UNIVERSITY OF CALIFORNIA
Los Angeles

Timing Recovery Using Soft Information Feedback and Efficiency of Array Codes

A dissertation submitted in partial satisfaction of the
requirements for the degree Doctor of Philosophy
in Electrical Engineering

by

Esteban Luis Vallés

2007

The dissertation of Esteban Luis Vallés is approved.

Mario Gerla

Mihaela van der Schaar

Committee Co-Chair: Richard D. Wesel

Committee Co-Chair: John D. Villasenor

University of California, Los Angeles

2007

To my family and friends

Table of Contents

List of Figures

List of Tables

Acknowledgments

I would like to thank my parents Anahí and Enrique, my sisters Ana Sofía and Maria Clara, my grandparents Memé, Pepé, Blanca and Marcelo, my uncles, aunts and cousins, for their constant support and love, that made all these years of work away from home go by as if I had never left them.

When I arrived at UCLA, Prof. John Villasenor welcomed me to his group, helping me to begin my academic life by working in exciting and practical problems. Our initial work together led to a series of projects that became the core of my thesis work. I am extremely grateful for his help and guidance.

Prof. Richard Wesel, was extremely important throughout my career. He introduced me to coding theory, first as a professor, and later as my co-advisor. His constant support in good times and especially in the difficult times that I have faced in the past years, were what drove me to never give up and successfully complete this doctoral degree. I am forever thankful for all his help and support, both in my personal and academic life.

I would also like to thank my committee members Prof. Mihaela van der Schaar and Prof. Mario Gerla for their invaluable feedback to make this a better dissertation.

Throughout these years in southern California, I received the help and support of many people, too many perhaps to fit in these two pages of acknowledgments. I will begin to thank the people at U.C. Irvine starting with the Maranesi family, who were like a second family to me, Ashish Bhargave, Raja Jurdak, Ali Hamadeh, Daniel Gilden, Maria Cruz Villa-Uriol, Olgu Icoz, and all the other amazing people that I met there. I especially want to thank my friend, and future best-man Enis

Akay. His friendship and support not only changed my life as student, but taught me that the word *brother* has no relation to where we are born or raised, or what our culture or religion is. Our friendship went beyond all these mundane differences and is a bond that will remain forever.

Life in Los Angeles has been a very gratifying experience. I met great people at school but I was also able to meet amazing friends outside campus. I would like to thank Mario Blaum for all of his support and words of advice. Meeting Mario and being able to work with him encouraged me to continue in the pursuit of my degree. I want to thank Leonel Lopez Lavalle, Andres Pascuzzi, Roman Sacyk, Rogelio Adobatti, and the many others who constantly made me feel at home. I want to specially thank Miguel Griot for being a great roommate, co-worker and teammate in our UCLA Gauchos team. I also want to thank Andres Vila Casado, who not only inspires our research group with his never ending ideas, but who also became a great friend throughout these years. Working together with him and Miguel, was one of the most enjoyable parts of my last years at UCLA. We created an extremely enjoyable work environment that will be really hard for me to find again. I am grateful to all the people at Image Lab including Michael Smith, David Choi, Roland Osborne, Connie Wang, Hyungjin Kim, Dong-U Lee, and specially Christopher Jones (my third advisor) with whom I greatly enjoyed working in our many projects together both at UCLA and at JPL. I would also like to acknowledge the great people in the CSL laboratory: Aditya Ramamoorthy, Jun Shi, Wen-Yen Weng, Herwin Chan and Bike Xie.

Despite being away from my home town, Bahía Blanca, my true friends were always there for me. I want to thank Beatriz Olleta, Ramiro Villamil, Antón Ferramola, Juan Nieto, Mauro Aguilera, Martin Añazco, Diego Ostrowsky, Federico Roses and my long time friend Guillermo Estrada for always finding a way to remain close to me.

Finally I want to thank Ana Laura Martinez, for her constant support and encouragement and for listening to my practice presentations with patience and love. I look forward to our life full of happiness together.

Vita

1999-2000	Teacher Assistant
	Universidad Nacional del Sur, Argentina
2000	Ingeniero Electrónico
	Universidad Nacional del Sur, Bahía Blanca, Argentina
2000-2002	Research Assistant
	University of California, Irvine
2002	M.S. in Electrical and Computer Engineering
	University of California, Irvine
2002-2007	Research Assistant / Teaching Assistant
	University of California, Los Angeles
2004	Hitachi Global Storage Technologies
	San Jose,CA
2005	Jet Propulsion Laboratory
	Pasadena,CA
2006	Hughes Research Laboratories
	Malibu,CA
2007	Ph.D. in Electrical Engineering
	University of California, Los Angeles

Publications

E. L. Vallés , C. Jones, R.D. Wesel and J.D. Villasenor " Carrier and Timing Synchronization of BPSK via LDPC Code Feedback", Asilomar Conference

on Signals, Systems and Computers. Pacific Grove, CA. Nov. 2006.

M. Simon, E. L. Vallés and C. Jones, "Joint Carrier-Phase Synchronization and LDPC Decoding". NASA Tech Brief for NTR 43656. Available online at http://www.nasatech.com/Briefs/

M. Simon, E. L. Vallés , C. Jones, R.D. Wesel and J.D. Villasenor, "Information-Reduced Carrier Synchronization of BPSK and QPSK Using Soft Decision Feedback". 44th Annual Allerton Conference on Communication, Control and Computing. Sept. 2006.

D. Lee, E. L. Vallés, C. Jones and J. Villasenor "Pilotless Iterative Symbol Timing Recovery via LDPC Code Constraint Feedback" NASA Tech Brief for NPO 43112. Available online at http://www.nasatech.com/Briefs/

D. Lee, E. L. Vallés, C. Jones and J. Villasenor "Joint LDPC Decoding and Timing Recovery Using Code Constraint Feedback".
IEEE Communications Letters, vol. 10, no. 3, pp. 189-191, Mar 2006.

E. L. Vallés, A.I. Vila Casado, M. Blaum, J. Villasenor and R.D. Wesel, "Hamming codes are Rate-Efficient Array Codes".
IEEE Globecom 2005, St.Louis,MO, Dec. 2005.

C. Jones, E. L. Vallés, M. Smith and J. Villasenor, "Approximate-Min* constraint node updating for LDPC code decoding". IEEE Military Communications Conference (MilCom), Vol. 1, pp. 157-162, Oct. 2003.

Abstract of the Dissertation

Timing Recovery Using Soft Information Feedback and Efficiency of Array Codes

by

Esteban Luis Vallés

Doctor of Philosophy in Electrical Engineering

University of California, Los Angeles, 2007

Professor John D. Villasenor, Co-Chair

Professor Richard D. Wesel, Co-Chair

The main contribution of the thesis provides techniques for symbol frequency and phase tracking in a pilotless Low-Density Parity-Check (LDPC) coded transmission over an AWGN channel. In traditional receiver architectures, symbol acquisition and tracking are performed using phase-lock techniques that are independent of the channel-code decoding process. In burst reception scenarios, bandwidth inefficient piloting must often be embedded in a transmission in order to accelerate acquisition to aid symbol time tracking at low signal-to-noise ratios (SNRs). In this thesis we show that outputs from the constraint node side of a bi-partite decoding graph can be used to improve the estimation of symbol frequency and phase. We focus on the problem of a symbol frequency and/or timing offset between transmitter and receiver and describe a method capable of handling very large offsets with

complexity that grows linearly with the maximum offset size. Combining information from the set of parity-check equations of a LDPC code observations with a properly calibrated phase locked loop allows successful tracking of a constant time delay, a frequency offset and a random phase walk.

The problem of carrier-phase estimation in a coded environment is also addressed. Once again we assume a pilotless scenario where, by feeding back soft information from an iterative LDPC decoder, the phase information of the carrier is estimated. The joint problem of symbol-timing estimation in a digital transmission affected by carrier phase offsets is also analyzed.

We conclude this dissertation by examining the rate efficiency of array codes. By allowing a slight increase in encoding and decoding complexity, we observed that the same error correcting properties of these codes can be achieved by Hamming and Reed-Solomon (RS) codes over $GF(q)$. For single phased burst correction, non-binary Hamming codes maximize the possible code rate and can be decoded with similar complexity as array codes. For multiple burst correction, RS codes offer the same error correcting capability as array codes with a higher code rate.

Chapter 1

Introduction

A communication system transports information from a data source to a single or multiple user destination via a communication channel. Most physical communication channels are analog in nature. The information from the source is generated in the form of digital bits, which are used to modulate the properties of an analog carrier waveform. In order to minimize the average symbol-error rate, error correcting codes are commonly used. Fig. 1.1 shows how channel coding techniques insert redundant data before the modulation is performed to protect the source information from noisy interference present in the channel. The receiver block performs a mapping of the output of the communication channel back into the digital domain. This is done by sampling the continuous waveform at instants chosen by the timing recovery block. The receiver then decodes the demodulated data and forwards its decisions to the information destination.

In this chapter we briefly explain how the different blocks in Fig. 1.1 work, and how this dissertation contributes to different aspects of a digital communication system.

1.1 Digital Communication Systems

Fig. 1.1 shows a block diagram of a typical digital communications system. The information source, generates information data bits m. These bits are passed to

a channel encoder that adds additional redundant symbols to the original bits m. This allows most of the errors - introduced in the process of modulation, transmission over a noisy medium and demodulation - to be corrected at the receiver side. The channel model assumed in this dissertation is an additive white gaussian noise (AWGN) process with samples that are independent from the source symbols. At the receiver end, after the timing recovery block produces digital samples from the received waveform, the channel decoder utilizes the redundant symbols to correct data-symbol errors present in r. In a classical error correcting codes (ECC) system, *hard-decision* bits from the demodulator are fed into a binary decoder. To understand this concept, suppose a communication system transmits symbols with two possible values $\pm A$, that are received with additive noise n. A hard-decision decoding technique will likely define a threshold ρ, and use a decision rule: $\hat{A} = A$ if $r > \rho$ or $\hat{A} = -A$ otherwise. With hard decoding, the only information used from the received signal is whether it is above or below a certain threshold. In contrast, *soft-decision* decoding algorithms use all the available information at the receiver before making a decision. In the previous example, besides considering whether the received symbol is above or below a threshold, a *soft-metric* will also consider the relative value of the signal compared to ρ. Soft-decision algorithms achieve a reduction in the required transmitted power per bit, or a lower average symbol-error rate for a given transmitted power as compared to hard decision algorithms. Soft-decision algorithms are used throughout this dissertation, unless noted otherwise.

1.2 Error Control Coding

The history of ECC started with the introduction of block coding in the form of Hamming codes around 1950 [1], at about the same time as the work of Shannon [2]. In block coding, a binary information sequence is segmented into *message* blocks of fixed length. Each message block m consists of k information bits which can generate a total of 2^k distinct messages. The encoder transforms each input message m into a binary n-tuple c referred to as the *codeword* of the message.

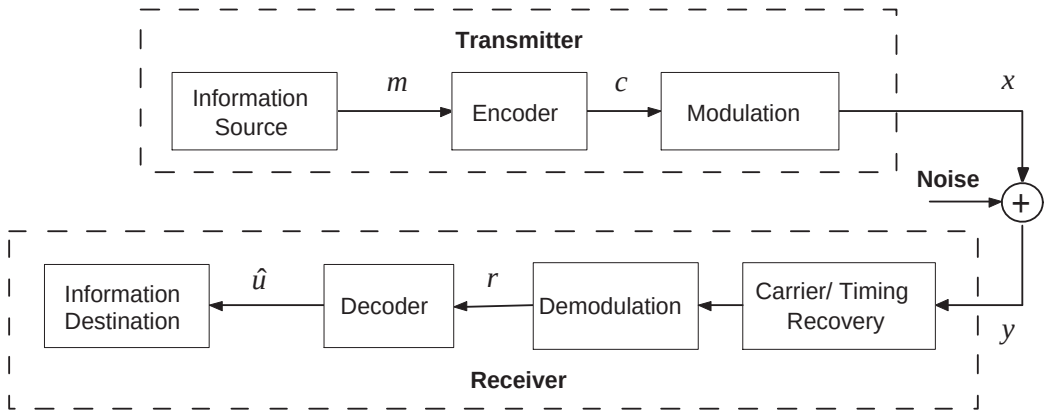


Figure 1.1: Digital Communications Channel

The set of 2^k possible codewords is called a *block code*. A desirable property for a block code to posses is linearity, which occurs when the 2^k codewords form a k-dimensional subspace of all n-tuples over the field $GF(2)$. A binary block code is linear if and only if the modulo-2 sum of any two codewords is also a codeword. A codeword can thus be represented as the product $c = m \cdot G$, where G is the *generator* matrix that spans the (n, k) linear code C. Note that G can be created with any k linearly independent codewords of an (n, k) linear code. For any $k \times n$ matrix G there exists an $(n - k) \times n$ matrix H with linearly independent rows, such that any vector in the row space of G is orthogonal to the rows of H. Hence the code can also be described by the *parity-check* matrix H such that $c \cdot H^T = 0$. We can also write H as,

$$H = \begin{bmatrix} H_1 & H_2 \end{bmatrix}, \tag{1.1}$$

where H_1 is an $(n - k) \times k$ matrix and H_2 is an $(n - k) \times (n - k)$ matrix. H_2 is constructed to be invertible, so that by row transformation through left multiplication with H_2^{-1}, we can obtain a *systematic* parity check matrix H_{sys} that is range equivalent to H,

$$H_{sys} = H_2^{-1}H = \begin{bmatrix} H_2^{-1}H_1 & I_{n-k} \end{bmatrix}. \tag{1.2}$$

Augmentation of the transposition of the left hand portion of the systematic parity check matrix H_{sys} with I_k yields the systematic generator matrix,

$$G_{sys} = \left[\begin{array}{cc} I_k & \left(H_2^{-1}H_1\right)^T \end{array} \right]. \tag{1.3}$$

The rows of G_{sys} span the codeword space. It is clear that, $GH^T = G_{sys}H_{sys}^T = 0$. It should be noted that even if the original H matrix is sparse (as in the case of LDPC codes), neither H_{sys} nor G_{sys} need to be sparse in general. G_{sys} is used for encoding and the sparse parity matrix H is used for iterative decoding.

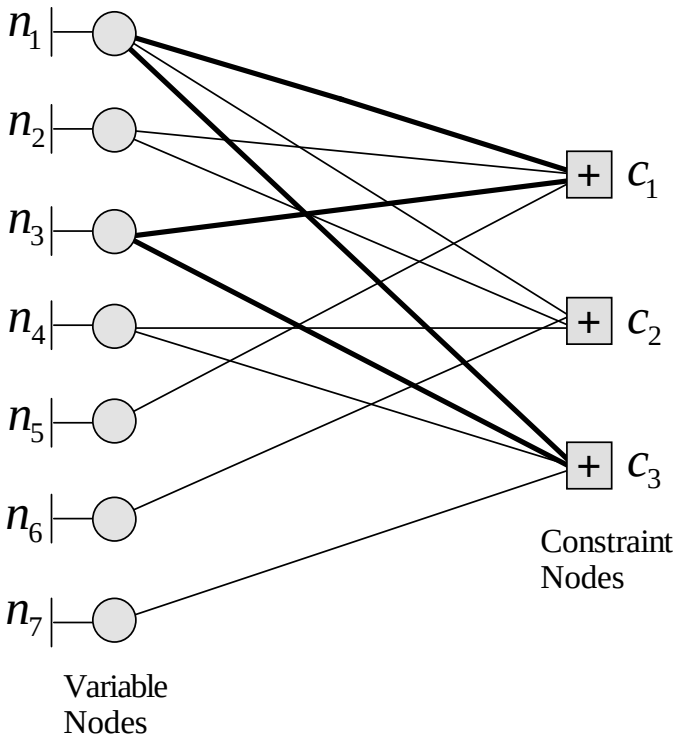
Any linear code can also be represented using a *bi-partite Tanner graph* [3]. These graphs have two sets of nodes - a set representing the transmitted bits and another set representing the constraints that the transmitted bits have to satisfy. For illustrative purposes consider a (7,4) Hamming code [1]. Its parity-check matrix is depicted in Fig. 1.2(a) together with its bi-partite graph, shown in Fig. 1.2(b). In this graph, n *variable nodes* form the left vertex set are connected to $(n-k)$ *constraint nodes* that form the right vertex set. Each variable node in the graph corresponds to one bit in the codeword, *i.e.* to one column of H. Each check node corresponds to a parity-check equation, *i.e.* one row of H. In the case of a systematic code, such as the one on Fig. 1.2(a), the first k variable nodes are *message* nodes and the last $(n-k)$ are *parity* nodes. If the entire set of variable nodes forms a valid codeword, then the exclusive-or performed on each constraint node will be zero. This can be mathematically expressed by the *syndrome* equation $s = r \cdot H^T$ where $s = 0 \leftrightarrow r \in C$. Iterative decoding algorithms work by passing messages between variable and constraint nodes, until a solution that satisfies $s = 0$ is found. Once this solution is found, the decision bits are forwarded to the information destination, as shown in Fig. 1.1.

1.2.1 Low-Density Parity-Check Codes

Low-density parity-check (LDPC) codes were proposed by Gallager in the early 1960s [4,5]. A LDPC code is a block code whose matrix H is sparse *i.e.* the parity-

$$H = \begin{array}{c} \begin{matrix} n_1 & n_2 & n_3 & n_4 & n_5 & n_6 & n_7 \end{matrix} \\ \begin{bmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 \end{bmatrix} \end{array} \begin{matrix} c_1 \\ c_2 \\ c_3 \end{matrix}$$

(a) Parity-check matrix



(b) Tanner graph

Figure 1.2: (a) (7,4) Hamming Code. (b) Corresponding Tanner graph. Cycle of length 4 shown in bold.

check matrix is primarily populated by zeros. Fig. 1.3 shows a parity-check matrix of a LDPC code with $(n, k) = (1944, 972)$. For this code, only 7181 out of its $1972 \times 972 = 1889568$ matrix positions (*i.e.* 0.37%) are non-zero. The structure of Gallager's codes (uniform column and row weight) led them to be called *regular* LDPC codes. Gallager provided simulation results for codes with block lengths on the order of hundreds of bits. However, these codes were too short to approach Shannon capacity (which is achievable in the limit of infinite block length) [2]. Furthermore, the computational resources to support longer random codes were decades away from being broadly accessible. Hence the LDPC coding received little attention from the research community for a period of over 30 years.

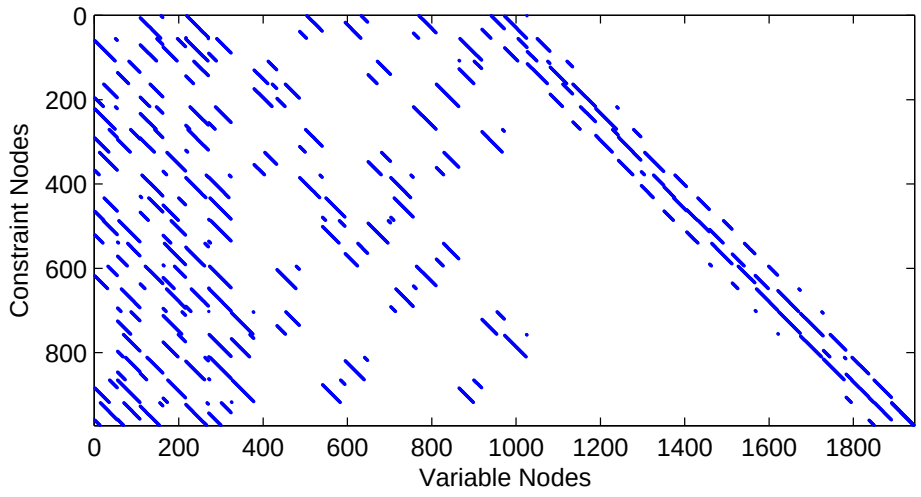


Figure 1.3: Parity check matrix for (n,k)=(1944,972) Irregular LDPC code

Following the ground breaking demonstration by Berrou *et al.* [6] of the impressive capacity-approaching capability of long random linear (turbo) codes, MacKay [7, 8] re-established interest in LDPC codes during the mid to late 1990s by pro-

viding a diverse set of construction methods for regular codes. Luby *et al.* [9] formally showed that properly constructed irregular codes can approach capacity more closely than regular codes. Richardson, Shokrollahi and Urbanke [10] created a systematic method called density evolution to analyze and synthesize *degree distributions* in asymptotically large random bipartite graphs under a wide range of channel realizations. As in the case of turbo codes, LDPC codes belong to the class of codes that can be efficiently decoded via *iterative* techniques. The work presented in the first five chapters of this dissertation exploit different properties of this type of error correcting codes.

1.2.2 Array codes

Array codes refer to a general class of algebraic error-correcting codes for use in detecting and correcting error bursts [11, 12, 13]. Each codeword is represented as a rectangular array. The dimensions of this array vary according to the error correcting capability of the code. These arrays define codewords in a block code by reading out the entries in the array in a particular order. However, they can also be considered as two-dimensional codewords that can be used in applications that store information in a two-dimensional medium, such as memories or hard drives. As it will be further explained in Chapter 6, one of the properties of these codes is that its parity-check equations, represented by constraint nodes, can be performed using exclusive-OR (XOR) operations over diagonal lines of the array. A binary parity-check matrix of an array code of length $n = 20$ is shown in for illustrative purposes in Fig. 1.4.

1.3 Digital Modulation Techniques

When digital data is transmitted over a band-pass channel, it is necessary to modulate the incoming data onto a carrier wave (usually sinusoidal) with fixed frequency limits imposed by the channel. The modulation process involves switching or keying the amplitude, frequency, or phase of the carrier in accordance with the incom-

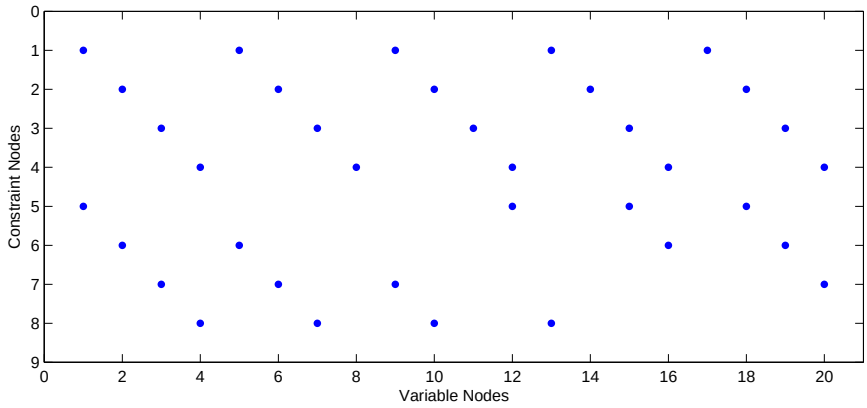


Figure 1.4: Parity check matrix for (n,k)=(20,12) array code.

ing data. Thus there are three basic modulation techniques for the transmission of digital data; they are known as amplitude-shift keying (ASK), frequency-shift keying (FSK), and phase-shift keying (PSK), which may be viewed as special cases of amplitude modulation, frequency modulation and phase modulation, respectively. In this this thesis we will concentrate on PSK modulations.

Modulation is defined as the process by which some characteristic of a carrier is varied in accordance with a modulating wave [14]. In digital communications, the modulating wave consists of an $M-ary$ encoded version of the data. Fig. 1.5 shows three PSK constellations for different values of M. For the carrier signal, we will use a sinusoidal wave. In our case, the feature used by the modulator to distinguish one signal from another is a step change in the phase of the carrier. Fig. 1.6 shows how the phase of a sinusoidal carrier is modulated by a sequence of alternating binary symbols. To perform demodulation at the receiver, we have the choice of *coherent*

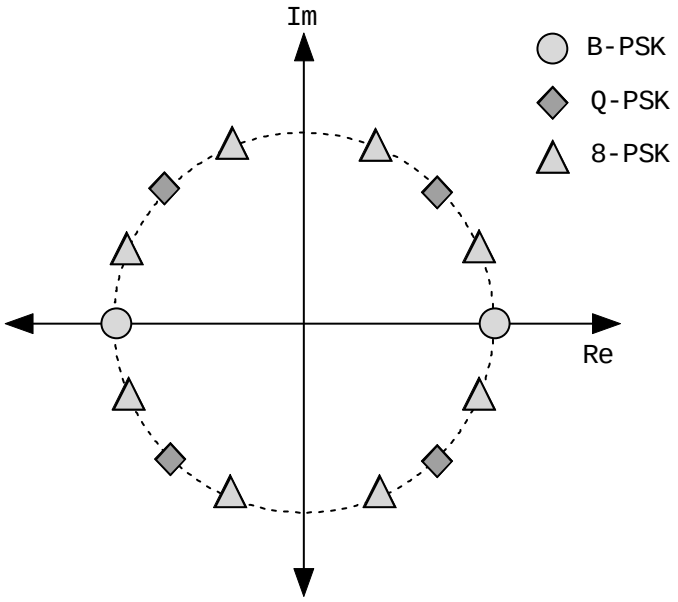


Figure 1.5: M-PSK constellations, for M=$\{2^1, 2^2, 2^3\}$

or *noncoherent detection*. In the case of coherent detection, the receiver requires the knowledge of the carrier's phase reference. Since this information is usually not known a-priori by the receiver, a carrier synchronization algorithm is usually used in the demodulation process. One of the contributions of this dissertation involves a carrier phase-recovery circuit that, unlike traditional receivers, utilizes information from the channel code decoder to aid the synchronization process in a pilotless environment. In noncoherent detection, knowledge of the carrier's phase is not required. The complexity of the receiver is therefore reduced at the expense of inferior error performance. The coherent reception of digitally modulated signals require that the receiver is synchronous to the transmitter. To accomplish this we need two basic modes of synchronization:

1. *Carrier Synchronization*: knowledge of both the frequency and phase of the carrier wave.

2. *Symbol Synchronization*: the receiver has to know the time instants at which the modulation can change its state so that it can determine when to sample

These two synchronization operations can be coincident or can occur sequen-

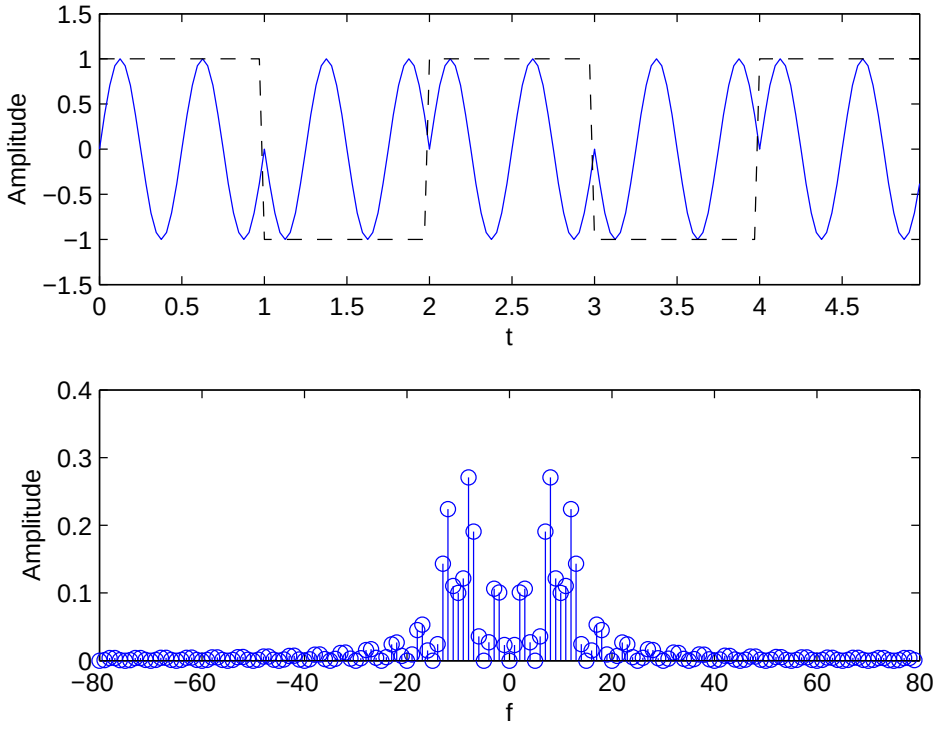


Figure 1.6: B-PSK signal as a function of time (top) and frequency (bottom).

tially one after the other. In order to aid the synchronization process it is also common to include a *pilot signal* in the transmission. This signal is known both at the transmitter and the receiver and is transmitted for supervisory, control, equalization, synchronization, or reference purposes but carries no information.

We thus see that a multitude of modulation/detection schemes exist, each scheme offering different advantages. The trade-off space includes the following design goals:

- Maximum data rate.
- Minimum probability of symbol error.
- Minimum transmitted power.
- Minimum channel bandwidth.

- Maximum immunity to interfering signals.
- Minimum circuit complexity.

Many of these goals have conflicting requirements so it is the choice of the engineer to try to select the extent to which each goal will be achieved based on what is feasible and the requirements of a specific system. This dissertation provides new techniques that increase the space of what is feasible, providing new choices in the trade-off space.

1.4 Outline

The rest of the dissertation is organized as follows. Chapter 2 introduces different soft-decoding iterative algorithms for LDPC codes. A new algorithm that we refer to as "Approximate Min* Belief Propagation" is presented. In Chapter 3 a pilotless symbol synchronization algorithm that uses information based on the constraint nodes of an LDPC code is introduced. This method assumes that perfect carrier synchronization occurs at the receiver. Chapter 4 presents a pilotless carrier-phase synchronization circuit, that also uses information from the LDPC decoder, assuming that the symbol timing and the carrier frequency are known. In Chapter 5, the combined problem of carrier-phase and symbol timing synchronization is addressed. Finally Chapter 6 focuses on maximizing the code rate of array codes. We propose using Hamming and Reed-Solomon codes for array code applications thus increasing the code rate at the expense of a slight increase in circuit complexity.

Chapter 2

Decoding Methods for LDPC Codes

In [4,5], Gallager introduced several decoding algorithms for LDPC codes. One of these algorithms is representable as a special case of the Sum-Product algorithm that has since been identified for general use in factor graphs [15] and Bayesian networks [16]. Some forms of the Sum-Product algorithm are better suited for implementation than others. It is possible to differentiate each potential technique through the examination of their respective operator sets and word length requirements. Given unlimited precision, however, all of these forms yield the same *a posteriori* (APP) estimation and we classify them collectively as the group of Full Belief Propagation (Full-BP) implementations.

Even Full-BP algorithms suffer performance degradation as compared to the optimum maximum likelihood (ML) decoder for a given code. This is due to the fact that bipartite graphs representing finite-length codes without singly connected nodes are inevitably non-tree-like. Cycles in bipartite graphs, such as the one shown in Fig. 1.2(b), compromise the optimality of Sum-Product decoders. The existence of cycles implies that the neighbors of a node are not in general conditionally independent (given the node), therefore Full-BP algorithms produce inaccurate a-posteriori probabilities. Code conditioning techniques [17] can be used to mitigate the non-optimalities of iterative decoders. Establishing the true ML performance

of LDPC codes with length beyond a few hundred bits is generally viewed as an intractable problem.

Most of the work in this particular field of coding relies on the use of Monte Carlo simulation to demonstrate code performance in terms of Bit Error Rate (BER) vs. Signal to Noise Ratio (SNR). At very low SNRs, errors occur often, and a sufficient statistic can be gathered readily with a workstation. However, at higher SNRs the situation is different. As SNR is increased the initial rapid decline of BER flattens somewhat. The knee of this flattening on a plot of BER vs. SNR indicates the beginning of the error-floor region of the code. Since the BER level of well conditioned codes lies below 10^{-8}, workstation simulation provides an inadequate means for finding a statistically sufficient set of error events to make accurate statements about the error floors of a given code. In this chapter, we present techniques for integer implementation and high-speed parallel realization of an LDPC decoder such that a system can be constructed on a FPGA board to search for error-floor events in time frames that are at least 3 orders of magnitude faster than simulations provided by workstations.

In the next section we begin by describing the concept of message passing iterative decoding techniques. In section 2.1.2 the core decoding algorithm is presented. This algorithm is a carefully optimized and a reduced-complexity version of the so-called Belief Propagation (BP) algorithm [16]. We note that the BP algorithm for LDPC decoding has been mapped to hardware in [18]. However, the authors of [18] did little to directly reduce the complexity of the decoding computations. This chapter develops a lower complexity (shift-and-add) decoding technique. While the new technique (even in the limit of infinite precision) does not implement BP decoding exactly, it suffers little or no performance loss in comparison to the BP decoder. Section 2.2 gives a discussion of finite precision issues and provides performance data for several quantization schemes. Section 2.3 describes the LDPC codec that has been developed at UCLA based on the constraint update technique of this chapter. Concluding remarks are made in section 2.4.

