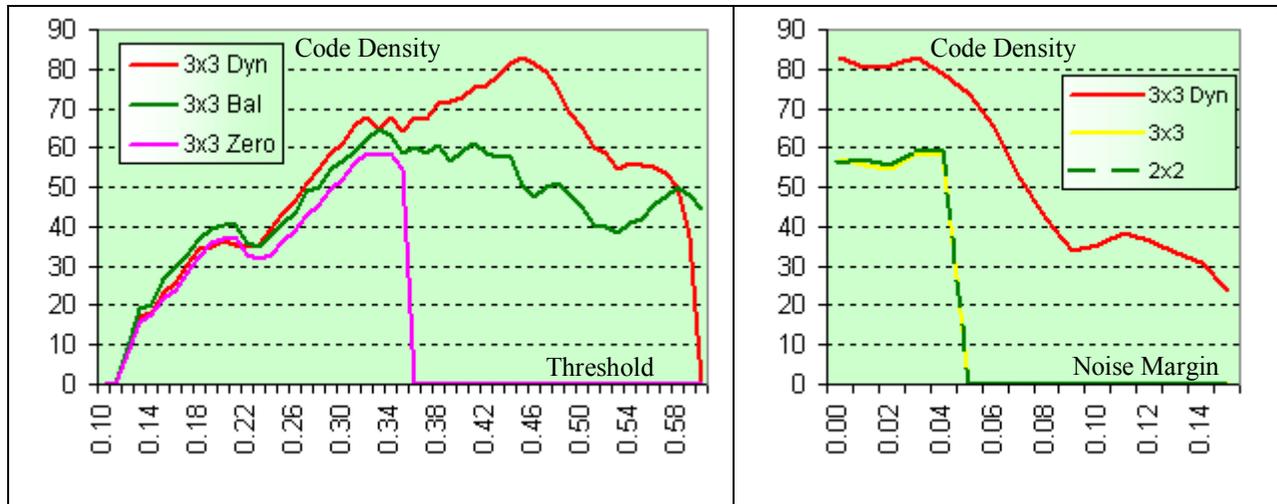
7.2.1 Either-Neither Algorithm - Simulation Results

The Either-Neither approach performed at linear execution time, which is an improvement over the full enumeration encoding. However, the code density generated by the Either-Neither algorithm was between 45% and 60% for a 3x3 code block size. This algorithm slightly outperformed static encoding in terms of code density, which is usually below 50%. However, its time complexity was linear as opposed to constant for the static approach. Therefore, the marginal gain in code density was not worth the time overhead.

Several experiments were conducted in search of better performance by changing the detection threshold and noise margin for the Either-Neither algorithm (Figure 26). Dynamic full enumeration performance curves were also plotted for a comparison in the figure. Yet no better performance than 60% was achieved. It was observed that the threshold for the best code density for Either Neither algorithm is at 34% of light intensity and that the noise margin should not exceed 4% of the light intensity. The graphs also show that the full enumeration approach performs significantly better and has a different optimal threshold.

As the histograms show, there is no difference in terms of noise margin between the simple and balanced Either-Neither algorithms. However in terms of threshold the balanced version performs significantly better than the simple one. This is because for the simple approach the code density drops to zero with thresholds above 0.36, since in case of bit-failures a zero is always written. If a one-bit were written, it may increase light intensity for its neighboring bit locations so that the intensity may be higher than the threshold for valid encoded one-bit locations.

Finally, we run Either-Neither algorithm with 3x3 code block size on the whole *Work-Set* file collection. The results are plotted in Figure 27 and compared to Dynamic Full Enumeration encoding performance. The charts show that the Either-Neither performance is only slightly above 60%, and usually by approximately 20% less than Full-Enumeration performance in terms of code density. The chart also shows that code density was the lowest for encoding ZIP files. For all file types the minimal code density was 43.91%, average was 67.76%, and the maximal reached code density was 90.68. Standard deviation was 5.94 percent.



Code Density (vertical axis) vs. Threshold (horizontal axis).

3x3 Dyn - Full enumeration

3x3 Bal – Balanced Either-Neither

3x3 Zero – Simple Either-Neither

Code Density (vertical axis) vs. Noise Margin (horizontal axis)

3x3 Dyn - Full enumeration

3x3 and 2x2 – Either-Neither

Figure 26: Threshold and Noise Margin for Either-Neither and Full Enumeration encoding algorithms

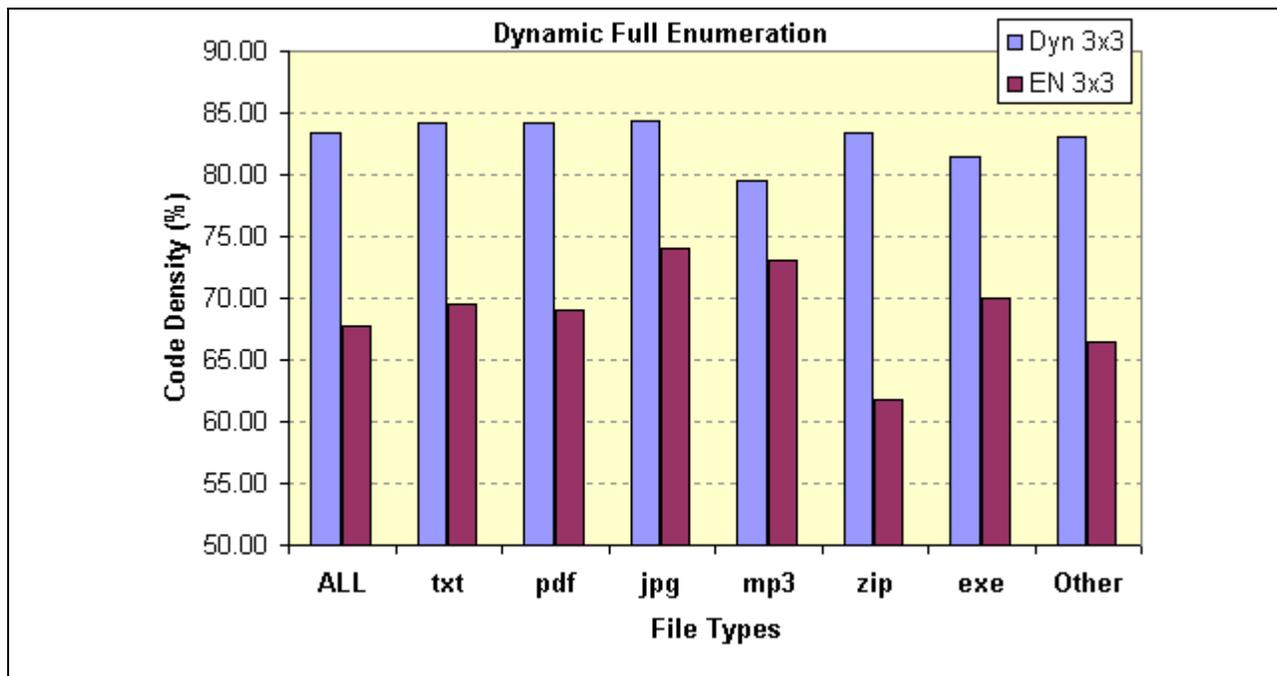


Figure 27: Either-Neither encoding performance

In the search for improved code density, different tactics were tried with respect to the bit fill order within a code block. The following strategies were tested: filling columns row by row, filling rows column by column, both diagonals starting from both top and bottom. However, no noticeable code density improvement or degradation was observed. This left Either-Neither as an interesting, although impractical algorithm for dynamic encoding.

7.3 DYNAMIC ENCODING SUMMARY

The highest code density that we were able to achieve was 87% obtained by performing the full enumeration algorithm. However, the full enumeration approach achieves such high code density at the cost of exponential complexity. In the attempt to reduce the complexity a linear time algorithm was designed and analyzed, however the code density of 60% was only marginally better than for static encoding. Therefore we decided to pursue alternative encoding schemes, exploring a new encoding class between the static and dynamic encoding classes. Thus we devised quasi-dynamic encoding, which eventually offered the best performance/complexity tradeoff and is described in the next chapter.

8.0 QUASI-DYNAMIC ENCODING PARADIGM

Static encoding assumes worst-case scenario for the data that surrounds a code block (CB) to be encoded. This results in a relatively low code density, often below 50%. In earlier chapters, we proposed a more aggressive technique called *dynamic encoding*, which yielded code densities up to 83%. Unfortunately, the high code density came at the cost of exponential time complexity. Therefore, in order to find a suitable tradeoff between the code density and the complexity alternative algorithms were searched. As a result, a quasi-dynamic encoding approach was developed, which for every code block dynamically selects or assembles a code from a list of precomputed encoding tables or table fragments. Our most successful quasi-dynamic encoding algorithm, called Re-coding, can be performed in constant time, in terms of code block size, and provides code densities up to 77%.

The *quasi-dynamic encoding approach* initially analyzes the code block CB surrounding information and then selects a *precomputed* code table that would work best for the given configuration on the data page. More flexibility is achieved by a slightly modified approach that instead of pre-computing the whole tables pre-computes only fragments of tables. The final encoding table is assembled dynamically on the fly using only the table fragments most suitable for the code block location. In fact, the whole encoding table does not even need to be assembled, but just the appropriate sub-table needs to be selected and used for encoding of the actual source word. Thus, more virtual code tables are available and better code density is achieved. For example, if the system encodes 7-bit data words into 9-bit words and allows a choice from two code words for each data word, then the number of virtual code tables is two to the power of number of source words, i.e., 2^{128} . By having more available code tables the approach is more flexible in the choice of the most suitable table and, therefore, can achieve better performance in terms of code density. This assumes that the code word pair selection is precomputed instead of the whole tables. The developed *Second Chance Re-coding* algorithm is based on this approach.

8.1 “SECOND CHANCE RE-CODING” ALGORITHM

The main idea behind the *Second Chance Re-coding* algorithm is that every source data word can be encoded by any of several code words. A code word that creates the least ISI will be selected to encode the particular source data word, depending on the information surrounding the CB. The second chance re-coding system is created in such a way that every source word is assigned one or more code words, and there are no code words that are assigned to more than one source data word.

For example, a 2/4 version of quasi-dynamic encoding could be defined as follows: every 2 bits will be encoded with 4 bits as in Table 10. For instance, if a source word “10” is received for encoding, then it can be encoded using either code word “1001” or “0110”, depending on which is more appropriate for the location.

Table 10: Second Chance Re-coding example

Source word	Code word 1	Code word 2
00	0101	1010
01	1100	0011
10	1001	0110
11	0000	1111

We continue with definition of the encoding and decoding algorithms, followed by a discussion about creating the *Second Chance Re-coding* table (**ENC**).

8.1.1 Encoding and Decoding

The *Second Chance Re-coding* algorithm (**Figure 28**, Encoding Phase) has a choice of 2 code words (**CW1** and **CW2**) for encoding each source word (**SW**). In order to encode a source data word SW the algorithm finds the first predetermined code word CW1 pre-assigned to the particular SW and tests it for destructive ISI with the surrounding data. If CW1 passes the test, i.e., no destructive ISI is created; it is used to encode the SW.

If CW1 does not pass the test, the other predetermined codeword CW2 is selected. Again, CW2 is tested for destructive ISI, and used if the test passes. If the second code word also fails, a blank, nondestructive codeword in ISI sense is written signifying that no information is encoded in this location, and the algorithm has to move to the next code block location. The blank code word is a special code word or a selection from a list of code words that

Phase 1: Create re-coding table. (Done offline once per memory system.)

For each codeword CW1

For each codeword CW2

For the whole training data set

Choose random surrounding of the code block

Test if CW1 is OK (if CW1 does not create destructive ISI)

Test if CW2 is OK

If (CW1 is OK) OR (CW2 is OK)

Increase probability counter P for (CW1, CW2) pair

End-all-loops

Create a table T with the probability counter (likelihood) indicators P for all codeword pairs (CW1, CW2)

Do “perfect roommate” matching on T so that the aggregate likelihood P for all of the selected distinctive non-intersecting codeword pairs is maximized.

The result is the encoding table ENC

Phase 2: Encoding

For each source word SW try CW1.

If CW1 OK, done.

Else, try CW2.

If CW2 OK, done. Source word SW encoded.

Else write blank, and move to the next code block location.

Phase 3: Decoding

For each received codeword CW,

If CW is marked as blank, move to the next location.

else do reverse lookup from the encoding table ENC.

Figure 28: Second Chance Re-coding algorithm

are not used for actual SW encoding. The blank code word can be either completely dark or contain a pattern that would serve for clock extraction purposes or contain a low level light sources that would improve the surrounding one-bits and yet be low enough not to create destructive ISI for surrounding zero-bits. The code word pair assignment to source words is chosen in such a manner that the likelihood of neither of the code words being accepted is minimized. The creation of a re-coding table with such a property is an interesting problem and it is discussed in detail in the section 8.1.2 below.

Decoding is straightforward: code words are detected on a data page and then decoded to source words using reverse lookup from the ENC table. The decoded SW is unique because every CW in the table is assigned only one code word. Also, the blank code words are assigned no source words; instead they are marked as blank. Thus, when the decoding algorithm reads a blank code word, it moves to the next CB location and retrieves a new CW from there.

The test of a code word for destructive interference involves assessment of ISI around the selected code block within a limited distance. This can be done computationally in constant time because the PSF matrix size is constant. Thus, the whole encoding is done in constant time. Decoding is a simple reverse lookup of a source word that corresponds to the code word received. The most interesting and, consequently, most difficult part is choosing code word assignments to source words and building the encoding table. Although building this table is computationally intensive, it needs to be built only once for a particular memory system and media.

8.1.2 Construction of the Re-coding Table

The re-coding table consists of code word pairs that achieve the following property with a maximized likelihood: in arbitrary surroundings of a memory data page either the first code word of the pair creates no destructive ISI, or the second code word creates no destructive ISI. Sometimes the property is not achieved and both code words create destructive ISI, however this should happen very rarely if the pairs for the table are carefully selected. We call this table “re-coding” because every source word will be encoded using the first code word of the pair, and then, in case of failure, re-coded using the second code word of the pair.

In order to create the re-coding table we first collected statistical information about how much destructive ISI each code word generates in random configurations of a data page (Figure 28, Phase 1). Some code words were discarded after this step because they created destructive ISI all the time. For example, creating a re-coding table for 3x3 block encoding out of 512 possible code words only 311 were kept in a good code word set as potential candidates for the re-coding table. Then we collected information on all possible code word pairs from the good code word set, counting the cases when either code word of a pair was success, i.e., created no destructive ISI in the same memory

data page configuration. The results were arranged in a 2D table with empirically determined likelihoods that the code word pairs are a success. A fragment of such table is shown in Figure 29 a). The columns and rows are labeled with code words, and the table cells represent the likelihood of the corresponding code word pair to be a success. For example, a codeword pair (c0, c2) has a probability of 0.86 to avoid destructive ISI in an arbitrary configuration.

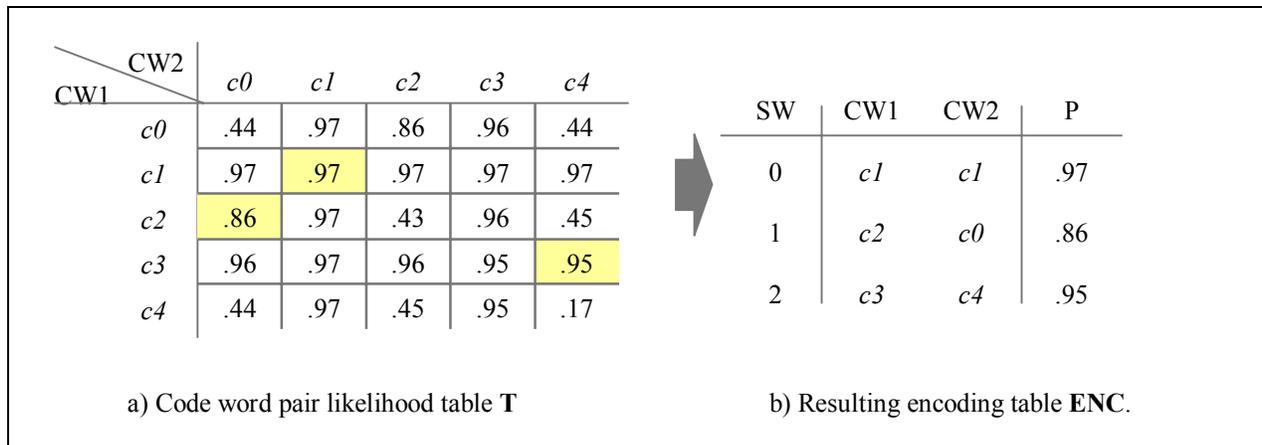


Figure 29: Re-coding codeword likelihood and encoding tables

In order to create a re-coding table from the likelihood table example in Figure 29 a) a list of non-intersecting pairs have to be selected that maximize the total likelihood of being success. For example, the selected pairs in the figure are colored in yellow. The chosen approach for re-coding table generation did this by maximizing the sum of selected code word pair likelihoods. This is similar to the “*stable roommates*” matching problem [65]. Fully exhaustive search for the best code word pairs is unaffordable for large data sets; for example, a table of 256 rows and columns can have $O(256^{128})$ different sets of pairs. Therefore, an approximation method was created to find a good acceptable solution. Once the pairs were selected, they were assigned to source words, and the encoding table was created.