2.1 Message Passing Decoding

Let x be the transmitted signal corresponding to a binary codeword c using BPSK modulation: $x = 2c - 1$. Let y be the received signal equal to the sum of x and channel noise. A message passing decoder tries to solve for the value of x based on the knowledge of y. There are many ways to describe a soft-decision message passing algorithm. The one used in this dissertation exchanges log-likelihood ratios (LLRs) of the available information. The iterative decoding algorithms described in this work are commonly known as *flooding algorithms*, and behave as follows.

We define as u a message incident on a variable node,

$$u = \ln\left(\frac{p\left(x = 0|Y\right)}{p\left(x = 1|Y\right)}\right), \tag{2.1}$$

and v as the incident message on a check node,

$$v = \ln\left(\frac{p\left(x' = 0|Y'\right)}{p\left(x' = 1|Y'\right)}\right), \tag{2.2}$$

where primes in x' and y' are used to differentiate between left and right-going messages. At time zero, every variable node has an associated received message u_i^0. Messages are exchanged between nodes along the edges of the graph in discrete time steps. First, each variable node sends a message to each connected check node taking values in some message alphabet, as we will discuss in section 2.1.1. Each check node processes the messages it receives and sends back, to each neighboring variable node, a new message. Each variable node now processes the new messages it received together with its associated received value from the channel to produce new messages which it then sends to its neighboring check nodes. For every time step, a *cycle* or *iteration* of message passing proceeds with check nodes processing and transmitting messages followed by the variable nodes processing and transmitting messages. An important condition on the processing is that a message sent from a node along an adjacent edge may not depend on the message previously received along the same edge . This guarantees that only *extrinsic* information is passed along. This is known to be an important property of good message-passing

decoders [19].

2.1.1 Belief Propagation Algorithm

Belief propagation, is an iterative algorithm for computing marginals of functions on a graphical model commonly used in computer science and information theory problems. Judea Pearl in [16] developed a message-passing scheme that updates the probability distributions for each node in a graph in response to observations of one or more variables. This algorithm is provably efficient on trees but has also demonstrated empirical success in numerous applications applications such as LDPC codes that are not tree-like due to the presence of cycles.

Constraint Update Equations

For illustrative purposes, assume a constraint node of degree $d_c = 4$ like the one shown in Fig. 2.1(a). The constraint node equation for c_1 can be written as:

$$v_1 + v_2 + v_3 + v_4 = 0. \tag{2.3}$$

Note that the indices i for the incoming messages v_i correspond to the incoming *edge* number. In the above equation message v_4 corresponds to a message sent from variable node n_5 to constraint node c_1.
The outgoing message U_3 going towards n_3 is defined as follows:

$$\begin{aligned} p_{U_3} = p_{n_3,c_1} &= P(n_3 = 1, c_1|Y) \\ &= P(v_3 = 1, v_1 + v_2 + v_3 + v_4 = 0|Y) \\ &= P(v_1 + v_2 + v_4 = 1|Y) \\ &= p_1(1-p_2)(1-p_4) + (1-p_1)(p_2)(1-p_4) + \\ &\quad (1-p_1)(1-p_2)p_4 + (p_1p_2p_4) \\ &= \frac{1}{2} - \frac{1}{2}\prod_{i\in\{1,2,4\}}(1-p_i) \end{aligned} \tag{2.4}$$

where probabilities are labeled $p_{to,from}$, p_{edge} and $p_i = p(v_i = 1)$.

In practice, message passing algorithms do not use probability measures since product operations performed on probabilities are not easily represented. An initial attempt to solving this numerical method consists in replacing probabilities by *likelihood ratios*. Consider an even simpler example than (2.4) that will help us to derive the general update equations. Let us assume $d_c = 3 \Rightarrow v_1 + v_2 + v_3 = 0$. For a check-node of this type the likelihood ratio of this probability measure going from c_1 to n_1 is:

$$\begin{aligned}
\lambda_{U_1} = \lambda_{n_1,c_1} &= \frac{(1-p_{n_1,c_1})}{(p_{n_1,c_1})} = \frac{\frac{1}{2} + \frac{1}{2}\prod_{i\in\{2,3\}}(1-p_i)}{\frac{1}{2} - \frac{1}{2}\prod_{i\in\{2,3\}}(1-p_i)} \\
&= \frac{1-p_2-p_3+2(p_2p_3)}{p_2+p_3-2(p_2p_3)} \\
&= \frac{1+\left(\frac{1-p_2}{p_2}\right)\left(\frac{1-p_3}{p_3}\right)}{\left(\frac{1-p_2}{p_2}\right)+\left(\frac{1-p_3}{p_3}\right)} \\
&= \frac{1+\lambda_2\lambda_3}{\lambda_2+\lambda_3}.
\end{aligned} \tag{2.5}$$

The last step before obtaining the message passing equations for the sum-product algorithm combines the logarithm of the likelihood equation (2.5) $ln(\lambda_{U_1})$ with the exponential representation of the hyperbolic tangent

$$tanh(\alpha) = \frac{1-e^{-2\alpha}}{1+e^{-2\alpha}}. \tag{2.6}$$

When (2.5) and (2.6) are combined together we obtain:

$$tanh\left(\frac{1}{2}ln(\lambda_{U_1})\right) = \frac{1-e^{\frac{\lambda_2+\lambda_3}{1+\lambda_2\lambda_3}}}{1+e^{\frac{\lambda_2+\lambda_3}{1+\lambda_2\lambda_3}}} = \frac{(1-\lambda_2)(1-\lambda_3)}{(1+\lambda_2)(1+\lambda_3)} = \tanh(\frac{1}{2}\lambda_2)\tanh(\frac{1}{2}\lambda_3). \tag{2.7}$$

The general equation for a constraint node of degree d_c is therefore:

$$\tanh\left(\frac{1}{2}\ln\lambda_{U_j}\right) = \prod_{i=1,i\neq j}^{d_c}\tanh\left(\frac{1}{2}\ln\lambda_{v_i}\right). \tag{2.8}$$

Variable Update Equations

We now consider a message going from a variable node of degree $d_v = 3$, going along the edge that connects the variable node n_1 with check node c_2, as shown in Fig. 2.1(b).

$$\begin{aligned}
&p_{V_2} = p_{c_2,n_1} = P\left(n_1 = 1|u_0,u_1,u_3,Y\right) = \frac{P(n_1=1,u_0,u_1,u_3,Y)}{P(u_0,u_1,u_3,Y)} \\
&= \frac{P(n_1=1,u_0,u_1,u_3,Y)}{P(n_1=1,u_0,u_1,u_3,Y)+P(n_1=0,u_0,u_1,u_3,Y)} \\
&\approx \frac{P(n_1=1,u_0,Y)P(n_1=1,u_1,Y)P(n_1=1,u_3,Y)}{P(n_1=1,u_0,Y)P(n_1=1,u_1,Y)P(n_1=1,u_3,Y)+(1-P(n_1=1,u_0,Y))(1-P(n_1=1,u_1,Y))(1-P(n_1=1,u_3,Y))}
\end{aligned}$$

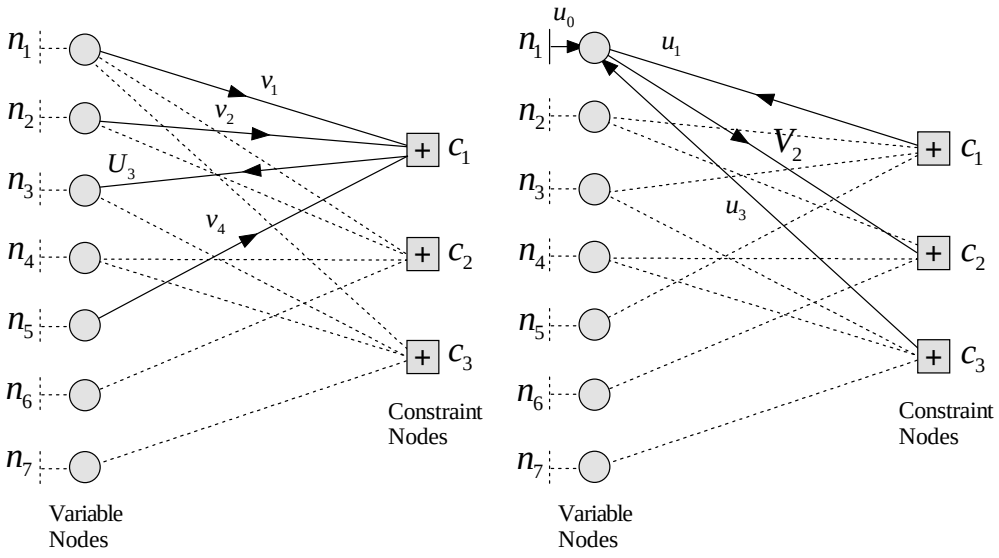
The last approximation is due to the fact that the independence assumption does not hold for graphs with cycles. Likelihood ratios applied here yield:

$$\begin{aligned}
\lambda_{V_2} &= \lambda_{c_2,n_1} \\
&= \frac{1-p_{c_2,n_1}}{p_{c_2,n_1}} \\
&\approx \frac{(1-P(n_1=1,u_0,Y))(1-P(n_1=1,u_1,Y))(1-P(n_1=1,u_3,Y))}{P(n_1=1,u_0,Y)P(n_1=1,u_1,Y)P(n_1=1,u_3,Y)} \\
&= \lambda_{u_0}\lambda_{u_2}\lambda_{u_3}
\end{aligned}$$

Clearly the generic log-likelihood (LLR) generic equation is:

$$\ln\left(\lambda_{V_j}\right) = \sum_{i=1,i\neq j}^{d_v}\ln\left(\lambda_{u_i}\right). \tag{2.9}$$

In the following sections the term *messages* will refer, unless noted otherwise, to LLRs.



(a) Constraint Node Update (b) Variable Node Update

Figure 2.1: Messages generated at constraint and variable nodes

2.1.2 Approximate-Min* BP decoding

In this section alternative technique for processing constraint node updates is presented that is suitable for hardware implementation of LDPC decoding. The method is stated and is then followed by a description of the steps taken to achieve its derivation.

We begin on the variable node (left hand) side of the bi-partite graph. Here, u messages arrive and V messages depart. At the constraint node (right hand) side of the graph v messages arrive and U messages depart. All four message types are LLRs such as (2.1) and (2.2).

For a given check node of degree d_c, there are d_c variable node messages (v_j's) arriving to it. The notation U^{APP} denotes a constraint node outgoing message determined by *all* arriving variable messages (and is defined to be the constraint node *a-posteriori probability*). The notation $U^{APP} \backslash v_j$ denotes the outgoing constraint message determined by all incoming edges with the exception of edge v_j. Note that message v_j represents *intrinsic* information that is left purposefully absent in the

extrinsic message $U^{APP}\backslash v_j$.

Algorithm 1 Approximate Min* B.P

1: Initialize $\left\{ v_{\min} = \min_{j=1..d_c} \left(|v_j| \right), \delta^0 = \infty \right\}$
2: **for** $k = 1 \cdots d_c$ **do**
3: **if** $k \neq min$ **then**
4: $\delta^k = \left| \Lambda^{BP^*} \left(\delta^{k-1}, v_k \right) \right|$
5: **else**
6: $\delta^k = \delta^{k-1}$
7: **end if**
8: **end for**
9: $\delta^{APP\backslash v_{\min}} = \delta^{d_c}$
10: $\delta^{APP} = \left| \Lambda^{BP^*} \left(\delta^{APP\backslash v_{\min}}, v_{\min} \right) \right|$
11: $\sigma^{APP} = \prod_{j=1}^{d_c} \operatorname{sgn}(v_j)$

On the constraint side of the graph, perform on each node, the computations described in Algorithm 1. Constraint message updates then follow from,

$$\begin{aligned} \underset{j=\{1..d_c\}\backslash \min}{U_j} &= \operatorname{sgn}(v_j) \operatorname{sgn}(\sigma^{APP}) \delta^{APP} \\ U_{\min} &= \operatorname{sgn}(v_{\min}) \operatorname{sgn}(\sigma^{APP}) \delta^{APP\backslash v_{\min}} \end{aligned} \tag{2.10}$$

where δ^k is a storage variable, and Λ^{BP^*} will be defined shortly.

From 2.9 we observe that variable node updating follows a linear process and is well suited to hardware adaptation in its unaltered form,

$$V^{APP} = \sum_{j=0}^{d_v} u_j \qquad \underset{j=1..d_v}{V_j} = V^{APP} - u_j. \tag{2.11}$$

where d_v is the variable node degree.

Derivation of the Approximate-BP constraint node update begins with the so called 'Log-Hyperbolic-Tangent' definition of BP constraint updating. In the

equation below, sign and magnitude are separable since the sign of LnTanh(x) is determined by the sign of x.

$$U^{APP} = \left[\prod_{j=1}^{d_c} \mathrm{sgn}(v_j)\right] \ln\left(\frac{1 + e^{-\sum_{j=1}^{d_c} \ln\left(\frac{1+e^{-|v_j|}}{1-e^{-|v_j|}}\right)}}{1 - e^{-\sum_{j=1}^{d_c} \ln\left(\frac{1+e^{-|v_j|}}{1-e^{-|v_j|}}\right)}}\right) \tag{2.12}$$

This equation is highly non-linear and requires substantial simplification before mapping to hardware. To begin, the above computation can be performed by first considering the *inner* recursion in (2.12),

$$\sum_{j=1}^{d_c} \ln\left(\frac{1+e^{-|v_j|}}{1-e^{-|v_j|}}\right). \tag{2.13}$$

A total of d_c table look-ups to the function $\Lambda^{lnBP} = \ln\left(\frac{1+e^{-|v_j|}}{1-e^{-|v_j|}}\right)$ followed by $d_c - 1$ additions complete the computation in (2.13). Furthermore, the linearity of the inner recursion allows intrinsic variable values to be 'backed-out' of the total sum before d_c *outer* recursions are used to form the d_c extrinsic outputs. To summarize, computation of all d_c extrinsic values (in (2.12)) follow from d_c table look-ups, $d_c - 1$ additions, d_c subtractions, and a final d_c table look-ups. The cost of computing the extrinsic sign entails $d_c - 1$ exclusive-or operations to form the APP extrinsic sign, followed by d_c incremental exclusive-or operations to back-out the appropriate intrinsic sign to form each final extrinsic sign.

Variable node computation (2.11) is more straightforward. However, a possible alternative to (2.11) is given in [20] where it is noted that codes lacking low degree variable nodes experience little performance loss due to the replacement of V_j with V^{APP}. However, codes that maximize rate for a given noise variance in an AWGN channel generally have a large fraction of degree-2 and degree-3 variable nodes [10]. Low degree nodes are substantially influenced by any edge input and V^{APP} may differ significantly from corresponding properly computed extrinsic values. We have found experimentally that using V^{APP} alone to decode capacity approaching

codes degrades performance by one dB of SNR or more.

We continue toward the definition of an alternative constraint update recursion by rearranging (2.12) for the $d_c = 2$ case,

$$\text{sgn}(v_1)\text{sgn}(v_2) \ln\left(\frac{1+e^{|v_1|+|v_2|}}{e^{|v_1|}+e^{|v_2|}}\right) = \ln\left(\frac{1+e^{v_1+v_2}}{e^{v_1}+e^{v_2}}\right). \tag{2.14}$$

Two applications of the Jacobian logarithmic identity ($\ln(e^a+e^b) = \max(a,b) + \ln\left(1+e^{-|a-b|}\right)$) [21] result in the Min* recursion that is discussed in the rest of the chapter,

$$\Lambda^{BP^*}(v_1,v_2) = \text{sgn}(v_1)\text{sgn}(v_2) \begin{pmatrix} \min(|v_1|,|v_2|) \\ +\ln\left(1+e^{-(|v_1|+|v_2|)}\right) \\ -\ln\left(1+e^{-||v_1|-|v_2||}\right) \end{pmatrix}. \tag{2.15}$$

Note that (2.15) is not an approximation. It is easy to show that $d_c - 1$ recursions on Λ^{BP^*} yield exactly U^{APP} in equation (2.12). Furthermore, the function $\ln\left(1+e^{-|x|}\right)$ ranges over $(ln(2), 0)$ which is substantially more manageable than the range of the function Λ^{LnBP}, $Range\left(\ln\left(\frac{1+e^{-|x|}}{1-e^{-|x|}}\right)\right) = (\infty, 0)$ from a numerical representation point of view. However, the non-linearity of the recursion (2.15) implies that updating all extrinsic information at a constraint node requires $d_c(d_c-1)$ calls to Λ^{BP^*}. This rapidly becomes more complex than the $2d_c$ look-up operations (augmented with $2d_c - 1$ additions) required to compute all extrinsic magnitudes based on the form in (2.12). Again, in this earlier case intrinsic values can be 'backed-out' of a single APP value to produce extrinsic values.

Instead of using the recursion in (2.15) to implement Full-BP we propose that this recursion be used to implement an approximate BP algorithm to be referred to as *Approximate-Min*-BP* (A-Min*-BP). The algorithm works by computing the proper extrinsic value for the minimum magnitude (least reliable) incoming constraint edge and assigning the U^{APP} magnitude in conjunction with the proper extrinsic sign to all other edges.

To provide intuition as to why this hybrid algorithm yields good performance, note first that a constraint node represents a single linear equation and has a known

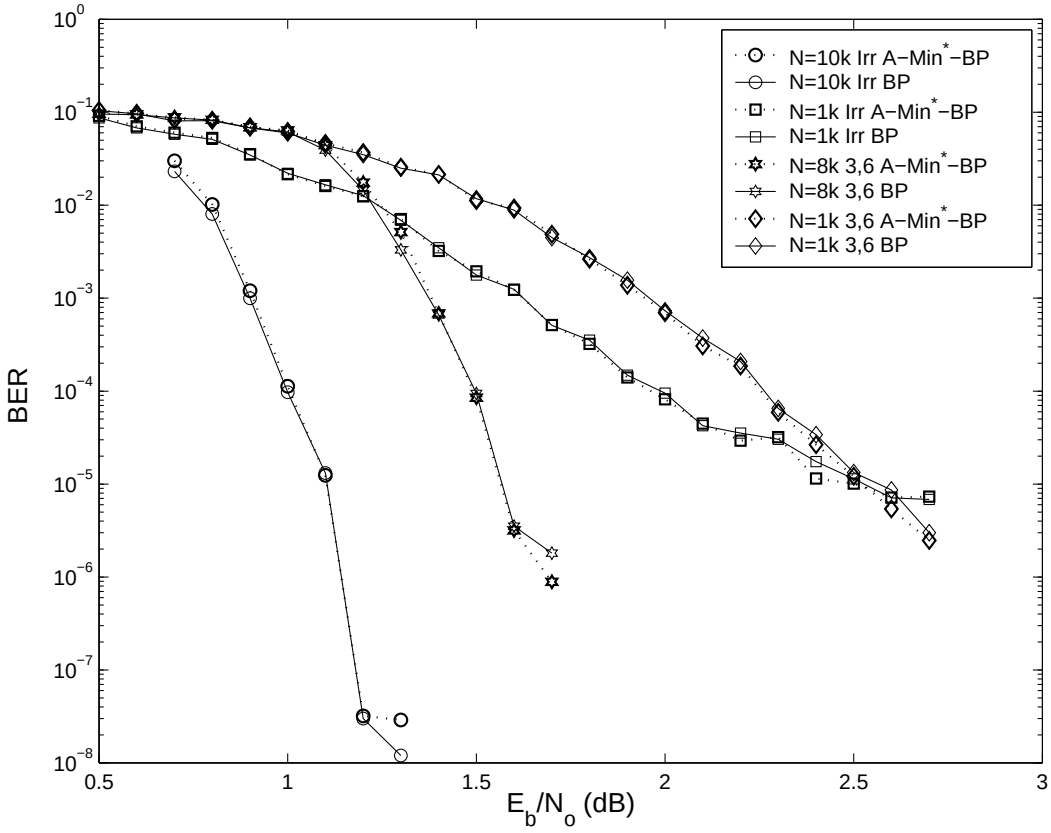


Figure 2.2: A-Min*-BP decoding compared to Full-BP decoding for four different rate 1/2 LDPC codes. Length 10k irregular max left degree 20, length 1k irregular max left degree 9, length 8k regular (3,6), length 1k regular (3,6). The proposed algorithm incurs negligible performance loss. Note that rate 1/2 BPSK constrained capacity is 0.18 dB (E_b/N_o).

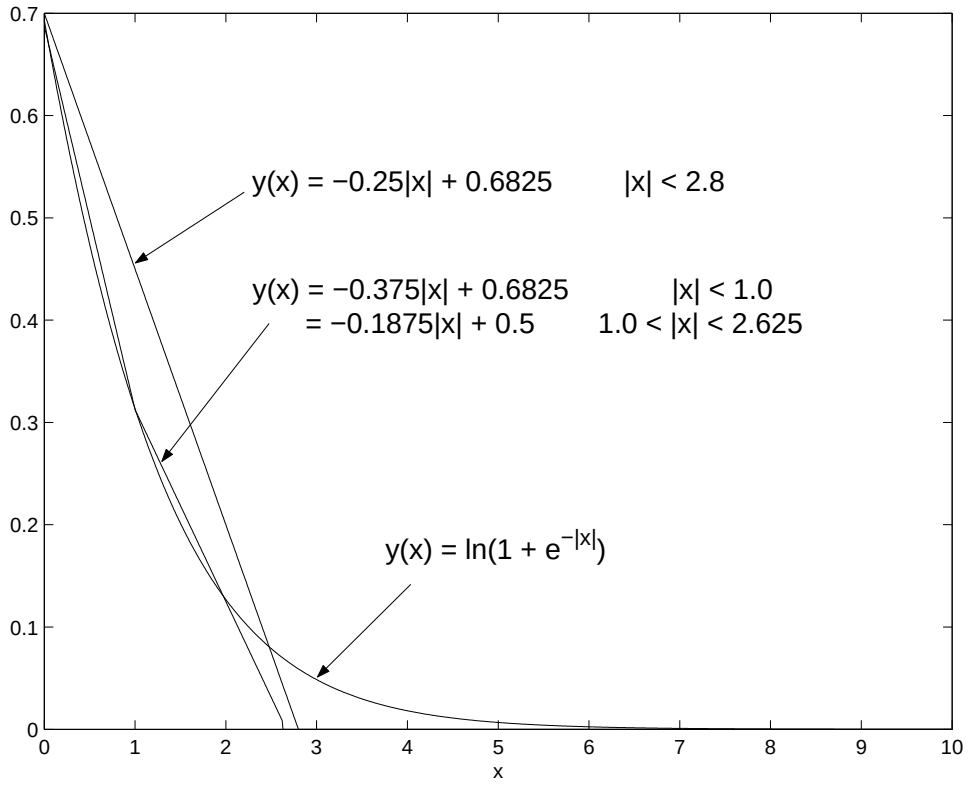


Figure 2.3: Non-linear function (Λ^{BP^*}) at the core of proposed algorithm and single / double line approximations to the function that can easily be implemented in combinatorial logic.

'solution' if no more than one input variable is unknown. Consider the following two scenarios. First, if a constraint has more than one unreliable input, then all extrinsic outputs are unreliable. Second, if a constraint has exactly one unreliable input, then this unknown input can be solved for based on the extrinsic reliability provided by the 'known' variables. In this second case all other extrinsic updates are unreliable due to the contribution of the unreliable input. The approximation in the suggested algorithm assigns less accurate magnitudes to would-be unreliable extrinsics, but for the least reliable input preserves exactly the extrinsic estimate that would be produced by Full-BP.

We next show that U^{APP} always underestimates extrinsics. Here the notation U_{mn} represents the extrinsic information that originates at constraint node m and excludes information from variable node n. Rearrangement of (2.12) (with standard

	Full-BP	A-Min*-BP	Offset-Min-BP
Table LookUps	$2d_c$	$d_c - 1$	0
Comparisons	0	$d_c - 1$	$2d_c - 3$
Additions	$2d_c - 1$	0	d_c
XORs	$2d_c - 1$	$2d_c - 1$	$2d_c - 1$
Tot Table Lookups	80000	35000	0
Tot Comparisons	0	35000	65000
Tot Additions	75000	0	40000
Tot XORs	75000	75000	75000
Tot Ops	230,000	145,000	180,000
Performance	Reference	NoLoss	0.1dB Loss

Table 2.1: Complexity comparison for three constraint update techniques. Full-BP and A-Min*-BP have essentially the same performance. The simplest technique, Offset-Min-BP, experiences about a 0.1dB loss [20]. Numerical values are shown for a rate 1/2 code with: $n - k = 5000$, and average right degree d_c=8.

intrinsic/extrinsic notation included [22]) yields the following,

$$\left|U^{APP}\right| = \ln \frac{1 + \prod\limits_{n' \in N(m)} \frac{1 - e^{-|v_{mn'}|}}{1 + e^{-|v_{mn'}|}}}{1 - \prod\limits_{n' \in N(m)} \frac{1 - e^{-|v_{mn'}|}}{1 + e^{-|v_{mn'}|}}} \tag{2.16}$$

$$\frac{1 - e^{-\left|U^{APP}\right|}}{1 + e^{-|U^{APP}|}} = \left(\prod_{n' \in N(m) \backslash n} \frac{1 - e^{-|v_{mn'}|}}{1 + e^{-|v_{mn'}|}} \right) \left(\frac{1 - e^{-|v_{mn}|}}{1 + e^{-|v_{mn}|}} \right). \tag{2.17}$$

Note first that the function $g(x) = \dfrac{1 - e^{-|x|}}{1 + e^{-|x|}}$ (a product of which comprises the right hand side of (2.17)) ranges over $(0, 1)$ and is non-decreasing in the magnitude of x. The first (parenthesized) term on the right hand side of (2.17) equals the extrinsic value U_{mn} under the operator $g(\cdot)$, i.e. $g(U_{mn})$. The second term scales this value by the intrinsic reliability $g(v_{mn})$. Hence, the monotonicity and range of $g(x)$ ensure that $\left|U^{APP}\right| < |U_{mn}|$. We provide the inverse function, $g^{-1}(x) = \ln \dfrac{1 + x}{1 - x}$, for reference.

Underestimation in A-Min*-BP is curtailed by the fact that the minimum reliability $g(v_{\min})$ dominates the overall product that forms U^{APP}. This term would

have also been included in the outgoing extrinsic calculations used by Full-BP for all but the least reliable incoming edge. The outgoing reliability of the minimum incoming edge incurs no degradation due to underestimation since the proper extrinsic value is explicitly calculated. Outgoing messages to highly reliable incoming edges suffer little from underestimation since their corresponding intrinsic $g(v_{mn})$ values are close to one. The worst case underestimation occurs when two edges 'tie' for the lowest level of reliability. In this instance the dominant term in (2.17) is squared. An improved version of A-Min*-BP would calculate exact extrinsics for the two smallest incoming reliabilities. However, the results in Fig.2.2, where the algorithm (using floating point precision) is compared against Full-BP (using floating point precision) for short and medium length regular and irregular codes, indicate that explicit extrinsic calculation for only the minimum incoming edge is sufficient to yield performance that is essentially indistinguishable from that of Full-BP.

The proposed algorithm is similar to the Offset-Min-BP algorithm of [20] where the authors introduce a scaling factor to reduce the magnitude of extrinsic estimates produced by Min-BP. The Min-BP algorithm finds the magnitude of the two least reliable edges arriving at a given constraint node. The magnitude of the least reliable edge is assigned to all edges except the edge from which the least reliable magnitude came (which is assigned the second least reliable magnitude). For all outgoing edges, the proper extrinsic sign is calculated. As explained in [22] these outgoing magnitudes overestimate the proper extrinsic magnitudes because the constraint node update equation follows a product rule (2.17) where each term lies in the range $(0, 1)$. The Min-BP approximation omits all but one term in this product. To reduce the overestimation, an offset (or scaling factor) is introduced to decrease the magnitude of outgoing reliabilities. The authors in [23] use density evolution to optimize the offset for a given degree distribution and SNR. The optimization is sensitive to degree sequence selection and also exhibits SNR sensitivity to a lesser extent. Nevertheless, using optimized parameters, performance within 0.1 dB of Full-BP performance is possible.

By way of comparison, A-Min*-BP improves performance over Min-BP be-

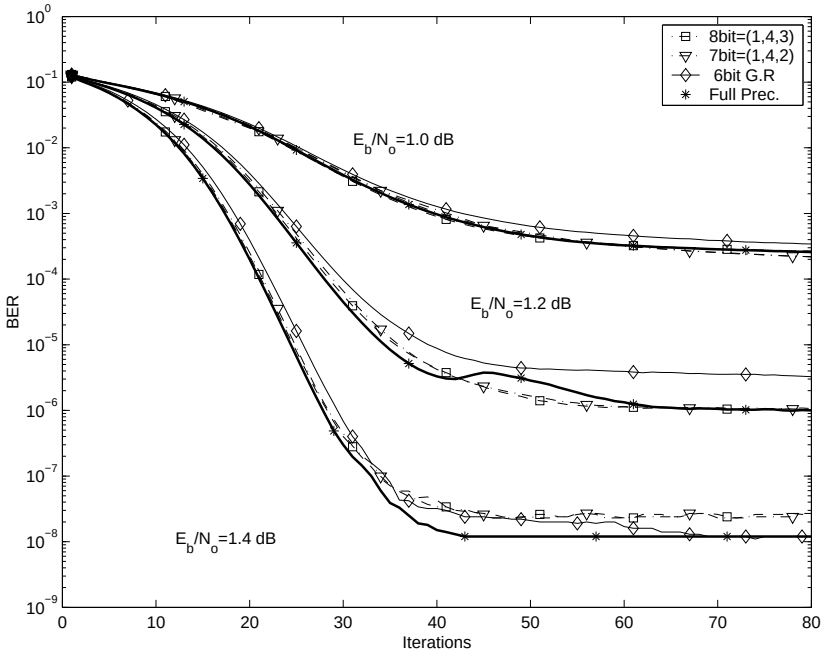
cause the amount by which U^{APP} underestimates a given extrinsic is less than the amount by which Min-BP overestimates the same extrinsic. Specifically, the former underestimates due to the inclusion of one extra term in the constraint node product while the latter overestimates due to the exclusion of all but one term in the product. A direct comparison to Offset-Min-BP is more difficult. However, a simple observation is that in comparison to Offset-Min-BP, A-Min*-BP is essentially 'self-tuning'.

The range and shape of the non-linear portion (Λ^{BP^*}) of the A-Min*-BP computation are well approximated using a single, or at most a 2-line, piecewise linear fit, as shown in Fig. 2.3. All of the fixed precision numerical results to be presented in section 2.2 use the 2-line approximation (as do the floating point results in Fig. 2.2). Hence, the entire constraint node update is implemented using only shift and add computations, no look-ups to tables of non-linear function values are actually required.

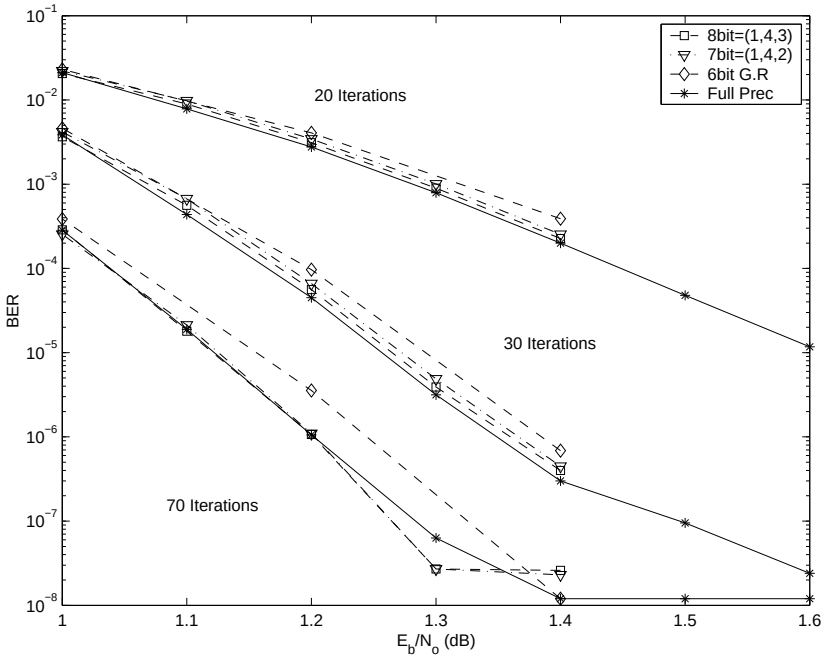
The cost of constraint node updating for Full-BP (implemented using (2.12)), A-Min*-BP, and Offset-Min-BP are given in Table 2.1. The latter two algorithms have similar cost with the exception that $d_c - 1$ table look-up operations in A-Min*-BP are replaced with d_c additions in Offset-Min-BP (for offset adjustment). Note that use of a table is assumed for the representation of Λ^{LnBP}. While Λ^{BP^*} is well approximated using a two line piecewise fit employing power of 2 based coefficients. Variable node updating occurs via (2.11) for all three algorithms.

2.2 Numerical Implementation

Minimum complexity implementation of the A-Min*-BP algorithm necessitates simulation of finite word length effects on edge metric storage (which dominates design complexity). Quantization selection consists of determining a total number of bits as well as the distribution of these bits between the integer and fractional parts *(I,F)* of the numerical representation. The primary objective is minimization of the total number of bits with the constraint that only a small performance degradation in the waterfall and error-floor BER regions is incurred. Quantization



(a) BER vs Iterations, for different fixed E_b/N_o.



(b) BER Vs. E_b/N_o, for different fixed numbers of iterations.

Figure 2.4: Performance of Approx.Min* BP

saturation levels ($Sat = 2^I$) that are too small cause the decoder to exhibit premature error-floor behavior. We have not analytically characterized the mechanism by which this occurs. However, the following provides a rule of thumb for the saturation level,

$$Sat = -\ln(p) \approx \ln\left(\frac{1-p}{p}\right) \qquad \text{where } p = e^{-Sat}.$$

This allows literal Log-Likelihood Ratio (LLR) representation of error probabilities that are as small as p. In practice, this rule seems to allow the error-floor to extend to a level that is about one order of magnitude lower than p.

In the results that follow, simple uniform quantization has been employed, where the step size is given by 2^{-F}. To begin, Fig. 2.4(a) shows that low SNR performance is less sensitive to quantization than high SNR performance. A small but noticeable degradation occurs when 2 rather than 3 fractional bits are used to store edge metrics and 4 integer bits are used in both cases. In summary, 7 bits of precision (Sign, 4 Integer, 2 Fractional) are adequate for the representation of observation and edge metric storage in association with the considered code.

When power of 2 based quantization is used, the negative and positive saturation levels follow $[-2^{I-1}, 2^{I-1} - 2^{-F}]$. An alternative approach arbitrarily sets this range between a maximum and a minimum threshold and sets the step size equal to $s = 2 * MaxRange/2^{TotalBits}$. This approach to quantization is more general than the previous since the step size is not limited to powers of 2. We have found that in the low SNR regime, smaller quantization ranges are adequate, but the optimal step size remains similar to that needed at higher SNRs. Thus, operation at lower SNRs requires fewer overall bits given the general range approach to quantization. For example for $E_b/N_o = 1.0$dB, when $MaxRange = 10$ and a total of 6 bits were used, no performance degradation was observed. For higher SNR values, $MaxRange = 16$ was the best choice. This agrees with the results obtained

using binary quantization with $(I, F) = (4, 2)$. The performance of this quantizer is described in Fig.2.4(a) by the curve labeled '6bit G.R.' (or 6 bit general range) where in this case the range is set equal to (-10,10)@1.0dB;(-12,12)@1.2dB;(-16, 16)@1.4dB and a total of 6 bits (1 sign, 5 quant-bits) is used. Hence in this case the general range quantizer is equivalent to the (1,4,1) power of 2 quantizer at high SNR. At lower SNRs, the best case range was smaller than (-16,16) such that general range quantization offers an added degree of freedom in precision allocation that is useful in the context of LDPC decoding.

2.3 LDPC Codec FPGA Implementation

We have implemented the above constraint update technique along with many other necessary functions in order to create a high throughput Monte Carlo simulation for arbitrary LDPC codes. The design runs on a Xilinx Virtex-II XC2V4000-6 FPGA evaluation board from Nallatech systems running at 133 MHz and is interfaced to a PC via a JAVA API. A block diagram is provided in Fig.2.5. The Gaussian noise generator developed by the authors in [24] is instantiated next to the decoder so as to avoid a noise generation bottleneck that exists when noise is generated in the PC and passed to the decoder. This block directly impacts the overall value of the system as a Monte Carlo simulator for error-floor testing as good noise quality at high SNR (tails of the Gaussian) is essential. Since the LDPC decoding process is iterative and the number of required iterations is non-deterministic, a flow control buffer can be used to greatly increase the throughput of the overall system. When compared to typical workstation throughputs of 10 - 20 kbps, Table 2.2 shows that even a single (non-parallel) slice decoder yields a 20x - 40x throughput improvement.

Through the use of JAVA as an soft interface to the board, we have been able to facilitate the initiation and monitoring of simulations from remote locations. Researchers around the world are able to upload their own LDPC codes for testing on the "UCLA Monte Carlo System".

Iterations	Length 1000 Irregular LDPC	Length 800 Irregular LDPC
1	2.19 [Mbps]	2.20 [Mbps]
5	665 [Kbps]	684 [Kbps]
10	355 [Kbps]	367 [Kbps]
15	243 [Kbps]	252 [Kbps]
20	184 [Kbps]	191 [Kbps]

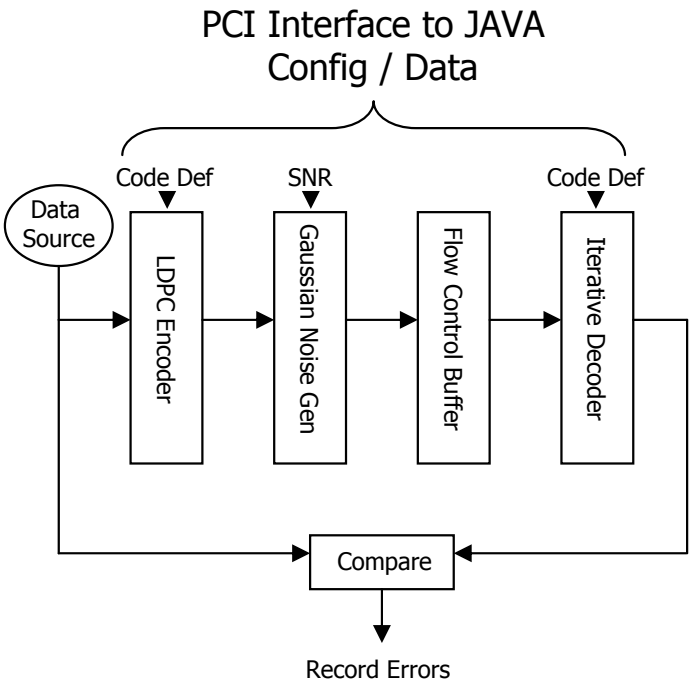


Figure 2.5: Architecture block diagram.

2.4 Summary

A reduced complexity decoding algorithm that suffers little or no performance loss has been developed and is justified both theoretically and experimentally. Finite word lengths have been carefully considered and 6 to 7 bits of precision have been shown to be adequate for a highly complex (a length 10,000 $d_{lmax} = 20$ irregular LDPC) code to achieve an error floor that is code rather than implementation limited.

Chapter 3

Iterative Pilotless Timing Recovery via LDPC Code Constraint Feedback

In traditional receiver architectures, symbol acquisition and tracking are performed using phase lock techniques that are independent of the channel code decoding process. In burst reception scenarios, bandwidth inefficient piloting must often be embedded in a transmission in order to accelerate acquisition and or to aid symbol time tracking at low signal-to-noise ratios. In this chapter we show that outputs from the constraint node side of a bi-partite decoding graph can be used to direct estimation of symbol frequency and phase in a pilotless LDPC coded transmission. We focus on the problem of a baseband symbol frequency and/or time offset between transmitter and receiver and describe a method capable of handling very large offsets with complexity that grows linearly with offset size. Relatively large timing offsets may occur in receivers that need to determine baud rates autonomously while finer offsets are often due to crystal oscillator mismatches and Doppler-induced frequency shifts. Combining the constraint node observations with a properly calibrated phase locked loop allows successful tracking of a constant time delay, a frequency offset and a random phase walk.

3.1 Overview

Recent advances in iteratively decoded channel codes such as LDPC codes make it possible to operate at capacity-approaching SNRs [4]. This in turn places more stringent requirements on the timing recovery portions of receivers, which must successfully acquire and track symbols at these lower SNRs. Acquisition and tracking have traditionally been performed upstream of, and independently of, channel decoding, and typically utilize a phase locked loop (PLL). However, the LDPC decoding process provides information that can be fed back to the timing recovery circuit to enable significantly improved performance relative to a system in which no such feedback is present. This chapter describes the use of information from constraint nodes of LDPC decoders for symbol synchronization, allowing operation at very low SNRs where other timing synchronization methods typically fail. In this chapter we also assume perfect carrier information and therefore work with a baseband model. Joint timing and carrier parameter estimation is addressed in Chapter 5.

LDPC codes are commonly represented using a bipartite graph containing two sets of nodes. In the graph corresponding to an (n, k) code, the n variable nodes correspond to the codeword symbols and the $n - k$ constraint nodes represent the constraints that the code places on the variable nodes in order for them to form a valid codeword. The decoding procedure involves iterative computation of values associated with these nodes, as described in Chapter 2. A constraint node represents a parity-check equation using a set of variable nodes as inputs. A valid decoded codeword is obtained if all parity-check equations are satisfied. After each iteration, the metrics associated with each constraint node can be evaluated to determine the status of the associated parity check. Normally, this information is utilized only within the LDPC decoding process to assess the convergence behavior of the iterative processing. However, we show here that it has value in the timing recovery process as well. The number and nature of satisfied constraint node equations provides a measure not only of code convergence but also of the underlying accuracy of the timing estimates used in acquiring the sampled data input to the

LDPC decoder.

The basic idea of coupling LDPC decoding with timing recovery has been explored in the past. For example, Barry *et al.* [25] and Liu *et al.* [26] have described modifying a decision-directed Mueller-Müller timing error detector (M&M TED) [27] by using feedback from decoded LDPC or turbo information to produce better timing estimations. This approach gives significantly improved performance over a system in which the data provided to the TED is derived directly from channel observations, before any iterative decoding is done. Sun and Valenti [28] investigate the use of multiple SNR estimators coupled with a Turbo decoder. Their BER performances are very close to the perfect timing case, but their model is able to track constant time delays only.

The previous work in this area has focused on the use of output codewords produced as the iterations progress. By contrast, we exploit the information available from the metrics computed for the constraint nodes of an LDPC code during the decoding process. In addition, we use a waveform model that more directly captures the distortions induced by relative transmitter/receiver motion and other receiver-side timing errors. The proposed solution is suitable not only for wired communications where the transmitter and receiver are at fixed positions. It is also suitable for mobile scenarios where relative motion exists between the transmitter and receiver. This occurs in a wide range of applications including deep space and satellite communications as well as general terrestrial mobile wireless environments.

The rest of this chapter is organized as follows. In Section 3.2 we describe the transmitter and timing offset model. In Section 3.3 the receiver architecture is presented. Simulation results are presented in Section 3.4. Finally concluding remarks are made in Section 3.5.

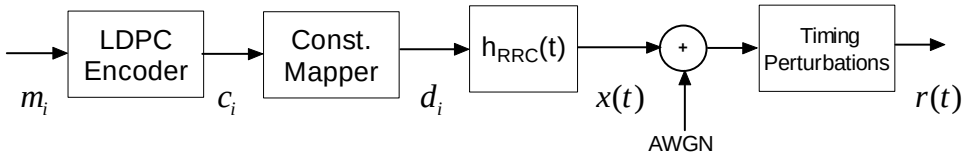


Figure 3.1: Baseband transmitter and channel model. The waveform is constructed as a series of superimposed root raised cosine pulses, after which noise is added and timing offsets are introduced.

3.2 Baseband Transmitter and Timing Offset Model

On the transmitter side, message symbols m_i are fed into the LDPC encoder as shown in Fig. 3.2. Note that the modulation and demodulation blocks from Fig. 1.1 have been removed and a baseband model is assumed, due to a perfect carrier information assumption. For a (n, k) LDPC code, for every k input bits the encoder generates a codeword of n bits. We consider a BPSK signal comprised of a series of n root raised-cosine pulses $h_{RRC}(t)$, transmitted at integer multiples of a symbol interval T and scaled by d_i, where $d_i \in \{\pm 1\}$ is the BPSK modulated value of the i^{th} symbol,

$$x(t) = \sum_{i=0}^{n-1} d_i h_{RRC}\left(t - iT\right). \tag{3.1}$$

In a system with no timing errors, $x(t)$ would be sampled at the receiver at multiples of a sampling interval T_s. This produces a sampled received sequence $r[k] = x(kT_s) + n(kT_s)$, where $n(kT_s)$ is AWGN noise introduced by the channel. The root-raised cosine pulses that form $x(t)$ are designed to have zero crossings at multiples of T. Therefore when no timing distortion occurs in the channel, the received waveform is ISI-free when sampled at multiples of the symbol interval as depicted in Fig. 3.2. When timing errors are present, the assumed time reference for the k^{th} sample at the receiver differs from the corresponding time reference at the transmitter according to some function $\tau[k]$, giving the value of the k^{th}sample $r[k] = x(kT_s + \tau[k]) + n(kT_s)$. Substituting in (3.1) gives,

$$r[k] = \sum_{i=0}^{n-1} d_i h_{RRC} \left(kT_s + \tau[k] - iT \right) + n(kT_s). \tag{3.2}$$

We consider the following timing error modalities.

3.2.1 Constant time offset

All pulses are affected by the same constant delay with respect to their ideal sampling time:

$$\tau[k] = D. \tag{3.3}$$

3.2.2 Random walk

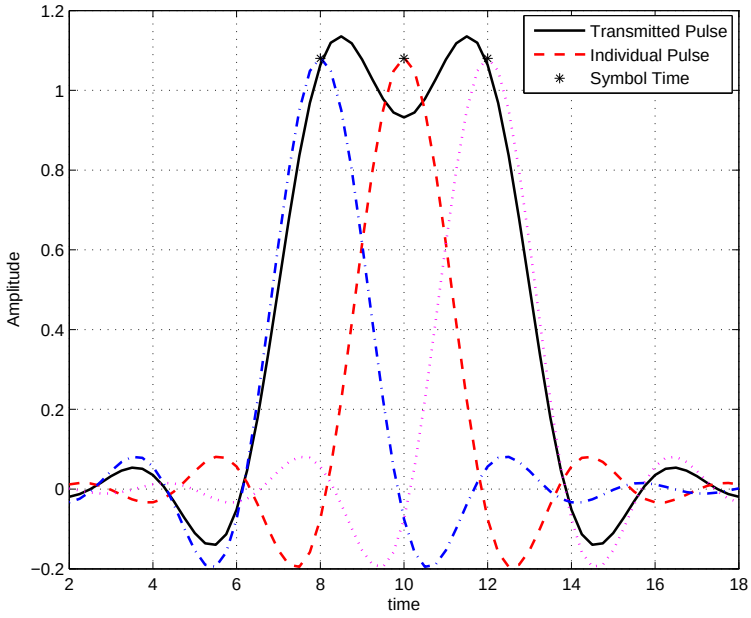
This models an accumulation process where the timing error at each sampling instant is given by the previous error further perturbed according to a zero mean Gaussian random variable with variance σ_d^2, denoted $\mathcal{N}(0, \sigma_d^2)$:

$$\tau[k] = \tau[k-1] + \mathcal{N}(0, \sigma_d^2)\, T_s. \tag{3.4}$$

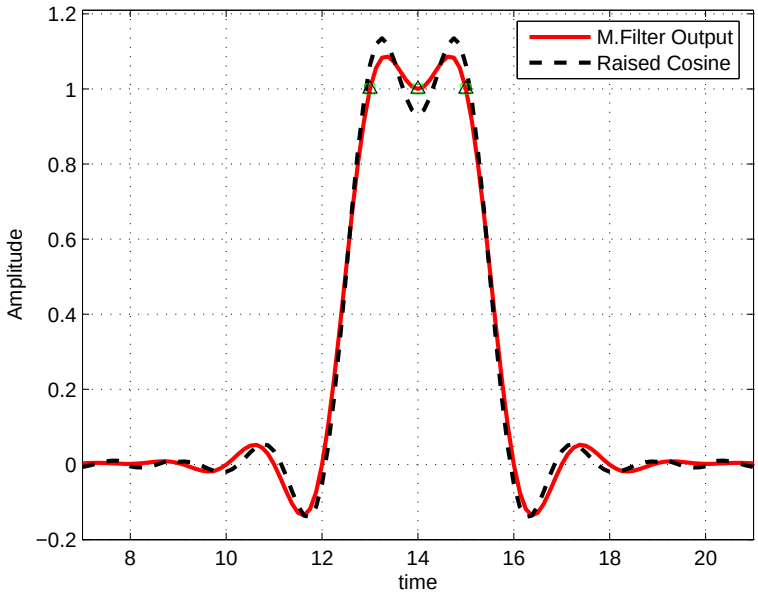
3.2.3 Constant frequency offset

Constant frequency offset can occur when there is a mismatch between the transmitter and receiver clock crystals or when there is a constant Doppler shift. A frequency offset F_{PPM} measured in parts per million, assuming an initial timing offset of zero for the first symbol, is described by:

$$\tau[k] = \tau[k-1] + \frac{F_{PPM}}{10^6}\, T_s. \tag{3.5}$$



(a) Transmitter



(b) Receiver

Figure 3.2: (a) Perfectly aligned superimposed RRC pulses at the transmitter (b) ISI free Raised Cosine waveform at the output of the matched filter.

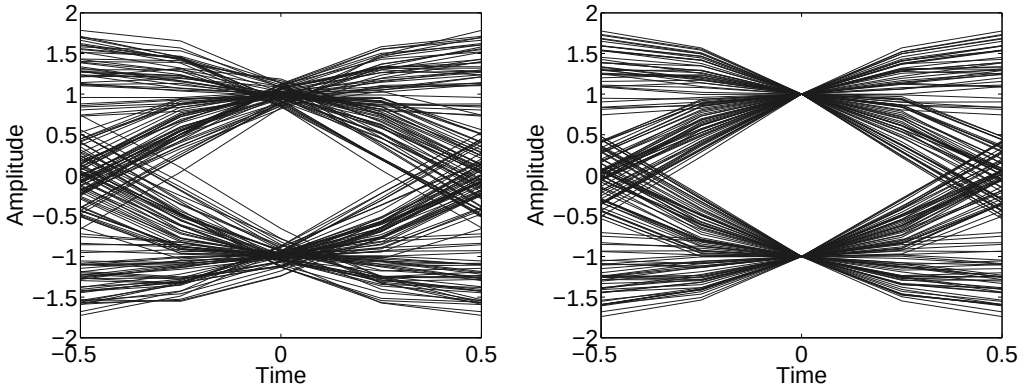
3.2.4 Generalized Doppler shift

The Doppler effect causes apparent change in the frequency of an electromagnetic wave due to relative movement between the transmitter and the receiver. A constant relative motion leads to a constant frequency offset, with the effect described in (3.5) above. However, we consider the more general case in which the relative velocity of the transmitter and receiver can vary with time. The instantaneous Doppler frequency shift f_d is a function of the symbol frequency $1/T$, relative velocity v between the transmitter and receiver, and the speed of light c; i.e. $f_d = v/(c\,T)$. Mapping this into an appropriate perturbation $\tau[k]$ of the sampling time for the k^{th} sample requires properly computing the cumulative impact of the velocity history.

The approach of mapping timing errors into perturbations of the sampling times at the receiver is slightly different from that commonly used in previous treatments of this subject. For example, in [25, 29, 30], timing errors are modelled as perturbations of the symbol (as opposed to sample) times of the individual pulses, leading to a waveform of the form,

$$x(t) = \sum_{i=0}^{n-1} d_i \; h_{RRC}\left(t + \tau_i - iT\right), \tag{3.6}$$

where τ_i is the timing error associated with the i^{th} symbol. However, applying differing timing shifts to individual pulses and then superimposing the shifted pulses gives rise to inter-symbol interference (ISI). Thus, while such a model may be appropriate for a system in which a principal source of timing error is symbol clock jitter at the transmitter, it is not well suited for the present work, which assumes a correctly constructed ISI-free waveform at the transmitter, and aims to model perturbations induced by the relative motion between the receiver and transmitter as well as other receiver-side timing estimation errors. Therefore, here we use a perturbation model where timing distortion is introduced on a sample-wise basis on the superimposed waveform $x(t)$ instead of on a symbol-wise basis prior to superimposing the pulses to construct the waveform. This difference is



(a) Symbol-wise perturbation (b) Sample-wise perturbation

Figure 3.3: Eye diagram shows ISI for both perturbation models.

illustrated in Fig. 3.3 which shows eye diagrams for the two approaches in a noise-free environment. Fig. 3.3(a) shows the eye diagram resulting from a perturbation of the symbol times in accordance with (3.6). In this particular example, 200 symbols are used. The perturbation τ_i is based on a random walk with a symbol-to-symbol standard deviation of 1% of the symbol period T. Fig. 3.3(b) shows the eye diagram when the model of (3.2) is applied (assuming zero noise), again using 200 symbols with an equivalent symbol-to-symbol random walk standard deviation, implemented sample by sample, of 1% of T.

Fig. 3.4 illustrates the relationship between the timing error model and an associated waveform. Fig. 3.4(a) shows an example frequency perturbation profile. In this example, the frequency offset between transmitter and receiver is initially zero, it then reaches a positive constant value, and finally falls again to zero. Fig. 3.4(b) shows an ideal waveform comprised of the sum of 3 raised-cosine functions, prior to the application of any timing perturbations. Note, however, that sample positions have been distributed in accordance to the $\tau[k]$ sequence determined by the perturbation profile. The portions of the waveform transmitted during periods of higher frequency offset appear compressed as viewed by the receiver. As shown in Fig. 3.4(c), the receiver is not aware of the timing distortion and samples the received waveform at evenly spaced intervals resulting in an overall compressed

waveform.

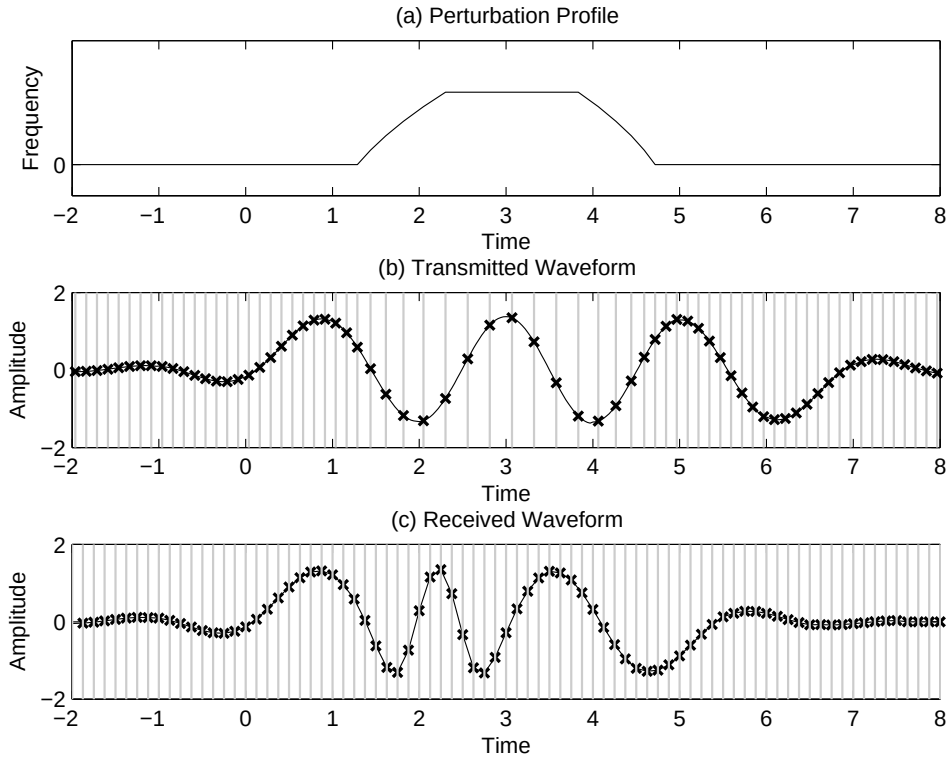


Figure 3.4: Introduction of frequency offsets via time-varying sampling times.

3.3 Receiver Model

Fig. 3.5 illustrates the receiver architecture which exploits feedback from the LDPC decoder to manage timing errors due to constant time delay, frequency offset and random walk. The received waveform is initially sampled at a intervals of T_s and stored into a buffer. T_s is chosen to be sufficiently small to avoid any aliasing. The interpolator computes interpolants at intervals of T_i using linear interpolation, which are then used for the matched filtering process. In this work, we use $T_i = \hat{T}/2$ and $T_s = \hat{T}/4$, where $\hat{T}$ is the receiver's assumption of the transmitter symbol period $\hat{T}$ (i.e. the symbol period that would be seen by the receiver in the absence of any timing perturbations).

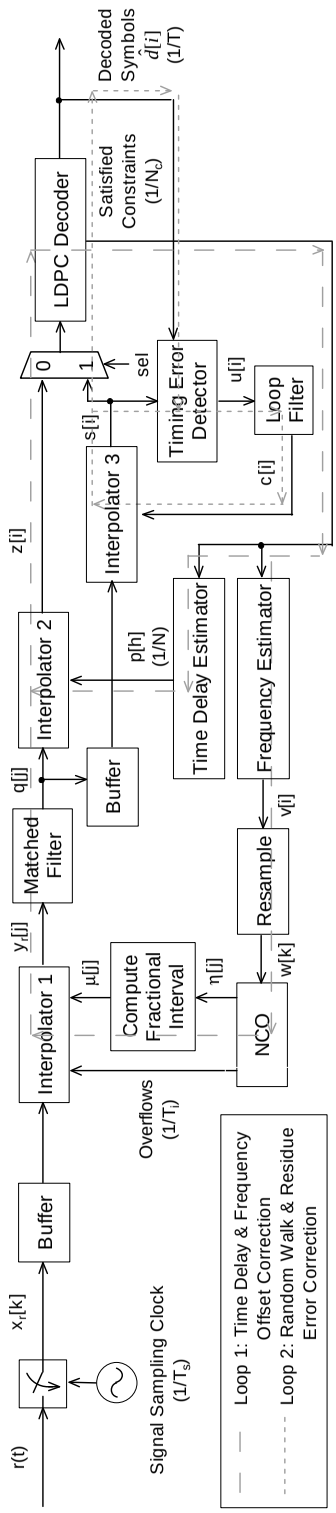


Figure 3.5: Receiver model for timing recovery. N_v and N_c are the number of variable and constraint nodes, respectively. Signals indexed with i, j, and k are at a rate $1/\hat{T}$, $1/T_i$, and $1/T_s$, respectively

Loop 1 in Fig. 3.5 is first executed to recover constant time delays and frequency errors. By using the number of satisfied constraints feedback from the LDPC decoder, the phase estimator and the frequency estimator provide increasingly accurate estimates of the phase and frequency errors. The phase estimator provides the interpolator (after the matched filter) with a time offset, which is used to correct the constant time delay. The frequency estimator provides a frequency control word which is resampled at a rate of $1/T_s$ and fed to the numerically controlled oscillator (NCO). This frequency adjustment loop is based on Gardner's timing processor model [31, 32], which ensures that the interpolator samples the incoming signal at the frequency specified by the control word. The NCO selects the desired base-point $x_r[kT_s]$, while the "compute fractional interval" block computes the appropriate fractional interval between $x_r[kT_s]$ and $x_r[(k+1)T_s]$. Matched filtering is performed by the means of an FIR filter followed by another interpolation step which corrects any remaining phase offset as determined by the time delay estimator. These interpolants (one per period $\hat{T}$) are supplied to the LDPC decoder where the number of satisfied constraints is computed. Note that it is important to have the matched filter inside the loop, since for the case of tracking frequency offsets, it is important to supply the matched filter with the "frequency corrected" samples to maximize its noise filtering performance.

After phase and frequency errors have been compensated in loop 1, loop 2 is utilized to handle random walks and perform conventional LDPC decoding at the same time. It can also correct residual phase and frequency errors remaining after loop 1. A conventional first-order PLL-based circuit with a Mueller-Müller timing error detector (M&M TED) is used. The M&M TED is decision-directed detector, which requires knowledge of the transmitted symbols. Since our M&M TED is not run in trained mode (where the transmitted symbols are known by the receiver), we drive it with the symbols decoded by the LDPC decoder, analogous to the approach used in the recent work of Barry *et al.* [25].

3.3.1 Tracking time delays and frequency offsets

A PLL-based circuit can be effective for tracking small time delays and frequency offsets, but is generally insufficient to track large time delays and frequency offsets, as it will be shown in Section 3.4. In addition, PLLs can suffer from instability problems when the loop gains are not properly calibrated. We address this by utilizing the satisfied-constraint feedback from conventional LDPC decoders. In this case the methods for tracking time delays and frequency offsets follow essentially the same principle, we describe the approach in detail for the frequency offset tracking case.

Fig. 3.6 shows the average percentage of satisfied constraints as a function of frequency estimation errors for different SNRs (E_b/N_0) and numbers of LDPC iterations. The $(n, k) = (1944, 972)$ irregular LDPC code proposed for the IEEE 802.11n standard is used throughout this work [33]. As one would expect, higher SNRs, more LDPC iterations, and smaller estimation errors lead to a higher percentage of satisfied constraints. More interestingly, the figure indicates the rate of falloff as the estimation error increases, and shows that the best frequency error discrimination occurs for errors within approximately 200 ppm (i.e. 0.02%). This information is used in determining the step size to use in the frequency offset search. Fig. 3.6 also indicates the benefits (in increasing the percentage of satisfied constraints) of increasing the number of iterations.

Instead of simply sweeping frequencies over the expected frequency offset range, we employ a windowed search method. We present two iterative methods with different convergence properties. The first method is explained in Algorithm 2. For each "search iteration", multiple LDPC decoding iterations are performed for each candidate frequency error at evenly spaced steps in the window. For the first search iteration, the frequency offset estimate giving the most satisfied constraints is found. The search window is recentered to this frequency offset, and then the search window and the step size are reduced by a factor of c_1 and c_2, respectively, and the next search iteration is performed. The *ComputeObs* function performs the following tasks: it first drives the NCO with the control word for the

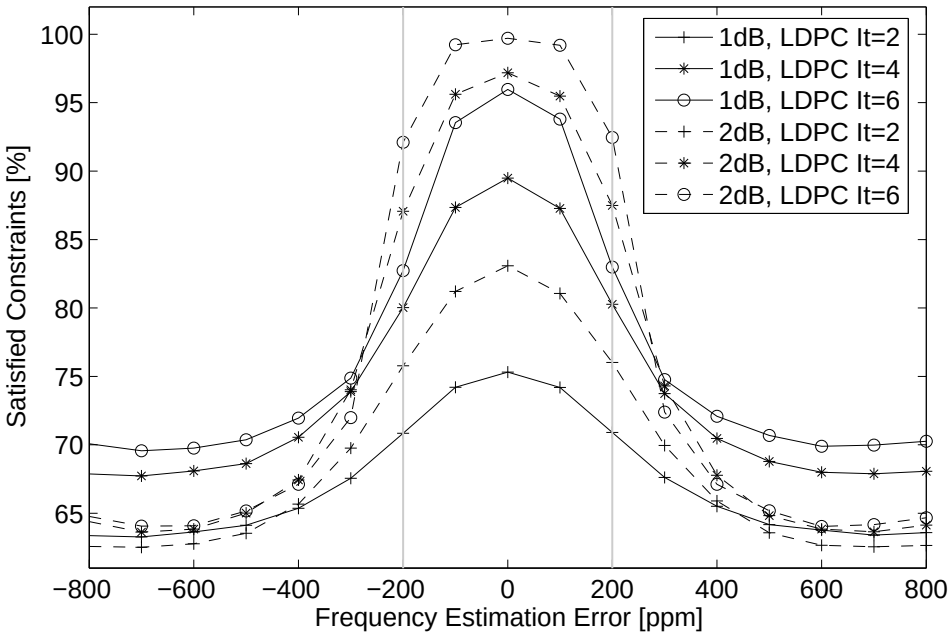


Figure 3.6: Percentage of satisfied constraints as a function of frequency estimation error. Curves for 2, 4, and 6 LDPC iterations are shown for E_b/N_0 of 1 and 2 dB. An average of 500 trials are used for each data point.

frequency estimate, then performs interpolation on the incoming waveform, and then conducts matched filtering followed by a second interpolation step to extract the symbols $z[iT]$. These symbols are fed to the LDPC decoder where the number of satisfied constraints are computed.

There are seven key parameters: The number of search iterations, the number of LDPC iterations per estimate, the minimum and maximum frequency error, the initial step size, and the reduction factors c_1 and c_2. When c_1 is large, the search window shrinks at a fast rate, but this increases the risk of causing the true frequency error to fall outside the search window. This can occur because while Fig. 3.6 shows the average percentage of satisfied constraints, for any specific realization the percentage may not fall monotonically with increasing frequency error. Shrinking the search window less aggressively gives more opportunity to leverage the average behavior as opposed to the behavior exhibited by a particular realization. Similar computation/accuracy tradeoffs exist with the choice of c_2. We

Algorithm 2 Frequency Search: Method A

```
1: // Parameters: SearchIter,LdpcIter,MinFreq,MaxFreq,Step,c1,c2
2: WindowSize = MaxFreq-MinFreq
3: CenterFreq = (MinFreq+MaxFreq)/2
4: BestConst = 0
5: for i=0 to SearchIter-1 do
6:    NoEstimates = ⌈(WindowSize/Step+1)⌉
7:    for j=0 to NoEstimates-1 do
8:       Estimate = CenterFreq+Step*(j-(NoEstimates-1))
9:       Z⃗ = ComputeObs(Estimate)
10:      SatConst = LdpcDecoder(Z⃗,LdpcIter)
11:      if SatConst>BestConst then
12:         Bestconst = SatConst
13:         BestEstimate = Estimate
14:      end if
15:   end for
16:   WindowSize = WindowSize/c1
17:   Step = Step/c2
18:   CenterFreq = BestEstimate
19: end for
```

have found $c_1 = 2$ and $c_2 = 2$ to be effective, meaning that the window size and the step size are halved after each search iteration.