If the distribution of source words is not uniform, the re-coding performance is improved by assigning codeword pairs of higher likelihood P to source words that appear more often in the input stream. Thus a file-type tuned encoding was envisioned and tested as described in the section 8.5 below.

In order to build the re-coding table statistical information about how much ISI each code word creates was collected. Ideally every code word should have been tested for every possible configuration of surrounding information. However this was unrealistic because even in the case of relatively small 3x3 code block assuming that ISI has no contribution beyond the distance of 5 pixels (as defined by the size of the PSF matrix chosen in chapter 6.0) the configuration space size would reach $2^{(5+3+5)*(5+3+5)} = 2^{121}$ different configurations for each of the 2^7 code words. Therefore a limited number of configurations were used for testing each code word. In fact, a study was conducted on how a training data set size affects the performance of the *Second Chance Re-coding* algorithm, as discussed in section 8.2 below.

8.2 RE-CODING TABLE GENERATION EXPERIMENTS

A set of experiments was performed to determine how training data size for re-coding table generation affects *Second Chance Re-coding* performance. Each experiment generated a re-coding table and executed the Re-coding algorithm on the *Work-Set*, which contained 1.3GB of files (See section 6.3.2 for test data set descriptions).

8.2.1 Training Set Selection for Re-coding Table Generation

The re-coding table generation depends on code word likelihood to create destructive ISI. In order to obtain such information a 16MB file was selected from the *Random-Set* and encoded using the full enumeration dynamic encoding algorithm. During the encoding every possible code word was tried for every CB location on the data page, and then one was kept depending on the input data stream. As the encoding progressed, the data page filled up with encoded information providing “destructive ISI free” memory page configurations for code word testing. Thus every code word was tested in a number of configurations.

The main question the experiments were designed to answer was: how large number of configurations is sufficient for obtaining the code word likelihood information that is used to generate the re-coding table, so that the re-coding algorithm performance using the generated table is competitive with the static encoding techniques, i.e. the code density is above 50%. In order to answer this question we performed series of re-coding table generation

experiments which differed in the size of the optical page that was filled with encoded data, and consequently the number of code block locations and times every code word was tested. The code block size was 3x3 bits. The results of the re-coding performance after each code table generation are shown in Table 11 and Figure 30.

Table 11: Re-coding table generation experiment results

Page Size in Bits	Page Size in CBs	Code Density			
		Min	Max	Average	Standard Deviation
32 x 32	100	69.33	77.78	75.57	1.09
64 x 64	441	73.29	77.78	75.80	1.02
128 x 128	1,764	67.68	77.78	75.92	0.99
256 x 256	7,225	70.85	77.78	75.82	0.91
512 x 512	28,900	70.83	77.78	75.81	0.83
1024 x 1024	116,281	66.39	77.78	75.81	1.04
2048 x 2048	465,124	71.39	77.78	75.82	0.89
4096 x 4096	1,863,225	71.74	77.78	75.88	0.89

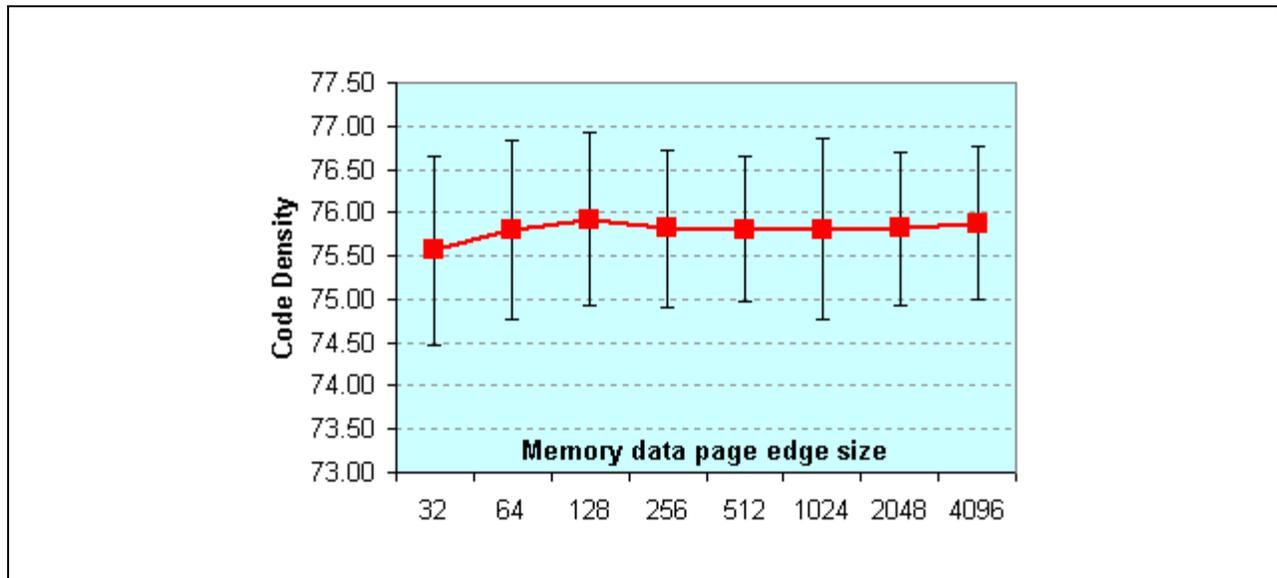


Figure 30: Re-coding table generation experiment results

The table lists the data page size in bits and code blocks for every experiment series. It also shows the minimal, maximal, and average code densities for Re-coding after the re-coding table was generated. The results are depicted in the figure and show code density for each data page edge, i.e., for each data page size and therefore for the number of times every code word was tested. The plot also shows the corresponding standard deviation bars, which are in the range of 1%. The experiments suggest that already after testing every code word 1.8 thousand times a competitive average code density of 75% is achieved, and that performing more tests for code word likelihood to create destructive ISI does not bring significant improvement. Thus it was decided to use less than 30000 tests for each code word when generating the re-coding table as provided by a data page of size 512x512, since more tests are unlikely to bring superior results.

8.2.2 Testing Re-coding Table Generation for Certain File Types

The next set of experiments was run to determine if Re-coding performance could be tailored to particular data types. Therefore, new re-coding tables were created using benchmark files from the *Ref-Set* as input for the re-coding table generation. The experiments were run for six file types: TXT, PDF, JPG, MP3, ZIP, and EXE, which are described in section 6.3.2. For each series of experiments the corresponding file from the *Ref-Set* was used to generate the re-coding table. A data page of 512x512 bits was used for this purpose providing 28,900 different configurations for each code word testing. As was concluded in the section 8.2.1, a training set of this size is sufficient for good Re-coding performance. Finally, after the re-coding table was generated, the whole *Work-Set* was encoded using the re-coding algorithm. The results were viewed as a whole and also for the target group of the specific file type. The results of the experiments are shown in the Table 12 and Figure 31.

Contrary to intuition the experiments showed that using certain type files for re-coding table generation did not favor higher code density for other files of the same type over any files. The graph in the figure shows that in almost all cases average code density for all files is higher than the specific type files for what the re-coding table was supposedly tailored. The only exceptions are TXT files and EXE files. It is interesting to notice that according to *Work-Set* analysis in section 6.3.2.2 these file types have rather low entropy and the arithmetic mean is away from the middle value. This suggests that the particular files have non-uniform data word distribution, and therefore tailoring to them may have slight advantage, although very minimal, less than a quarter percent of average code

density. Therefore the conclusion is that tailoring re-coding table to specific file types does not bring significant improvement in terms of code density.

Table 12: Generating Re-coding tables for specific file types

File Type	Code Density				
	Min All Files	Max All Files	Avg All Files	Standard Deviation	Average for Files of the Trained Type
TXT	70.80	77.78	75.75	0.93	75.96
MP3	71.25	77.78	75.83	0.97	74.77
PDF	73.20	77.78	75.91	0.92	75.03
ZIP	71.88	77.78	75.94	0.93	74.65
EXE	71.61	77.78	75.12	1.26	75.34
JPG	71.43	77.78	75.99	0.95	75.08

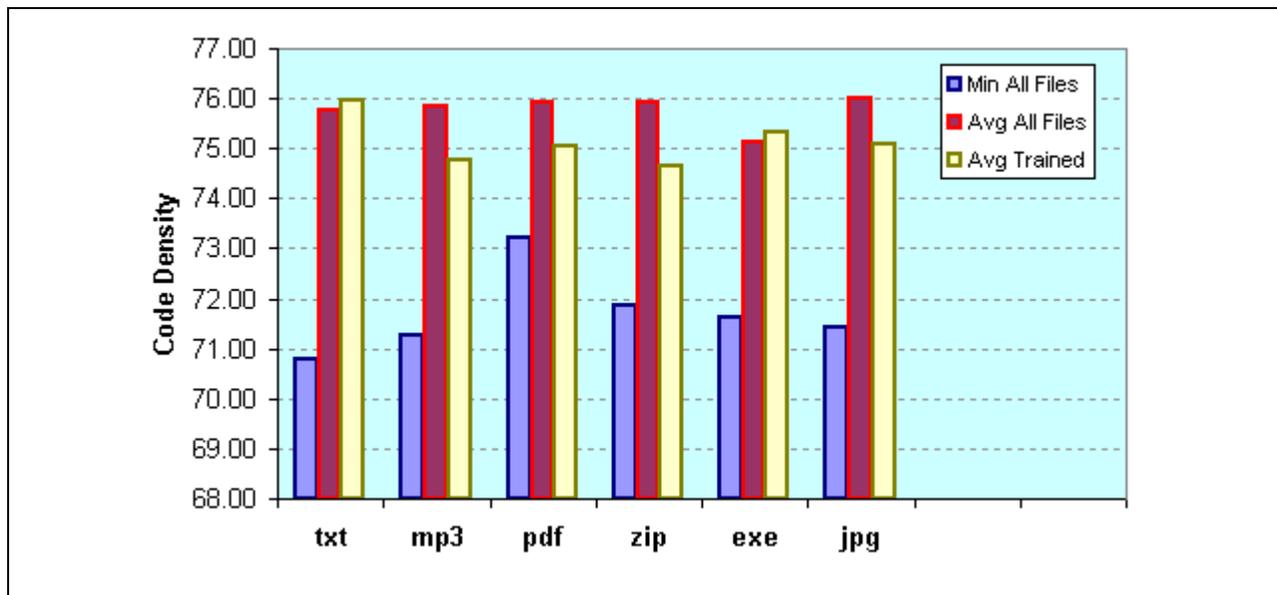


Figure 31: Generating Re-coding tables for specific file types

Another attempt to improve the performance of the encoding in terms of code density and potential parallel implementations is the order how code blocks are assigned or decoded on a data page. We call it a “*page fill strategy*” and discuss it in the next section.

8.3 PAGE FILL STRATEGIES

In this section, we outline possible strategies to fill out an optical page with code blocks for the encoding process. Two main alternatives are considered: filling out in a sequential fashion and filling out using a pseudo random pattern. The general idea behind the page fill strategies is to allow code blocks to have as much freedom in terms of not being impaired by their neighbors through inter-symbol interference (ISI). Thus, one idea is to keep code blocks out of each others reach through ISI as long as possible. This is the idea behind most page fill strategies described in this chapter. The ISI is calculated point spread function, which is described in the next section.

8.3.1 Point Spread Function Matrix

The inherent property of the dynamic encoding is that every new code block depends on the code blocks encoded previously, the blocks that are in the immediate vicinity. This is because of the inter-symbol interference (ISI). The ISI is calculated by applying the Point Spread Function (PSF) to each code block and surrounding information light sources. Ideally PSF does not have a boundary around the light source to which it is applied, i.e. no matter how far any cell is located, there is some light energy contributed to this location from the light source. In reality, the contributed light decreases significantly by increasing the distance from the source and at a certain distance it can be considered negligible. Therefore, the PSF function is pre-computed and stored in a matrix with a limited size to speed up the ISI estimation. The size of the matrix determines the accuracy of the ISI estimate because it does not take in account ISI contributed from the light sources that are beyond its size. If we assume that integrating PSF over infinite area the total light energy is one unit, i.e.,

$$\iint_{\infty} PSF() = 1$$

Then the unaccounted light energy for PSF matrix of size $A \times A$ is equal to

$$Unaccounted_ISI = 1 - \iint_{A \times A} PSF()$$

The matrix size is chosen to be 11x11 cells with the light source at its center, according the analysis shown earlier in Figure 13 and section 6.2. Thus, if the PSF matrix size is 5 cells in each direction from the light source, then it implies assumption that the light contributed to the pixels that are outside 5 cell range is negligible, where the light source is assumed to be in one of the corners of the matrix (Figure 32). Consequently, due to the assumption that the

unaccounted ISI can be negligible, no code block will be affected by another code block if the distance between them is more than the size of the PSF matrix.

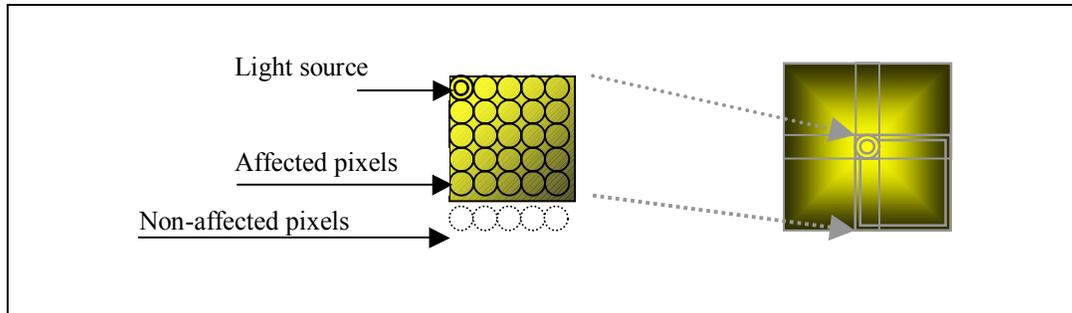


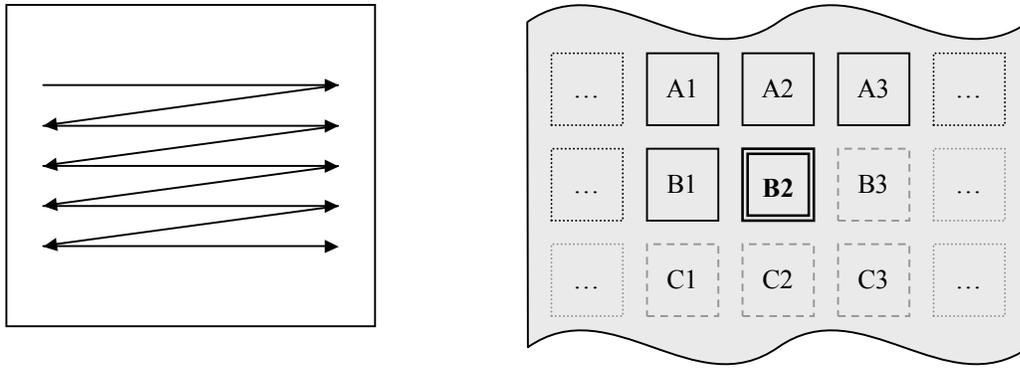
Figure 32: Pixel crosstalk and ISI in optical memory media.

The next section presents the simplest data page fill order: the linear fill. Other fill order strategies are presented consequently, and finally the simulation results showing performance of the described fill strategies are compared.

8.3.2 Linear Fill

A straightforward way to fill out a page is in the left-to-right, top-down fashion. This way the location for the first code block is on the left top corner of the page. The encoding algorithm assumes that all the rest of the code block locations have no information written to them, therefore, are completely dark, and provide no light energy that would create ISI. This assumption is made in order to minimize ISI to the areas of the memory data page that have not been written yet, and thus allow more freedom for choice of code words in these areas. As the encoding algorithm proceeds by encoding code blocks one by one, it always assumes that the information above and in the same row to the left has been encoded, and will take that information in account when constructing the valid code set.