Fig. 3.7 explores the relationship between the number of search iterations, LDPC iterations per estimate, and the accuracy of the resulting frequency estimation. The figure shows diminishing returns after three search iterations. It also shows that using four LDPC iterations generally gives little added benefit over three iterations. Thus, we utilize three LDPC iterations per estimate and perform a total of three search iterations. With this choice of parameters, the frequency estimate is accurate to within 40 ppm.

An alternative algorithm (Algorithm 3) uses only one "Search Iteration" by using a smaller step size. After computing the number of satisfied constraints for the different hypothesis, an interpolation function based on the shape of Fig. 3.6 produces a more accurate estimation.

With regard to computational complexity, the total number of *ComputeObs* function calls is given by the product of the number of search iterations and the

Algorithm 3 Frequency Search: Method B

```
// Parameters: LdpcIter,MinFreq,MaxFreq,Step
WindowSize = MaxFreq-MinFreq
CenterFreq = (MinFreq+MaxFreq)/2
BestConst = 0
NoEstimates = ⌈(WindowSize/Step+1)⌉
for j=0 to NoEstimates-1 do
    Estimate = CenterFreq+Step*(j-(NoEstimates-1))
    Z⃗ = ComputeObs(Estimate)
    SatConst = LdpcDecoder(Z⃗,LdpcIter)
    if SatConst>BestConst then
        Bestconst = SatConst
        BestEstimate = Estimate
    end if
end for
Computer Interpolated offset
```

number of candidate frequency errors explored per search (SearchIter × NoEstimates). The total number of LDPC decoding iterations is given by the product of the number of *ComputeObs* calls and the number of LDPC iterations per estimate (SearchIter × NoEstimates × LdpcIter). For example, for a frequency offset search window from -2000 ppm to 2000 ppm, initial step size of 400 ppm, and three LDPC iterations at each frequency estimate, the first proposed algorithm requires a total of $3 \cdot 4000/400 + 1 = 33$ *ComputeObs* calls and $33 \cdot 3 = 99$ LDPC decoding iterations. On the other hand, the second proposed algorithm will use a step size of 250 ppm with three LDPC iterations per estimate, yielding a total of $3 \cdot 4000/250 + 1 = 51$ iterations that on average are within 70 ppm of the real offset value. The choice of a particular method depends on the constraint of the particular application. The first method is preferred for applications that require great precision, whereas the later is preferred for scenarios where the available computing resources are limited.

The accuracy and the computational burden are independent of the actual value of the frequency error as long as it is contained within the initial search window. Moreover, the complexity grows linearly with the frequency offset. An additional

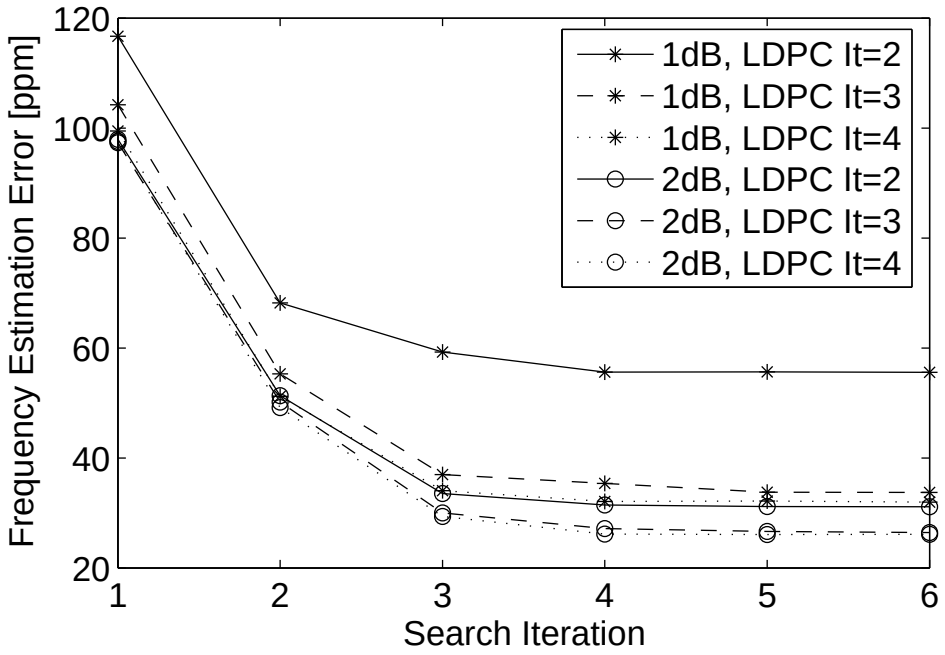


Figure 3.7: Relationship between the number of search iterations and the number of LDPC iterations per frequency estimate on the frequency estimation error. An initial step size of 400 ppm is used. An average of 500 trials using randomly selected frequency offsets over ± 2000 ppm are used for each data point.

advantage of this approach is that there is no PLL in this stage of the processing, hence there are no loop gains that need to be calibrated.

Time delays can be tracked in the same manner. Based on experiments analogous to those used for frequency offsets, we chose a step size of $0.2T$, one search iteration, and three LDPC iterations per estimate. Again, c_1 and c_2 are chosen to be two. At the cost of increased computational complexity it is also possible to track waveforms where both time delays and frequency offsets are present. For this case, Algorithm 4.2 is slightly modified where time delay tracking is in the outer loop and the frequency offset tracking is in the inner loop.

3.3.2 Tracking random walks

Random walks are tracked with a first-order PLL-based circuit utilizing the LDPC decoded symbols to aid the error estimation. The M&M TED generates an estimate

of the timing error according to,

$$u[i] = s[i]\hat{d}[i-1] - s[i-1]\hat{d}[i], \tag{3.7}$$

where $s[i]$ is the symbol from the interpolator and $\hat{d}[i]$ is the decoded symbol from the LDPC decoder. In practice, $u[i]$ is noisy and needs to be attenuated via a loop filter. We employ a first-order loop filter with an estimate updated according to,

$$c[i] = c[i-1] + Kp \times u[i], \tag{3.8}$$

where Kp is the proportional gain and $u[i]$ is the output of the M&M TED.

3.4 Experimental Results

For all experiments, we perform root raised-cosine pulse shaping with a roll-off factor of 0.3, and use a 12-tap FIR filter for matched filtering. The loop gain of loop 2 is fixed to $Kp = 0.001$. Fig. 3.8 shows the effects of frequency offsets and random walks on the bit error rate (BER) for the case when no timing recovery is applied. It illustrates that the BER penalty of timing errors can be quite high, especially for higher values of E_b/N_0. Hence, it is crucial to keep the timing error as low as possible. As noted earlier in association with Fig. 3.7 in Section 3.3.1 the method we present is able to track the frequency to within 40 ppm. Fig. 3.8 shows that tracking to that accuracy will preserve nearly all of the BER performance.

Fig 3.9 shows the BER and frame error rate (FER) performance. The "perfect timing" curves show the bounding BER and FER performance that would be obtained if the receiver was given perfect knowledge of the timing offsets. The other curves, in order of decreasing performance relative to the ideal case, show the performance of "Loop 1 + Loop 2" (i.e. the approach described in Section 3.3 and shown in Fig. 3.5), the case where only Loop 1 is used, and the case where only a PLL is used, for two different frequency offset values. "Loop 1 + Loop 2" is better than loop 1 alone because, as noted previously, loop 2 is able to correct

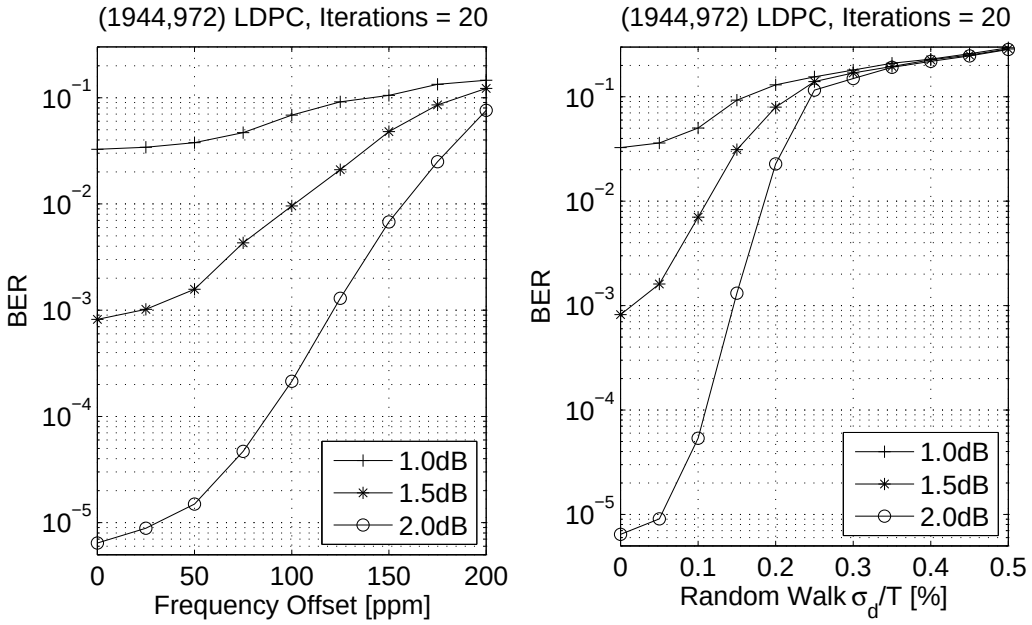


Figure 3.8: Effects of frequency offsets and random walks on the BER when no timing recovery is applied. Curves for three different values of E_b/N_0 are shown. The figure shows that frequency tracking accuracy should be below about 50 ppm to avoid signicant loss of BER performance.

residual frequency errors remaining after loop 1. Note that the performance of the iterative PLL-only technique is severely degraded with a 500 ppm offset.

Fig. 3.10 shows the BER/FER performance for different timing offset models. For constant time delays and frequency offsets, the receiver is able to get within $0.1 \sim 0.2$ dB of the ideal (i.e. perfect timing) code performance. The curve for the random walk gets farther away from the ideal case with increasing SNR. For the case when time delays, frequency offsets and random walks are present at the same time, the performance is within 0.1 dB of the random walk only case, suggesting that random walk is the dominant source of error.

3.5 Summary

A two-stage pilotless symbol timing recovery architecture is proposed where the first stage corrects large-scale time delays and frequency offsets and the second

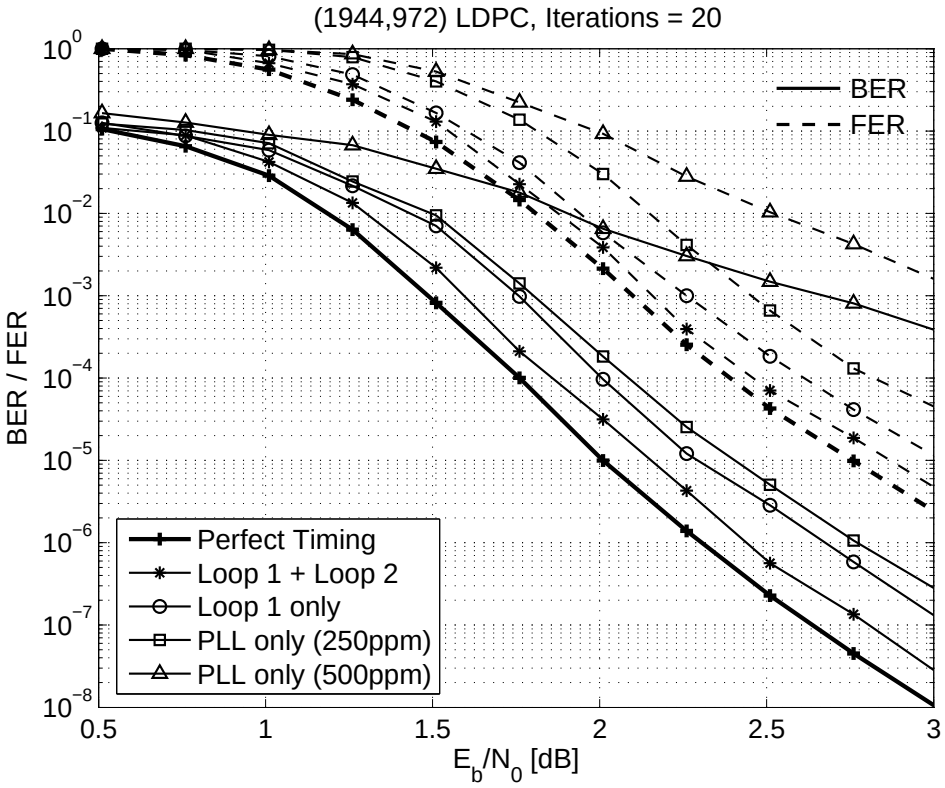


Figure 3.9: Bit error rate (BER) and frame error rate (FER) performance curves. The "PLL only" architecture employs a second order loop filter with the loop gains optimized for each case. For "loop 1 + loop 2" and "loop 1 only" models, random frequency offsets over ±2000 ppm are used.

stage tracks random walks and corrects residual time and frequency offsets. In the first stage, constraint node feedback from the LDPC decoder is employed in a search algorithm based on successively narrower windows to find the correct time delay and/or frequency offset. In the second stage, a PLL-based loop is employed where decoded symbols from the LDPC decoder are used to aid a timing error detector. Simulation results show performance within $0.1 \sim 0.2$ dB of the ideal (i.e. perfect timing knowledge at the receiver) code performance, and within 0.1 dB of the random walk only case, when time delays, frequency offsets and random walks are present at the same time.

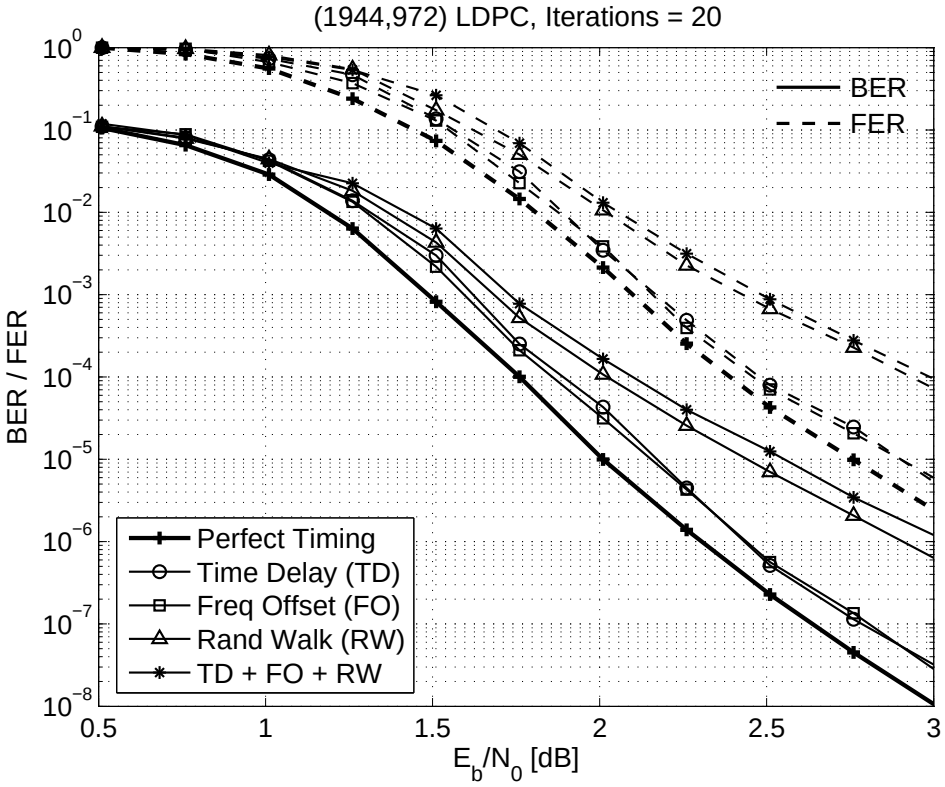


Figure 3.10: BER/FER performance for different timing offsets. For time delay and frequency offsets, random offsets over $\pm 0.5T$ and ± 2000 ppm, respectively are used. For the random walk, $\sigma_d/T = 0.5\%$ is used.

Chapter 4

Carrier Phase-Synchronization via LDPC Code Feedback

This chapter addresses the carrier-phase estimation problem under the low SNR conditions often encountered in turbo- and LDPC-coded applications. In [34] a decision directed carrier synchronization (DDCS) circuit that uses soft information from an iterative LDPC decoder was presented for BPSK and QPSK constellations. In this work we present a method that is able to handle arbitrary constellations. Loop SNR equations are derived and the performance for different constellations is shown. An extension of the DDCS algorithm is described that uses a search and tracking method based on measuring the number of satisfied check equations at the constraint node side of the decoder and is able to track arbitrary carrier phase.

4.1 Introduction

In recent years there has been increasing interest in highly efficient error-correction codes such as turbo codes and low density parity check (LDPC) codes [4,9]. These codes approach the Shannon channel capacity of the system and operate at very low symbol signal-to-noise ratios (SNRs) thus requiring carrier synchronization schemes that can track the carrier at these SNRs.

A significant research effort is underway in the area of joint decoding and carrier

phase estimation. As described by Noels *et al.* [35], two somewhat distinct groups of joint decoding and synchronization algorithms have evolved. The first of these approach the problem by modifying iterative detection/decoding algorithms and/or graphs to include parameter estimation. A partial list of such work includes [36, 37, 38, 39, 40, 41, 42, 43].

Of particular interest has been the work of Colavolpe *et al.* [43] where phase-tracking processing nodes were introduced in the iterative decoding graph. Dauwels *et al.* also investigated [40] specially adapted message-passing update rules. Howard *et al.* [41] proposed a pilotless modulation technique for turbo-coded differential 8PSK modulation which uses 35 iterations to compensate a $\pi/8$ phase offset at $E_b/N_o = 4.5$ dB. We also note the work of Nuriyev [39] on adapting density evolution to evaluate the performance of joint carrier-phase estimation in a pilot-assisted environment.

The second group of algorithms passes messages between an independent phase estimation block and an essentially unmodified iterative decoder. The resulting architectures are often said to employ *turbo synchronization* [35]. Algorithms of this type can can be found in [44, 45, 46, 47, 48].

The technique in this chapter falls into this second category and has the potentially attractive feature that little modification is required with either the iterative decoder or the carrier recovery block (which consists primarily of a phase-locked loop (PLL)) [32]. Specifically, the work leverages the fact that LDPC symbol estimates can 'wipe-off' modulated symbols in a decision-directed carrier recovery loop to enhance the carrier information such that a classic residual carrier PLL is able to provide increasingly accurate phase estimates over LDPC iterations. The method incurs a latency penalty (by way of increased iterations) as carrier phase is acquired. However, complexity in terms of system description and area (in the case of a real-time implementation) remains similar to that of state of the art residual carrier recovery techniques currently used, for example, in NASA's deep-space network. Moreover, the proposed architecture can be used in conjunction with other types of phase tracking loops in order to track residual carrier-phase errors at the decoding stage.

The authors in [44] propose a related approach but have described a phase estimate based on the instantaneous average of an entire block of received symbols. Additionally, Lottici and Luise [46] have developed a blind recovery technique for QAM receivers. The work in this chapter is also based on blind, or pilotless, operation and we motivate this in part by recalling a result from Anastasopoulos [36] who showed pilotless techniques to be more efficient at lower SNRs where pilot insertion loss is considerable.

The rest of this chapter is organized as follows. The next section provides a detailed description of the decision-directed carrier synchronization (DDCS) method. We derive the tracking performance of the PLL in terms of its mean-square phase error when operating in the linear (high loop SNR) region as is typical. In Section 4.3 we illustrate a digital baseband implementation that achieves the same performance as the piecewise constant analog model considered in Section 4.2. Results for different modulations are presented in Section 4.4. Finally, Section 4.5 gives conclusions.

4.2 Tracking Performance

On the transmitter side, we consider a baseband signal comprised of root raised-cosine pulses $p(t) = h_{RRC}(t)$, transmitted at multiples of a symbol interval T_s:

$$M(t) \quad = m_I(t) + jm_Q(t) = \left(\sum_{k=-\infty}^{\infty} d_{I_k} p\left(t - kT_s\right) \right) + j \left(\sum_{k=-\infty}^{\infty} d_{Q_k} p\left(t - kT_s\right) \right).$$

Multiplication by a sinusoidal carrier signal yields the transmitted waveform:

$$\begin{aligned} Y_T(t) &= M(t)e^{(j\omega_c t)} \\ &= y_{T_I}(t) + jy_{T_Q}(t) \\ &= \left(m_I \cos(\omega_c t) - m_Q \sin(\omega_c t)\right) + j\left(m_I \sin(\omega_c t) + m_Q \cos(\omega_c t)\right), \end{aligned} \tag{4.1}$$

where ω_c is the carrier frequency. A constant envelope modulation with $\{d_{I_k}, d_{Q_k}\} \in \left\{\pm\frac{\sqrt{P}}{2}\right\}$ (P is the carrier power) is initially assumed, although the proposed carrier recovery method works for any type of constellation as described in Section 4.4. In the transmission process, the signal in (4.1) is rotated by an angle θ_c and affected by a bandpass AWGN process:

$$N(t) = (n_I(t) + jn_Q(t))\, e^{j(\omega_c t + \theta_c)},$$

where $n_I(t)$ and $n_Q(t)$ have single-sided noise power spectral density (PSD) equal to N_0.

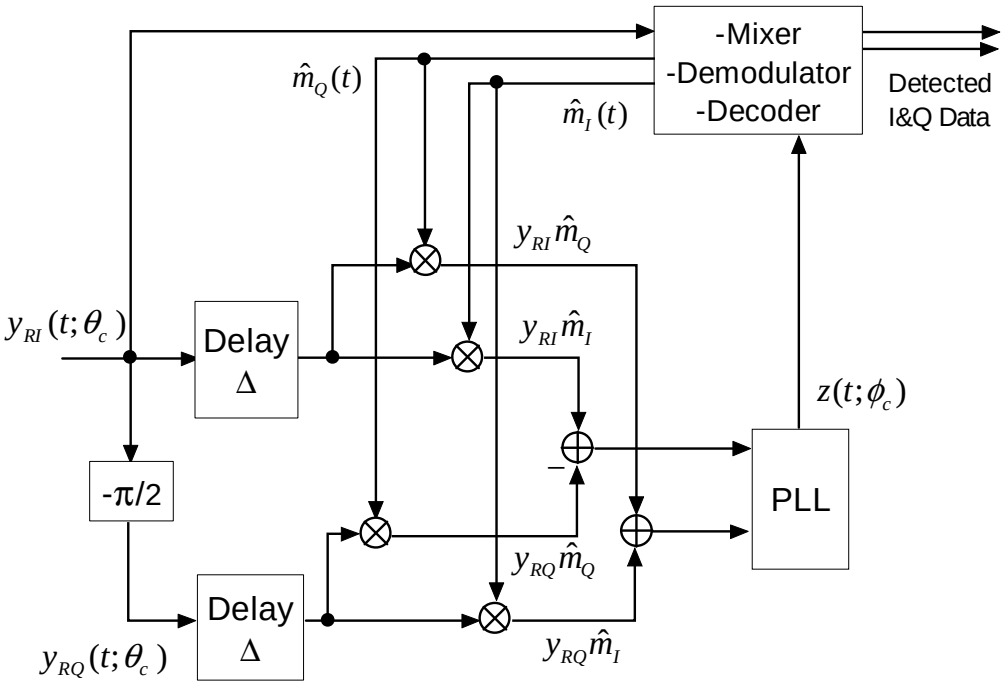


Figure 4.1: Analog receiver using soft-decision feedback.

Accurately estimating θ_c is the goal of carrier synchronization. A sample analog receiver, shown in Fig. 4.1, will be referred to in deriving the proposed carrier synchronization method. An alternative baseband implementation of this receiver is detailed in Section 4.3. At the receiver, consider an input modulation of the

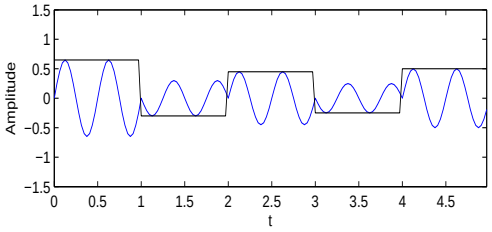
form:

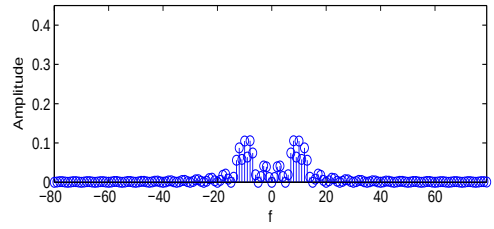
$$\begin{aligned} Y_R(t) &= (M+N)\, e^{(j\omega_c t+\theta_c)} = y_{R_I}(t) + j y_{R_Q}(t) \\ &= (m_I + n_I)\cos(w_c t+\theta_c) - (m_Q+n_Q)\sin(w_c t+\theta_c) \\ &\quad + j\left((m_Q+n_Q)\cos(w_c t+\theta_c) + (m_I+n_I)\sin(w_c t+\theta_c)\right), \end{aligned} \tag{4.2}$$

where the time reference (t) has been dropped from the symbol vectors M and N (and their components) for ease of notation.

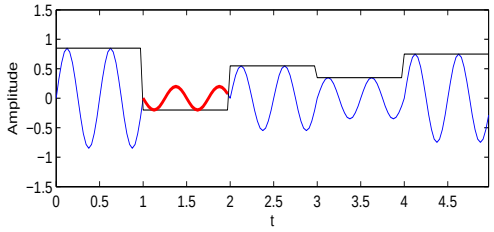
The next step is to delay the input by the decoder delay Δ and to remove the modulation present in (4.2) to produce a signal consisting of a pure tone. This signal can then be tracked by a digital PLL in order to generate an estimate of the carrier phase $\hat{\theta}_c$. To illustrate the concept behind the DDCS algorithm, consider a very simple example where five BPSK symbols consisting of alternating 1's and -1's affected by symbol-wise noise (note that the amplitude of each pulse not the transmitted value of unity) are transmitted as shown in Fig. 4.2(a). Modulation is removed by multiplying the received waveform by soft-estimated symbols from the LDPC decoder. The resulting unmodulated frequency spectrum, shown in Fig. 4.2(b), clearly begins to show the presence of a carrier. Note that this occurs regardless of the fact that some soft estimations of the symbols may still be incorrect and have an incorrect sign. Since the information feedback consists of soft information, the incorrect information has in general low reliability. Fig. 4.2(c) shows the frequency spectrum obtained after the decoder has converged to the correct sent codeword. The strong tone that appears at the carrier frequency can now be tracked by a PLL.

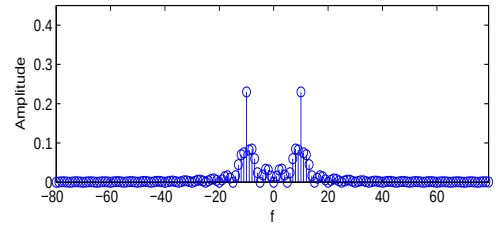
In this work, two different techniques for removing the modulation information present in the received waveform have been examined. These two alternatives differ in the information feedback vector used to remove the modulation in the carrier. Both techniques use a soft symbol estimate $\hat{M}(t) = \hat{m}_I(t) + j\hat{m}_Q(t)$ produced by the LDPC decoder to remove the modulation information. The first proposed technique uses the complex conjugate (CC) of the estimated symbol, $\hat{M}^*(t)$, as feedback. The second proposed technique normalizes the symbol estimates to as-



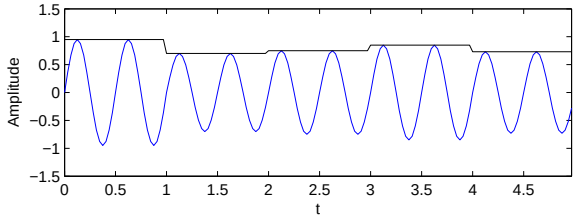


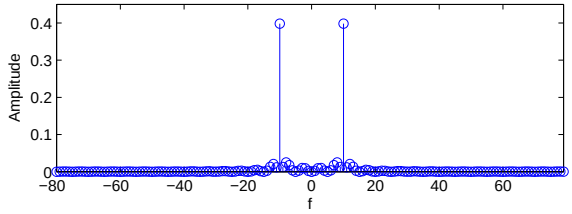
(a) Received Symbols + Noise





(b) DDCS Waveform: Initial Iterations





(c) DDCS Waveform: Final Iterations

Figure 4.2: DDCS BPSK Example. (a) Received Symbols($R = [1; -1; 1; -1; 1]$) + Noise is assumed. (b) DDCS waveform after a small number of iterations. Example shows that the decoder makes an error on the estimated value of the second symbol. (c) DDCS waveform near convergence. Figure assumes that the LDPC decoder has correctly estimated the received symbols R.

sure unit amplitude, $\frac{\hat{M}^*(t)}{||\hat{M}^*(t)||}$. The intuition behind these two choices is that when using CC feedback, both the magnitude and phase of the feedback vector are feed back to the carrier recovery circuit. On the other hand when the normalized complex conjugate (NCC) is used as a metric, only the phase of the estimated symbol is sent from the decoder to the DDCS circuit. The choice of the modulation removal technique depends on the signal constellation chosen for transmission and on the range of possible values that the carrier recovery circuit is intended to track. As will be described in Section 4.4, our experiments show that for constant envelope modulations such as MPSK both methods yield similar performance although in some cases where the carrier phase offset is significant, the NCC technique outperforms the CC method in terms of frame error rate (FER) performance. For non-constant envelope modulations such as 16QAM and APSK, the results of our simulations show that using the NCC as a metric is a better choice, especially for cases with large carrier offsets. There are many other possible alternatives that can be implemented by manipulating the LDPC output symbol estimates in different ways. This choice will depend on the modulation used and should be carefully designed jointly with the loop gains present in the PLL in order to minimize FER.

4.2.1 Complex Conjugate (CC) Feedback

This technique uses the complex conjugate of the estimated symbol output from the LDPC decoder, $\hat{M}^*(t)$, to remove the modulation information present in (4.2):

$$U(t) = Y_R(t)\hat{M}^*(t) = Y_R(t)\left(M^*(t) + \frac{N_s(t)}{\gamma}\right) \tag{4.3}$$

where

$$N_s = \sum_{k=-\infty}^{\infty} (n_{s_I k} + j n_{s_Q k}) p\left(t - kT_s\right)$$

models the decoder symbol-estimation error and γ is the LDPC decoder *a-posteriori* signal amplitude. Both $n_{s_I k}$ and $n_{s_I k}$ are i.i.d. zero mean Gaussian random variables with variance σ^2. By observing the symbol-estimation noise term $\frac{N_s(t)}{\gamma}$

we can see how $\hat{M}(t) \rightarrow M(t)$ as the LDPC decoder iterations increase and the reliability of the *a-posteriori* values increase,

$$\begin{aligned} u_I(t) &= (m_I\hat{m}_I + m_Q\hat{m}_Q)\cos(\omega_c t + \theta_c) + (m_I\hat{m}_Q - m_Q\hat{m}_I)\sin(\omega_c t + \theta_c) \\ &\quad + (n_I\hat{m}_I + n_Q\hat{m}_Q)\cos(\omega_c t + \theta_c) + (n_I\hat{m}_Q - n_Q\hat{m}_I)\sin(\omega_c t + \theta_c) \\ &= (\alpha_s + \alpha_n)\cos(\omega_c t + \theta_c) + (\beta_s + \beta_n)\sin(\omega_c t + \theta_c), \end{aligned} \tag{4.4}$$

$$\begin{aligned} u_Q(t) &= \underbrace{(m_I\hat{m}_I + m_Q\hat{m}_Q)}_{\alpha_s}\sin(\omega_c t + \theta_c) - \underbrace{(m_I\hat{m}_Q - m_Q\hat{m}_I)}_{\beta_s}\cos(\omega_c t + \theta_c) \\ &\quad + \underbrace{(n_I\hat{m}_I + n_Q\hat{m}_Q)}_{\alpha_n}\sin(\omega_c t + \theta_c) - \underbrace{(n_I\hat{m}_Q - n_Q\hat{m}_I)}_{\beta_n}\cos(\omega_c t + \theta_c) \\ &= (\alpha_s + \alpha_n)\sin(\omega_c t + \theta_c) - (\beta_s + \beta_n)\cos(\omega_c t + \theta_c). \end{aligned} \tag{4.5}$$

The signals above are then input to a PLL whose voltage-controlled oscillator (VCO) output can be expressed as

$$r_{vco}(t, \hat{\theta}_c) = \sin(\omega_c t + \hat{\theta}_c).$$

Multiplying $u_I(t)$ and $u_Q(t)$ by $r_{vco}(t, \hat{\theta}_c)$ and $r_{vco}(t, \hat{\theta}_c - \pi/2) = \cos(\omega_c t + \hat{\theta}_c)$ and combining the results of these products yields,

$$\begin{aligned} z(t, \phi_c) &= u_Q(t)\cos(\omega_c t + \hat{\theta}_c) - u_I(t)\sin(\omega_c t + \hat{\theta}_c) \\ &= \left(\frac{\alpha_s + \alpha_n}{2}\right)\Big[\sin(2\omega_c t + \hat{\theta}_c + \theta_c) + \sin(\phi_c)\Big] \\ &\quad - \left(\frac{\beta_s + \beta_n}{2}\right)\Big[\cos(2\omega_c t + \hat{\theta}_c + \theta_c) + \cos(\phi_c)\Big] \\ &\quad - \left(\frac{\alpha_s + \alpha_n}{2}\right)\Big[\sin(2\omega_c t + \hat{\theta}_c + \theta_c) - \sin(\phi_c)\Big] \\ &\quad - \left(\frac{\beta_s + \beta_n}{2}\right)\Big[-\cos(2\omega_c t + \hat{\theta}_c + \theta_c) + \cos(\phi_c)\Big] \\ &= \alpha_s\sin(\phi_c) + \underbrace{(\alpha_n\sin(\phi_c) - (\beta_s + \beta_n)\cos(\phi_c))}_{v(t,\phi_c)} \end{aligned} \tag{4.6}$$

where $\phi_c = \theta_c - \hat{\theta}_c$. Note that no low-pass filter is required in (4.6) since the

frequency components at $2\omega_c$ have opposite signs and cancel each other.

Next we pass $z(t)$ through a matched filter to produce for the k^{th} interval $(k+1)T_s \leq t \leq (k+2)T_s$, [1]

$$\begin{aligned} e_k &= \int_k^{(k+1)} z(t,\phi_c)dt \\ &= (d_{I_k}+n_{I_k})\underbrace{\left(d_{I_k}+\frac{ns_{I_k}}{\gamma}\right)}_{\hat{d}_{I_k}}\sin(\phi_c) + (d_{Q_k}+n_{Q_k})\underbrace{\left(d_{Q_k}+\frac{ns_{Q_k}}{\gamma}\right)}_{\hat{d}_{Q_k}}\sin(\phi_c) \\ &\quad -(d_{I_k}+n_{I_k})\left(d_{Q_k}+\frac{ns_{Q_k}}{\gamma}\right)\cos(\phi_c) + (d_{Q_k}+n_{Q_k})\left(d_{I_k}+\frac{ns_{I_k}}{\gamma}\right)\cos(\phi_c) \\ &= (d_{I_k}^2+d_{Q_k}^2)\sin(\phi_c) + v(k,\phi_c) \\ &= P\sin(\phi_c) + v(k,\phi_c), \end{aligned} \tag{4.7}$$

where

$$\begin{aligned} v(k,\phi_c) &= \left(n_{I_k}\hat{d}_{I_k} + \frac{d_{I_k}ns_{I_k}}{\gamma} + n_{Q_k}\hat{d}_{Q_k} + d_{Q_k}\frac{ns_{Q_k}}{\gamma}\right)\sin(\phi_c) \\ &\quad -\left((d_{I_k}+n_{I_k})\hat{d}_{Q_k} + (d_{Q_k}+n_{Q_k})\hat{d}_{I_k}\right)\cos(\phi_c), \\ n_{I_k} &= \int_k^{(k+1)} n_I(t)dt, \\ n_{Q_k} &= \int_k^{(k+1)} n_Q(t)dt, \end{aligned} \tag{4.8}$$

are zero mean Gaussian noise RVs with variance ${\sigma_n}^2 = N_o/2$. Clearly from the above, the slope of the $S-$curve [49], K_g, for constant envelope modulations is given by $K_g = P$.

We now compute the autocorrelation function of $v(k,\phi_c)$ (treated as a piecewise continuous process $v(t,\phi_c)$) from which we shall obtain the equivalent noise PSD affecting the loop. For operation in the neighborhood of $\phi_c = 0$, it is reasonable to

[1] Without loss in generality, we herein ignore the decoder delay Δ and assume T_s=1 and a unit energy pulse.

consider only the autocorrelation function of $v(k,0)$:

$$\begin{aligned} v(k,0) &= (d_{Q_k} + n_{Q_k})\hat{d}_{I_k} - (d_{I_k} + n_{I_k})\hat{d}_{Q_k} \\ &= (d_{Q_k} + n_{Q_k})\left(d_{I_k} + \frac{ns_{I_k}}{\gamma}\right) - (d_{I_k} + n_{I_k})\left(d_{Q_k} + \frac{ns_{Q_k}}{\gamma}\right) \\ &= \underbrace{(d_{Q_k}\hat{d}_{I_k} - d_{I_k}\hat{d}_{Q_k})}_{-\beta_{s_k}} + \underbrace{(n_{Q_k}\hat{d}_{I_k} - n_{I_k}\hat{d}_{Q_k})}_{-\beta_{n_k}} \\ &= -(\beta_{s_k} + \beta_{n_k}). \end{aligned}$$

Assuming that the real and imaginary components of $N_s(t)$ are independent and the noise samples are independent from symbol interval to symbol interval, then the autocorrelation is triangular,

$$R_v(\tau) = E\{v(t,0)v(t+\tau,0)\} = \begin{cases} \sigma_v^2\left(1 - \frac{|\tau|}{T_s}\right) & |\tau| \le T_s \\ 0 \text{ otherwise} \end{cases}$$

with

$$\begin{aligned} \sigma_v^2 &= E\{v^2(k,0)\} \\ &= E\left\{(d_{Q_k}\hat{d}_{I_k} - d_{I_k}\hat{d}_{Q_k})^2\right\} + E\left\{(n_{Q_k}\hat{d}_{I_k} - n_{I_k}\hat{d}_{Q_k})^2\right\} \\ &\quad + 2E\left\{(d_{Q_k}\hat{d}_{I_k} - d_{I_k}\hat{d}_{Q_k})(n_{Q_k}\hat{d}_{I_k} - n_{I_k}\hat{d}_{Q_k})\right\}. \end{aligned} \tag{4.9}$$

The cross-term in (4.9) is zero, and the two remaining terms can be written as:

$$\begin{aligned} E\left\{(d_{Q_k}\hat{d}_{I_k} - d_{I_k}\hat{d}_{Q_k})^2\right\} &= E\left\{\left(d_{Q_k}\left(d_{I_k} + \frac{ns_{I_k}}{\gamma}\right) - d_{I_k}\left(d_{Q_k} + \frac{ns_{Q_k}}{\gamma}\right)\right)^2\right\} \\ &= E\{d_{Q_k}^2\}\left(E\{d_{I_k}^2\} + \frac{\sigma^2}{\gamma^2}\right) + E\{d_{I_k}^2\}\left(E\{d_{Q_k}^2\} + \frac{\sigma^2}{\gamma^2}\right) \\ &\quad - 2E\left\{d_{I_k}^2 d_{Q_k}^2\right\} \\ &= \frac{\sigma^2}{\gamma^2}\left(E\{d_{I_k}^2\} + E\{d_{Q_k}^2\}\right) = 2P\sigma^2/\gamma^2 \end{aligned} \tag{4.10}$$

and

$$
\begin{aligned}
E\left\{(n_{Q_k}\hat{d}_{I_k} - n_{I_k}\hat{d}_{Q_k})^2\right\} &= E\left\{n_{Q_k}^2\left(d_{I_k} + \tfrac{ns_{I_k}}{\gamma}\right)^2\right\} + E\left\{n_{I_k}^2\left(d_{Q_k} + \tfrac{ns_{Q_k}}{\gamma}\right)^2\right\} \\
&\quad - 2E\left\{n_{I_k}n_{Q_k}\hat{d}_{I_k}\hat{d}_{Q_k}\right\} \\
&= \sigma_n^2\left(E\{d_{I_k}^2\} + \frac{\sigma^2}{\gamma^2}\right) + \sigma_n^2\left(E\{d_{Q_k}^2\} + \frac{\sigma^2}{\gamma^2}\right) \\
&= 2\sigma_n^2\frac{\sigma^2}{\gamma^2} + \sigma_n^2\left(E\{d_{I_k}^2\} + E\{d_{Q_k}^2\}\right) \\
&= 2\sigma_n^2\left(P + \frac{\sigma^2}{\gamma^2}\right)
\end{aligned}
\tag{4.11}
$$

where $E\{n_{I_k}^2\} = E\{n_{Q_k}^2\} = \sigma_n^2 = \frac{No}{2}$, $E\{d_{I_k}^2 + d_{Q_k}^2\} = P$ and $E\{ns_{I_k}^2\} = E\{ns_{Q_k}^2\} = \sigma^2$. Combining (4.10) and (4.11) yields:

$$
\sigma_v^2 = 2\sigma_n^2\left(P + \left(1 + \frac{P}{\sigma_n^2}\right)\frac{\sigma^2}{\gamma^2}\right). \tag{4.12}
$$

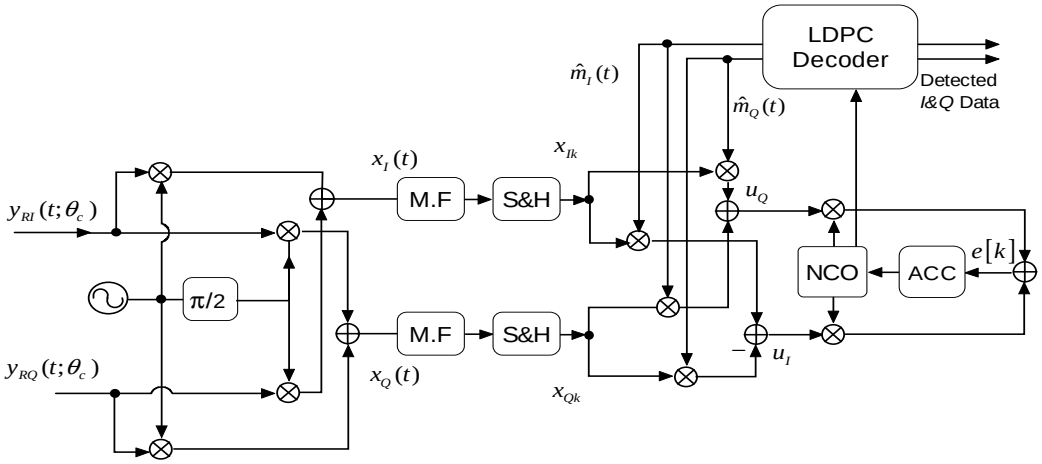


Figure 4.3: Digital circuit for baseband implementation.

The equivalent single-sided noise PSD, N_e, can be written as:

$$
N_e = 2\int_{-\infty}^{-\infty} R_v(\tau, 0)d\tau = N_0\left(P + \left(1 + \frac{P}{\sigma_n^2}\right)\frac{\sigma^2}{\gamma^2}\right).
$$

Finally, the mean-square phase error in the loop is given by:

$$\sigma^2_{\phi_c} = \frac{N_e B_L}{K_g^2} = \frac{N_0 B_L}{P^2}\left(1 + \left(\frac{1}{P} + \frac{1}{\sigma_n^2}\right)\frac{\sigma^2}{\gamma^2}\right) \triangleq \frac{1}{\rho S_L}, \tag{4.13}$$

where B_L is the noise bandwidth, $\rho = P/(N_0 B_L)$ is the loop SNR in a conventional PLL and

$$S_L^{DDCS} \triangleq \left(\frac{1}{P} + \left(\frac{1}{P^2} + \frac{1}{\sigma_n^2}\right)\frac{\sigma^2}{\gamma^2}\right)^{-1}, \tag{4.14}$$

is the degradation of the loop SNR analogous to the "squaring loss" in a conventional Costas loop (CL). The quantity γ^2/σ^2 represents the decoder *soft SNR estimate*. As the iteration proceeds, the estimated data SNR increases and likewise the squaring loss decreases (i.e., $S_L^{DDCS} \to P$). By comparison, for a Costas loop, the expression for the squaring loss is given by

$$\begin{aligned} S_L^C &\triangleq (1 + 1/(2R_d))^{-1}, \\ R_d &= P/N_0, \end{aligned} \tag{4.15}$$

and thus remains fixed, independent of the iteration process, for a given symbol SNR [49].

To numerically evaluate the performance in (4.14), one needs to quantify the functional dependence of the decoder soft-estimate of the data SNR and the input symbol SNR. Under the assumptions in which (4.15) and (4.14) were derived, for the SNR regime where the DDCS circuit operates, the expected gain in SL is approximately 4 dB. However when both circuits were simulated at low SNR scenarios we experienced a gain of around 12dB in SL. This behavior is shown in Fig. 4.4 for a system that uses BPSK modulation with $\theta_c = \pi/4$. Fig. 4.4 compares the L_{SNR} performance for both loops under using a rate-1/2 irregular LDPC code of length $n = 1944$ with BPSK modulation and $\theta_c = \pi/4$. One of the reasons why the gap in performance between both circuits is larger than predicted is that in our simulations the DDCS system channel observations are updated on every iteration. After the DDCS loop processes a block of n symbols, an average of the carrier phase $\hat{\theta}_c$ is used to de-rotate the received channel observation vector. The

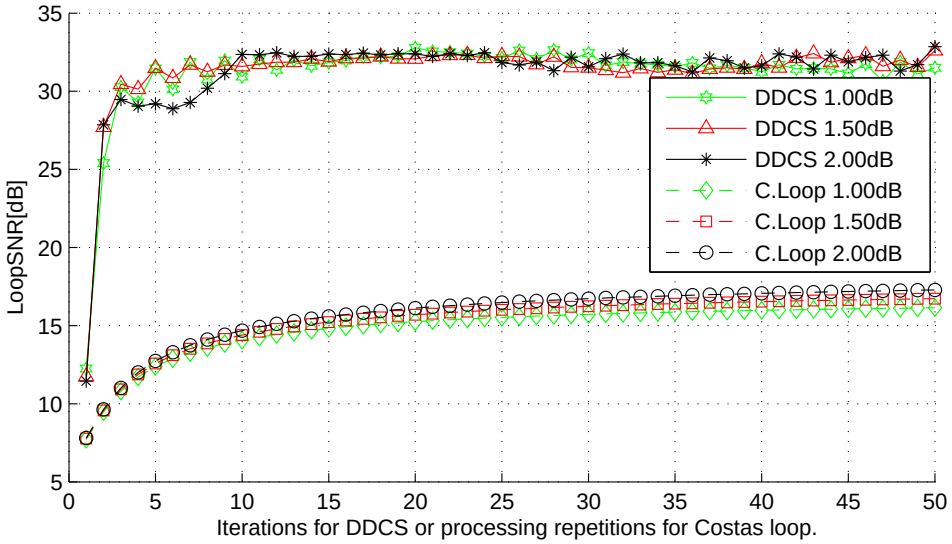


Figure 4.4: Comparison of Loop SNR for a BPSK system with $\theta_c = \pi/4$ using the DDCS circuit and a Costas loop.

LDPC decoder performs a variable and check-node update before starting a new DDCS loop-iteration. On the other hand, the CL is independent of the decoder decisions and uses the same channel information every time it runs. This implies that for the CL case, the horizontal axis of Fig. 4.4 in fact represents the number of times that each block (of size n) is processed by the loop. For example, after 10 loop updates, the DDCS circuit has processed $S = 10n$ different symbols, while the CL overprocessed the same $S = n$ symbols 10 times.

For the DDCS circuit, steady state is reached after 10 iterations ($10 \times 1944 = 19440$ total symbols processed). The CL converges to its steady state operation after overprocessing each block of 1944 symbols approximately 20 times. The speed of convergence is highly dependent of the gains of the loop-filter shown in Fig. 4.3. A second order filter with transfer function $H(z) = (K_p + K_i z^{-1})/(1 - z^{-1})$ was used for both circuits with gains $[K_p, K_i] = [8.85.10^{-4}, -8.75.10^{-4}]$ for the CL and $[K_p, K_i] = [8.91.10^{-5}, -8.75.10^{-5}]$ for the DDCS circuit. Both phase estimators were unbiased, however the second moment of ϕ_c was larger for the CL case than for the DDCS circuit causing the variance of ϕ_c to be larger for the CL case.

4.2.2 Normalized Complex Conjugate (NCC) Feedback

An alternative technique to remove the modulation information present in (4.2) normalizes the complex conjugate of the estimated symbol output from the LDPC decoder to unit amplitude,

$$U(t) = Y_R(t)\frac{\hat{M}^*(t)}{||\hat{M}(t)||}. \tag{4.16}$$

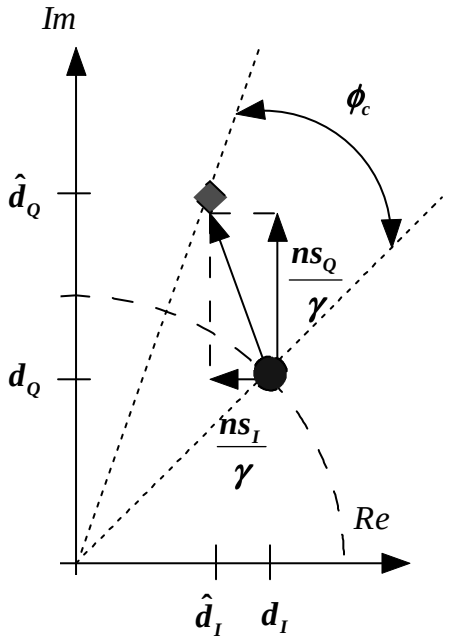
By analyzing the squaring loss equation corresponding to this method, we can see that a factor of $||\hat{M}(t)||$ will now divide (4.4),(4.5) and (4.6). Let s_{I_k} and s_{Q_k} be the sign of $\hat{d}_{I_k}$ and $\hat{d}_{Q_k}$. The error signal in (4.7) can now be written as:

$$\begin{aligned} e_k &= \int_k^{(k+1)} z(t,\phi_c)dt \\ &= (d_{I_k} + n_{I_k})\left(s_{I_k} + \frac{ns_{I_k}}{\gamma}\right)\sin(\phi_c) + (d_{Q_k} + n_{Q_k})\left(s_{Q_k} + \frac{ns_{Q_k}}{\gamma}\right)\sin(\phi_c) \\ &\quad - (d_{I_k} + n_{I_k})\left(s_{Q_k} + \frac{ns_{Q_k}}{\gamma}\right)\cos(\phi_c) + (d_{Q_k} + n_{Q_k})\left(s_{I_k} + \frac{ns_{I_k}}{\gamma}\right)\cos(\phi_c) \\ &= \sqrt{P}\sin(\phi_c) + v_N(k,\phi_c). \end{aligned} \tag{4.17}$$

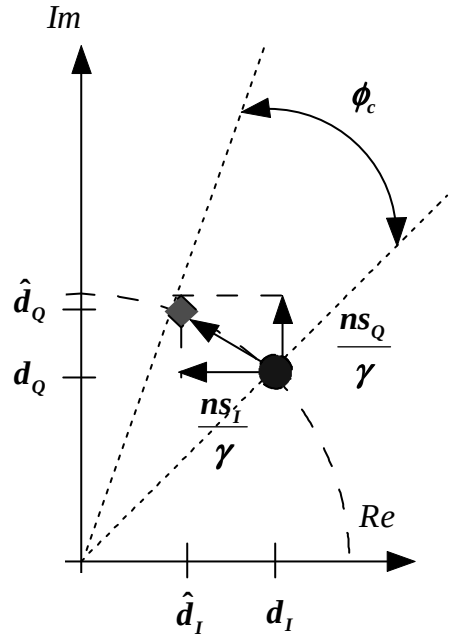
where n_{I_k}, n_{Q_k} and $v_N(k,\phi_c)$ are defined as in (4.8). By comparing (4.7) and (4.17) it is clear that the term proportional to carrier phase error is simply scaled by a factor of $\sqrt{P}$. In addition, the noise affecting the loop has also been affected by the normalization since, as shown in Fig. 4.5, the estimation vectors used for feedback are now constrained to the unit circle. Simulation results shown in Section 4.4, show a small gain in FER performance is obtained when using the NCC metric, in particular for cases that exhibit a large carrier phase offsets.

4.3 A Baseband Digital Implementation

The circuit in Fig. 4.3, shows a practical implementation of the carrier recovery loop presented in the previous section. The first difference with the block diagram in Fig. 4.1 is that demodulation (conversion to baseband) of the input signal (4.2)



(a) CC Feedback



(b) NCC Feedback

Figure 4.5: Vector diagrams of different feedback metrics. (a) Shows the case when using a complex conjugate metric, while (b) shows the case where the feedback metric has been normalized to unit amplitude.

is done with the carrier phase error still present, using the I and Q reference signals (arbitrarily assuming them to have zero phase relative to the received signal):

$$\begin{aligned} x_I(t;\theta_c) &= y_{R_I}\cos(\omega_c t) + y_{R_Q}\sin(\omega_c t) \\ &= (m_I + n_I)\cos(\theta_c) - (m_Q + n_Q)\sin(\theta_c), \\ x_Q(t;\theta_c) &= -y_{R_I}\sin(\omega_c t) + y_{R_Q}\cos(\omega_c t) \\ &= (m_I + n_I)\sin(\theta_c) + (m_Q + n_Q)\cos(\theta_c). \end{aligned}$$

The demodulated signals are then passed through matched root raised-cosine filters,

$$\begin{aligned} v_{I_k} &= (d_{I_k} + n_{I_k})\cos(\theta_c) - (d_{Q_k} + n_{Q_k})\sin(\theta_c), \\ v_{Q_k} &= (d_{I_k} + n_{I_k})\sin(\theta_c) + (d_{Q_k} + n_{Q_k})\cos(\theta_c), \end{aligned} \tag{4.18}$$

with n_{I_k} and n_{Q_k} as in (4.8). Next we multiply v_{I_k} and v_{Q_k} by the symbol estimate provided by the LDPC decoder. From this point forward (4.3) is used to remove the modulation from the received signal, unless specified otherwise. The discrete version of (4.6) is obtained by combining (4.18) with (4.3), $U_k = V_k \hat{M}_k^*$. The real and imaginary components of U_k can be written as:

$$\begin{aligned} u_{I_k} &= v_{I_k}\hat{d}_{I_k} + v_{Q_k}\hat{d}_{Q_k} \\ &= (d_{I_k}\hat{d}_{I_k} + d_{Q_k}\hat{d}_{Q_k})\cos(\theta_c) + (d_{I_k}\hat{d}_{Q_k} - d_{Q_k}\hat{d}_{I_k})\sin(\theta_c) \\ &\quad + (n_{I_k}\hat{d}_{I_k} + n_{Q_k}\hat{d}_{Q_k})\cos(\theta_c) + (n_{I_k}\hat{d}_{Q_k} - n_{Q_k}\hat{d}_{I_k})\sin(\theta_c) \\ &= (\alpha_{s_k} + \alpha_{n_k})\cos(\theta_c) + (\beta_{s_k} + \beta_{n_k})\sin(\theta_c) \end{aligned} \tag{4.19}$$

$$\begin{aligned} u_{Q_k} &= v_{Q_k}\hat{d}_{I_k} - v_{I_k}\hat{d}_{Q_k} \\ &= (d_{I_k}\hat{d}_{I_k} + d_{Q_k}\hat{d}_{Q_k})\sin(\theta_c) - (d_{I_k}\hat{d}_{Q_k} - d_{Q_k}\hat{d}_{I_k})\cos(\theta_c) \\ &\quad + (n_{I_k}\hat{d}_{I_k} + n_{Q_k}\hat{d}_{Q_k})\sin(\theta_c) - (n_{I_k}\hat{d}_{Q_k} - n_{Q_k}\hat{d}_{I_k})\cos(\theta_c) \\ &= (\alpha_{s_k} + \alpha_{n_k})\sin(\theta_c) - (\beta_{s_k} + \beta_{n_k})\cos(\theta_c) \end{aligned} \tag{4.20}$$

which are the baseband equivalents of (4.4) and (4.5). These signals are input to a digital PLL whose number-controlled oscillator (NCO) produces an estimate of the carrier phase denoted by $\hat{\theta}_c$. Combining u_{I_k} and u_{Q_k} with $w_{ck} = \cos(\hat{\theta}_c)$ and

$w_{sk} = \sin(\hat{\theta}_c)$, provides the baseband error signal:

$$\begin{aligned} e_k(\phi_c) &= u_{Q_k}\cos(\hat{\theta}_c) - u_{I_k}\sin(\hat{\theta}_c) \\ &= (\alpha_s + \alpha_n)\sin(\phi_c) - (\beta_s + \beta_n)\cos(\phi_c) \\ &= (d_{I_k}^2 + d_{Q_k}^2)\sin(\phi_c) + v(k, \phi_c), \end{aligned} \tag{4.21}$$

where as before $\phi_c = \theta_c - \hat{\theta}_c$ denotes the phase error in the loop. Comparing (4.21) with (4.6) we see that they are identical and thus the performance of the digital PLL would also be described by (4.13) together with (4.14).

4.4 Iterative Processing and Numerical Results

We have evaluated the performance of the all-digital baseband approach described in Section 4.3 via joint decoding with a rate-1/2 (1944, 972) irregular LDPC code developed in [33]. In Figs. 4.6 - 4.9, we can see the FER performance for the two proposed methods for modulation removal. The NCC technique that removes symbol modulation by using the unit vector $\frac{M^*}{||\hat{M}(t)||}$ as a feedback metric outperforms the CC technique specially for cases the cases non-constant envelope constellations such as 16QAM. For the constant envelope constellations, the gap in performance between NCC and CC is reduced, although NCC seems to have a significant advantage for scenarios with a strong carrier phase offset. Overall the metric that normalizes the feedback vector seems to be the better choice in terms of FER performance.

Let ψ be the rotational invariance angle of the constellation. For the constellations considered in this work we have $\psi_{\text{BPSK}} = \pi$, $\psi_{\text{QPSK}} = \psi_{\text{16QAM}} = \pi/2$, $\psi_{\text{8PSK}} = \pi/4$. Let γ_c be the maximum carrier phase rotation that the system can handle without severely degrading the FER performance. From our experimental results shown in Figs. 4.6 - 4.9 we observe that $\gamma_c \approx 0.45\pi$ for BPSK, $\gamma_c \approx 0.19\pi$ for QPSK, $\gamma_c \approx 0.15\pi$ for 16QAM and $\gamma_c \approx 0.10\pi$ for 8PSK. In all these cases $\gamma_c < \frac{\psi}{2}$.

Slightly perturbing the constellations to increase the rotational invariance ψ

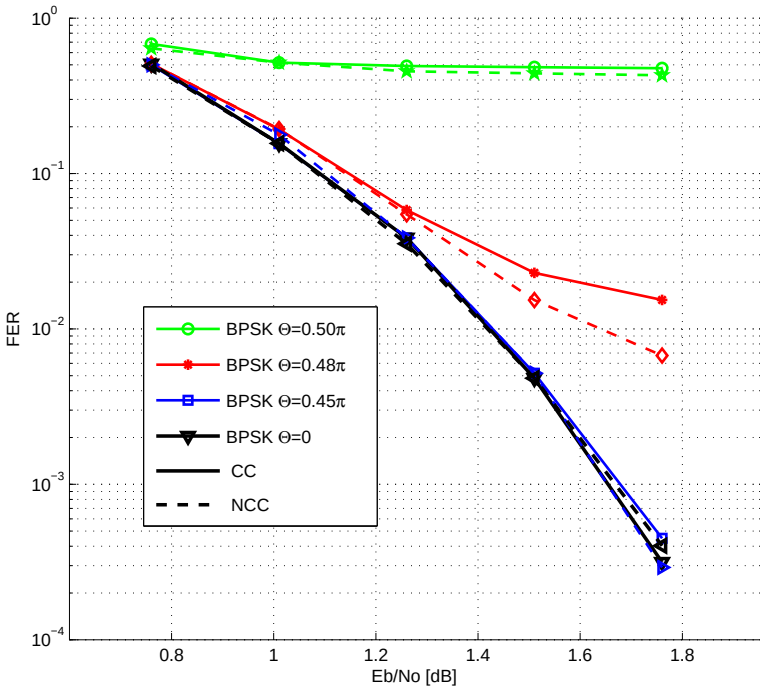


Figure 4.6: BPSK Performance.

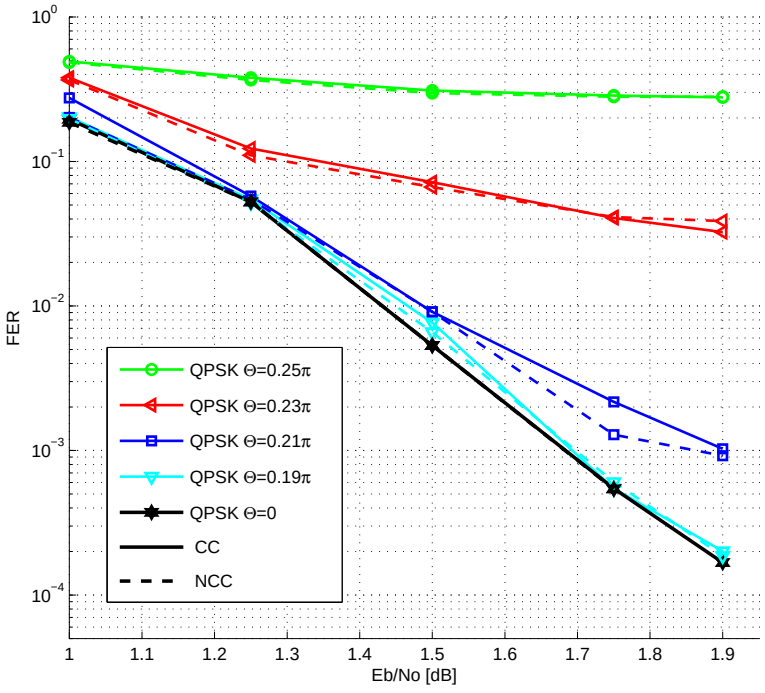


Figure 4.7: QPSK Performance.

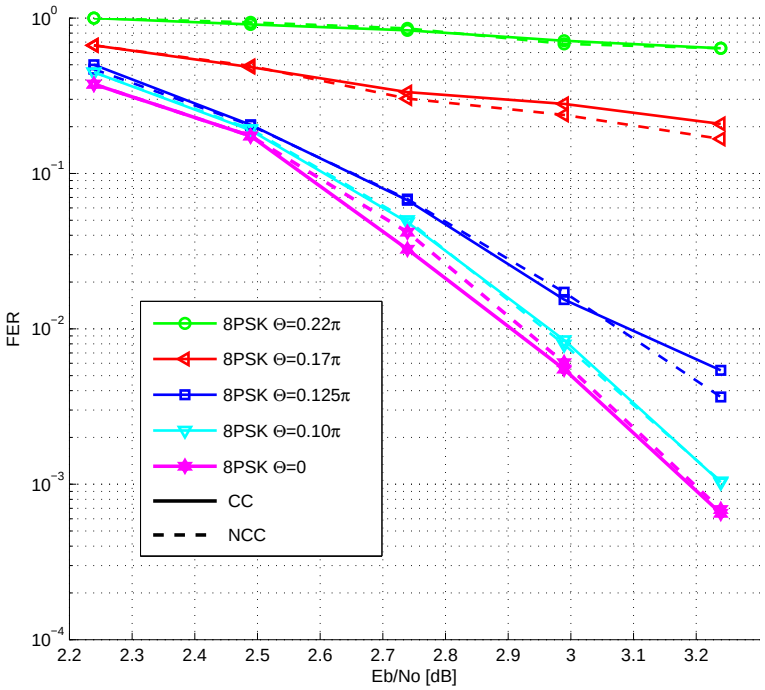


Figure 4.8: 8PSK Performance.