All of the simulation experiments were conducted assuming the sequential fill of the page method (Figure 33), except for those testing other fill strategies, as presented in the section 8.3.4.



(a) Fill order in the page

(b) Code block order in the page

Figure 33: Sequential fill of the data page.

8.3.3 Uniform Fill

An alternative way to fill out a page is in a pseudo-random fashion, filling up the whole memory page uniformly as opposed to the linear fashion. Several strategies were devised that tend to fill the memory page uniformly. Consider dividing a page in regions R_x in such a way that the PSF matrix completely fits in each region. This way by writing a new CB in each region, it will not affect a CB at the same relative position in any other region, because the distance between the code blocks is longer than the size of the PSF matrix (Figure 34). Eventually as the regions are filled with the code blocks, they are going to start influence each other through ISI, but it will happen only in the late stages of the encoding. The decoder must follow the same page fill order; therefore, the fill order must be implicit and match the encoding order. Therefore, we propose to use uniform fill order, generated by a predetermined location generator function and making sure that every position on the page is visited only once and the whole page is covered.

8.3.3.1 Z-Fill and X-Fill Strategies

For a uniform fill the memory page is subdivided in equal size regions, where each region can contain several code blocks. The regions are sized and shaped in such a way that a code block in the position A_x of a region R_x will not interfere with A_x position in any other region. When all regions have one CB encoded, the fill algorithm moves on to the next stage of the encoding and encodes another CB at the location B_x for each region. ISI is possible due to the close proximity of the A_x code blocks to B_x CBs, therefore the encoding has to consider ISI generated and

received by neighbors of the Bx blocks. When all the locations for Bx are filled, the CB placement algorithm moves on to the next location Cx in each region, and finally to Dx.

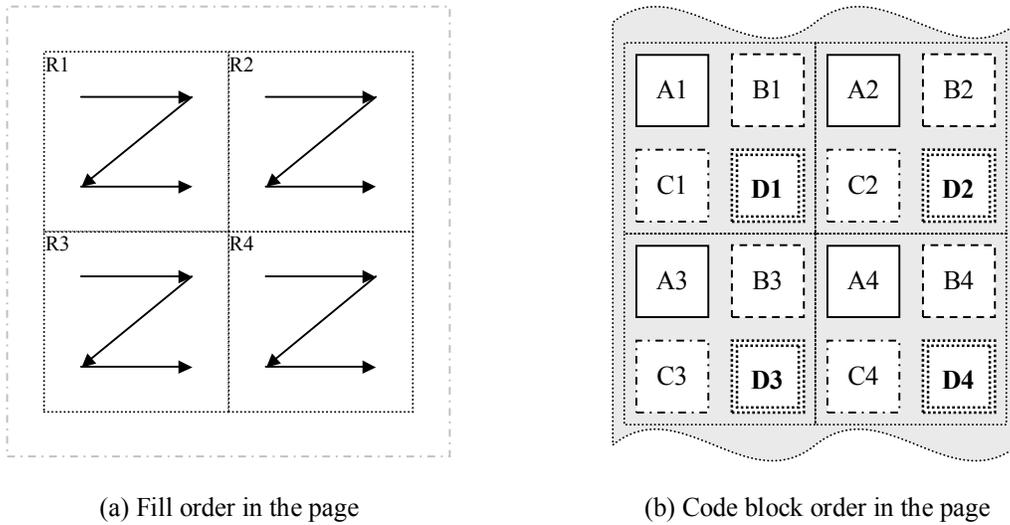


Figure 34: Z-Fill pattern fill of a data page

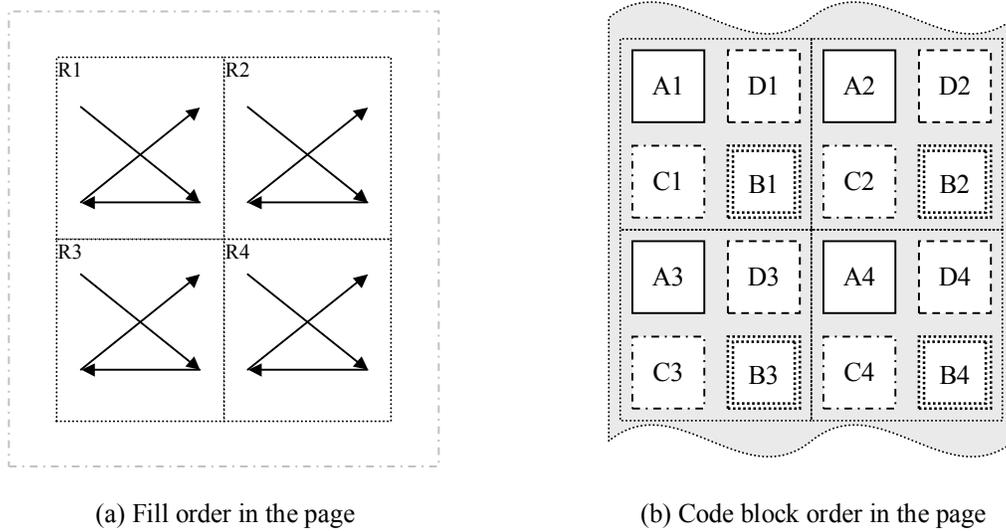


Figure 35: X-Fill pattern fill of a data page

The fill pattern described above is called Z-Fill due to the shape of the fill pattern (Figure 34). An alternative uniform fill order is so called X-Fill, where the regions are filled with CBs in the X-shaped order as shown in Figure 35. More similar fill strategies are possible, such as U-fill.

8.3.3.2 Pseudo-Random Fill Strategy

Another alternative is a pseudo-random fill (R-Fill), where the next CB position generation algorithm is pseudo-random for both encoding and decoding in order to preserve the decoding order. Note, that a truly random position generator cannot be used, unless the random position sequence is saved somewhere on the media and retrieved before decoding. This can be done, but will result in undesirable space overhead, therefore it is better to use implicit code block location ordering. Alternatively a seed for the random generator could be saved, and the same seed used to start a random location sequence upon detection.

There are advantages to the uniform filling of the page. Note, that all the code blocks A_x can be processed independently of each other since there is no cross influence due to the large distance between them. This way all of these code blocks can be processed in parallel on as much computational hardware as there is available. The same is true for the groups of code blocks B_x , C_x and D_x .

In the case when the chosen CB size is much smaller than the PSF matrix size, more than 4 code blocks may fit in each rectangle. This leads to fill geometries of three by three code blocks and higher. Nevertheless, the idea of the page fill order remains the same: fill out the page uniformly by concurrently filling each rectangle. This paradigm allows for more code freedom and data density at the early stages of page fill and more parallelism for encoding and decoding operations.

8.3.4 Page Fill Simulation Results

Although there might be strategic advantages to choosing Z-Fill or X-Fill over the Linear-Fill, the experiments showed that in terms of average code density there was no significant difference. We performed experiments with the same optical memory configuration and source data sets to compare the page fill strategies for different file types. Memory data page size was 512x512 bits, code block size was 3x3 bits, and the encoding algorithm was *Second Chance Re-coding*. The results shown in Table 13 and Figure 36 reveal that the Linear-Fill performs the best in terms of average code density, and the Random-Fill performs the worst. We suspect that the poor Random-Fill performance is due to the fact that although in the beginning the memory page is empty and new code blocks are not

likely to be in close proximity to each other, and thus no destructive ISI is created, as the page fills up the new code block locations tend to hit locations with more neighbors in all directions, and therefore more ISI restrictions, thus limiting the code for choices of valid code words. Apparently, filling out a memory page in orderly fashion, be it linear or uniform fill order, produces less restricted locations in terms of ISI, and allows for better code densities.

Table 13: Fill order performance comparison

Page fill order	Code Density			
	Min	Max	Average	Standard deviation
Linear	71.74	77.78	75.88	0.89
X-Fill	39.44	77.78	75.82	1.83
Z-Fill	48.63	77.78	75.27	2.56
Random	45.38	77.78	73.93	1.96

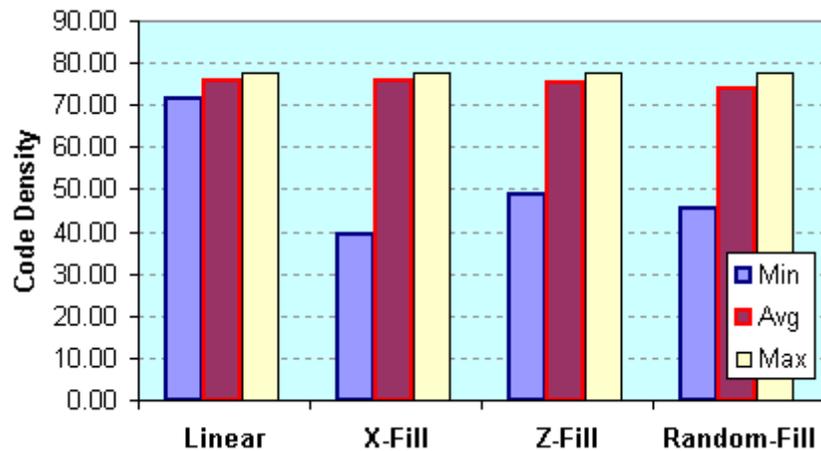


Figure 36: Fill order performance comparison

Since the average code density for X-Fill is very close to the best average code density produced by Linear-Fill algorithm, the X-Fill strategy is favorable for implementations on highly parallel encoding and decoding architectures. Thus the encoding and decoding time and consequently the transfer rate are improved due to uniform fill properties without practically any loss of code density.

It is interesting to note that all uniform fill strategies have rather low minimal code density. However, there are very few files that have code densities below 70%, otherwise the average code density would be much lower, especially because it is so close to the maximal code density (within 3%) and the standard deviation is also within 3%. Figure 37 shows code density distribution for different page fill orders plotted for the *Work-Set* files. It shows that less than 8% of the files have code density below 71% for any page fill strategy. This means that most of the files result in good code density when encoded to a memory page using any of the page fill strategies.

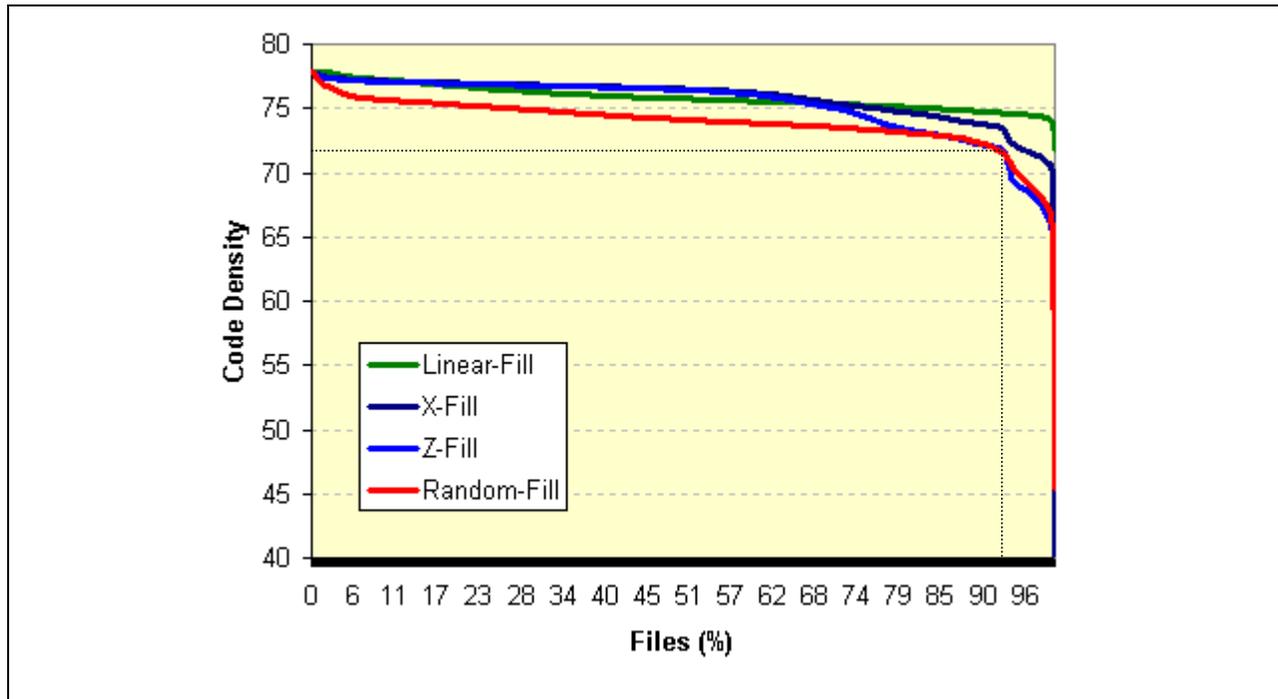


Figure 37: Page fill order code density distribution over *Work-Set* files

8.4 SECOND CHANCE RE-CODING PERFORMANCE TESTS

The *Second Chance Re-coding* algorithm was used to test the performance of the quasi-dynamic encoding approach. It was tested in the presence of three types of noise: inter-symbol interference (ISI), jitter/misalignment, and media non-uniformity. The simulation of the Re-coding algorithm used the *Work-Set* source data set as described in Experiment Design (chapter 6.3) above, containing various types of files, such as binary executables, text and PDF files, image and audio files, and compressed archive files.

The simulation experiments encoded every 7-bit source word (SW) using 9-bit code word (CW) arranged in 3x3 pixel array code blocks (CB). Every source word required at least two code words for the quasi-dynamic Re-coding algorithm. Some of the available code words created too much ISI to be used at all. Therefore, the number of unique code words exceeded the number of unique source words more than twice, which resulted in at least 2-bit difference between source word and code word sizes. Increasing the difference decreased the code rate. Increasing the code word size escalated the complexity of the perfect matching problem, which needed to be solved once for each memory system. Therefore 7/9 was chosen as the optimal code rate for the simulation experiments. This code rate has potential code density of 77.8%.

Based on the dynamic encoding simulation results the threshold and noise margin parameters were chosen to be 0.45 and 3% of the maximal one pixel light intensity (Figure 20 and Figure 21). These parameters were observed to provide the densest code when using full enumeration encoding with the two-photon memory model. Therefore, these parameters were good candidates to give good performance for the Re-coding algorithm.

Three sets of experiments were performed to test the performance of the Re-coding algorithm. In the first set only ISI was responsible for the noise. The other two simulation setups considered jitter and material non-uniformity noise sources in addition to ISI. The following sections discuss the results of the simulations.

8.5 RE-CODING PERFORMANCE IN THE PRESENCE OF ISI

The initial evaluation of the Re-coding algorithm considered only ISI when performing encoding and decoding. The encoding was performed on various types of binary, text, and multi media files contained in the *Work-Set* as described in section 6.3.2.2. The results were evaluated with respect to the achieved code density. The minimal code density ranged between 66%, and 77%. The average code density of 76% was very close to the maximal. These results were superior to static encoding, which rarely exceeded 50%.

The tests confirmed that there were no bit errors. As shown in the histogram in Figure 38, the blue line representing zero-bits and the green line representing one-bits do not cross. There is a clearly marked threshold at 45% of light intensity on the X-axis and clearly defined noise margins on both sides of the threshold were strictly enforced. This was due to the nature of the code design, which encoded data to patterns that avoided creating destructive ISI. The following sections of this dissertation analyze the Re-coding algorithm in the presence of noise sources in addition to ISI.

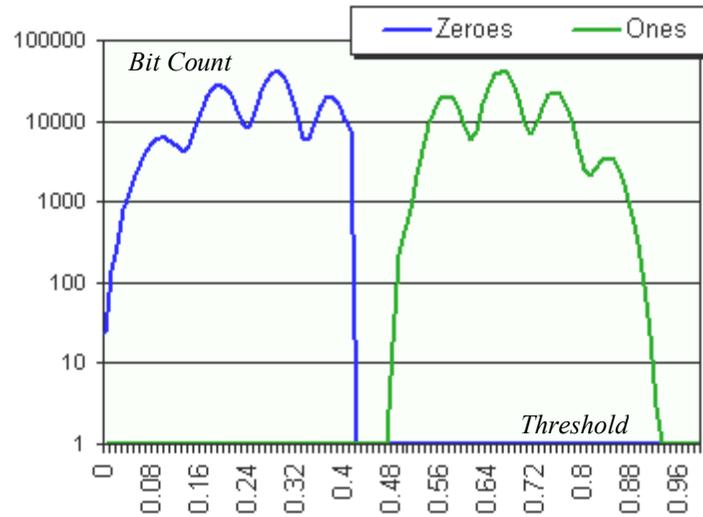


Figure 38: Quasi dynamic encoding algorithm verification test results

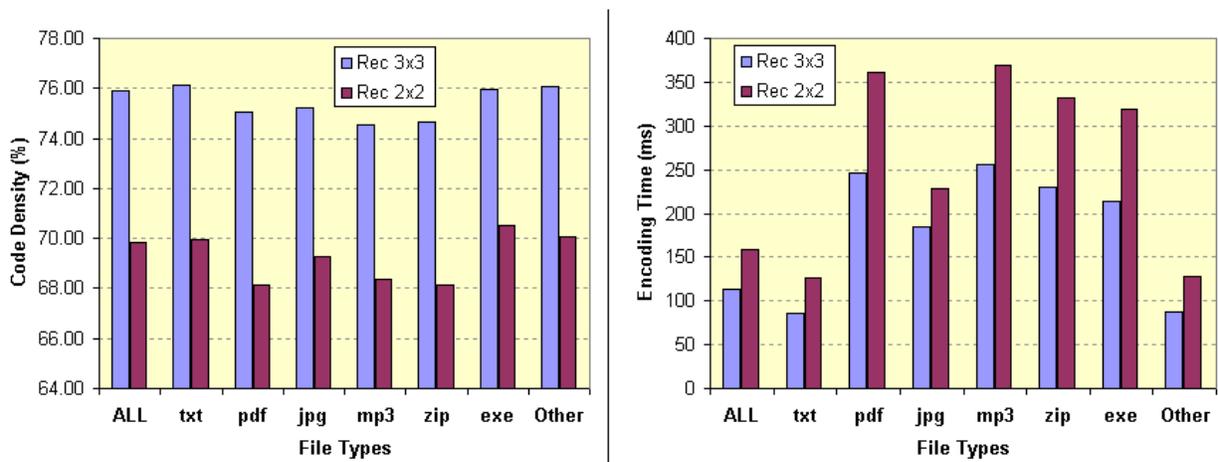


Figure 39: Quasi-dynamic encoding code density and time performance

Figure 39 shows the results of experiments running quasi-dynamic algorithm on different types of data files. We plotted average code density and average encoding time per 512x512 memory page. The results are also given in

numbers in Table 14. Two versions of *Second Chance Re-coding* algorithm were selected for these experiments with respect to the code block size: 2x2 and 3x3 code blocks. As the table and histograms show, the average code density for 3x3 encoding was 75% over all files, while it was only 69% for 2x2 version. The maximal code density achieved by 3x3 version of the algorithm was 77%, while maximal code density for 2x2 version reached only 75%. It was also observed that the execution time for 2x2 code block size experiments was higher than 3x3 block size experiments. This is due to the fact that although it takes slightly less time to encode 2x2 code block than 3x3 block, there are 9/4 more 2x2 code blocks on a data page than 3x3 code blocks. Therefore more code block encoding operations per data page need to be performed, which leads to higher execution time.

Table 14: Second Chance Recoding simulation results

File types	Average Code Density		Average Encoding Simulation Time	
	3x3 CB size	2x2 CB size	3x3 CB size	2x2 CB size
<i>All files</i>	75.88	69.82	113.49	159.07
txt	76.12	69.95	86.04	126.51
pdf	75.07	68.15	246.75	360.68
jpg	75.20	69.27	184.36	228.14
mp3	74.56	68.35	256.13	369.92
zip	74.66	68.14	230.44	331.32
exe	75.98	70.50	214.00	319.51
<i>Other types</i>	76.08	70.05	87.94	127.31
MIN value	71.74	60.22	0	0
MAX value	77.78	75.00	515	454
Standard deviation	0.89	1.76	107.6	149.7

The histograms also show encoding of different file types. It was observed that both 2x2 and 3x3 versions of encoding follow the same trend with respect to the file types, and 3x3 encoding always perform better in both code

density and encoding time respects. The encoding time was obtained by timing the memory simulator; therefore, it should be regarded as relative measure only for comparison purposes between the versions of algorithms.

The following sections estimate the tolerance of quasi-dynamic encoding to such noise sources as jitter and material non-uniformities.

8.6 RE-CODING PERFORMANCE IN THE PRESENCE OF JITTER

Typically, jitter and misalignment corrupt data in optical mass storage systems. Therefore, a series of experiments were performed to test the Re-coding algorithm in the presence of these noise sources.

We modeled jitter by shifting every bit location with respect to a readout detector on the optical data page. The maximal shift was expressed in percentage of pitch between two neighboring pixels. For example, zero percent jitter was assumed when each pixel was centered with respect to a photo-detector. Five percent jitter on the X-axis was assumed when a projection of a pixel was shifted along the X-axis five percent of the pitch between the photo-detectors (Figure 40).

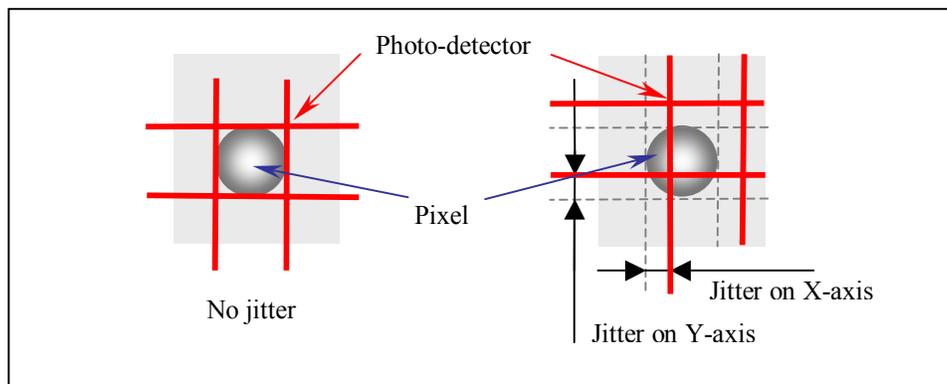


Figure 40: Jitter measurement

We carried out a series of experiments gradually increasing the *jitter limit* from 0% to 30% by 0.1%. For each jitter limit three types of experiments were performed:

- “Random” jitter - Uniform jitter amplitude distribution: the jitter was uniformly distributed between 0 and the *jitter limit* over all pixels on the optical data page. Random jitter was applied to both axes.
- “Constant” jitter - Shift of the whole data page: this approach represents misalignment. Every pixel was shifted diagonally by a constant *jitter limit* over both X and Y axes.
- “Radial” jitter - Constant amplitude jitter with uniformly distributed radial direction for pixel displacement with respect to the corresponding photo detector. The distance between a pixel and the corresponding photo detector center is kept constant, however the angle of the direction is randomly changed for every pixel.

The simulation results in Figure 41a show that as the jitter limit increases from 0% to 30% all types of jitter follow the same trend line and start introducing measured bit error rates of 10^{-6} at 5.6%. Figure 41b shows the close-up at the jitter interval between 5% and 8%, where the Re-coding algorithm fails to correct all bit errors. Another interesting fact is that although the “Random” jitter starts introducing bit errors earlier than the “Radial” and “Constant” jitter experiments, the blue line very soon crosses the other two, i.e. the “Radial” and “Constant” jitters become more destructive with respect to the bit errors. Finally, Figure 41c shows that both “Constant” and “Radial” jitters have very similar performance, while the “Random” jitter generates fewer errors.

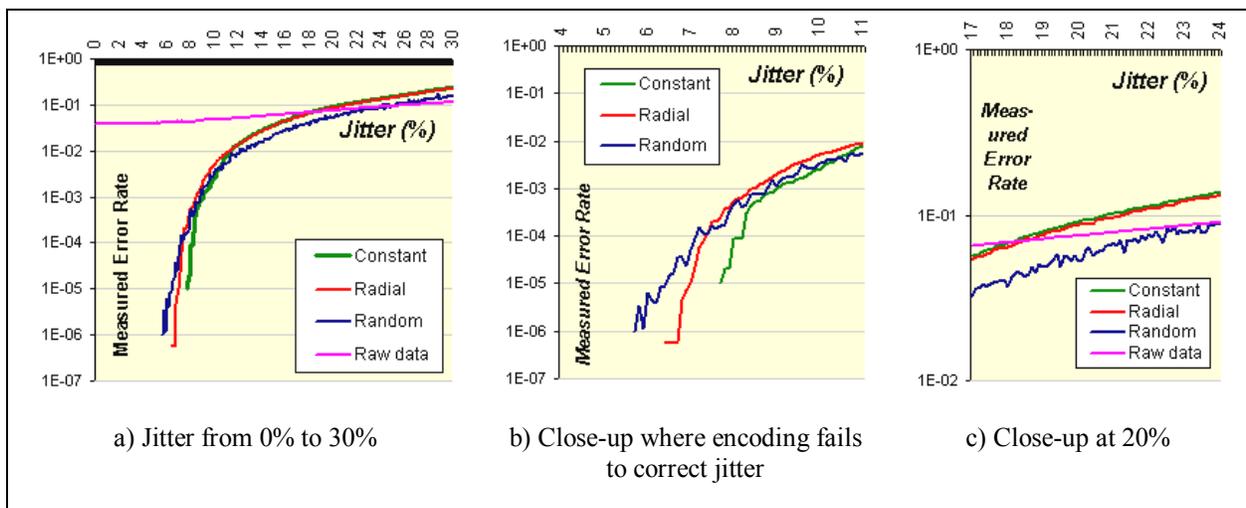


Figure 41: Re-coding performance in the presence of jitter.

A line labeled “Raw data” represents data stored to the media without encoding. This allows comparison between encoded and un-encoded data and illustrates the increased error rates due to the lack of encoding. However at about 18% of jitter can be observed that it does not matter any more whether the data was encoded or not, because in both cases the error rate is 0.1.

8.7 RE-CODING PERFORMANCE IN THE PRESENCE OF MEDIA NON-UNIFORMITIES

Ideally, storage systems and storage media would be uniform and noise-free. Unfortunately, there are factors that change readout quality across the media. In the case of optical storage, the affected properties are transparency, reflection, refraction, or dust residing on the media. As a result, the image read from 2D page-oriented storage media may become darker and brighter, either across the whole page or for some areas of the image. The granularity of these defects may range between individual pixels representing data bits and the whole data page. Modulation encoding schemes must protect from such media non-uniformities. Therefore, the performance of the Re-coding algorithm was tested in the presence of noise sources simulating media and readout system non-uniformities in addition to ISI.

The experiments for testing the Re-coding algorithm in the presence of media non-uniformities were conducted in the following fashion: first, the simulator encoded data with Re-coding algorithm and generated a data page of optical intensities. Then it generated a noise matrix that stored noise amplitude for each pixel. Finally, the simulator added the noise matrix to the optical data page and decoded the resulting page. The decoded data was compared to the original data, and a bit error rate was measured.

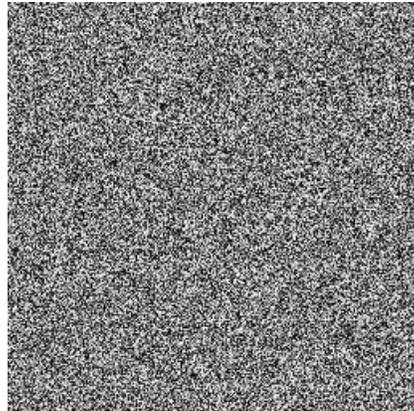
The simulator performed series of experiments by increasing maximal noise limit from 0% to 30% of the maximal light intensity. For each noise limit four experiments were performed differing in types of noise applied. The four noise matrices were generated as follows:

- Constant additive noise, modeled by a constant matrix set to the maximal noise limit (Figure 42a): this represents global changes in the entire optical memory system, which may require shifting the bit detection threshold.

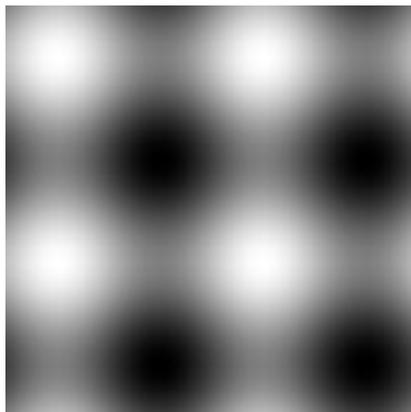
- Additive random noise with intensity uniformly distributed between zero and the noise limit (Figure 42b). This represents high frequency uniform noise across the optical memory page.
- Low frequency uniform noise: a noise was added to pixels depending on the location of the pixel. The noise matrix was generated using a two-dimensional *sin* function with X and Y coordinates as parameters and the noise limit as the maximal possible absolute value of the function. The period of the sin function was chosen to be half of the data page width, thus, simulating a low frequency uniform noise, as illustrated in Figure 42c.
- Non-uniform low frequency noise: this noise matrix was generated from a 16x16 matrix of random intensities by interpolating it to the data page sized noise matrix with B-Spline filter. The resulting matrix was normalized to the noise limit. The result was a matrix with random low frequency noise, as shown in Figure 42d. This is the most likely scenario for low-frequency material non-uniformities.



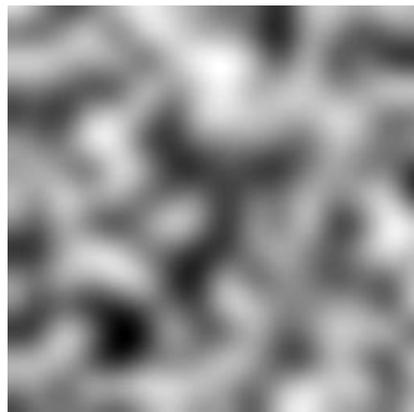
a) Constant noise



b) Random uniform high frequency noise



c) Uniform low frequency noise



d) Non-uniformly distributed low frequency noise

Figure 42: Noise matrix examples

The simulation results were plotted in the Figure 43a. The different noise sources were labeled as follows: “Const” for constant noise, “AWGN” for uniform high frequency noise, “2D-Sin” for low frequency noise defined using sin function, and “Bilinear” for low frequency non-uniform noise. It was observed in the close-up Figure 43b that the Re-coding algorithm could correct data as the noise increased to 7.2% of the maximal light intensity. Increasing noise resulted in measured bit error rate of 10^{-5} and higher. For higher noise levels error correction schemes, such as Reed-Solomon ECC should provide data integrity.

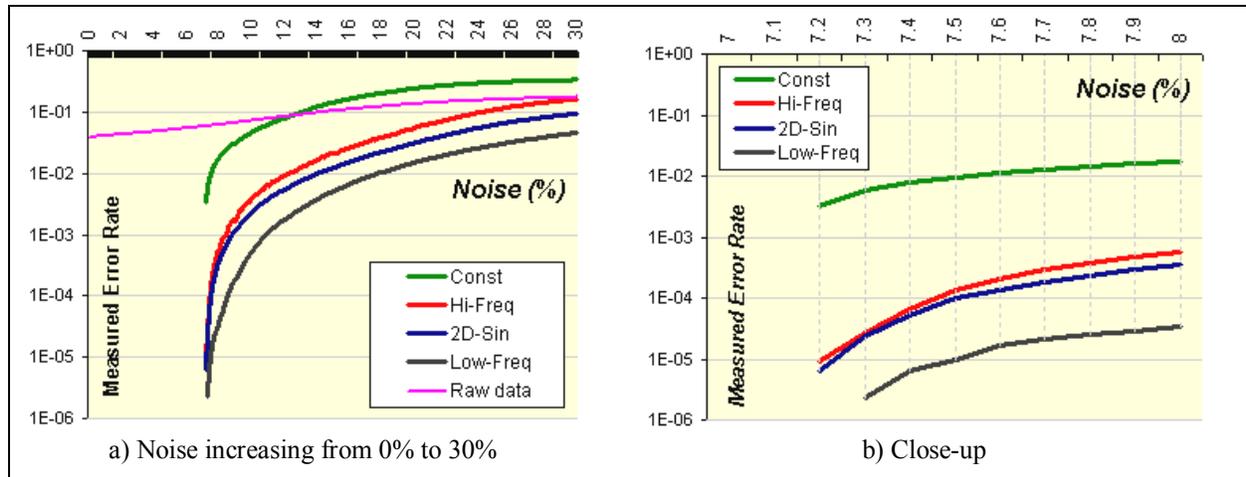


Figure 43: Re-coding performance in the presence of noise.