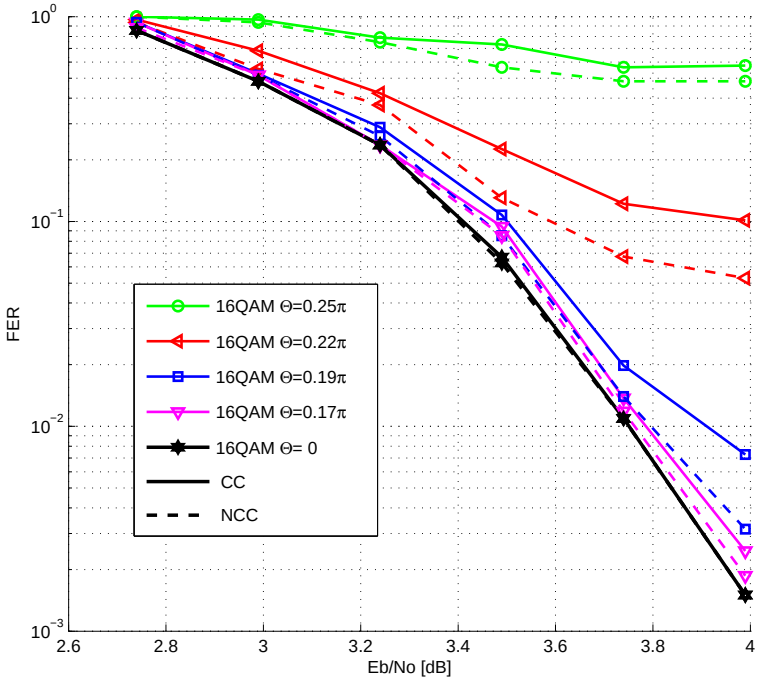
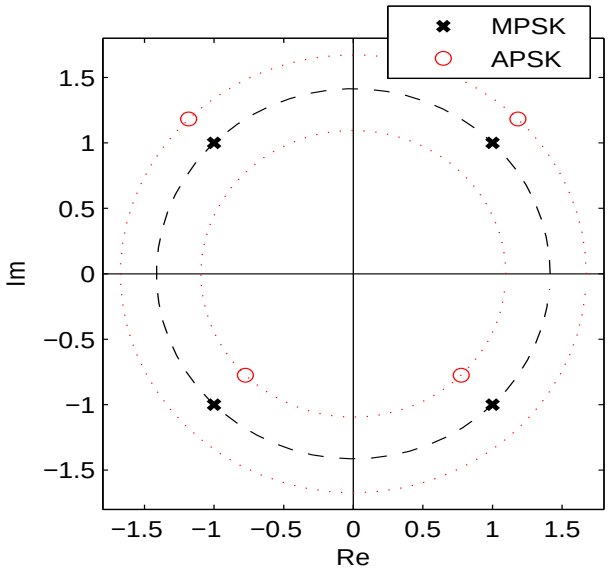
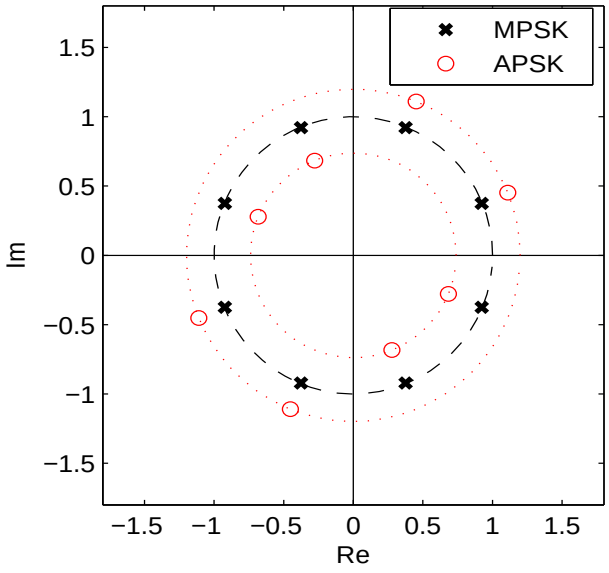


Figure 4.9: 16QAM Performance.



(a) 2+2-APSK



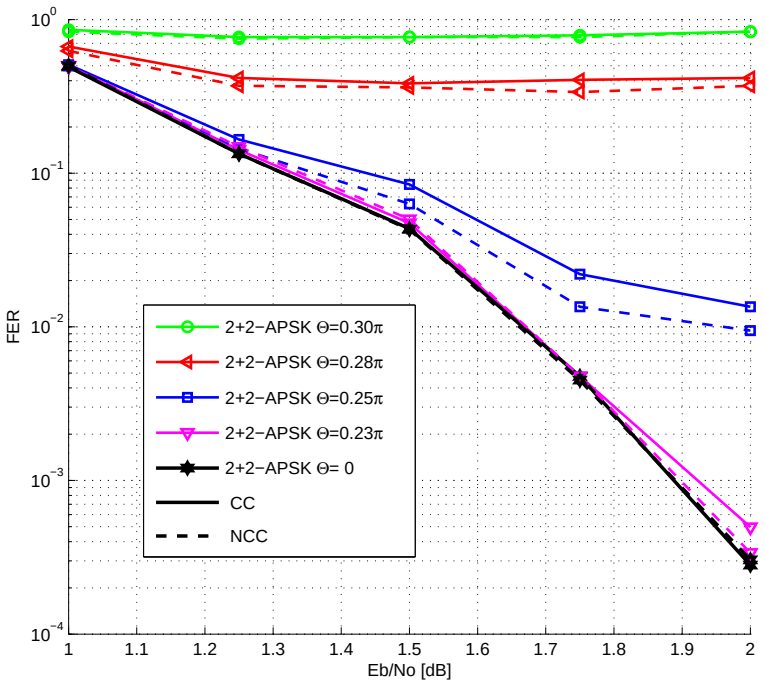
(b) 4+4-APSK

Figure 4.10: APSK constellations that enable tracking of larger phase offsets.

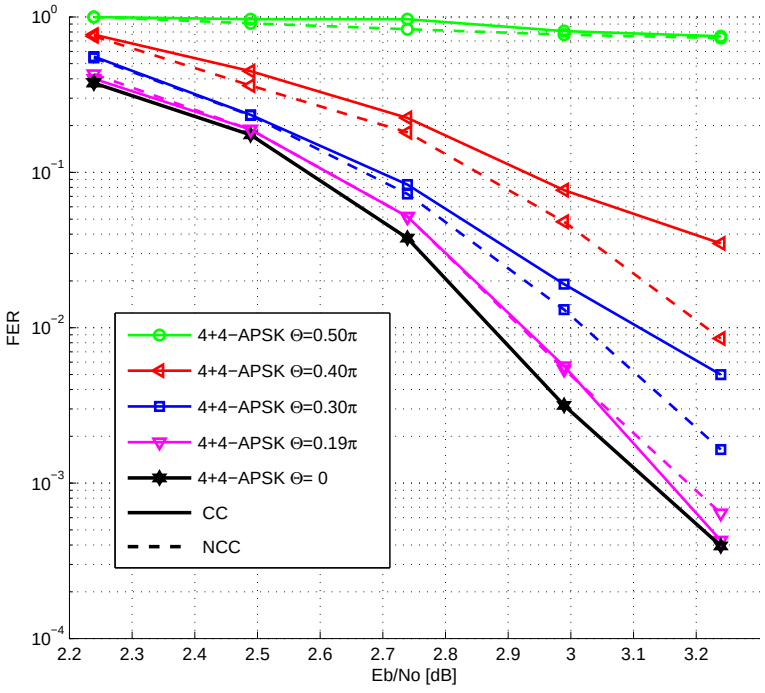
increases the maximum phase tracking angle. This perturbation produces amplitude phase-shift keying (APSK) modulations like such as the ones shown in Fig. 4.10. The APSK constellations considered in this work have a rotational invariance $\psi = \pi$. The corresponding FER performances are shown in Fig. 4.11(a) for the 2+2-APSK case and in Fig. 4.11(b) for the 4+4-APSK. The overall energy of the constellation was preserved by equally increasing and decreasing the power of the different constellation points. For the 4+4-APSK case, the outer points were moved from the unit circle to $r_1 = 1.128$ and $r_2 = 0.738$. A similar technique was used for the 2+2-APSK case using $r_1 = 1.1832$ and $r_2 = 0.774$. A comparison between two MPSK constellations and the corresponding APSK constellations is shown in Fig. 4.12. For 2+2-APSK, we conclude that the maximum phase rotation that can now be tracked increase can be slightly increased from 0.19π to 0.23π at the expense of a significant FER degradation for scenarios with lower carrier phase noise. However for the 4+4-APSK case, the proposed modification in the constellation points doubled the maximum phase rotation angle from 0.10π to around 0.19π without degrading the cases of lower carrier phase noise.

While it is always desirable to increase γ_c when possible, the overall phase tracking ability of the system is not limited the value of γ_c. Depending on the type of modulation used, techniques that use the DDCS algorithm can be implemented to track all possible values of $\theta_c \in \{-\pi, \pi\}$. One method for tracking offsets in $\{-\pi, \pi\}$ of a BPSK modulation was introduced in [34]. The decoder procedure begins by measuring the average power across a single codeblock of the signals v_{I_k} and v_{Q_k}. If the sine component (v_{Q_k}) has average power greater than the cosine component (v_{I_k}), then these two components are swapped. This procedure may leave (or induce) a remaining error of π radians. To resolve this ambiguity we run a single PLL pass followed by several (up to 4) LDPC iterations. The orientation that produces the maximum number of satisfied odd-degree check equations is selected and the decoding procedure is re-initialized [2]. Similar techniques are proposed in [44, 45]. For higher order modulations, a search method can be implemented

[2]Even degree checks remain satisfied under a rotation of all inputs by π.



(a) 2+2-APSK



(b) 4+4-APSK

Figure 4.11: FER Performance of APSK constellations that enable tracking of larger phase offsets.

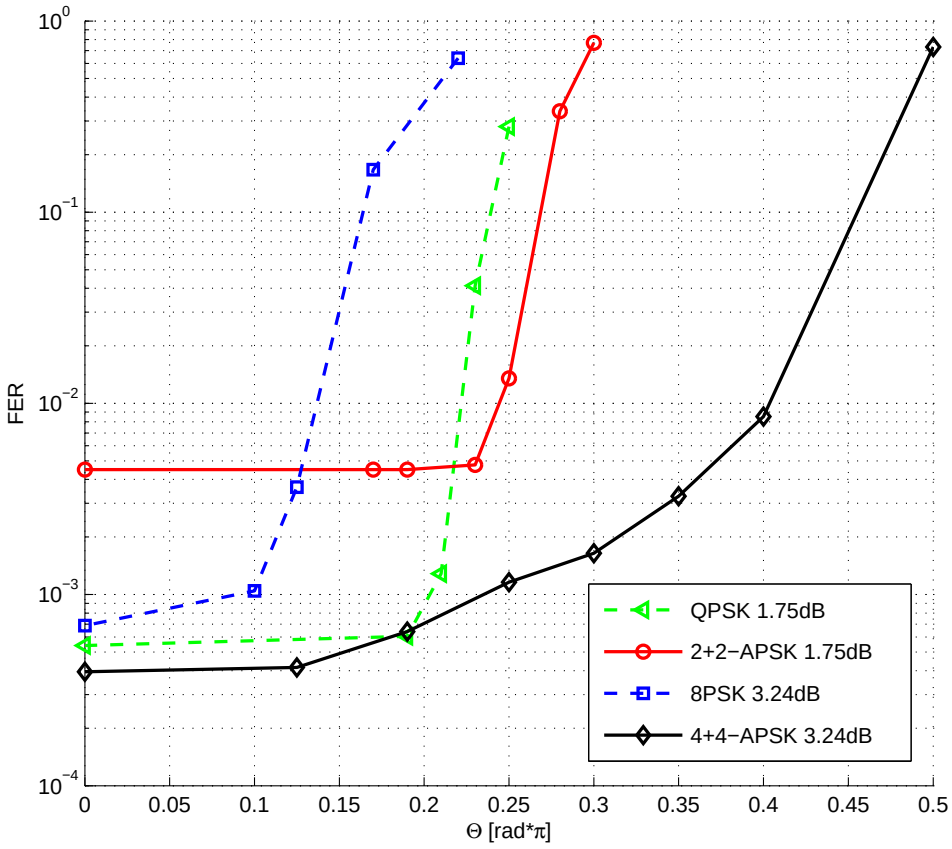


Figure 4.12: FER as a function of the carrier phase offset θ_c for a fixed E_b/N_0.

based on the window search method for symbol timing recovery presented in [50,51]. The carrier phase spectrum is divided into fractions of size γ_c. The search space will therefore consist of the intervals $\{[0, \gamma_c), [\gamma_c, 2\gamma_c), \ldots, [2\pi - \gamma_c, 2\pi)\}$. The received waveform is pre-rotated to fall into each potential interval before beginning the DDCS estimation. A small number of LDPC iterations can be performed for each potential interval and the one with the highest number of satisfied constraints is chosen to start the final DDCS estimation. The utility of this metric as a feedback mechanism is illustrated in Fig. 4.13, which shows the average percentage of satisfied constraints as a function of the absolute value of the carrier phase estimation error for different SNRs (E_b/N_0) and numbers of LDPC iterations. For example, in order to estimate the carrier phase offset for a QPSK system step size of $\gamma_c = 0.2\pi$ is chosen. A small number of iterations is performed in the

intervals $\{[0, 0.2\pi), [0.2\pi, 0.4\pi), \ldots, [1.8\pi, 2\pi)\}$. Three iterations are performed for each hypotheses which gives a total of 30 iterations to reduce the phase uncertainty to $|\theta_c| < \gamma_c$.

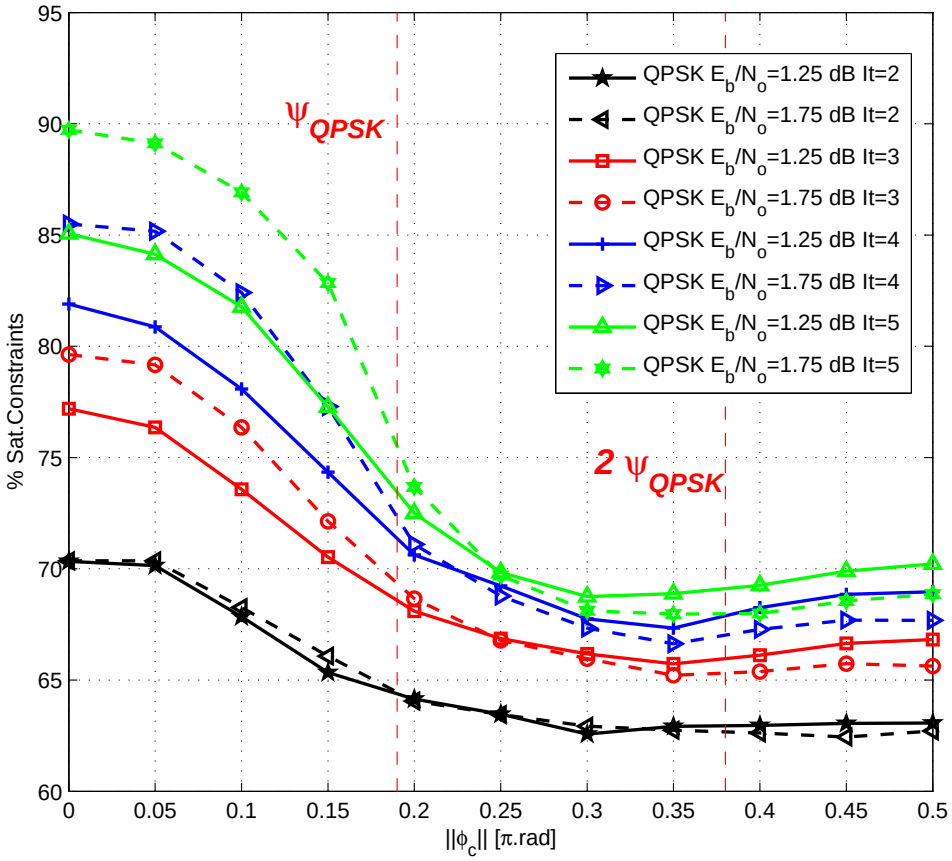


Figure 4.13: Percentage of satisfied constraints as a function of frequency estimation error. Curves for 2, 3, 4, and 5 LDPC iterations are shown for E_b/N_0 of 1.25 and 1.75 dB.

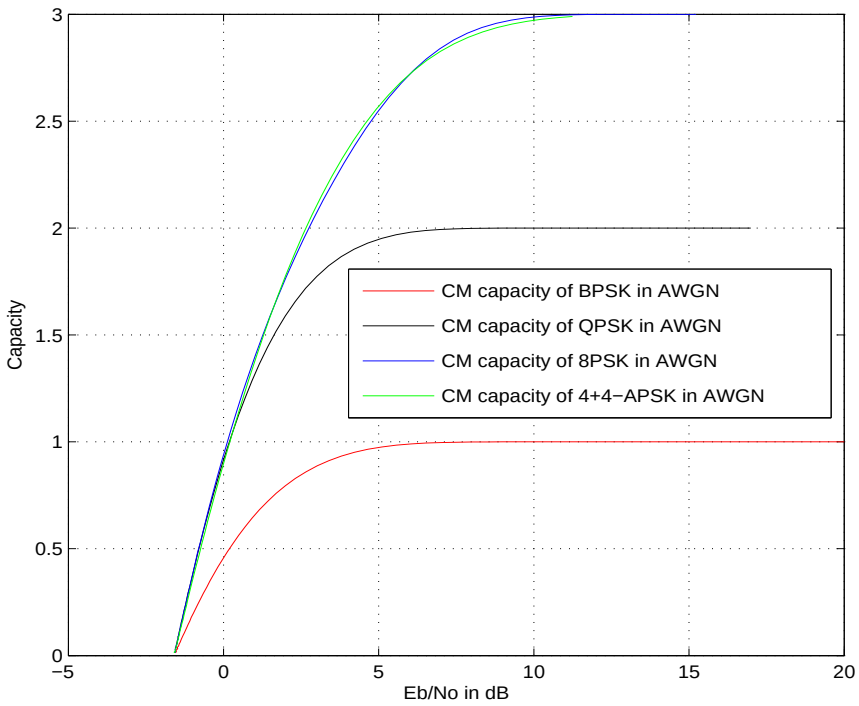
4.5 Conclusion

We have demonstrated a means for improving the carrier synchronization function for iteratively decoded MPSK using information derived from the decoder (decision-directed) to remove the modulation prior to the carrier tracking operation. The motivation for doing this is to overcome the penalty in noisy reference

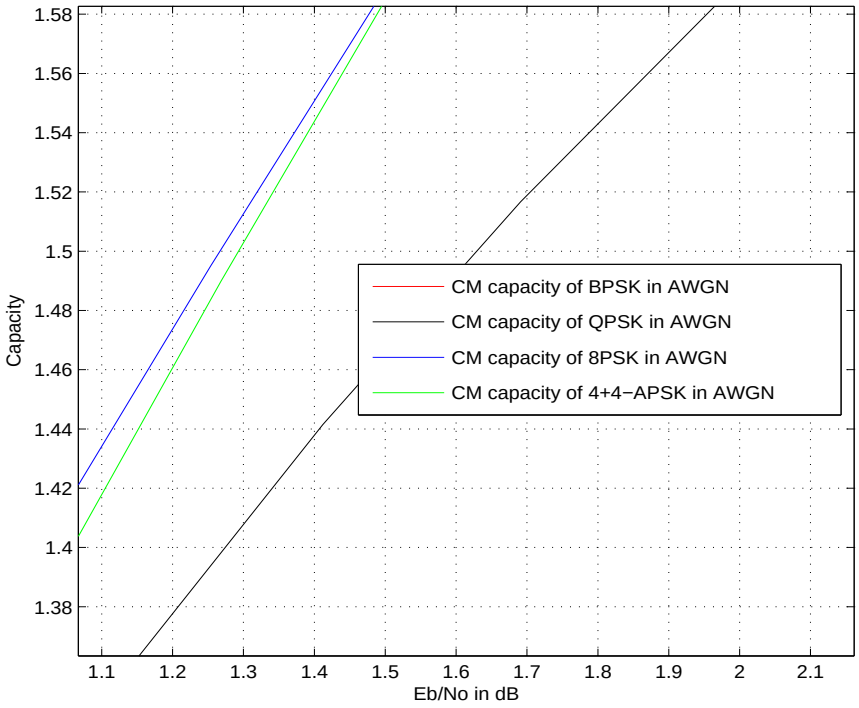
loss attributed to the large squaring loss at low SNRs that is characteristic of the traditional MPSK carrier sync loops such as the Costas-type loop. In contrast with the decision-directed carrier synchronization loop with hard decision feedback as proposed in [49] and [52], the scheme described in this work makes use of soft decision information and does not require estimating the decoder error probability. This occurs as a consequence of the assumption here of a fixed carrier synchronization structure, i.e., a PLL, whose design does not change with knowledge obtained from the decoder. While in the soft decision feedback case considered here such a structure would only be asymptotically optimum (in the MAP motivation sense) at high SNR, it nevertheless provides a simple yet performance-efficient carrier synchronization loop in SNR regions of interest for coded applications. The proposed architecture can be used in conjunction with other types of phase tracking loops in order to track residual carrier-phase errors at the decoding stage.

4.6 Future Work and Open Problems

A close look at Fig. 4.11(b) and Fig. 4.8 shows that 4+4-APSK outperforms 8PSK for all values of θ_c, even for $\theta_c = 0^o$. For the scenarios where θ_c is large this effect is due to the 4+4-APSK constellation having a larger value of ψ than the 8PSK case. On the other hand, for lower values of θ_c, where the carrier phase-estimation circuit does not play a fundamental role, we cannot explain the difference in performance with certainty. One possible explanation to this phenomena would be that the constrained capacity [8] of the 4+4-APSK constellation was greater than the 8PSK case at rate $\frac{1}{2}$. If this was the case, the 4+4-APSK constellation would require less SNR than its 8PSK counterpart. However, simulations performed using [53] show that for a code rate of $\frac{1}{2}$, 4+4-APSK requires an $E_b/N_o = 0.03\ dB$ *greater* than the 8PSK constellation. Fig. 4.14 and Fig. 4.15 show a graph of constrained capacity vs. SNR. Another possibility is that when iteratively decoding an LDPC code, it may be useful to have some of the incoming symbols with a stronger reliability (even if this implies reducing the energy of some symbols). The decoder could then use the *reliable*(*i.e.* with larger energy) information to

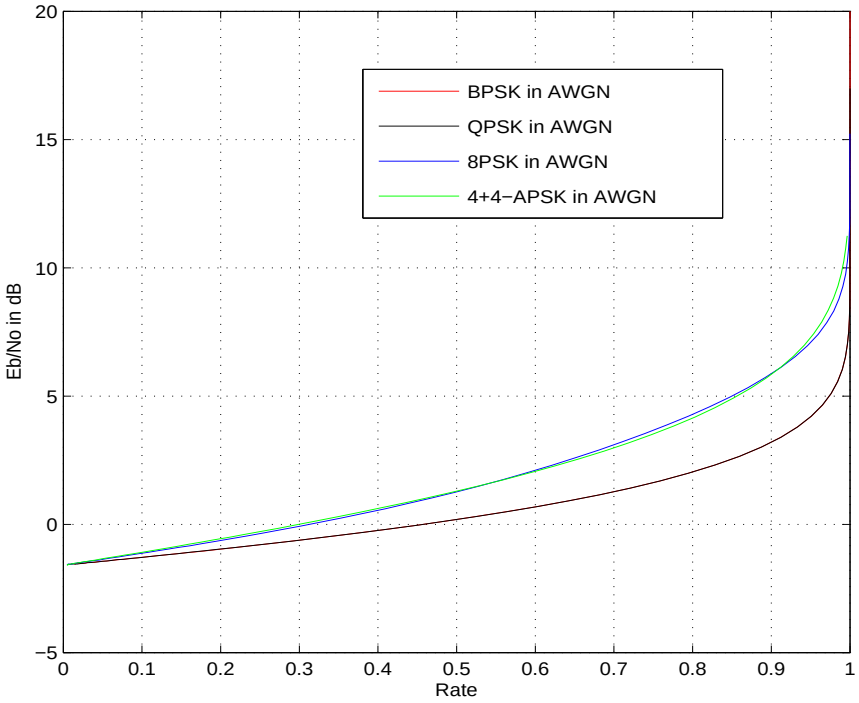


(a) Capacity vs E_b/N_o

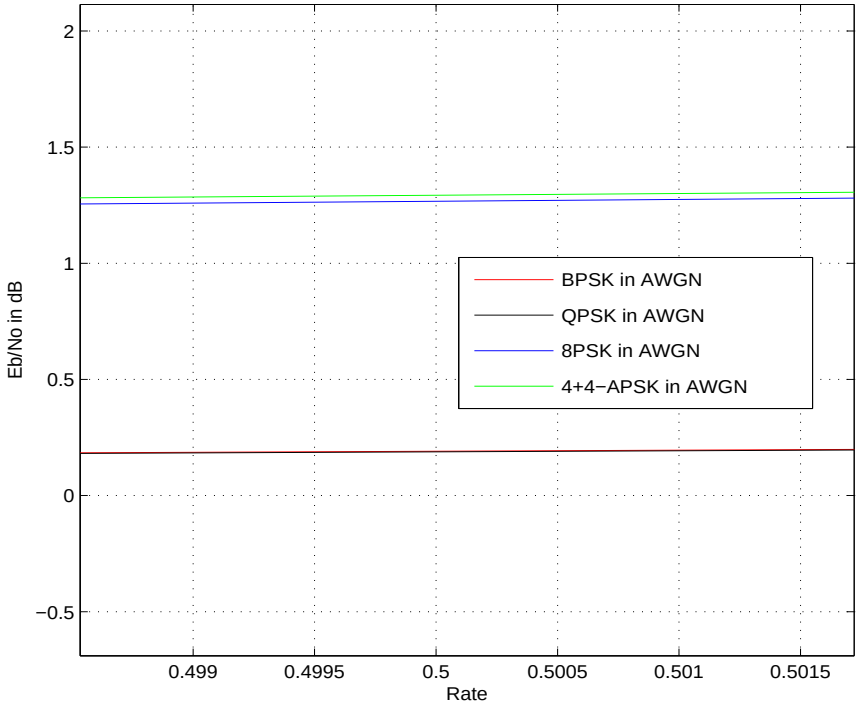


(b) Capacity vs E_b/N_o at C=1.5 bits

Figure 4.14: Constrained capacity for different constellations.



(a) Rate vs E_b/N_o



(b) CRate vs E_b/N_o at R=0.5

Figure 4.15: Rate vs E_b/N_o for different constellations.

compensate for the unreliable symbols located on the inner circle of the APSK constellation. In fact, the left degree distribution for the LDPC code used is $\{\alpha_2 \approx 0.47, \alpha_3 \approx 0.36, \alpha_9 \approx 0.15, \alpha_{10} \approx 0.02\}$, so density evolution [10] also tells us that not all bits should be equally protected. However the proof of this conjecture remains an open problem.

Chapter 5

Joint Carrier and Timing Synchronization of BPSK via LDPC Code Feedback

In traditional receiver architectures, symbol acquisition and tracking are performed using phase lock techniques that are independent of the channel-code decoding process. In Chapter 3, feedback from the constraint-node side of a bi-partite graph is used to estimate symbol frequency and timing offset in a baseband pilotless transmission. In Chapter 4, soft information feedback from an LDPC decoder is used to recover carrier phase information under the assumption of perfect symbol timing. In this chapter we address the problem of joint carrier-phase and symbol timing recovery, specifically for the case of BPSK modulations. The theory and results presented can be easily extended to bi-dimensional modulations. The proposed system is able to perform within $0.2 \sim 0.3$ [dB] of the genie aided BPSK code performance with perfect knowledge of carrier phase and symbol timing.

The rest of this chapter is organized as follows. The next section provides a detailed description of the transmitter and receiver models and gives an overview of the joint parameter estimation process. In Section 5.2, the circuit for symbol timing estimation is introduced. A digital implementation of the carrier synchronization circuit is illustrated in Section 5.3. Section 5.4 presents numerical results derived

from a simulation of the BPSK scheme with a particular LDPC code. Finally, Section 5.5 documents our conclusions.

5.1 Transmitter and Receiver Models

On the transmitter side, we consider a baseband signal comprised of N root raised-cosine pulses $h_{RRC}(t)$, transmitted at multiples of a symbol interval T and scaled $d_i \in \{\pm 1\}$: $m(t) = \sum_{i=0}^{N-1} d_i h_{RRC}(t - iT.)$. Multiplication by a sinusoidal carrier signal yields the transmitted waveform:

$$y_{Tx}(t) = \sqrt{2P} m(t). \sin(w_c t), \tag{5.1}$$

where P is the signal power.

When symbol timing errors are present, the assumed time reference for the k^{th} sample at the receiver r_k differs from the corresponding time reference at the transmitter $r_k = m(kT_s + \tau_k)$. The timing error modalities considered in this work combine constant time offsets ($\tau_k = D$), random walks ($\tau_k = \tau_{k-1} + \mathcal{N}(0, \sigma_d^2) T_s$) and constant frequency offsets ($\tau_k = \tau_{k-1} + \frac{F_{PPM}}{10^6} T_s$) where T_s is the sampling period and the frequency offset F_{PPM} is measured in parts per million. The received waveform can be modeled as:

$$y_{Rx}(t) = \sqrt{2P} r(t). \sin(w_c t + \theta_c) + n(t) \tag{5.2}$$

where

$$\begin{aligned} r(t) &= \sum_{i=0}^{N-1} d_i h_{RRC}(t + \tau(t) - iT), \\ n(t) &= \sqrt{2}\left[N_c(t) cos(w_c t + \theta_c) - N_s(t) sin(w_c t + \theta_c)\right] \end{aligned}$$

θ_c is the carrier phase and $n(t)$ is a bandpass AWGN process.

The decoding circuit is shown in Fig.5.1. The input signal $y_{Rx}(t)$ is converted to

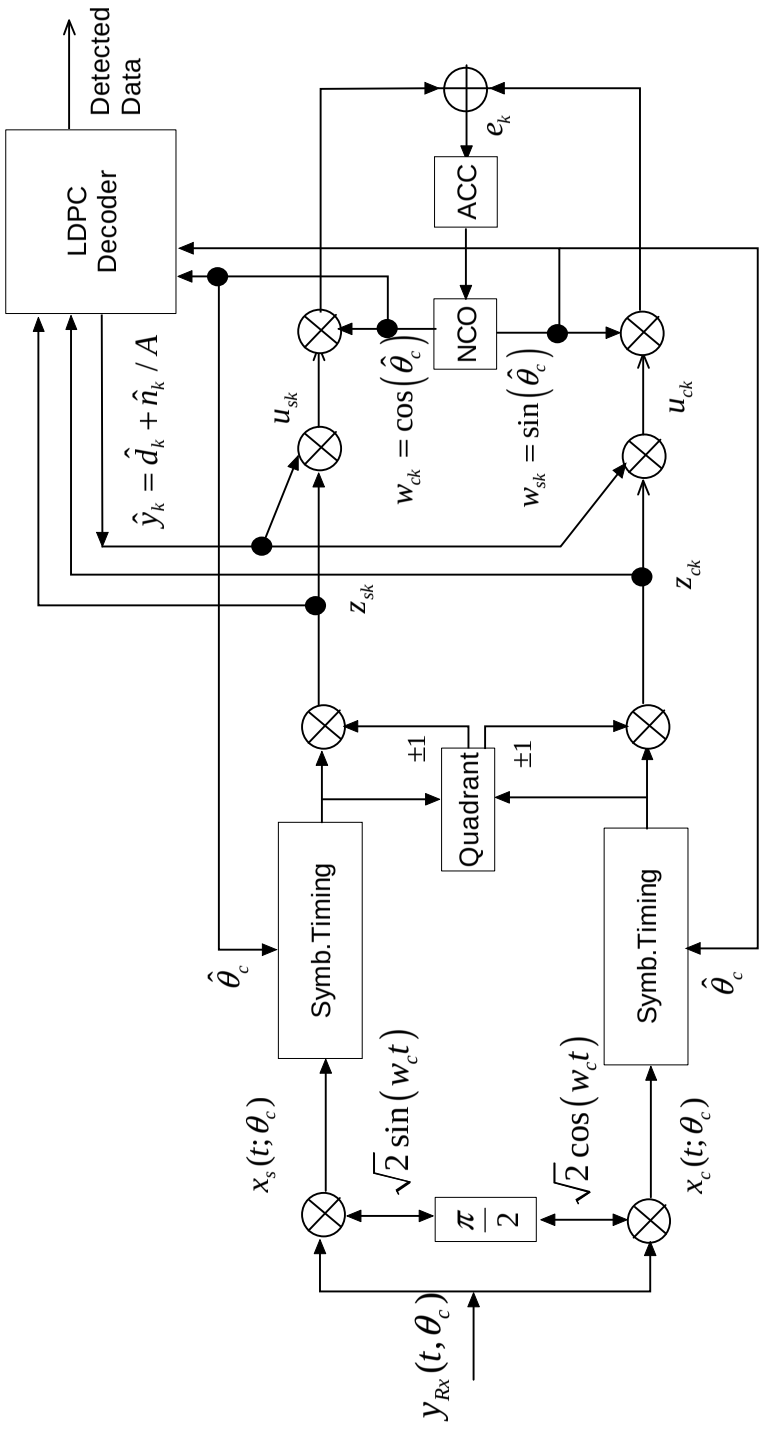


Figure 5.1: Digital Implementation of BPSK receiver.

baseband and low-pass filtered to remove frequencies at $2w_c$ which yields:

$$\begin{aligned} x_s(t) &= \sqrt{P}m(t)sin(\theta_c) + N_c(t)cos(\theta_c) - N_s(t)sin(\theta_c) \\ x_c(t) &= \sqrt{P}m(t)cos(\theta_c) - N_c(t)sin(\theta_c) - N_s(t)cos(\theta_c) \end{aligned} \tag{5.3}$$

The In-phase/Quadrature (I&Q) signal components in (5.3) are then sampled and matched filtered resulting in two digital signals:

$$\begin{aligned} z_{sk} &= \sqrt{P}T_s d_k sin(\theta_c) + N_{ck}cos(\theta_c) - N_{sk}sin(\theta_c) \\ z_{ck} &= \sqrt{P}T_s d_k cos(\theta_c) - N_{ck}sin(\theta_c) - N_{sk}cos(\theta_c) \end{aligned}$$

in the interval $kT_s \leq t \leq (k+1)T_s$.