8.7.1 Adapting to Low Frequency Noise

Constant and low frequency additive noise is corrected by adjusting the readout intensity threshold prior to data decoding. The threshold-adjustment algorithm takes advantage of a small data packet saved with each optical page or page area, depending on the correction granularity. The data packet contains the count of one-bits recorded on the area at the time of encoding. The data packet is encrypted with a strong ECC algorithm ensuring correct decoding in the presence of noise. After the data page is decoded, the actual decoded number of one-bits is compared with the stored number. If they match, the decoding of the page is assumed to be successful. Otherwise the threshold is adjusted according the difference between stored and sampled number of one-bits. If the stored number is higher then the threshold is lowered, otherwise it is raised.

The threshold adjustment method involves data redundancy and thus an overhead. Assuming 7/9 Re-coding approach, one CB of size 3x3 bits can store 7 bits. The maximum number that can be encoded in 7 bits is $2^7 = 128$,

thus this label can count one-bits over an area of 128 bits. Assuming full redundancy, i.e. encoding this number in two code blocks, the overhead will be 18 pixels for every 128 pixels. Therefore, the total data density for 7/9 Re-coding with threshold adjustment is $((128-18)*7/9) / 128 = 66\%$, which is 11% less than Re-coding without the threshold adjustment. It is still better than static encoding with typical data density of 50%. Designers of the actual memory storage system can decide if the additional 11% overhead is worth implementing the threshold adjustment algorithm.

An alternative approach is for the threshold adjustment algorithm is to read a pre-recorded calibration data page, sample it and compare the results with predefined patterns. Then the algorithm can decide about the threshold depending on the matching calibration patterns.

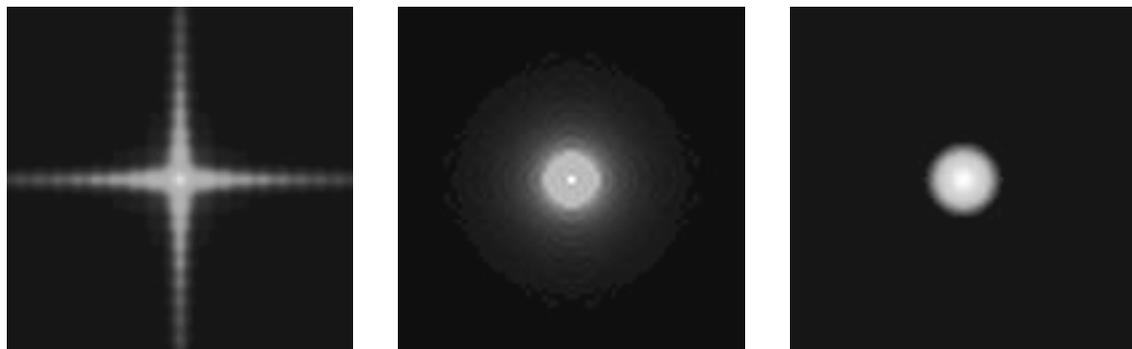
Only the Re-coding algorithm was tested against jitter and material non-uniformities. Dynamic encoding was tested only against ISI and found to have too high time complexity to be practical. Therefore testing dynamic encoding against jitter and non-uniformities noise was not performed and is left as one of the directions for future research.

9.0 ANALYSIS OF THE TWO-PHOTON MEMORY MODEL

The quasi-dynamic re-coding approach was tested using point-spread functions (PSF) generated by the two-photon memory system developed at Call-Recall, Inc. The PSF data kindly provided by Call-Recall Inc. represents a real system test bed for dynamic encoding. In this chapter, we describe the two-photon memory measurement data and methods used to extract the PSF functions. Finally, we perform dynamic encoding using the extracted PSF data and estimate the areal data density limits that the Re-coding algorithm is able to provide.

9.1 VARIATIONS OF POINT SPREAD FUNCTIONS

Most of the experiments in this research were performed using a *diffraction-limited square-aperture* Point Spread Function (PSF) formula to model projection of optical media data page to photo-detectors, as described in the chapter about our simulation model. An alternative approach is to use a different PSF modeling *diffraction-limited round-aperture*. Yet another PSF version is using Gaussian beam for light source simulation in the presence of large amounts of small *independent aberrations*. Figure 44 illustrates all three PSF kinds as images defining the ISI pattern of the projected light. The choice of a PSF depends on the assumptions about the theoretical model of the optical storage readout system. However, to simulate a realistic PSF we devised another approach: PSF was extracted from an experimental two-photon memory system, which was built and tested in Call-Recall Inc. laboratory.



a) Square aperture PSF

b) Round aperture PSF

c) Gaussian Beam

Figure 44: Various Point Spread Functions

The two-photon memory PSF was acquired by recording an image of a one-bit light source being received onto a photo-detector plane. The ideally aligned PSF was recorded with the light source on-axis with the optical system center. Alternative PSF images were also recorded representing misalignment ranging between 100 and 250 microns from the axis (Figure 45). The acquired images were processed by our conversion software, which sampled, integrated, and transferred the light intensities to a matrix representing PSF. One of the software parameters defined the size of each PSF matrix cell with respect to the recorded image. Thus, it was possible to generate PSF matrices that represent different pixel spacing, and thus different areal bit density.

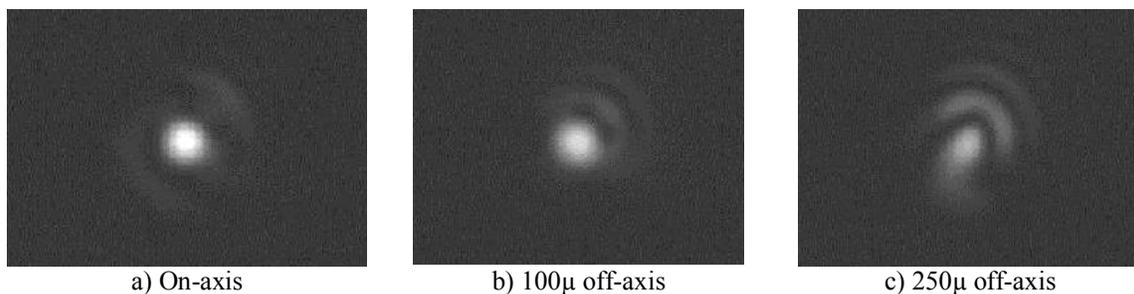


Figure 45: PSF recorded from a 2-photon memory system (provided by Call-Recall Inc., CA).

The calculated PSF is an 11 by 11 cell matrix of values representing light intensity received at every cell location from an assumed light source at the matrix center. Thus, the values in each cell represent the normalized light intensity received by a photo detector at that location. Each PSF matrix cell represents a “one” bit written onto an optical medium. Thus, the PSF matrix represents a contribution of a bit at the matrix center to any of the bits within distance of 5 cells.

The PSF matrix was used to evaluate the inter symbol interference (ISI) by convoluting the matrix with a binary data page. The resulting page of light intensities represented the analogue data page with ISI taken into consideration, modeling the data image read by the photo detector array. The image is sampled by applying a threshold and, then, decoded.

9.2 DYNAMIC ENCODING FOR TWO-PHOTON MEMORY SYSTEM

The goal of this study was to evaluate dynamic modulation encoding on a two-photon memory system developed at Call Recall Inc, California. The memory system stores information in data pages, where every bit is represented as a fluorescent light source $0.7\mu - 2\mu$ in diameter, with 5μ distance between the data pages [43]. In our study, dynamic modulation encoding was applied to achieve higher data density than with static encoding and improve data reliability. The tradeoffs between code density, detection threshold, and pitch between the bits were evaluated.

In the experiments described below we estimate the effective areal density after applying the dynamic encoding. The effective areal density is the number of bits effectively encoded to an area of specific size of the storage media. Naturally, effective areal density is lower than areal density for raw data due to the modulation encoding overhead. However, unencoded raw data typically has bit error rates as high as 10^{-4} [14, 48]. Therefore, we proposed using the dynamic encoding, which provides high code density.

The dynamic encoding assumes data readout detection by a threshold, i.e. every bit read at light intensity below the threshold is considered to have value '0', and every above is '1'. Code density can be improved by fine-tuning the threshold. Another parameter of the dynamic encoding is the point spread function (PSF), which describes light distribution from a bit source. Usually this distribution pattern is wide enough to create inter-symbol interference (ISI) with the neighboring bits. Dynamic encoding pre-evaluates ISI and arranges bit patterns for the codewords so that ISI is minimized.

The point-spread function used in this research was extracted from the images provided by Call Recall Inc. Each image represented a bitmap of light intensities (Figure 46) received at different locations on a photo-

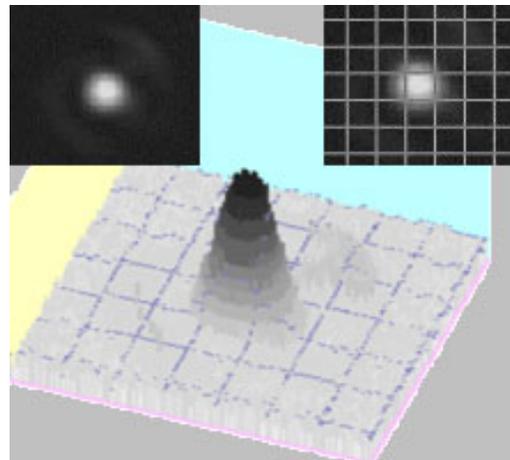


Figure 46: PSF 2D and 3D plots

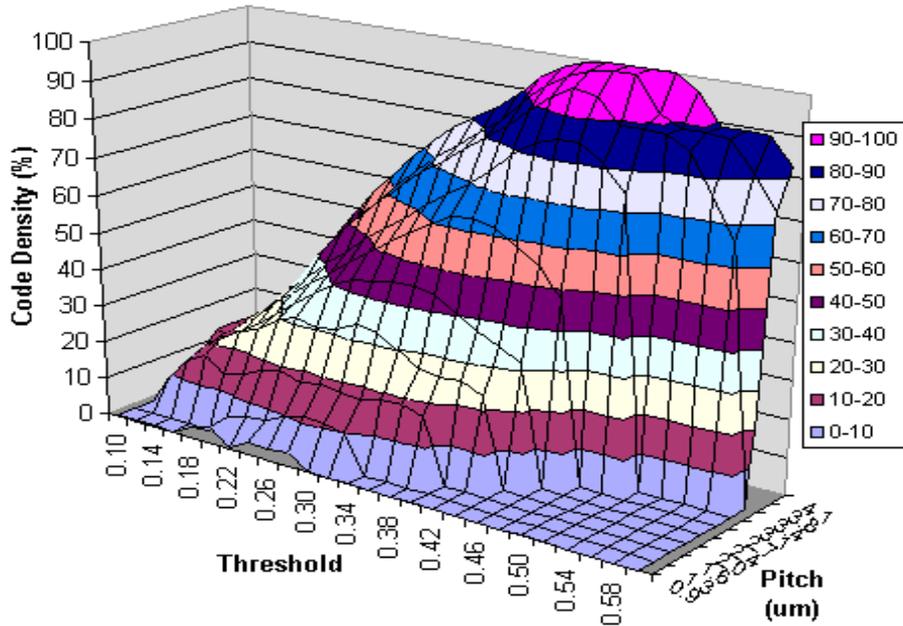


Figure 47: Code density plot

detector plane. In order to evaluate dynamic encoding the image was superimposed with a grid that defined a pitch between bit locations on the memory page. Then the light intensity values were integrated for each grid cell and normalized over the whole grid. The result was a PSF matrix, which represented ISI intensity values for neighboring bit locations.

Several PSF matrices were created for different grid pitch values ranging between 0.9μ and 4.1μ modeling different raw data densities. Then encoding simulations were performed for every PSF matrix and a range of thresholds to determine the code density. The results constitute the surface plot in Figure 47, in which pitch versus threshold are plotted against code density. The top area on the surface indicates the values of the threshold and pitch at which the bits were sufficiently spaced apart so that ISI became negligible, i.e. there were no cross talk, which resulted to 100% code density. However by spacing bits apart we enlarge the area required for the bits to be stored on the media; therefore, it is not sufficient to consider only code density as a measure of a successful modulation encoding. Instead, the effective areal density must be considered, which is obtained by dividing the code density by the area per one bit, i.e., pitch squared. The results of this operation are shown on a surface plot on Figure 48. Once the areal density values, defined as kilobytes per square millimeter were computed and plotted on the vertical axis, it became apparent that the “sweet” spot with the highest effective areal density is located at a different pitch and thresholds.

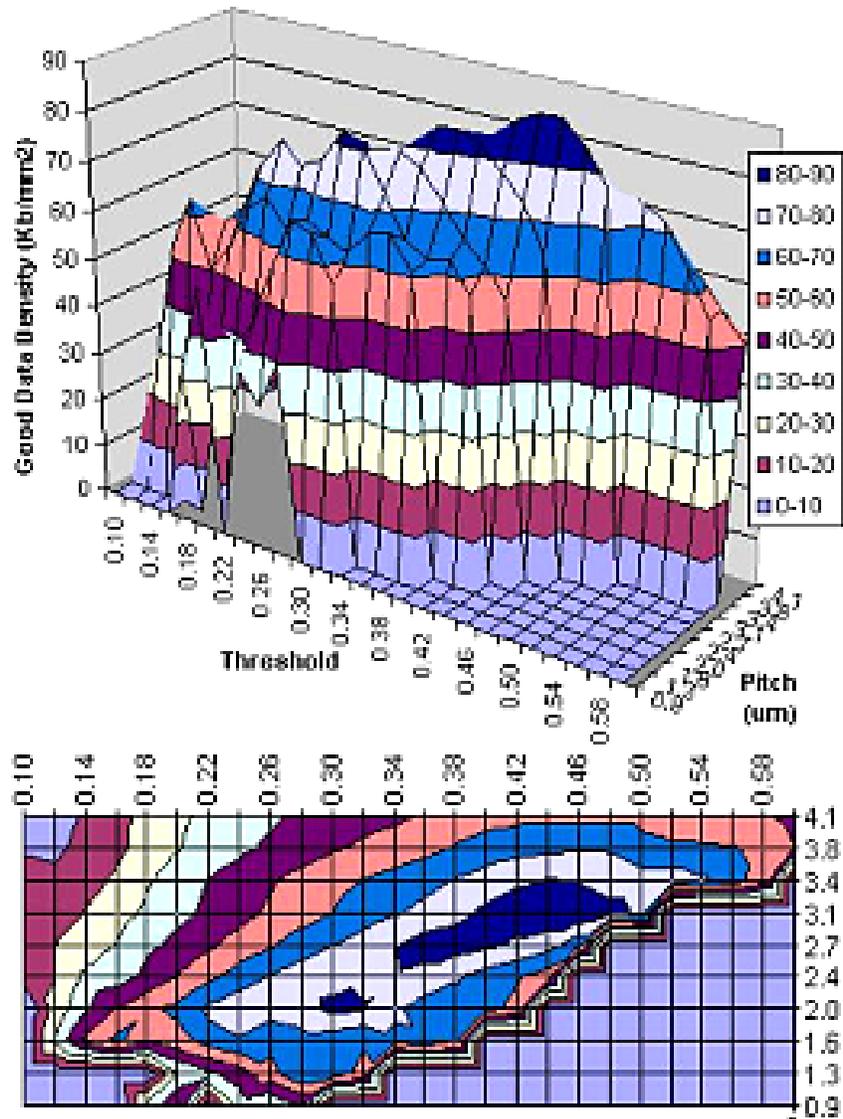


Figure 48: Effective Areal density plot for $2\mu^2$ bit size.

The best effective areal density is achieved for pitch of 3.1μ and thresholds between 42%-46% of the maximum light intensity.

The results of the dynamic encoding simulation on Figure 48 show that for a $2\mu^2$ bit diameter and 3.1μ pitch the 2-photon memory system can achieve the effective areal density of 89 Kbit/mm^2 (Table 15). Reducing the bit size to $1\mu^2$ and assuming that the bit has the same profile defined by the PSF increases the data density to 356 Kbit/mm^2 (224 Mbit/inch^2) at the best pitch of 1.5μ . For the smallest bit size $0.7\mu^2$, as reported by Call Recall [43], and 1.1μ

pitch dynamic encoding enables data densities as high as 726Kbit/mm² (457Mbit/inch²). Finally, scaling the bit size to 0.3μ² as projected pixel sizes in near future optical memories by [43] would lead to areal density of 3.9 Mbits/mm² (2.4 Gbits/inch²) at the best pitch of 0.5μ while tolerating ISI. Decreasing the pitch even further would be impractical due to destructive ISI. Assuming 100 data pages written at different depths as reported by [43], the cumulative areal density would reach 240 Gbits/inch² per media access area which surpasses the current target of 100 Gbit/inch² marked by commercial magnetic storage drives [11]. In fact, to be more precise, in the latter case the data density should be expressed in effective volumetric density when writing to several levels of depth in volume. Since the projected pixel sizes are to approach 0.3 x 0.3 x 2 microns, this allows storing pages as close as 2 microns [43]. Then the theoretical effective volumetric density becomes $3.9 * 1000 / 2 = 1.95 \text{ Gbits/mm}^3$ (31Tbits/inch³). However, if we limit the levels of pages to 100 and assume 5mm thickness of the two-photon media disk, then the effective volumetric density is $3.9 * 100 / 5 = 78 \text{ Mbits/mm}^3$ (1.2Tbits/inch³).

Table 15: Areal density for two-photon memory system

Bit (pixel) diameter (μ)	Pitch (μ)	Areal density (Kbits/mm ²)	Areal density (Mbits/inch ²)
2	3.1	89	57
1	1.5	356	224
0.7	1.1	726	457
0.3	0.5	3900	2400

Realistically, some crosstalk is possible between the stored pages leading to inter-page interference (IPI). This should also be taken in account when talking about volumetric data storage at high densities. Therefore, adapting dynamic and quasi-dynamic encodings to 3D storage taking IPI into account is one of the future directions of this research.

10.0 SUMMARY, CONCLUSIONS, AND FUTURE DIRECTIONS

10.1 SUMMARY AND CONCLUSIONS

This dissertation presents original data modulation-encoding techniques for page-oriented data storage (PODS). These modulation schemes use a dynamic, adaptive approach to prevent destructive inter-symbol interference (ISI). The proposed dynamic and quasi-dynamic modulation algorithms achieve improved code density over traditional static modulation schemes by analyzing surrounding information on a data page for every code block location prior to encoding.

Several dynamic algorithms are proposed and analyzed using the two-photon optical memory system simulation setup described in this dissertation. The proposed algorithms differ with respect to the manner in which they trade code density with time complexity, as shown in Table 1 and the chart in Figure 49. The proposed algorithms are compared against static modulation schemes for PODS, which approach only 50% code density [6,7]. The dynamic full enumeration algorithm Dyn3x3var, requiring exponential time complexity in terms of code block size, achieves the highest average code density, 83%. However, due to the variable code rate approach there is a danger of error propagation while decoding a memory page. Therefore either an error correction code preventing such error propagation should be applied on top of the modulation encoding, or a constant code rate version of the dynamic encoding should be used instead of the variable code rate. Unfortunately, the constant code rate algorithms yield lower code densities, only 75% on average as shown in the table (algorithm labeled “Dyn3x3 7/9”). Finally, the dynamic full enumeration encoding approach has exponential complexity leading to slower encoding and decoding times. Therefore, in order to reduce time complexity, several linear time algorithms were designed.

Three linear time complexity strategies were explored. The first, called *Either-Neither Algorithm* evaluated every bit location of a code block in order to establish whether it could be used for encoding both ‘1’ and ‘0’. Simulations showed a rather low increase of code density over the static algorithms, which typically have 50% code density.

Table 16: Dynamic encoding algorithm comparison: Time complexity vs. code density.

Encoding	Time Complexity	Code Density		
		Min	Avg	Max
Static 7/16	Constant	< 50	< 50	< 50
Dyn2x2var	Exponential	50	70	74
Dyn3x3var	Exponential	74	83	87
Dyn3x3 7/9	Exponential	72	75	77
EitherNeither	Linear	43	67	90
Quasi-Dyn. 7/9	Constant	66	76	77

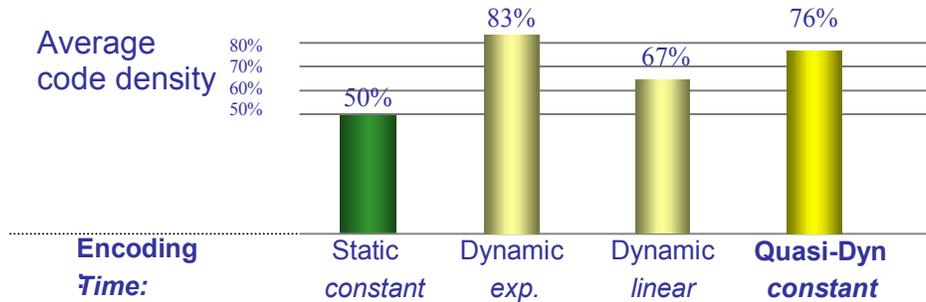


Figure 49: Modulation encoding techniques. Comparison chart. Code density.

The second linear time strategy assigned a score for each bit location depending on the bit's effect on the surrounding media. The locations with the highest score were used to encode source data bits. The data density for this approach was comparable to the *Either-Neither Algorithm*.

The third algorithm assumed that the total light intensity map generated for an area had a so-called mass center. The opposite of the mass center, or the "darkest center", was called mass anti-center. Code blocks were categorized by the location of their mass centers. The speculation that there would be a correlation between the mass anti-center of the encoding area on the data page and the acceptable mass centers of the code block, which would speed up valid

code set construction, proved invalid. However, no correlation was found. Almost half of the code mass centers matched while the other half had them in different locations. This discouraged the mass-center oriented approach.

As the full enumeration algorithm required high time complexity and the linear time algorithms were not sufficiently competitive with the static encoding in terms of code density, new variations of dynamic encoding approaches were explored. Finally a new class of modulation algorithms called *quasi-dynamic encoding* was discovered.

Similarly to the dynamic approach, the *quasi-dynamic approach* analyzes data page information surrounding the target code block. Then, instead of computing the encoding table from scratch, the quasi-dynamic algorithm selects the most appropriate table from a subset of precomputed tables. A version of an algorithm from this class, called *re-coding*, provides 76% code density in the average case while having constant time complexity. This algorithm was selected for further analysis, including tolerance to noise and potential bit densities provided by the encoding.

The quasi-dynamic encoding was tested in the presence of jitter and media non-uniformities. The simulations showed that the modulation algorithm was capable of correcting jitter up to 6% in both vertical and horizontal directions, where the jitter was expressed in percentage of the distance between data bits. Regarding the tolerance against media non-uniformities, the simulations showed that the encoding was capable to tolerate signal intensity variations within 6% induced by the variations of the media uniformity.

The quasi-dynamic encoding was tested on a two-photon memory model using a point-spread function extracted from the two-photon memory system at Call-Recall, Inc. The results of the simulation showed that the dynamic modulation encoding enabled areal density of 240 Gbit/inch² for the two-photon memory system by storing data pages on 100 layers, which was 2.4 times higher than 100 Gbit/inch² found in conventional magnetic storage, which is capable of storing data in only one layer.

10.2 FUTURE DIRECTIONS

The new approach presented in this dissertation provides numerous research avenues that can be further explored.

Following are a few possible future research directions:

- Inter-page cross talk may occur at high data page densities for PODS [66]. Adaptation of the devised algorithms to a 3D environment may address these problems, taking in account neighboring page information.
- Another problem arising for PODS is data readout alignment, which suggests embedding clock signals in the 2D and 3D encoding schemes.
- Ref. [51] presents an interesting approach for iterative data page readout for PODS. There is a possibility to merge dynamic encoding and iterative page readout thus further improving data integrity or encoding performance.
- Adaptation of the dynamic and quasi-dynamic encoding algorithms for other types of volumetric memories, such as holographic optical memory.
- Adaptation of the quasi-dynamic encoding technique to very dense parallel optical communication channels. Researchers at McGill University have shown that encoding can lower the energy requirements for information transmission as well as improve the data integrity [67]. Explore the possibilities to improve data integrity over the channels and also minimize the energy requirements for the transmission.

APPENDIX A: PROGRAMMABLE OPTOELECTRONIC INPUT HEAD

We have designed a prototype for a smart optical input head. This device is a programmable chip that is capable of directly receiving parallel optical data, converting it to digital data and processing the digital data according to the device configuration. The device can be configured to perform parallel operations, such as data decoding. The device designed and implemented in our lab is a proof of concept device, therefore its functionality and number of channels are limited. However, the device implements architecture that is scalable and potentially can perform complicated operations such as modulation encoding and decoding for PODS systems.

The prototype was designed as an optically reconfigurable field programmable gate array (OFPGA) [9] in our laboratory. OFPGA is implemented on a chip with a 4x4 optical detector array, which is capable of receiving and processing optical information. The chip can be reconfigured dynamically to perform various computations.

The OFPGA integrated circuit is designed as system based on a reconfigurable computing paradigm [68] and providing high-speed configuration. In these environments, the computation resources provided by the field programmable gate array are re-used in multiple configurations throughout a single application program. Thus, when there are not enough hardware resources to perform a whole computation on-chip at once, the computation is split in configurations and performed each at a time. Additionally, the OFPGA device can accept highly parallel optical input data for immediate processing.

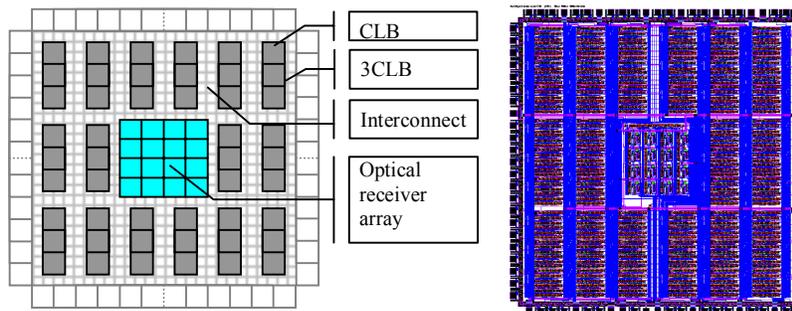


Figure 50: OFPGA logical and physical layouts

A.1 ARCHITECTURE

The proof of concept OFPGA device consists of 48 3-input 1-output configurable logic blocks (CLB), which can send and receive signals via configurable interconnect (Figure 50). The configuration data can be uploaded electronically or optically using a 4x4 optical receiver array, which can also be used as optical data input during the execution of the configuration by the device.

The CLBs are arranged in a 6 by 9 grid. All CLBs are divided in groups of three (3CLB). There are 16 3CLB groups. Each of those groups receives a separate optical signal for configuration or data by the optical receiver array. Therefore, all of the 3CLB groups can be configured simultaneously, even though the configurations are different. Each of the CLB in a 3CLB group can also be configured separately. This makes partial device re-configuration possible, including the case when several CLBs execute some computation while others are being reconfigured. In the center of the CLB grid resides a 4x4 array of optical receiver cells. Each cell has two optical detectors that differ in sensitivity and speed. The receiver cell pitch is 250 microns.

A.2 CONFIGURABLE LOGIC BLOCK DESIGN

The configurable logic blocks (CLBs) are composed of a bit serial configuration register, input multiplexer, a programmable Boolean logic function, and a flip-flop (Figure 51). Since configuration input data line may carry information from the optical receivers, we can use it as an “optical” input for the CLB. The flip-flop can be clocked by either a global clock signal or one of the inputs of the CLB. Thus, the flip-flop can be used for either data storage or frequency divider. Finally, the CLB has carry logic and a carry output signal, that allows it to be used as a full adder and combined with other CLBs to make an n-bit adder.

A.3 INTERCONNECT

The OFPGA internal interconnect provides routing of the following classes of signals (Figure 52): Short – CLB to any of its 8 immediate neighbor CLBs; Long – CLB to some of the other CLBs that are 3 interconnect units distant; I/O – perimeter CLB inputs and outputs to the I/O pins of the device; Global In – global data from input pins to all CLBs; Carry – carry signal routing when the CLBs are used as Full Adders. Optic – signals received by optical receiver array to the inputs of the CLBs; Control lines, such as clock, configuration data and enable signals.

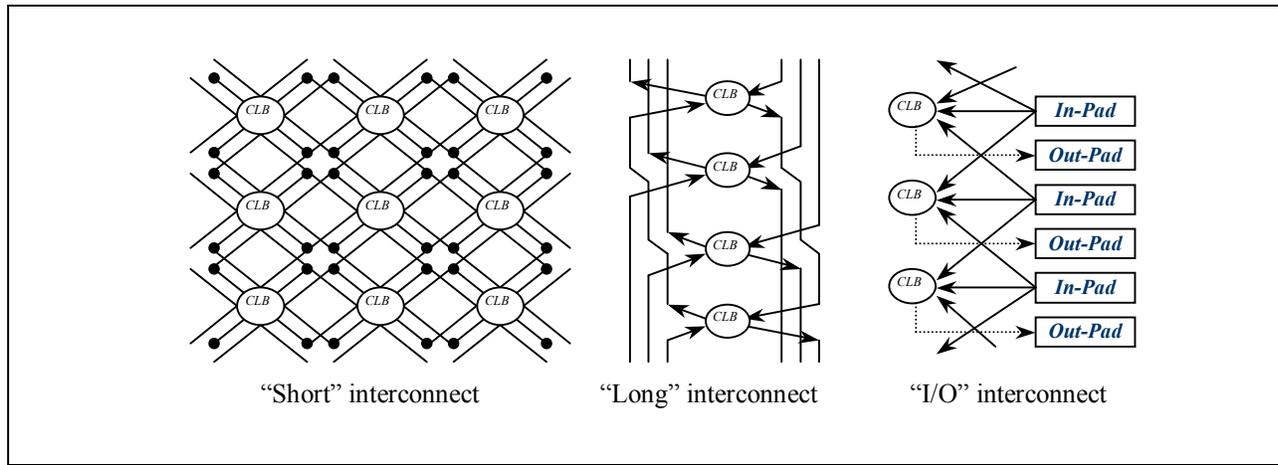


Figure 52: OFPGA interconnect

A.4 OPTICAL RECEIVER ARRAY

The optical information is received by 4x4 array of receiver cells. Each cell has two versions of square shaped

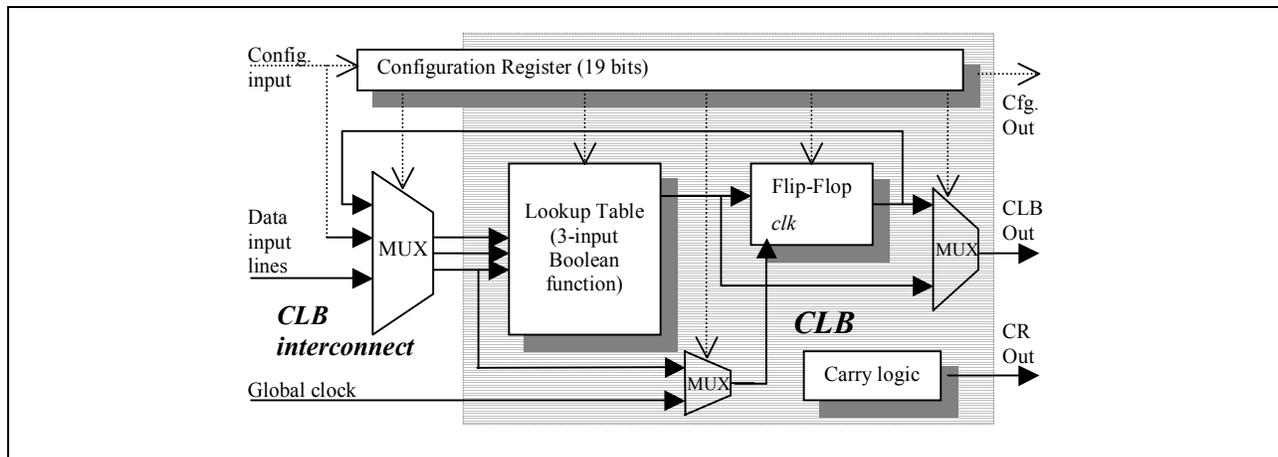


Figure 51: OFPGA configurable logic block

optical detectors implemented in silicon (with 17 μ m and 34 μ m on each edge) that differ in sensitivity and speed. The larger cell is more sensitive because more area is exposed to the optical signal, but it is slower because it has higher capacitance due to the larger size. Similarly the smaller cell has faster performance although it needs higher incoming optical power to perform reliably. The receiver array is located in the middle of the chip for easier integration with an optical system. The receiver cell pitch is 250 microns, the same as the pitch of typical VCSEL arrays that would be used as transmitters of the optical configuration data.

A.5 CONFIGURATION MODES

The configuration data for one CLB and its related interconnect fits in 19 bits of information. This information is clocked in serially for each CLB. There are 4 possible configuration modes: 2 optical and 2 electrical. The optical ones differ by the size of optical detectors they use. The electrical configuration modes provide fast “all-same” or “all-different” configuration options. For both optical modes the 3CLB groups are configured in parallel, thus only $19 \times 3 = 56$ clock cycles are necessary to configure the whole device. In the third mode, the configuration data is input serially, thus requiring $19 \times 48 = 912$ clock cycles for the whole device. For the fourth mode, used for testing, all the CLBs receive the same configuration data; thus, the system can be configured in only 19 clock cycles.