The "symbol timing" recovery process described in Section 5.2 is now initialized. After the symbol-timing block corrects time delays, random walks and sampling frequency errors, parameter information is interchanged in an iterative fashion with the "carrier synchronization" block described in Section 5.3 to complete the parameter estimation process.

5.2 Symbol Timing Recovery

In Fig. 5.2 we illustrate the receiver architecture which exploits feedback from the LDPC decoder to manage timing errors. The received waveform is initially sampled at intervals of T_s and stored into a buffer. The interpolator computes interpolants at intervals of T_i using linear interpolation, which are then used for the matched filtering process [31]. In this work, we use $T_i = \hat{T}/2$ and $T_s = \hat{T}/4$, where $\hat{T}$ is the receiver-side assumption of the transmitter symbol period $\hat{T}$ (i.e. the symbol period that would be seen by the receiver in the absence of any timing perturbations).

The timing recovery circuit from Fig. 5.2 consists of two loops. Loop 1 is first executed to iteratively recover constant time phase and symbol-frequency offsets. The phase error estimator provides the interpolator (after the matched filter) with a time offset, which is used to correct the constant time delay. The symbol-frequency

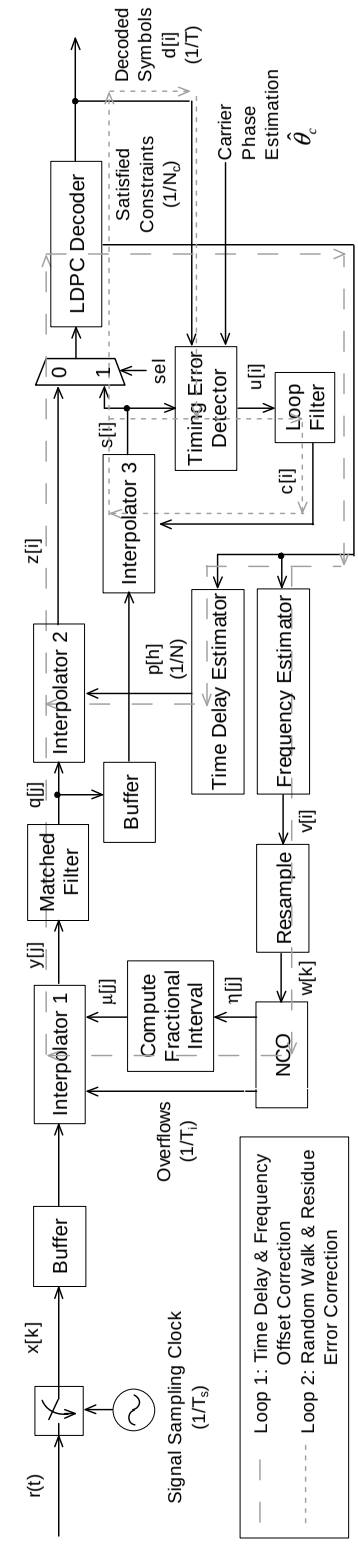


Figure 5.2: Generic 'Symbol Timing" block. Signals indexed with i, j, and k are at rate $1/T$, $1/T_i$, and $1/T_s$ respectively

estimator provides a frequency control word which is resampled at a rate of $1/T_s$ and fed to the numerically controlled oscillator (NCO).

Both the constant time delays and sampling frequency offset estimation processes use information from the iterative channel decoder based on the percentage of satisfied LDPC constraints. The utility of this metric as a feedback mechanism is illustrated for the case of symbol-frequency offsets in Fig. 5.3, which shows the average percentage of satisfied constraints as a function of frequency estimation error for different SNRs (E_b/N_0) and numbers of LDPC iterations. A similar plot, with similar tradeoffs, can be constructed for the relationship between the constant time delay estimation error and satisfied LDPC constraints. More interestingly, Fig. 5.3 indicates the rate of falloff as the estimation error increases, and shows that the best frequency error discrimination occurs for errors within approximately 200 ppm. This information is used in determining the step size to use in the frequency offset search. Fig. 5.3 also indicates the costs (in computations) and benefits (in increasing the percentage of satisfied constraints) of increasing the number of iterations.

In [51] phase and symbol-frequency estimates are generated in an iterative fashion using a window search method. An initial window and step size are chosen and a fixed number of LDPC iterations are performed at each hypothesis point. For example, in order to estimate a symbol-frequency offset of $\pm$2000 ppm (i.e. $\pm$0.2%) an initial step size of 400 ppm is used with three decoder iterations for each offset hypothesis. The window is then re-centered to the point with the highest number of satisfied constraints. The window step size is reduced by half to 200 ppm and three LDPC iterations are performed again for each new point. The process is repeated a third time with a resolution of 100 ppm. For this example, the method in [51] utilizes a total of $11[\text{points}] \times 3[\text{windows}] \times 3[\text{Iter. per point}] = 99[\text{iterations}]$ to correct an offset of $\pm$2000 ppm. In this work, in order to correct the same sampling frequency offset, a fixed step size of 250 ppm, with 3 LDPC iterations per point, was used. Instead of re-computing the window center and size, an interpolation technique generates the final frequency estimate based on the points with the highest percentage of satisfied constraints. This allows a reduction of the

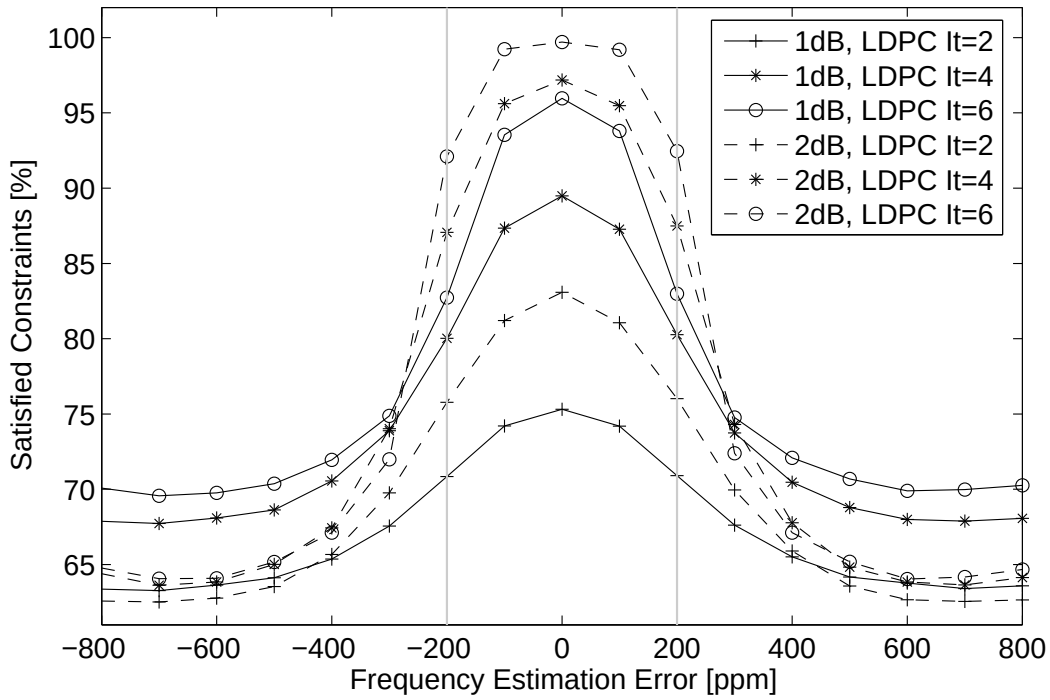


Figure 5.3: Percentage of satisfied constraints as a function of frequency estimation error. Curves for 2, 4, and 6 LDPC iterations are shown for E_b/N_0 of 1 and 2 [dB]. An average of 500 trials were used for each data point.

total number of iterations (16[points] $\times$ 3[Iter. per point] = 48[iterations]) by a factor of two. As long as the frequency offset is contained within the initial search window, the algorithm will converge with an accuracy that increases with increasing SNR. The complexity of this method grows linearly with the width of the range of frequency offsets contained in the initial search window. It is possible to track waveforms where both time delays and symbol-frequency offsets are present at the cost of quadratic computational complexity. A two-dimensional search strategy can be employed where for a given time delay candidate, the satisfied constraints of all frequency offset candidates are computed.

After large-scale phase and frequency errors have been identified in loop 1, loop 2 is used to handle random walks, correct residual time delay and sampling frequency errors, and to perform the remaining LDPC decoding. A conventional first-order PLL-based circuit with a decision-directed Mueller-Müller timing error

detector (M&M TED) [27] is used in loop 2. After every LDPC iteration, the M&M TED is provided with the symbols decoded by the LDPC decoder, analogous to the approach used in the recent work of Barry *et al.* [25].

At this point, an updated version of the signals z_c and z_s is sent to the carrier-phase recovery loop to produce a new estimate $\hat{\theta}_c$. This information is then fed to the LDPC decoder to continue with the iterative parameter recovery process. From this point forward, every update from the carrier-phase estimation loop is followed by an update from "loop 2" in the symbol-timing circuit in an iterative fashion.

5.3 Carrier Phase Synchronization

The carrier recovery circuit for BPSK modulation used in this work is the data-aided circuit originally proposed in [34]. This circuit converts the received modulated carrier to an unmodulated carrier (pure tone) before applying it to a phase-tracking loop. This is done by multiplying z_{ck} and z_{sk} by the normalized soft decision feedback sample $\hat{y}_k = d_k + \hat{n}_k/\gamma$, where as before over a given iteration $\hat{n}_k$ are modeled as i.i.d. zero mean Gaussian RVs with variance σ^2. This is equivalent to the complex conjugate (CC) feedback method presented in Chapter 4. The result of this multiplication removes the modulation and produces:

$$\begin{aligned} u_{sk} = \; & z_{sk}\hat{y}_k = \sqrt{P}T_s sin(\theta_c) \\ & +[(d_k + \hat{n}_k/\gamma)(N_{ck}cos(\theta_c) - N_{sk}sin(\theta_c)) \\ & +\hat{n}_k/\gamma\sqrt{P}T_s d_k sin(\theta_c)] \\ = \; & \sqrt{P}T_s sin(\theta_c) + v_{sk}, \end{aligned}$$

$$\begin{aligned} u_{ck} = \; & z_{ck}\hat{y}_k = \sqrt{P}T_s cos(\theta_c) \\ & +[(d_k + \hat{n}_k/\gamma)(-N_{ck}sin(\theta_c) - N_{sk}cos(\theta_c)) \\ & +\hat{n}_k/\gamma\sqrt{P}T_s d_k cos(\theta_c)] \\ = \; & \sqrt{P}T_s cos(\theta_c) + v_{ck}, \end{aligned}$$

which is then input to a digital PLL whose number-controlled oscillator (NCO)

produces an estimate of the carrier phase denoted by $\hat{\theta}_{ck}$. Multiplying u_{ck} and u_{sk} by $w_{sk} = sin(\hat{\theta}_{ck})$ and $w_{ck} = cos(\hat{\theta}_{ck})$, respectively, and then differencing the results of these products provides the error signal:

$$\begin{aligned} e_k &= u_{sk}w_{ck} - u_{ck}w_{sk} \\ &= \sqrt{P}T_s sin(\phi_{ck}) + v_{sk}cos(\hat{\theta}_{ck}) - v_{ck}sin(\hat{\theta}_{ck}) \end{aligned}$$

where as before $\phi_{ck} = \theta_{ck} - \hat{\theta}_{ck}$ denotes the phase error in the loop.

5.3.1 Loop SNR Performance

The performance of carrier-phase synchronization loops is commonly expressed as a function of the "loop SNR" (L_{SNR}). For a PLL based system, this can be expressed as:

$$L_{SNR}^{PLL} = \frac{1}{\sigma_{\phi_c}^2} = \rho_{PLL} = \frac{P_c}{N_o B_L} \tag{5.4}$$

where P_c is the carrier power, N_o is the noise PSD and B_L is the loop bandwidth [32].

The degradation of this performance in the case of BPSK is represented by a quantity called the "squaring loss", which is a measure of the degradation of the receiver signal-to-noise (SNR) ratio and is associated with the mean-squared phase error of the loop. At low symbol SNR, the squaring loss of an I&Q loop, such as the Costas loop, can be severe enough to prevent tracking:

$$L_{SNR}^{Costas} = \frac{1}{\sigma_{\phi_c}^2} = \rho_C . S_{L_C} = \frac{P_t}{N_o B_L}\left(1 + \frac{1}{2R_d}\right)^{-1} \tag{5.5}$$

where P_t is the total transmitted power, N_o is the noise PSD, B_L is the loop bandwidth and R_d is data SNR at the input of the receiver. Note that (5.5) is independent of the iteration process.

If the data sequence and its timing parameters were completely known, then a BPSK signal could be converted to a pure tone simply by multiplying the BPSK signal by the data waveform. One could then track the unmodulated carrier with

improved performance by use of a PLL, which from (5.4) we see that it does not exhibit squaring loss. Short of complete knowledge of the data waveform and in the presence of noise, the best approximation of a pure tone could be obtained by feeding back an estimate of the data waveform corresponding to tentative decisions on the data symbols.

Although initially available data-waveform estimates ($\hat{y}_k$) are generally of low quality, they can be used to initiate the carrier synchronization process by reducing the number of data transitions at the input. Once phase lock is achieved, the improved phase estimates can be fed back to the data detector, yielding improved symbol estimates for feedback, and thereby achieving even better phase tracking. This iterative process eventually leads to virtual elimination of squaring loss, so that the performance of the system approaches that of a phase-locked loop operating on an unmodulated carrier signal. For the proposed system we have that:

$$L_{SNR} = \frac{P_T}{N_o B_L}\left(1+\frac{\sigma^2}{\gamma^2}\right)^{-1} = \frac{P_T}{N_o B_L}\left(1+\frac{2}{\gamma}\right)^{-1} \tag{5.6}$$

where γ^2/σ^2 represents the decoder soft-estimate of the data SNR, and the last equality follows from the "symmetry condition" [54] for LDPC and turbo codes. In (5.6) we can see that as the iteration proceeds, the estimated data SNR increases and likewise the squaring loss decreases and approaches unity. By comparison, for a Costas loop, the expression for the squaring loss in (5.5) remains fixed, independent of the iteration process, for a given symbol SNR.

5.4 Numerical Results

We have evaluated the performance of the all-digital BPSK baseband approach, assuming perfect knowledge of the carrier frequency and simulating the signals in (5.3). Joint parameter estimation and decoding was performed using a rate-1/2 (1944, 972) irregular LDPC code developed in [33] and currently in the IEEE 802.11n standard. After a complex rotation to resolve phase ambiguity (discussed below), the signals z_c and z_s are multiplied by the decoder output $\hat{y}$ to form u_c

and u_s. As described in previous sections and shown in [34], if the PLL input has a small fraction of total modulated symbols in a block successfully removed, then it can begin to produce a reasonable phase estimate, even at relatively low SNRs. We have found that the estimation/decoding process can be successfully started by assigning $\hat{y} = z_s$ (Subsequent iterations derive $\hat{y}$ from the decoder). After this assignment, the PLL in the carrier synchronization loop operates once across all symbols in a codeword. LDPC decoder log-likelihood ratio inputs are then produced by combining the updated PLL phase estimates with z_c and z_s

$$\begin{aligned} Q_k &= \tfrac{2}{\sigma_{llr}^2}\left(z_{sk}w_{ck} + z_{ck}w_{sk}\right) \\ &= \tfrac{2}{\sigma_{llr}^2}\left(\sqrt{P}T_s d_k cos(\phi_c) - N_{ck}sin(\phi_c) - N_{sk}cos(\phi_c)\right) \end{aligned}$$

where

$$\sigma_{llr}^2 = PT_s^2/(2E_s/N_o).$$

An "extrinsic" LLR feedback mechanism was employed in which prior LDPC inputs are subtracted from current outputs before new inputs (from the most recent PLL update) are added. Also, state information in the decoder (in particular the most recent extrinsic information arriving from check-nodes) is preserved between LDPC-to-PLL-to-LDPC iterates. Each new carrier phase estimate is immediately followed by an update from loop 2 in Fig.5.2.

We conclude this section by noting that phase ambiguity (for offsets greater than $\pm\pi/2$ can be resolved by first measuring the average power across a single codeblock of the signals z_c and z_s. If the sine component (z_s) has average power greater than the cosine component (z_c), then these two components are swapped. This procedure may leave (or induce) a remaining error of π radians. To resolve this ambiguity we run a single PLL pass followed by several (up to 4) LDPC iterations. The orientation that produces the maximum number of satisfied odd-degree check equations is selected and the decoding procedure is reinitialized [1]. Similar techniques are proposed in [45, 44].

Results in Fig.5.4 for a carrier phase offset $\phi = \theta - \hat{\theta} = \pi/4$, a symbol-frequency

[1] Even degree checks remain satisfied under a rotation of all inputs by π.

offset of $\pm$2000ppm, a time delay of $\pm 0.5T$ and a random walk of $\sigma_d/T = 0.5\%$ show a degradation of $0.2 \sim 0.3$ [dB] from the code performance where carrier phase and symbol timing are known perfectly.

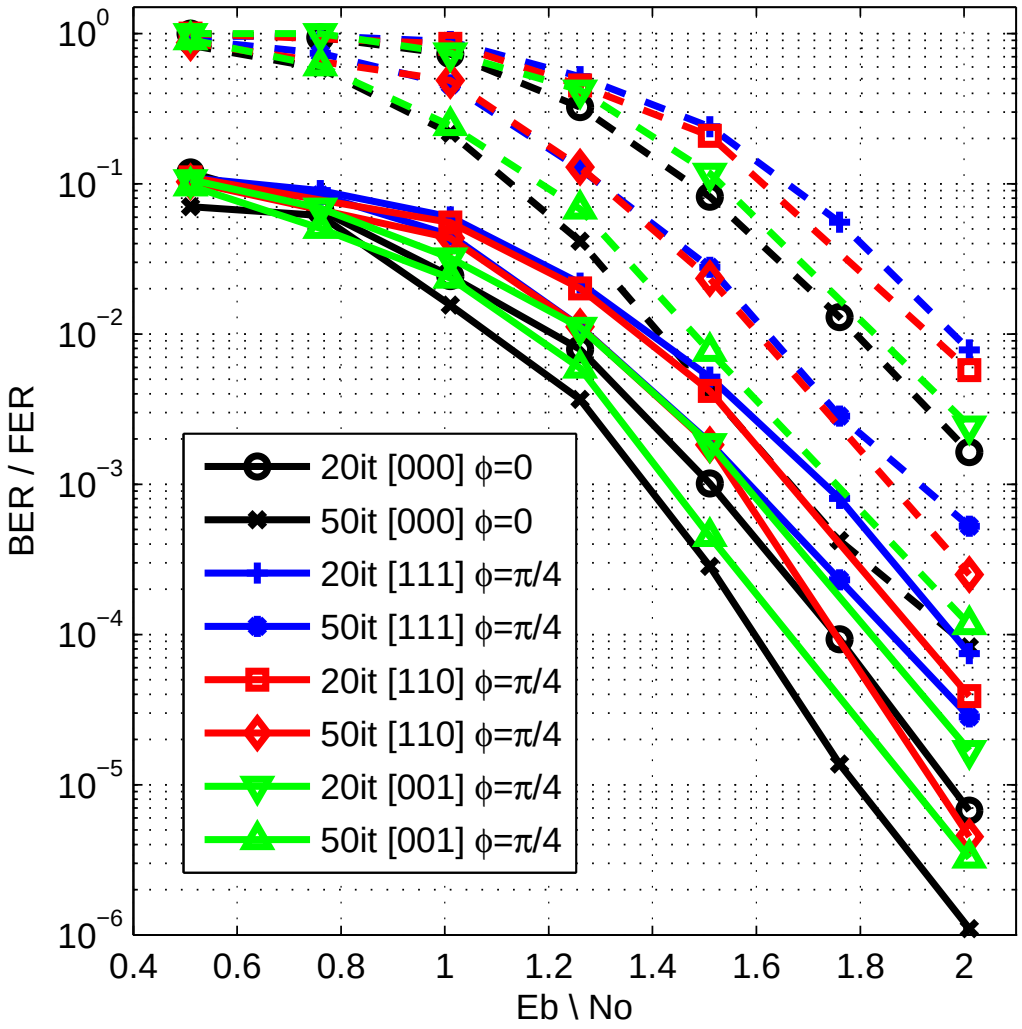


Figure 5.4: BER(solid)/FER(dashed) performance. Legend format [XYZ] indicates the presence/absence of: (X=1)$\rightarrow$Symbol-frequency offset, (Y=1)$\rightarrow$Time delay, (Z=1)$\rightarrow$Random walk. In all cases $\phi = \pi/4$.

5.5 Summary

We have demonstrated a means for improving the symbol timing and carrier-phase estimation for iterative decoded BPSK using information derived from the decoder extrinsics. For carrier synchronization, the signal modulation is removed prior to the carrier tracking operation. The motivation for doing this is to overcome the penalty in noisy reference loss attributed to the large squaring loss at low SNRs that is characteristic of the traditional BPSK carrier sync loops such as the Costas-type loop. The scheme described in this chapter makes use of soft decision extrinsic information and does not require estimating the decoder error probability. A pilotless symbol timing recovery architecture for tracking time delay, frequency offsets and random walks using LDPC feedback was also presented. The complexity of this window search method reduces by half the number of iterations needed in [51]. Performance within $0.2 \sim 0.3$ [dB] of the ideal code performance can be achieved for large time delays, frequency timing offsets and any carrier phase offset.

Chapter 6

Hamming and Reed Solomon codes as Rate-Efficient Array Codes for Burst Error Correction

Array codes are error-correcting codes of very low complexity that were initially used for burst error and burst erasure correction in Redundant Arrays of Inexpensive Disks (RAID) architectures and storage applications. Their algebraic structure is analogous to Reed Solomon (RS) codes with symbols defined over Galois rings rather than Galois fields. This allows very desirable properties like simple encoding and decoding but decreases the rate. For single phased burst correction, non-binary Hamming codes maximize the possible code rate and can be decoded with similar complexity as array codes. For multiple-burst correction, RS codes offer the same error correcting capability as array codes with a higher code rate. In this chapter we propose using Hamming and RS codes for array code applications thus increasing the code rate at the expense of a slight increase in complexity.

6.1 Introduction

Burst error correcting codes are used in many fields such as multi-track storage, satellite communications and disk arrays. Array codes [13, 55, 56], Fire codes [57,

58] and Reed-Solomon codes [59, 58] are well-known codes that have good burst-error-correcting capabilities. The main advantage of using array codes for burst correction is that the encoding and decoding only requires the use of simple bit manipulation such as bit shifting, reducing the overall complexity [11]. Therefore when the implementation complexity is an issue and hardware efficient encoders and decoders are needed, array codes are an attractive option.

In this chapter we show that Hamming codes and RS codes can be used for the same applications as array codes, achieving a higher rate with a similar decoding algorithm at the expense of a slightly higher encoding and decoding complexity. The increase in rate is significant for short code lengths and decreases as the rate of the code approaches unity. For the case of single burst correction, an error burst of length 6 requires an array code with $[n; k] = [42, 30]$ and a code rate of $R = 0.714$. The same burst can be corrected with a $[n; k] = [390, 378]$ Hamming code with $R = 0.969$ thus obtaining a rate increase of 35.7%. As n increases, and the code rate approaches one, the gain in rate decreases. To correct a burst of length 22, the gain in code rate decreases to 9.51%. In the case of multiple burst correction, for two bursts of length 6 the gain in rate is 118.75% and for bursts of length 22 the rate gain falls to 21%.

In the next section an introduction to array codes is presented. Array codes are compared with Hamming and RS codes in Section 6.3. The encoding and decoding algorithms are presented in Section 6.4. Concluding remarks are made in Section 6.5.

6.2 Array codes

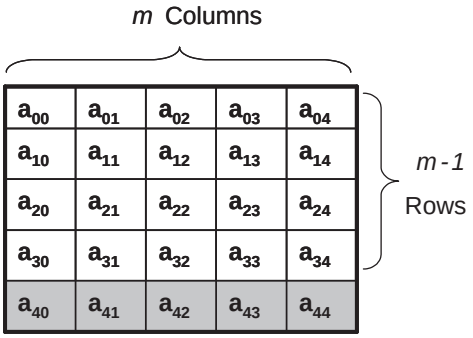
Array codes refer to a general class of algebraic error-correcting codes for use in detecting and correcting error bursts [11, 12, 13]. Each codeword is represented as a rectangular array. The dimensions of this array vary according to the error correcting capability of the code. However, in every case the array dimensions are $m \times m$ where m is a prime number. In this work when referring to an "array code" we assume a code with the structure of [11] as opposed to [12]. We assume, for

illustrative purposes, that each column in the array represents a disk. In RAID-3 or RAID-4 type of architectures [60, 61], a number of disks (columns in the array) carry data and one or more disks (columns) carry parity information.

To allow a simple algebraic description, the m^{th} row of the array is set to zero by design [11]. This row carries no information and is commonly described in the literature as the "imaginary row". The effective dimension of the array codes considered is therefore $(m-1) \times m$. Error correcting codes are defined in terms of the total number of information bits k, and the codeword length n. The parity-check matrix H of a block code has dimensions $(n-k) \times n$. For an array code both n and k are uniquely determined by m. An array code that corrects e bursts has $n = (m-1)(m)$, $k = (m-1)(m-2e)$ and rate $R = k/n = (m-2e)/m$. Fig. 6.1(a) illustrates a rectangular array for the case of $m = 5$ and $e = 1$. For this example, single burst correction requires that the first three columns of the array carry data information and the last two parity information.

Two different types of error bursts can be corrected using array codes. The first case consists of correcting bursts of length $L \leq (m-1)$ that occur within one column of the array. This type of error is called a *phased error burst* [58] since the burst cannot spread across columns of the array. From the disk array point of view, this is equivalent to having errors on a single disk of the array. Array codes can correct all error bursts of this type [11]. The second type of burst is a *non-phased error burst*. In this case the only constraint is that the length of the burst is $L \leq (m-1)$. As its name indicates, errors can start on one column (disk) of the array and propagate to a neighboring column (disk).

A syndrome-decoding analysis provides insight into the efficiency of array codes. In the event of a phased error burst, the resulting syndrome vector $s = r \cdot H^T$ is unique and contains all the available information of the error event. Fig. 6.1 shows the parity-check matrix for an array code with the structure from [11] where block columns consist of an identity matrix I_{m-1} on top of another $(m-1) \times (m-1)$ sub-matrix. Phased bursts excite only one block column in the parity check matrix. The syndrome s may therefore be decomposed into the first $m-1$ bits $s[1 : m-1]$, which are exactly the error burst and the remaining $m-1$ bits $s[m : 2(m-1)]$,



a_{00}	a_{01}	a_{02}	a_{03}	a_{04}
a_{10}	a_{11}	a_{12}	a_{13}	a_{14}
a_{20}	a_{21}	a_{22}	a_{23}	a_{24}
a_{30}	a_{31}	a_{32}	a_{33}	a_{34}
a_{40}	a_{41}	a_{42}	a_{43}	a_{44}

(a) Array

$$H = \left[\begin{array}{cccc|cccc|cccc|cccc|cccc} a_{00} & a_{10} & a_{20} & a_{30} & a_{01} & a_{11} & a_{21} & a_{31} & a_{02} & a_{12} & a_{22} & a_{32} & a_{03} & a_{13} & a_{23} & a_{33} & a_{04} & a_{14} & a_{24} & a_{34} \\ \hline 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ \hline 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}\right]$$

(b) Parity-check matrix

Figure 6.1: (a) Array code for $m = 5$. The grey shaded row is the "Imaginary row". (b) Corresponding parity-check matrix.

which must be used to identify the position of the burst. In this way, the array code facilitates correction of any single phased burst.

A binary block code generates 2^{n-k} distinct syndromes. This implies that at most $2^{n-k} = 2^{2(m-1)}$ distinct error events (including the "no-error" error event) can be corrected. However, for array codes there are only $m \cdot (2^{(m-1)})$ phased errors possible. Hence the algebraic structure of array codes although beneficial from a practical point view, does not use all possible syndromes given its code dimensions for phased burst correction. These extra syndromes facilitate correction of some, but not all, non-phased bursts. Correcting non-phased bursts was

not a requirement for the RAID application for which array codes were originally designed.

We assume a systematic encoder for the array codes in which the k data bits are grouped in $m-2e$ columns and the $2e(m-1)$ parity bits are grouped together in $2e$ columns. By viewing the columns as symbols in a Galois ring, array codes can be interpreted as cyclic codes over the polynomial ring $R_m = GF(2)/(x^m-1)$ [55,56]. This structure gives the code many desirable properties like algebraic decoding and a low-density structure [4, 55, 62]. However, as we mentioned earlier, it prevents the code from using all possible syndromes for phased-error correction causing a reduction in code rate. Hamming and RS codes have a similar algebraic structure to array codes with the difference that symbols are defined over a Galois field. The algebraic manipulations typically required to decode this type of code have higher complexity than the ones used for array-code decoding. On the other hand, the use of codes defined over a Galois field permits a larger fraction of the possible syndrome values to be used for correcting phased bursts, causing an overall rate growth.

Raphaeli pointed out in [63] that although array codes cannot correct all non-phased bursts, the probability of finding a burst that cannot be corrected decreases with the code length. This is possible by making use of "extra" syndromes. Correcting non-phased error bursts might be beneficial. However, if the specific purpose of the code is to correct only phased error bursts, the non-phased correction capability introduces a needless reduction in rate. Fig. 6.2 shows the difference between the number of phased errors and the possible syndromes that can be generated from the parity-check matrix as a function of m. As m increases the difference between the number of elements in the ring and in the field grows exponentially. This explains the results in [63] where it was noted that the probability of finding uncorrectable non-phased error bursts decreases with the code length.

The following sections present a decoding method for Hamming and RS codes based on array code decoding that allows these rate-efficient codes to be used at the expense of a slight increase in decoder complexity [64].

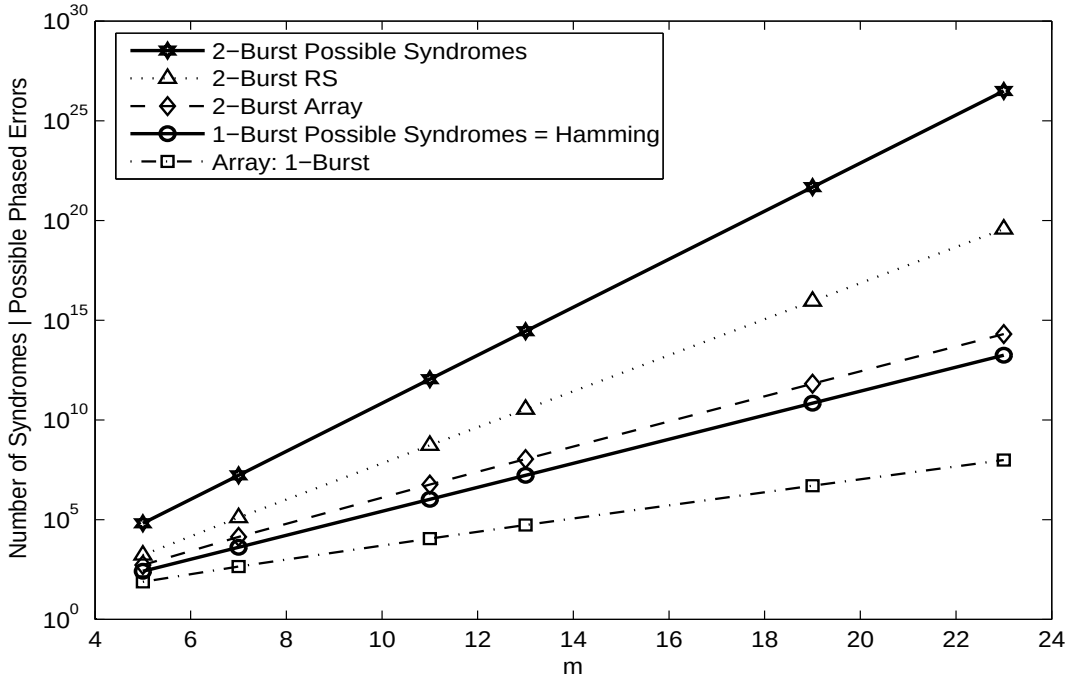


Figure 6.2: Comparison of maximum phased errors and total possible syndromes.