A.6 PERFORMANCE

The waveform in Figure 53 is a snapshot of a data word analyzer from an actual test of the device and it illustrates OFPGA performance by monitoring three CLB outputs. The waveform in the figure shows an example of four stages of the OFPGA. First, OFPGA is configured as a three bit counter in 19 clock cycles. Next, in the framed section labeled “Counter” the device performs a three-bit counter operation for 16 clock cycles. The diagram shows the CLB output signals that have become the counter output signals. Afterwards the OFPGA is re-configured to be a shift register and, finally, in the framed section labeled “Shift Register” it operates as the one. The reconfiguration in this experiment was done optically at clock speed of 4 MHz., allowing reconfiguration times between 5 μ s for partial and 15 μ s for full reconfiguration. Electrical tests showed that the device is capable running at 100MHz.

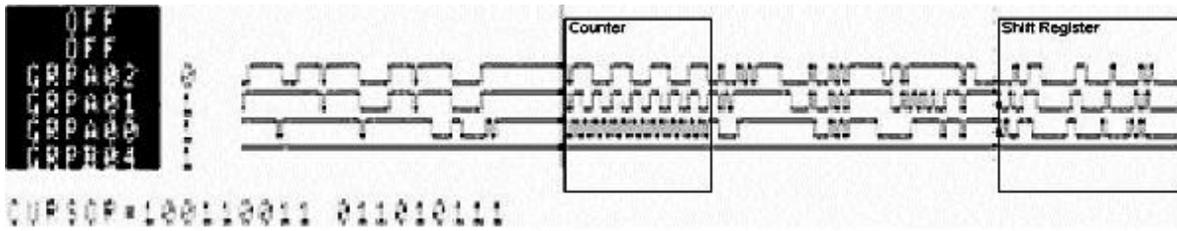


Figure 53: OFPGA waveforms

The OFPGA has been designed as a proof-of-concept device with fabrication in 1.2 μm SCMOS technology. State of the art systems use smaller feature sizes, which allow faster operation speeds, more optical detectors, and increased CLB count. Modern optical detectors and receivers are capable to perform at gigahertz frequencies. Thus, one can foresee OFPGA circuits with hundreds of CLBs and configuration times under a microsecond. This would enable a new class of high-speed reconfigurable processors based on an optoelectronic technology. Such devices could be used successfully as smart read/write heads with dynamic data modulation capabilities for mass storage systems.

Extending the OFPGA with a VCSEL array would enable optical data transmission after it's processing. Thus the extended OFPGA would become a device capable of processing, receiving, and transmitting data in both optical and electrical domains, which makes it an attractive device for advanced communications and optical data storage applications.

APPENDIX B: PAGE ORIENTED STORAGE SYSTEM SIMULATION SOFTWARE

The optical memory simulation software (OMSS) was created to test various modulation encoding and decoding algorithms for optical page-oriented data storage (PODS) systems. It simulates data recording and retrieval processes for a PODS memory using point spread function (PSF), which describes each bit storage profile. The PSF together with the distance between stored bits determine inter-symbol interference (ISI), which is calculated by the simulation software. Additional noise effects such as jitter and material non-uniformities can be added to the simulation process.

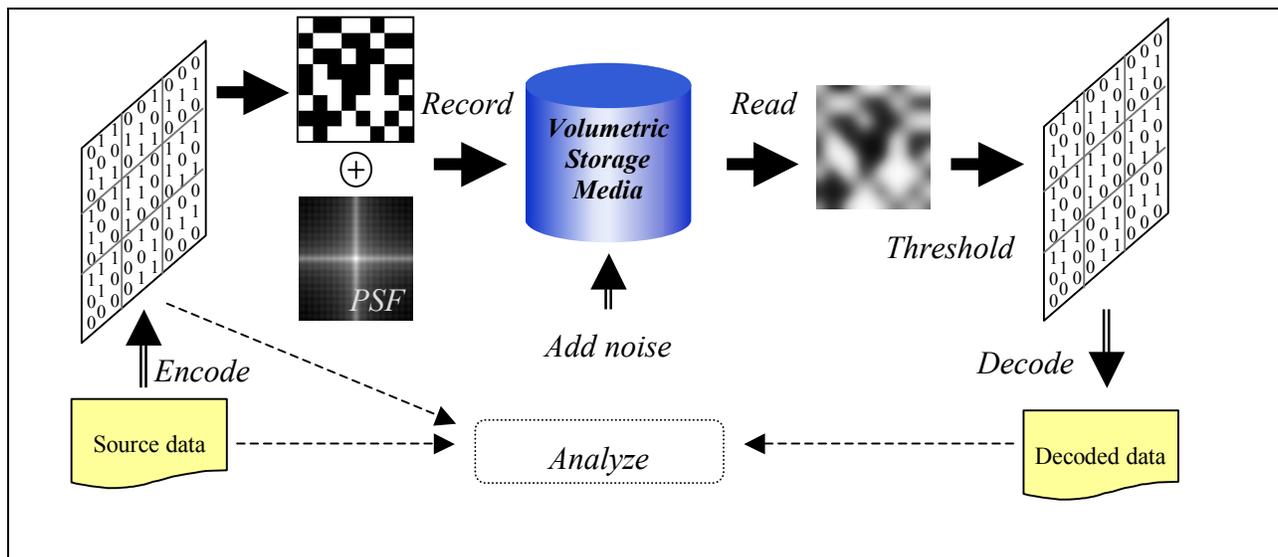


Figure 54: Simulation flow

The simulation flow is shown in Figure 54. Data to be encoded is read from a source file, then encoded using the selected encoding algorithm. The result is a binary 2D array representing a data page in the PODS. Convoluting the PSF that defines recording process profiles for each bit with the bit array simulates the recording process. The result is an array of analog intensity values for each bit. Jitter is simulated by appropriately displacing the PSF matrix before convoluting it with each bit. This analog array can be further manipulated by adding noise, described in detail later in this chapter. Applying a threshold to each bit intensity value simulates the readout of the data page. Thus every bit with intensity below the threshold value is detected as '0' and every bit with intensity value above the

threshold is detected as ‘1’. The detected bits make a binary 2D array that is decoded using the pre-selected modulation-encoding algorithm. The decoded data is compared to the original data to measure the bit error rate.

Besides the simulation capabilities the software includes other features that were extensively used for data generation and analysis. Such features include a random data file generation using Mitchell-Moore algorithm [63], 2D array plotter for visual representations of a memory page and point spread functions, and PSF matrix generation from imported PSF images.

B.1: SIMULATION SOFTWARE INTERFACE

The simulation software can be run either from a command line or graphical user interface. The synopsis of the command line parameters is given in the TableXX. The encoding and decoding parameters for the start with a dash ‘-’, while the other features such as data plot generation start with a plus ‘+’ character. We use numbers as examples for parameters that accept numeric values in the table. File names are given in italics, and should include full path. The parameters are followed by an input data file name or a folder name containing the input files.

Table 17: Simulator command line parameters

Parameter	Explanation
-OM 64x64x4	Define optical memory (OM) size with three dimensions. In this example the dimensions are 64 by 64 bit pages and 4 layers of pages. The acceptable values are memory pages of size between 1 x 1 and 4096 x 4096 bits, and up to 1024 pages (layers). Encoding will stop when either the input data file is fully encoded or there is no more space in the simulated memory. The layer number will be ignored if the “-RP - Reuse Pages” parameter is used, described below.
-RP	Reuse memory pages – only one layer will be used for the volumetric memory simulation. Thus, infinite number of pages is theoretically possible, allowing to simulate storage and

	retrieval of data files of arbitrary size.
-CB 3x3	Define the geometry of a code block, i.e. its width and height. The example tells the simulator that code blocks of size 3 by 3 bits are used for encoding.
-T 450	Define detection threshold for the encoding algorithms and data retrieval. The threshold is given in thousands of a maximal light intensity. The example defines the threshold to be at 0.45 of the maximal light intensity.
-T 100x600x10	Define a range of the threshold. Multiple experiments will be performed for thresholds between the given minimal and maximal values and increasing it by a given step value. All values are given in thousands of light intensity. The example tells the simulator to change the threshold from 0.10 to 0.60 by increasing it by 0.01 of the light intensity value for each encoding experiment. Each experiment encodes the whole input data file or until the simulated memory is full.
-NM 30	Define noise margin in thousands of maximal light intensity value. the example defines the noise margin to be 0.03% on both below and above the threshold.
-NM 0x150x10	Define a range of the noise margin. Multiple experiments will be performed for noise margins between the given minimal and maximal values and increasing it by a given step value. All values are given in thousands of light intensity. The example tells the simulator to change the noise margin from 0% to 15% by increasing it by 1% of the light intensity value for each encoding experiment. Each experiment encodes the whole input data file or until the simulated memory is full.
-a <algorithm>	Select the encoding and decoding algorithm. The following choices are available: DYN – dynamic encoding, full enumeration, EN – Either-Neither linear time algorithm, OE – Optimized Either-Neither algorithm, TREE – Codeword Tree algorithm, essentially optimized Full-Enumeration algorithm for faster performance, 49 – Static encoding for PODS, code rate 4/9, developed at University of Southern California [6],

	<p>716 – Static encoding for PODS, code rate 7/16, developed at University of Southern California [7],</p> <p>COPY – Copy input file to memory with no encoding,</p> <p>REC – Quasi-dynamic Second Chance Re-coding algorithm.</p>
-Z <fill-order>	<p>Select the algorithm for filling the memory data page with code blocks. The following fill-orders are available:</p> <p>L – linear, left-right top-down fashion,</p> <p>Z – Z-fill order,</p> <p>X – X-fill order,</p> <p>R – pseudo-random fill order.</p>
-JD 0x120x5	<p>Define data reading to media jitter. First two numbers define the maximal jitter interval in 0.1 percents of the distance between two pixel locations for a series of experiments. The third number defines the step for jitter increase between the series of experiments. The example defines maximal jitter change between 0% and 12% increasing it by 0.5% for every next experiment.</p>
-JE1 0x120x64 -JE2 0x120x64 -JE3 0x120x64	<p>Define data recording to media jitter. JE1 – Random amplitude and direction jitter, JE2 – Constant amplitude and angle jitter (misalignment over the whole page), JE3 – Radial jitter with changing angle but constant displacement. First two numbers define the maximal jitter interval in 0.1 percents of the distance between two pixel locations for a series of experiments. The third number defines the number of different jitter cases to generate, determined by the number of PSF matrices generated, from which one will be chosen at random for each data page bit. Note, that “JD” described above also must be specified as it defines the maximum jitter increase between experiments. The examples define maximal jitter change between 0% and 12%, and 64 different jitter PSF matrices (jitter configurations) will be generated.</p>
-U1 0x100x100 -U2 0x100x100	<p>Define non-uniformity noise. U1 – constant noise, U2 – high frequency noise, U3 – low frequency noise. Minimum noise, maximum noise, step. First two numbers define the</p>

-U3 0x100x100	<p>maximal noise interval in 0.1 percents of the distance between two pixel locations for a series of experiments. The third number defines the step for jitter increase between the series of experiments.</p> <p>Note, that “JD” described above also must be specified as it defines the maximum jitter increase between experiments.</p>
-o <i>fname</i>	<p>Output file name. The contents of the encoded optical memory will be written to this file. The file is a TAB-delimited 2D array formatted as such for easy import to a spreadsheet.</p>
-log <i>fname</i>	<p>Log file name. All standard output and error messages will be logged to this file and also displayed on the console (window).</p>
-psf <i>fname</i>	<p>Point spread function (PSF) definition file name. PSF is defined as a TAB-delimited 2D array</p>
-t <i>fname</i>	<p>Encoding table file name. This file will be used as a lookup table for static encoding if COPY parameter is specified.</p>
-r	<p>Recursive processing. Input data file is assumed to be a folder. All files in this folder and it’s subfolders will be encoded.</p>
-x	<p>Do both encoding and decoding, measure bit error rate.</p>
-genRec	<p>Generate the encoding table for Quasi-dynamic Second Chance Re-coding.</p>
-d 7	<p>Input data word length. The encoder will use this many bits for input source word. Code block must fit this many bits multiplied by a code rate of the selected algorithm. Acceptable values are between 1 and 36, or AUTO for automatic selection of the source word size, determined by the code rate and code block size. For a variable code rate use ‘-1’ as the source word size. The example tells simulator to use source words of 7 bits each and constant code rate.</p>
+P<options> -P<options>	<p>Enable (+) or disable (-) print options. The <options> can be a list of options separated by spaces, where each option can be one of the following (or abbreviated as in parentheses):</p> <ul style="list-style-type: none"> setup (s) Print setup (command line parameters). basic (b) Print basic information: input file name, size, code density.

	<p>lineinfo (l) Print memory page line-updates information. Useful for tracking the progress of encoding for large data files or memory sizes.</p> <p>codelen (c) Print codeword length statistics.</p> <p>omfile (o) Generate “.om” files. An “.om” file gives a snapshot of a 32x32 bit area of a data page in both binary (encoded) form and analog, with ISI present form. Also includes statistics about which codewords were selected how many times.</p> <p>gcodes (g) Print Good-bad codewords info (only for code blocks smaller or equal to 3x3 bits).</p> <p>fcodes (f) Print “friendly” codeword statistics (for code blocks smaller or equal to 3x3 bits). Generates statistical information about code word usage frequency and generates Re-Coding table, which can be later used for Quasi-dynamic encoding.</p> <p>allCBs (a) Print every code-block location information, not just a summary for each line. Useful for debugging purposes.</p> <p>jitEnc Print encoding jitter progress and report messages.</p> <p>jitDec Print decoding jitter progress and report messages.</p> <p>ll Print everything, except options <i>gcodes</i>, <i>fcodes</i> and <i>allCBs</i>.</p> <p>none Print no messages.</p>
+b	Load a batch file with PSF image as bitmap files and parameters such as center coordinates and grid size. Create PSF matrix and save as a text file. Perform encoding to test the PSF performance.
+p	Plot 2D surface from the input text file that has a 2D array of numbers. nput (text) File format: First 2 numbers are the dimensions of the matrix, next 2 numbers (optional) are the min and max values, followed by contents of the matrix. Numbers are space or tab-delimited. The output is a bitmap file with the same name as the input file except with “.bmp” extension added.
+3	Make a 3D surface plot from a bitmap, which is provided as the input file. Bitmap color

	values define elevation of the surface points.
+R +Rr	Analyze randomness of the input file and print the results. if “+Rr” is used, analyse all files in the input folder and its subfolders.
+S	Shut down the computer. Useful as a last call in a batch file after a long simulation left overnight.

The graphical front-end of the simulation software is shown in **Figure 55**. At the top of the dialog box are fields for the following files and folders: the executable of the encoder (retail or debug versions), the working directory, the input file or folder to be encoded and stored on the simulated memory, encoding point spread function (PSF) matrix file, decoding PSF matrix file, table file for static encoding option, output file for the simulated memory content export, and the log file where copies of all the messages and reports are stored.

The graphical interface allows convenient selection of the simulation parameters such as selection of the encoding algorithm, printing options for reports, simulated optical memory (OM) size, code block (CB) size, source data word size, thresholds and noise margins for simulation runs, memory page fill order algorithm, and media non-uniformity noise and jitter parameters. It also allows selecting options such as whether to recursively encode all files and subfolders located in the input folder, whether to reuse optical memory pages, thus simulating infinite size optical memory, whether to perform decoding right after encoding, and whether to generate a re-coding table for Quasi-dynamic encoding.

After making all necessary parameter selections the user can start the simulation process by pressing “Run” button. Alternatively, the user can generate the command line string with parameters in proper syntax and copy it to clipboard by pressing “to Clipboard” button. The generated command line string can be used in a batch command file to create a list of simulation runs that will be performed at a later time. Finally, after the simulations have finished the user can press “View Log” to open the log text file and view the simulation results.

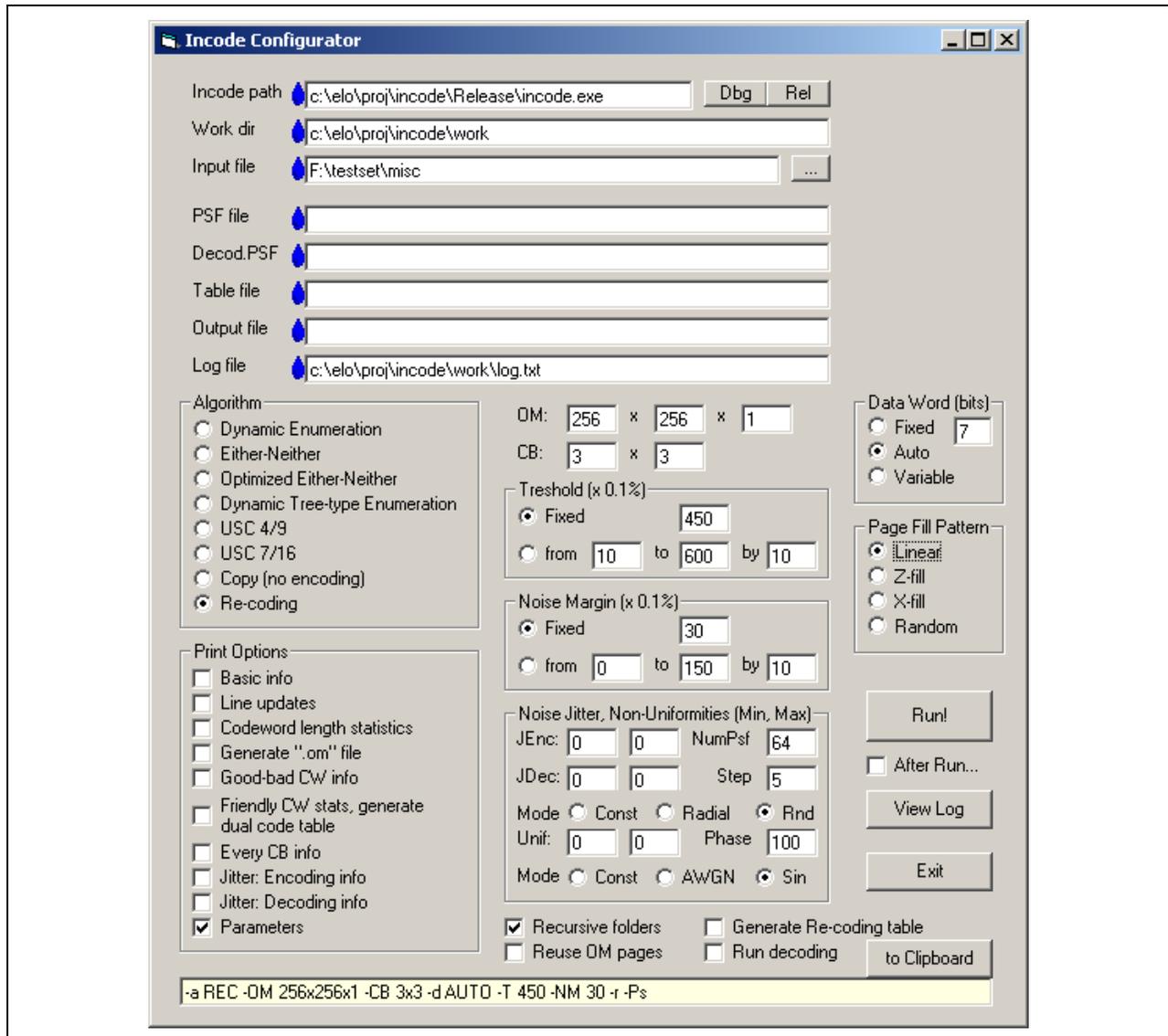


Figure 55: Simulation software graphical user interface

B.2: SIMULATION SOFTWARE IMPLEMENTATION

The simulation software was written in C++. The size of source code is approximately 5000 lines. The graphical front-end was written in Visual Basic.

BIBLIOGRAPHY

- 1 D. Psaltis, F. Mok, "Holographic memories," *Scientific American*, vol. 273 no. 5, Nov. 1995, pp. 70-76.
- 2 J. Ashley et al., "Holographic data storage," *IBM Journal of Research & Development*, vol. 44, no. 3, 2000, 341-368.
- 3 H. Zhang et al., "Multi-layer Optical Data Storage Based on Two-photon Recordable Fluorescent Disk Media," *Proceedings of Eighteenth IEEE Symposium on Mass Storage Systems*, 2001.
- 4 L.R. Carley, G.R. Ganger, and D.F. Nagle, "MEMS-Based Integrated-Circuit Mass-Storage Systems," *Communications of the ACM*, November 2000, Vol.43, No.11.
- 5 J.F. Hutton, George A. Betzos, Maureen Schaffer, and Pericles A. Mitkas, "Error correcting codes for page-oriented optical memories," *Proceedings of SPIE, Materials, Devices, and Systems for Optoelectronic Processing*, vol. 2848, 1996, 146-156.
- 6 D.E. Pansatiankul, A.A. **Sawchuk**, "Multidimensional modulation codes and error correction for page-oriented optical data storage," *Proceedings of Optical data storage topical meeting (ODS'2001)*, 2001, pp. 94-97.
- 7 D.E. Pansatiankul, A.A. Sawchuk, "Fixed-length two-dimensional modulation coding for imaging page-oriented optical data storage systems", *Applied Optics*, vol. 42, pp. 275-290.
- 8 L. Selavo, D.M. Chiarulli and S.P. Levitan, "Real-time adaptive encoding for 3D optical memories," *Three- and four-dimensional optical data storage conference, Proceedings of SPIE*, vol. 4459, 2001, pp. 344-351.
- 9 L. Selavo, S.P. Levitan and D.M. Chiarulli, "An Optically Reconfigurable Field Programmable Gate Array," *Proceedings of Optics in Computing (OC'99)*, 1999.
- 10 J.W. Toigo, "Avoiding a Data Crunch," *Scientific American*, May 2000.
- 11 "Seagate Sets Areal Density Storage Record", Seagate, News + Info, Technology, Nov. 2003, <http://www.seagate.com/newsinfo/technology/d4g.html>.
- 12 J. F. Heanue, M.C. Bashaw, and L. Hesselink, "Volume holographic storage and retrieval of digital data," *Science* vol. 265, 1994, pp. 749-752.
- 13 J. Dovic et al "Multi-track DVD-ROM," *Optical data storage topical meeting 2001*, vol. 4342, April 2001, pp. 112-114.
- 14 D.E. **Pansatiankul**, *Multi-dimensional modulation coding for page-oriented data storage systems*, doctoral dissertation, Dept. Electrical Engineering, Univ. of Southern California, May 2002.
- 15 "The era of giant magnetoresistive heads", White paper, IBM Storage Systems Division, 1998, <http://www.storage.ibm.com/hdd/technolo/gmr/gmr.htm>
- 16 C. Tsang et al., "5 Gbits/in² Recording Demonstration With Conventional AMR Dual Element Heads And Thin Film Disks," *IEEE Trans. Mag.*, vol. 33, no. 5, Sept. 1997, p. 2866.

- 17 “Removable small form factor storage devices for consumer electronics and information appliances,” White paper, IBM Storage Systems Division, 1999, <http://www.storage.ibm.com/hdd/micro/whitepaper/smallformfactorsd.htm>
- 19 I.S. Reed, G. Solomon, “Polynomial codes over certain finite fields,” *J. SIAM*, vol. 8, no. 2, June 1960, pp. 300-304.
- 20 S. Lin and D.J Costello, Jr, *Error Control Coding: Fundamentals and Applications*, Prentice-Hall, 1983.
- 21 L. Selavo, D.M. Chiarulli, and S.P. Levitan, “Improved Data Density Using Dynamic Encoding in a 2-Photon Memory,” *Integrated Photonics Research and Optics in Computing (IPR-OiC’2004)*, 2004, pp. 127-128.
- 22 T. Pratt, “Removable media storage devices”, VECTORS white paper, DELL, March 2000.
- 23 H.J. Verboom, “Selective ISI cancellation (SISIC) for high density optical recording using d=1 RLL channel-codes,” *Optical data storage topical meeting 2001*, vol. 4342, April 2001, pp. 85-87.
- 24 Amir and J. Finkelstein, World Patent WO98/36554, 1998.
- 25 T. Higuchi et al., “50Gbyte read-only dual-layer disk for the high NA objective lens and blue-violet lasers,” *Optical data storage topical meeting 2001*, vol. 4342, April 2001, pp. 31-33.
- 26 J. Kim, T.D. Milster, “Design aspects of waveguide hybrid advance mems (WHAM),” *Optical data storage topical meeting 2001*, vol. 4342, April 2001, pp. 19-21.
- 27 Y. Nishi, H. Kando and M. Tearo, “Simulations of re-crystallization in phase change recording material,” *Optical data storage topical meeting 2001*, vol. 4342, April 2001, pp. 46-48.
- 28 I. Ichimura et al., “In-groove phase-change optical recording for a capacity over 24GB,” *Optical data storage topical meeting 2001*, vol. 4342, April 2001, pp. 139-141.
- 29 N. Yamada et al, “Phase change material for use in rewritable dual layer optical disk utilizing a blue-violet laser,” *Optical data storage topical meeting 2001*, vol. 4342, April 2001, pp. 22-24.
- 30 “165 gigabytes of data storage on a CD-sized disc”, White paper, Norsam Technologies, 1997, <http://www.norsam.com/hdrom.htm>
- 31 N.Richard et al, “Optimized factor of merit of the magneto-optical Kerr effect of ferromagnetic thin films,” *The European Physical Journal B*, 2000, pp. 419-422.
- 32 D. Psaltis and G.W. Burr, “Holographic data storage,” *Computer*, vol. 31, no. 2, 1998, pp. 52-60.
- 33 J. H. Hong, I. McMichael, T. Y. Chang, W. Christian, and E. G. Paek, “Volume holographic memory systems: techniques and architectures,” *Opt. Eng.*, vol. 34, 1995, pp. 2193-2203.
- 34 H-Y.S. Li and D. Psaltis, “Three dimensional holographic disks,” *Applied Optics*, vol. 33, 1994, pp. 3764-3774.
- 35 D. Psaltis, “Parallel optical memories,” *Byte*, vol. 17, no. 9, 1992, pp. 179-182.
- 36 A. Pu and D. Psaltis, “High-density recording in photopolymer-based holographic three-dimensional disks,” *Applied Optics*, vol. 35, 1996, pp. 2389-2398.

- 37 W.R. Babbit, "Time-domain frequency-selective optical data storage in a solid state material," *Optical Communications*, vol. 65, 1988, p.185.
- 38 C. De Caro, A. Renn, and U.P. Wild, "Hole burning, Stark effect, and data storage:2:holographic recording and detection of spectral holes," *Applied Optics*, vol. 30, 1991, pp. 2890-2898.
- 39 B. Kohler, S. Bernet, A. Renn, and U.P. Wild, "Storage of 2000 holograms in a photochemical hole-burning system," *Optics Letters*, vol. 18, 1993, pp. 2144-2146.
- 40 E.S. Maniloff, S.B. Altner, S. Bernet, F.R. Graf, A. Renn, and U.P. Wild, "Recording of 6000 holograms by use of spectral hole burning," *Applied Optics*, vol. 34, 1995, pp. 4140-4148.
- 41 A. Renn and U.P. Wild, "Spectral hole burning and hologram storage," *Applied Optics*, vol. 26, 1987, pp. 4040-4042.
- 42 F.M. Schellenberg, W. Lenth, and G.C. Bjorkland, "Technological aspects of frequency domain data storage using persistent spectral hole burning," *Applied Optics*, vol. 25, 1986, 3207-3216.
- 43 E. Walker et al., "Two-photon volumetric optical disk storage systems experimental results and potentials," *Proceedings of Integrated Photonics Research and Optics in Computing (IPR-OC'2003)*, 2003, pp. 170-172.
- 44 A.S. Dvornikov, S. Esener, and P.M. Rentzepis, "Three-dimensional optical storage memory by means of two-photon interaction," *Optical Computing Hardware*, J. Jahns and S. H. Lee, eds., Academic Press, Massachusetts, 1994, pp.287-325.
- 45 S. Hunter, F. Kiamilev, S. Esener, D.A. Parthenopoulos, and P.M. Rentzepis, "Potentials of two-photon based 3-D optical memories for high performance computing," *Applied Optics*, vol. 29, 1990, pp. 2058-2066.
- 46 F.B. McCormick, "2-photon optical storage technology," *Optical Processing and Computing*, Y. Fainman, ed., SPIE International Technical Working Group Newsletter, Special Issue on Optical Data Storage, vol. 9, no. 1, April 1998, p. 1.
- 47 D.A. Parthenopoulos and P.M. Rentzepis, "Three-dimensional optical storage memory," *Science*, vol. 245, 1989, pp. 843-845.
- 48 M.M. Wang, S.C. Esener, F.B. McCormick, I. Cokgor, A.S. Dvornikov, and P.M. Rentzepis, "Experimental characterization of a two-photon memory," *Optics Letters*, vol. 22, 1997, pp. 558-560.
- 49 T.D. Milster, Y. Zhang, T. Y. Choi, S. K. Park, J. Butz and W. Bletscher, "Potential for Volumetric Bit-Wise Optical Data Storage in Space Applications," *Proceedings of NASA's Earth Science Technology Conference (ESTC2004)*, June 2004.
- 50 T. Tsujioka, M. Irie, "Theoretical study of the recording density limit of a near-field photochromic memory," *JOSA B*, vol. 15, no. 3, March 1998, pp. 1140-1146.
- 51 N. Intharasombat, A.A. **Sawchuk**, "Iterative detection for imaging page-oriented optical data storage," *Proceedings of Optics in Computing (OC'2003)*, 2003, pp.173-175.
- 52 W.D. Koek, "Temporary holographic data storage using bacteriorhodopsin," *Special Multi-Part Issue on: Holographic Materials for Data Storage, SPIE's International Technical Group Newsletter*, vol. 14, no. 2, 2003, p. 4.

- 53 J.L. Kann et al., "Mass Storage and retrieval at Rome Laboratory," *Proceedings of 5th NASA Goddard Mass Storage Systems and Technologies Conference*, 1996, pp. 389-406.
- 54 G.W. Burr, "Holography for information storage and processing," *SPIE Conference on Wave Optics and Photonic Devices for Optical Information Processing II*, Paper 5181-10, August 2003.
- 55 J.L. Kann and F.B. McCormick, "Numerical simulations of scattering in a two-photon optical data storage system," *Applied Optics*, vol. 37, 1998, pp. 4173-4182.
- 56 L. Zhang and M.A. Neifeld, "Blockwise data detection for spectral hole-burning memories," *Applied Optics*, vol. 40, 2001, pp. 1832-1842.
- 18 "The PC Guide," Site Version: 2.2.0 - Version Date: April 17, 2001, <http://www.PCGuide.com/>.
- 58 A.J. Viterbi, "Error bounds in convolutional codes and an asymptotically optimal decoding algorithm", *IEEE Transactions on Information Theory*, 1967.
- 59 D. Forney Jr, "The Viterbi Algorithm," *Proceedings of the IEEE*, vol. 61, no. 3, 1973.
- 60 D.E. Pansatiankul, A.A. Sawchuk, " Variable-length two-dimensional modulation coding for imaging page-oriented optical data storage systems", *Applied Optics*, vol.42, 2003, pp. 5319-5333.
- 61 Y. Wu, M.A. Neifeld, "Focal Plane Image restoration using the 2D4 algorithm," *Optics in Computing Technical Digest*, 2001, pp 21-23.
- 62 J. Walker, "ENT - A Pseudorandom Number Sequence Test Program," October 20th, 1998, <http://www.fourmilab.ch/random/>
- 63 D.E. Knuth, "Art of Computer Programming, Volume 2: Seminumerical Algorithms (3 Edition)", Addison-Wesley Pub Co, 1997, pp. 1-41.
- 64 M. Haahr, "Introduction to Randomness and Random Numbers," June 1999, <http://www.random.org/essay.html>
- 65 R.W. Irving, "An efficient algorithm for the 'stable roommates' problem," *Journal of Algorithms*, vol. 6, 1985, pp. 577-595.
- 66 N. Intharasombat, T. Ho, and A.A. **Sawchuk**, "Overcoming Inter-Page Interference (IPI) in 3D Optical Data Storage Systems," *Integrated Photonics Research and Optics in Computing (IPR-OiC'2004)*, April 2004, pp. 123-124.
- 67 J. Faucher, M.B. Venditti, E. Laprise, D.V. Plant, "An optoelectronic-VLSI chip with forward error correction to improve the reliability of parallel optical data links," *International Topical Meeting on Optics in Computing 2002*, 2002, pp. 147-149.
- 68 M.F. Sakr, et al., "Reconfigurable processor employing optical channels," *Optics in Computing '98, Proceedings of SPIE*, vol. 3490, 1998, pp. 564-567.