6.3 Hamming and RS codes for burst correction

Hamming codes were the first major class of linear binary codes designed for error correction [1]. The parameters for the family of binary Hamming codes are typically expressed as a function of a single integer $m_h \geq 2$, not necessarily prime. A Hamming code on $GF(2)$ has code length $n = 2^{m_h} - 1$, message length $k = 2^{m_h} - 1 - m_h$, redundancy $n - k = m_h$ and error correcting capability $t = 1$ bit. The parity-check matrices for binary Hamming codes has each nonzero binary m_h-tuples appearing once as a column. The same approach is used to construct $q-$ary non-binary Hamming codes. There are exactly $(q^{m_h} - 1)$ distinct nonzero $q-$ary m_h-tuples, but not all pairs of these m_h-tuples are linearly independent. For each $q-$ary m_h-tuple there are $(q-1)$ distinct nonzero m_h-tuples that are multiples of that m_h-tuple, pairs of which are clearly dependent. The $q-$ary Hamming code parity-check matrix H is constructed by selecting exactly one m_h-tuple from each set of multiples. This can be done by selecting as columns of H all distinct $q-$ary

m_h-tuples for which the uppermost nonzero element is 1. The parity-check matrix thus has $n = (q^{m_h} - 1)/(q - 1)$ columns and defines a $q-$ary Hamming code with $k = (q^{m_h} - 1)/(q - 1) - m_h$ and error-correcting capability $t = 1$ symbol of m_h bits. Out of all possible Hamming codes defined over a Galois field, we consider the ones with $m_h = n - k = 2$ and $GF(q = 2^{m-1})$ since they have the same block structure as array codes with identity sub-matrices on the top row. This results in a $[n; k] = [2^{m-1} + 1; 2^{m-1} + 1]$ Hamming code where every symbol has $m - 1$ bits.

Consider the case of $m = 5$ as an example to compare the properties of array codes with Hamming codes. For this example the field where the Hamming code is defined is $GF(q = 2^{(m-1)} = 16)$. In $GF(16)$, every nonzero element has a multiplicative order that divides 15. An element may have order 1, 3, 5, or 15. An element with order 15 is primitive. We can construct $GF(16)$ with the primitive polynomial $p(z) = z^4 + z + 1$, and the primitive element α as a root of this polynomial. The elements of $GF(16)$ can be found in Table 6.1. With I as the $(m - 1) \times (m - 1) = 4 \times 4$ identity matrix, the binary parity check matrix for a Hamming code defined on $GF(16)$ is

Element		Polynomial in α		V_{α^i}
α	=	α	=	0 0 1 0
α^2	=	α^2	=	0 1 0 0
α^3	=	α^3	=	1 0 0 0
α^4	=	$\alpha + 1$	=	0 0 1 1
α^5	=	$\alpha^2 + \alpha$	=	0 1 1 0
α^6	=	$\alpha^3 + \alpha^2$	=	1 1 0 0
α^7	=	$\alpha^3 + \alpha + 1$	=	1 0 1 1
α^8	=	$\alpha^2 + 1$	=	0 1 0 1
α^9	=	$\alpha^3 + \alpha$	=	1 0 1 0
α^{10}	=	$\alpha^2 + \alpha + 1$	=	0 1 1 1
α^{11}	=	$\alpha^3 + \alpha^2 + \alpha$	=	1 1 1 0
α^{12}	=	$\alpha^3 + \alpha^2 + \alpha + 1$	=	1 1 1 1
α^{13}	=	$\alpha^3 + \alpha^2 + 1$	=	1 1 0 1
α^{14}	=	$\alpha^3 + 1$	=	1 0 0 1
α^{15}	=	1	=	0 0 0 1

Table 6.1: Representation of elements of $GF(16)$. V_{α^i} indicates the binary vector representation of the element α^i

$$H_{Hamm} = \begin{bmatrix} 0 & I & I & I & \cdots & I \\ I & 0 & \alpha & \alpha^2 & \cdots & \alpha^{2^{(m-1)}-1} = \alpha^{15} \end{bmatrix} \alpha = \begin{bmatrix} 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}. \quad (6.1)$$

An array code with the same burst-correction properties has the following binary parity check matrix:

$$H_A = \begin{bmatrix} I & I & \cdots & I \\ \bar{\beta} & \bar{\beta}^2 & \cdots & \bar{\beta}^{m-1} = \bar{\beta}^4 \end{bmatrix}. \quad (6.2)$$

Since array codes are defined using an all-zero row in the $m \times m$ array that carries no information, $\bar{\beta}^i$ consists of the first $m-1$ rows and columns of

$$\beta^i = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}^i \Rightarrow \bar{\beta} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}. \quad (6.3)$$

In the case of array codes, β^i is equivalent to cyclically shifting the elements of β i-times to the left. This yields sub-matrices $\bar{\beta}$ with a low-density structure that have a maximum of 1 one per column and per row. This is not the case of Hamming codes where the α^i sub-matrices are non-structured. Note that although the Hamming code has two additional block columns that form a $I_{2(m-1)}$ identity, this construction still allows a decoding algorithm which is very similar to the one used for array codes. The rate of a Hamming Code code is $R_{\text{Hamming}} = (2^{m-1} - 1) / (2^{m-1} + 1)$. The increase in rate compared to original array codes is shown in Fig. 6.3. As was mentioned in Section 6.1, there is a great increase in code rate for short code lengths but the gain is reduced as the code length grows. It is important to note that while this technique increases the rate of the code, the *density of errors* per

block that can be corrected decreases. The maximum length of a correctable error burst remains constant $L \leq m-1$, but the codeword length increases from $m(m-1)$ to $(2^{m-1}+1)(m-1)$. For a disk array application, this implies that we could add more data disks for a given number of parity disks.

The RS codes are a well-known family of multiple-error correction [58, 65]. RS codes over $GF(q)$ are defined by a $(d-1) \times n$ check matrix:

$$H_{RS} = \begin{bmatrix} I & \alpha^{b} & \alpha^{2b} & \cdots & \alpha^{(n-1)b} \\ I & \alpha^{b+1} & \alpha^{2(b+1)} & \cdots & \alpha^{(n-1)(b+1)} \\ \vdots & \vdots & \vdots & \cdots & \vdots \\ I & \alpha^{b+d-2} & \alpha^{2(b+d-2)} & \cdots & \alpha^{(n-1)(b+d-2)} \end{bmatrix}. \tag{6.4}$$

The design parameters are α which is an element of $GF(q)$ of order n, b is any integer $(0 \leq b < n)$ and d is an integer $(2 \leq d \leq n)$. If α is a primitive element of $GF(q)$, then we can use a RS code of length $n = q-1$ that is maximum distance separable (MDS) and since it achieves the Singleton bound has the maximum possible rate for its dimensions. For double burst correction, the choice $b = 0$, $d = 2e+1$ defines a RS code with the same structure of an array code but with a significant higher rate. An array code that corrects e error bursts with $e \geq 2$ has a parity-check matrix that can generate $2^{2e(m-1)}$ syndromes. Array codes use only $\begin{pmatrix} m \\ e \end{pmatrix} (2^{m-1})$ syndromes while RS codes use $\begin{pmatrix} 2^{m-1}-1 \\ e \end{pmatrix} 2^{(m-1)}$. The rate of a $[n,k] = [2^{(m-1)}-1, 2^{(m-1)}-1-2e]$ RS code that corrects e error bursts is $R_{RS} = (2^{m-1}-2e)\,/\,(2^{m-1})$. Fig. 6.3 shows that if a RS code is used to correct a burst of length 6 the gain in rate is 118.75% and for bursts of length 22 the rate gain falls to 21%. Although this code does not achieve the Hamming bound, it offers a significant increase in code rate and, as we show in the next section, it can use a similar decoding algorithm to the one used for array codes. This method remains simple only for short code lengths. However other decoding algorithms like Berlekamp-Massey or Euclid's algorithm [59] may be more appropriate for longer block lengths or when the number of bursts that need to be corrected is greater than 2.

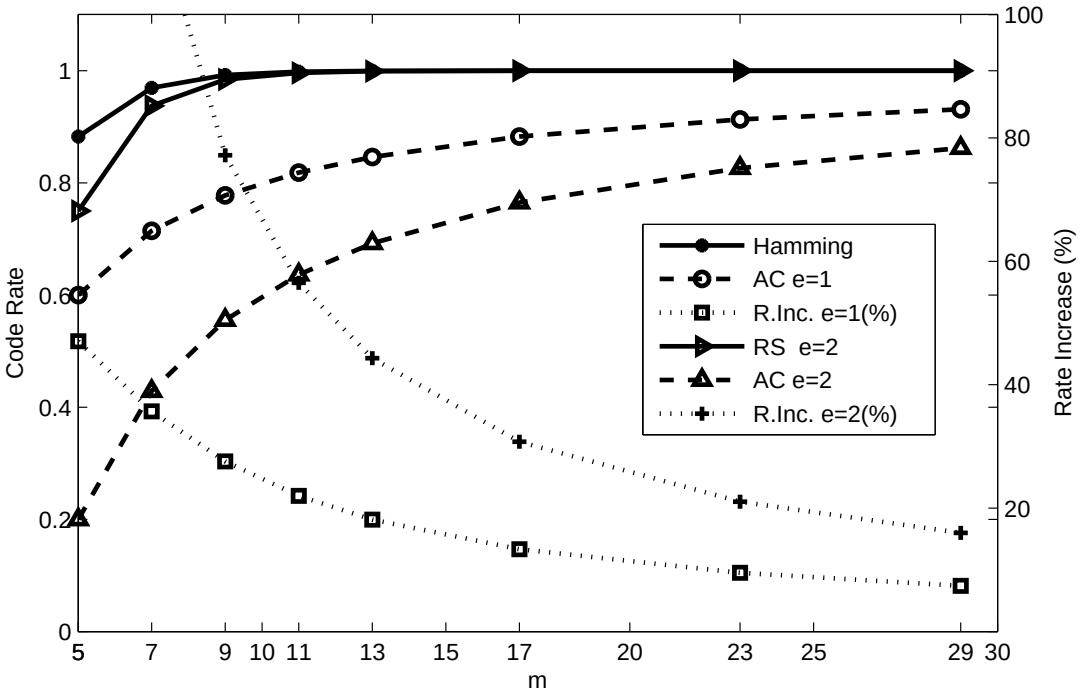


Figure 6.3: Comparison of code rates of array codes with Hamming and RS codes.

6.4 Encoding and Decoding Algorithms

6.4.1 Single-burst correcting codes

For the array codes in [11], parity bits on the last two columns of the array are computed such that a *horizontal* and *diagonal-parity* equations are satisfied. These equations are as follows:

- Horizontal Parity:

$$\sum_{j=0}^{m-1} a_{i,j} = 0 \qquad 0 \leq i \leq (m-2), \tag{6.5}$$

- Diagonal Parity:

$$\sum_{l=0}^{m-1} a_{\langle j-l \rangle_m, l} = 0 \qquad 0 \leq j \leq (m-2). \tag{6.6}$$

where $\langle k \rangle_n$ denotes k modulo n. We can see from the equations above that horizontal-parity bits and diagonal-parity bits are not independent. The encoding operation begins by encoding bits along a diagonal of the array that does not involve any horizontal-parity bits that have not yet been determined. For example, for the array depicted in 6.1(a) start with $j = 2$ for the diagonal-parity equation $a_{2,0} + a_{1,1} + a_{0,2} + a_{4,3} + a_{3,4} = 0$ and compute $a_{3,4}$. Note that the use of the "imaginary row" allows $a_{4,3} = 0$ to be in the equation, preventing and indexing problem. Once $a_{3,4}$ has been determined, compute the horizontal-parity equation for $i = 3$ and determine $a_{3,3}$. The process continues now with the diagonal equation with $i = 1$ and goes on until all parity bits are determined.

Let us compute the number of XOR operations required to determine the parity bits in an array code. There are $(m-1)$ horizontal and $(m-1)$ diagonal parity bits and each calculation requires $(m-2)$ XOR's. Therefore $2(m-1)(m-2)$ XOR's are required for encoding array codes that correct bursts of length $L \leq (m-1)$. Let us introduce some notation that will be useful to describe the encoding complexity of Hamming codes. Let the codeword $c = [v\ p_1\ p_2]$ be a concatenation of the information vector v of length $q - 1 = 2^{m-1} - 1$ symbols and two parity symbols p_1 and p_2, where each symbol has $m - 1$ bits. The parity-check matrix H_{Hamm} of the Hamming codes that we consider, can be divided into the following sub-matrices:

$$H_{Hamm} = \begin{bmatrix} H_1 & I & 0 \\ H_2 & 0 & I \end{bmatrix}. \tag{6.7}$$

The sub-matrices H_1 and H_2 can be further divided into $q-1$ square sub-matrices of size $m - 1$. The sub-matrices of H_1 are identity matrices I_{m-1} and the sub-matrices of H_2 are the α^i, square sub-matrices mentioned in the previous section. In order to compute the parity symbol $p_1 = v.H_1^T$, a total of $k - 1$ symbol XOR's which corresponds to $(2^{(m-1)} - 2)(m - 1)$ XOR bit operations are required. The second parity symbol $p_2 = v.H_2^T$, requires for $\left(2^{(m-1)} - 2\right)(m-1)^2/2$ bit XOR's. This implies that a Hamming code that uses $2(m - 1)$ parity bits requires a total of $\left(2^{(m-1)} - 2\right)(m^2 - 1)/2$ bit XOR operations.

The Hamming and array codes considered have the same burst correction capacity but different code lengths. In order to compare the number of operations required to encode these codes, we use as a reference the number of information bits used in a Hamming code that corrects bursts of length $L \leq (m-1)$. This is equivalent to fixing the number of disk units used for storage of parity information in a disk array. Given two redundant disks with $(m-1)$ bits each, a Hamming code defined over $GF(q = 2^{(m-1)})$ can use these disks to protect $2^{(m-1)} - 1$ information disks. To protect the same number of information using an array code, $\left(2^{(m-1)} - 1\right) / (m-2)$ arrays of size $(m-1) \times m$ are needed. The total number of bit XOR's required to encode this number of array codes is $2(m-1)\left(2^{(m-1)} - 1\right)$. The increase in XOR operations for Hamming codes is approximately equal to $(m+1)/4$.

For example if $m = 17$, we can use a $[n; k] = [1048592; 1048560]$ Hamming code, with 32 parity bits and a rate of nearly 1. To encode the same number of bits using an array code, we need 4369 array codes of rate 0.88 that use a total of $4369 \cdot 32 = 139808$ parity bits. To encode the 4369 array codes $4369 \cdot 480 = 2097120$ bit XOR's are needed. The corresponding Hamming code uses 9436896 bit XOR's which represents an increase of complexity by a factor of 4.5. This increase in encoding computation is a result of generating the bits corresponding to symbol p_2, since the computation complexity required for p_1 is the same for both codes.

At the decoder, the error burst appears on the first $m-1$ bits of the syndrome. The position of the burst is derived by using the last $m-1$ bits of the syndrome. Since any of the n received symbols may be in error, the decoding complexity is of order $O\left(2^{m-1}\right)$. Algorithm 3 is actually an extension of the algorithm presented in [11] for the general case when the bottom half of the parity-check matrix has the structure mentioned above for Hamming codes. The bit shifts that array codes use to derive the error position are replaced by XOR operations required for binary vector multiplication. On each step of the decoding algorithm $(m-2)(m-1)$ XOR's are needed to generate the vector $\bar{v}$ used to determine the error position. On each iteration, for the sample case of $m = 17$, 240 XOR's are needed to obtain $\bar{v}$. Array codes use a shift register to create $\bar{v}$. The error position is equal to the

number of cyclic shifts i required so that $\text{Shift}(\bar{v}, i) = s(m : 2(m-1))$.

Algorithm 4 Hamming Code: Error Correcting Algorithm

```
1: s = r · H^T
2: if s ≠ 0⃗ then
3:    if s(1 : m − 1) = 0⃗ then
4:       Position=1;
5:    else if s(m − 1 : 2(m − 1)) = 0⃗ then
6:       Position=2;
7:    else
8:       for i= 1 to 2^{m−1} − 3 do
9:          if (s(1 : m − 1) · α^i) = s(m : 2(m − 1) then
10:            Position=i+2;
11:            Break ⇒(Solution Found)
12:         end if
13:      end for
14:   end if
15: end if
```

6.4.2 Multiple burst-correcting codes

For multiple burst-correcting array codes the number of parity bits is $2e(m-1)$. Similar to the single burst-correcting case $(m-2)$ bit XOR's are required to encode each bit, requiring a total of $2e(m-1)(m-2)$ XOR's for a length $n = (m-1)m$ code. RS codes can be encoded with a complexity that is linear with the code length. A systematic generator matrix can be derived from H_{RS}. A structure for this matrix cannot be easily expressed as a function of α^i preventing us from doing a detailed complexity comparison with array codes as we did in the previous section. However, we can see that the analysis is similar to the single burst-correcting case, with an increase in XOR operations for the RS encoding that is due to loosing the low-density property that array codes have.

The extension of the previous single burst-error-decoding algorithm for the case of e error bursts has complexity order $O(n^e)$. On each step there are between $2e$ and $4e$ matrix multiplications of binary vectors of length $(m-1)$. This complexity can be acceptable for short code lengths and double error correction. For cases of

higher complexity the algorithm becomes comparable to an exhaustive search and other well known algorithms like Berlekamp-Massey [11, 58] are a better option since decoding complexity does not depend on the code length. This increasing complexity problem also occurs for multiple burst-correcting array codes. As in the case of Hamming codes, both RS and array codes have the same order of decoding complexity. Algorithm 4 for double burst correction is shown below. It begins by diving the received syndrome into four vectors $s = [s_0 \cdots s_3]$ that are equal to different combinations of combinations of the error bursts: $s_k = \left(e_0\alpha^{k\cdot i} + e_1\alpha^{k\cdot j}\right)$ where i and j are the two error positions. For every combination of error positions, the algorithms solves for e_0 and e_1 using two equations and later verifies if the solution holds for all remaining equations. The index p can be found using the primitive polynomial of the Galois field where the RS code is defined.

Algorithm 5 RSCode: Double Error-Correcting Algorithm

$s = r \cdot H^T$
for k= 0 to $2e$ **do**
  $s_k = s\left(k(m-1)+1 : (k+1)(m-1)\right);$
end for
for i= 0 to $2^{m-1}-1$ **do**
  for j= 0 to $2^{m-1}-1$ **do**
    Find p such that: $\alpha^p = \alpha^i + \alpha^j$
    $\hat{e}_0 = \left(s_0\alpha^j + s_1\right)\alpha^{\left(2^{m-1}-1-p\right)}$
    $\hat{e}_1 = \left(s_0\alpha^i + s_1\right)\alpha^{\left(2^{m-1}-1-p\right)}$
    if $(s_2 = (\hat{e}_0\alpha^{2i} + \hat{e}_1\alpha^{2j})$ and $s_3 = (\hat{e}_0\alpha^{3i} + \hat{e}_1\alpha^{3j}))$ **then**
      Break $\Rightarrow$(Solution Found)
    end if
  end for
end for

6.5 Summary

The most rate-efficient codes that meet the single burst-error correction capability of array codes are Hamming codes. These codes can correct all phased single error bursts of length $L \leq (m-1)$. For the case of multiple burst-error corrections RS

codes are the more rate-efficient than an array code with the same error correction properties. Furthermore, both Hamming and RS codes have an encoding and decoding complexity comparable to array codes. The decoding procedures is an extension of the algorithm used to decode array codes where bit shifts are replaced by binary vector multiplications. Still this is more complex than what the original array codes require. For very short code lengths the gain in rate obtained is significant, but as the code length increases and the code rate approaches one, the rate increase is reduced.

Bibliography

[1] R. Hamming, "Error detecting and error correcting codes," *Bell Syst. Tech. J.*, vol. 26, no. 2, pp. 147–160, 1950.

[2] C. E. Shannon, "A mathematical theory of communication," *Bell Syst. Tech. J.*, vol. 27, pp. 379–423 and 623–656, July and Oct. 1948.

[3] R. M. Tanner, "A recursive approach to low complexity codes," *IEEE Trans. on Inform. Theory*, vol. 27, pp. 533–547, Sept. 1981.

[4] R. G. Gallager, "Low-density parity-check codes," *IRE Trans. Inform. Theory*, vol. IT-8, pp. 21–28, Jan. 1962.

[5] ——, *Low-density parity-check codes*. Cambridge, MA: MIT Press, 1963.

[6] C. Berrou, A. Glavieux, and P. Thitimajshima, "Near Shannon limit error-correcting coding and decoding: Turbo codes," *Proc. IEEE Int. Conf. on Comm. (ICC)*, pp. 1064–1070, May 1993.

[7] D. J. C. MacKay, "Good error-correcting codes based on very sparse matrices," *IEEE Trans. on Inform. Theory*, vol. 45, pp. 399–431, Mar. 1999.

[8] ——, *Infomration Theory, Inference, and Learning Algorithms*. Cambridge Universiy Press, 2003.

[9] M. Luby, M. Mitzenmacher, A. Shokrollahi, and D. Spielman, "Improved low-density parity-check codes using irregular graphs," *IEEE Trans. on Inform. Theory*, vol. 47, pp. 585–598, Feb. 2001.

[10] T. Richardson, A. Shokrollahi, and R. Urbanke, "Design of capacity approaching irregular low density parity check codes," *IEEE Trans. on Inform. Theory*, vol. 47, pp. 618–637, Feb 2001.

[11] M. Blaum and R. Roth, "New array codes for multiple phased burst correction," *IEEE Trans. on Inform. Theory*, vol. 39, no. 1, pp. 66–67, Jan. 1993.

[12] M. Blaum, J. Brady, J. Bruck, and J. Menon, "Evenodd: An efficient scheme for tolerating double disk failures in raid architectures," *IEEE Trans. on Computers*, vol. 42, no. 2, pp. 192–202, Feb 1995.

[13] M. Blaum, P. Farrell, and H. van Tilborg, *Handbook of Coding Theory*. V.S.Press and W.C.Huffman. Elsevier Science B.V., 1998, ch. 22.

[14] S. Haykin, *Digital Communications*. John Wiley & Sons, 1988.

[15] G. D. Forney, "Codes on graphs: normal realizations," *IEEE Trans. on Inform. Theory*, vol. 47, pp. 520–548, Feb. 2001.

[16] J. Pearl, *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, 1988.

[17] T. Tian, C. Jones, J. Villasenor, and R. D. Wesel, "Selective avoidance of cycles in irregular LDPC code construction," *IEEE Trans. on Comm.*, vol. 52, no. 8, pp. 1242–1247, Aug. 2004.

[18] T. Zhang, Z. Wang, and K. Parhi, "On finite precision implementation of low density parity check codes decoder," in *Proc. IEEE Int. Symp. on Circuits and Systems,*, vol. 4, pp. 202–205, May 2001.

[19] T. Richardson and R. Urbanke, "The capacity of low-density parity-check codes under message passing decoding," *IEEE Trans. on Inform. Theory*, vol. 47, pp. 599–618, Feb 2001.

[20] J. Chen, A. Dholakia, E. Eleftheriou, M. Fossorier, and X.-Y. Hu, "Near optimum reduced-complexity decoding algorithms for LDPC codes," in *Proc. IEEE Int. Sym. Inform. Theory*, p. 455, 2002.

[21] E.Eleftheriou, T. Mittelholzer, and A. Dholakia, "A reduced-complexity decoding algorithm for low-density parity-check codes," *IEE Electron. Letters*, vol. 37, pp. 102–104, Jan. 2001.

[22] J. Chen and M. Fossorier, "Near optimum universal belief propagation based decoding of LDPCcodes," *IEEE Trans. on Comm.*, vol. 50, no. 3, pp. 406–414, Mar. 2002.

[23] ——, "Density evolution for bp-based decoding algorithms of LDPC codes and their quantized versions," in *Proc. IEEE Global Telecomm. Conf.*, vol. 2, Nov. 2002, pp. 1378– 1382.

[24] D. Lee, W. Luk, J. Villasenor, and P. Cheung, "A Gaussian Noise Generator for Hardware-Based Simulations," *IEEE Trans. on Computers*, vol. 43, no. 12, pp. 1523–1534, Dec. 2004.

[25] J. Barry, A. Kavčić, S. McLaughlin, A. Nayak, and W. Zeng, "Iterative timing recovery," *IEEE Sig. Proc. Mag.*, vol. 21, no. 1, pp. 89–102, 2004.

[26] J. Liu, H. Song, and B.V.K. Vijaya Kumar, "Symbol timing recovery for low-SNR partial response recording channels," in *Proc. IEEE Global Telecomm. Conf.*, Nov. 2002, pp. 1129–1136.

[27] K. Mueller and M. Müller, "Timing recovery for digital synchronous data receivers," *IEEE Trans. on Comm.*, vol. 24, no. 5, pp. 516–531, Jun. 1976.

[28] J. Sun and M. Valenti, "Joint synchronization and SNR estimation for Turbo codes in AWGN channels," *IEEE Trans. on Comm.*, vol. 53, no. 7, pp. 1136–1144, Aug. 2005.

[29] L.E. Franks, "Carrier and bit synchronization in data communication - a tutorial review," *IEEE Trans. on Comm.*, vol. 28, no. 8, pp. 1107–1121, Aug. 1980.

[30] D. Kim, M. J. Narasimha, and D.C. Cox, "Design of optimal interpolation filter for symbol timing recovery," *IEEE Trans. on Comm.*, vol. 45, no. 7, pp. 877–884, Aug. 1997.

[31] F. L. Gardner, "Interpolation in digital modems – part I: fundamentals," *IEEE Trans. on Comm.*, vol. 41, no. 3, pp. 501–507, 1993.

[32] ——, *Phaselock Techniques*, 3rd ed. John Wiley and Sons, Inc., 2005.

[33] A.I. Vila Casado, W. Weng, and R. Wesel, "Multiple rate low-density parity-check codes with constant block length," in *Proc. IEEE Asilomar Conf. on Sig., Syst. and Comput.*, vol. 2, Nov. 2004, pp. 2010–2014.

[34] M. Simon, E. Valles, C. Jones, R. Wesel, and J. Villasenor, "Information-reduced carrier synchornization of BPSK and QPSK using soft decision feedback," *Proc. IEEE 44th Allerton Conf. on Comm., Control and Comput.*, Sep. 2006.

[35] N. Noels, V. Lottici, A. Dejonghe, H. Moeneclaey, M. Luise, and M. Vandendorpe, "A theoretical framework for soft-information-based synchronization in iterative (turbo) receivers," *EURASIP Journal on Wireless Communications and Networking*, pp. 117–129, 2005.

[36] A. Anastasopoulos and K. Chugg, "Adaptive iterative detection for phase tracking in turbo-coded systems," *IEEE Trans. on Comm.*, vol. 49, no. 12, pp. 2135–2143, Dec. 2001.

[37] I. Motedayen-Aval and A. Anastasopoulos, "Polynomial-complexity noncoherent symbol-by-symbol detection with application to adaptive iterative decoding of turbo-like codes," *IEEE Trans. on Comm.*, vol. 51, no. 2, pp. 197–207, 2003.

[38] B. Mielczarek and A. Svensson, "Phase offset estimation using enhanced turbo decoders," *Proc. IEEE Int. Conf. on Comm. (ICC)*, vol. 3, pp. 1536–1540, April 2002.

[39] R. Nuriyev and A. Anastasopoulos, "Pilot-symbol-assisted coded transmission over the block-noncoherent AWGN channel," *IEEE Trans. on Comm.*, vol. 51, no. 6, pp. 953–963, 2003.

[40] J. Dauwels and H.-A. Loeliger, "Phase estimation by message passing," *Proc. IEEE Int. Conf. on Communications*, pp. 523–527, June 2004.

[41] S. Howard and C. Schlegel, "Differentially-encoded turbo coded modulation with APP channel estimation," in *Proc. IEEE Global Telecomm. Conf.*, Dec. 2003, pp. 1761–1765.

[42] G. Colavolpe, G. Ferrari, and R. Raheli, "Noncoherent iterative (turbo) decoding," *IEEE Trans. on Comm.*, vol. 48, no. 9, pp. 1488–1498, 2000.

[43] G. Colavolpe, A. Barbieri, and G. Caire, "Algorithms for iterative decoding in the presence of strong phase noise," *IEEE J. Select. Areas Comm.*, vol. 23, no. 9, pp. 1748– 1757, Sept. 2005.

[44] W. Oh and K. Cheun, "Joint decoding and carrier phase recovery," *IEEE Comm. Letters*, vol. 5, no. 9, pp. 375–377, 2001.

[45] A. Burr and L. Zhang, "A novel carrier phase recovery method for turbo-coded QPSK system," *Proc. European Wireless (EW'02). Florence, Italy*, pp. 917–821, Feb. 2002.

[46] V. Lottici and M. Luise, "Embedding carrier phase recovery into iterative decoding of turbo-coded linear modulations," *IEEE Trans. on Comm.*, vol. 52, no. 4, pp. 661–669, 2004.

[47] C. Morlet, I. Buret, and M.-L. Boucheret, "A carrier phase estimator for multi-media satellite payloads suited to RSC coding schemes," *Proc. IEEE Int. Conf. on Comm. (ICC)*, vol. 1, pp. 455–459, June 2000.

[48] C. Langlais and M. Helard, "Phase carrier for turbo codes over a satellite link with the help of tentative decisions," *2nd International Symposium on Turbo Codes and Related Topics. Brest, France.*, vol. 5, pp. 439–442, 2000.

[49] M. Simon and V. A. Vilnrotter, "Iterative information-reduced carrier synchronization using decision feedback for low SNR applications," *TDA Progress Report*, vol. 42-130, Aug. 15, 1997. [Online]. Available: http://tmo.jpl.nasa.gov/progress_report/42-130/130A.pdf

[50] E. Valles, C. Jones, J. Villasenor, and C. Jones, "Carrier and timing synchronization of BPSK via LDPC code feedback," *Proc. IEEE Asilomar Conf. on Sig., Syst. and Comput.*, Nov. 2006.

[51] D. Lee, E. Valles, J. Villasenor, and C. Jones, "Joint LDPC decoding and timing recovery using code constraint feedback," *IEEE Comm. Letters*, vol. 10, no. 3, pp. 189–191, Mar. 2006.

[52] M. Simon and A. Tkacenko, "An iterative information-reduced QPSK carrier synchronization scheme using decision feedback for low SNR applications," *TDA Progress Report*, vol. 42-164, Feb. 15, 2006. [Online]. Available: http://tmo.jpl.nasa.gov/progress_report/42-164/164H.pdf

[53] M. Valenti, "Iterative solutions coded modulation library (ISCML)." [Online]. Available: http://www.iterativesolutions.com

[54] S. Chung, T. Richardson, and R. Urbanke, "Analysis of sum-product decoding of low-density parity-check codes using a Gaussian approximation," *IEEE Trans. on Inform. Theory*, vol. 47, pp. 657–670, Feb. 2001.

[55] J. L. Fan, *Constrained Coding and Soft Iterative Decoding.* Springer, 2001.

[56] ——, "Array codes as low-density parity-check codes," *Proc. of Second Int. Symposium on Turbo Codes (ISTC)*, pp. 423–426, Brest, France, Sept 4-7, 2000.

[57] R. Blahut, *Theory and Practice of Error Control Codes.* Addison-Wesley, 1983.

[58] S. Lin and J. D. Costello, *Error Control Coding: Fundamentals and Applications.* Prentice-Hall., 1983.

[59] S. Wicker, *Error Control Systems for Digital Communication and Storage.* Prentice Hall, 1995.

[60] D. Patterson and J. Hennessy, *Computer Architecture: A Quantitative Approach.* Kaufmann, 1996.

[61] G. Gibson, *Redundant Disk Arrays.* MIT Press, 1992.

[62] K. Yang and T. Helleseth, "On the minimum distance of array codes as LDPC codes," *IEEE Trans. on Inform. Theory*, vol. 49, no. 12, pp. 3268–3271, Dec. 2003.

[63] D. Raphaeli, "The burst error correcting capabilities of a simple array code," *IEEE Trans. on Inform. Theory*, vol. 51, no. 2, pp. 722–728, Feb. 2005.

[64] E. Valles, A.I. Vila Casado, M. Blaum, J. Villasenor, and R. Wesel, "Hamming codes are rate-efficient array codes," in *Proc. IEEE Global Telecomm. Conf.*, vol. 3, Dec. 2005, pp. 1320–1324.

[65] I. S. Reed and G. Solomon, "Polynomial codes over certain finite fields," in *Journal of the Society of Industrial and Applied Mathematics*, vol. 8, pp. 300–304, 1960.