Technical University of Crete
Department of Computer and Electronic Engineering

Forward Error Correction for experimental wireless ftp radio link over analog FM

Supervisor: Nikolaos Sidiropoulos
Committee: Athanasios Liavas
Alexandros Potamianos

Iliakis Evangelos

TABLE OF CONTENTS

TABLE OF FIGURES

C h a p t e r 1

THE WIRELESS FTP COMMUNICATION SYSTEM

1. INTRODUCTION

This Thesis is a part of a team project for designing, implementing and evaluating the Wireless FTP Communication System. The aim of this project was to develop an efficient and low cost experimental wireless radio link, between two personal computers over analog FM for data communication.

In particular the objective was to design and develop a software modem operating with the appropriate hardware equipment. The WFTP system is a computer based communication system mainly used for file transfer. The analog FM radio link was implemented by the use of an FM transmitter and the corresponding radio receiver. In order to achieve the transmission of bits between two personal computers over the wireless radio link, we had to transform the binary information into analog data. Thus the most important part of the software modem is the modulator and the demodulator units in the transmitter and the receiver respectively. However the software modulator generates the samples of the analog signal that conveys the binary information. Likewise the software demodulator processes the samples of the analog signal that conveys the binary information. Therefore the only missing parts of our system were the D/A and A/D converters in the transmitter and the receiver respectively. Because of the constraints on the budget and the overall cost of the system, we had to find a cheap and efficient way to generate the desired analog signal. Consequently, we used the playback and the recording features of the soundcards. In the transmitter, the sound card performs the digital to analog conversion and generates the audio signal which is transmitted through the low-power FM radio transmitter. Likewise in the receiver the sound card performs the analog to digital conversion of the analog received signal and produces the corresponding digital data.

So far we have concluded to a basic structure of the WFTP system consisting of six major components:

- ⇒ Desktop PC
- ⇒ Laptop PC
- ⇒ Software for the transmitter
- ⇒ FM transmitter
- ⇒ Software for the receiver
- ⇒ Radio receiver

A general block diagram of the WFTP system is illustrated in the following figures.

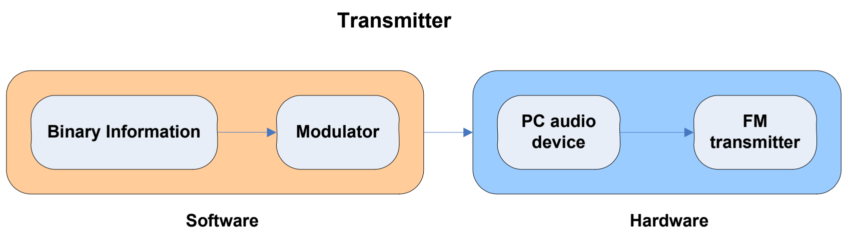


Figure 1-1 : Block diagram of the transmitter

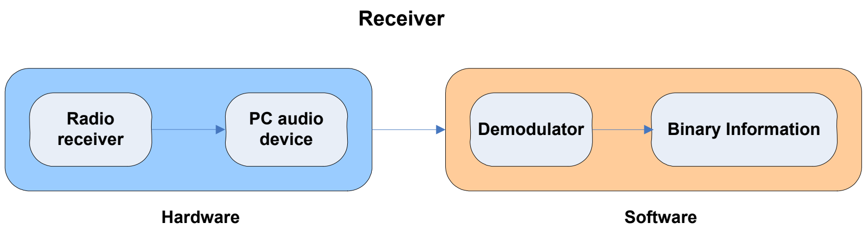


Figure 1-2 : Block diagram of the receiver

Considering this basic structure of the WFTP Communication system we defined the foremost objectives of our system:

- ⇒ Reliable file transfer.
- ⇒ Low bit error rate on the order of 10^{-6}
- ⇒ Achievement of the highest possible transfer rates.

In order to accomplish our goals we developed and implemented different software modules that were integrated in a completely operational communication system. In the subsequent sections it follows a thorough analysis of the hardware and software components of the WFTP system.

1.1 Hardware

The hardware equipment in WFTP system consists of two personal computers, a low-power radio transmitter, a dipole antenna and a radio receiver.

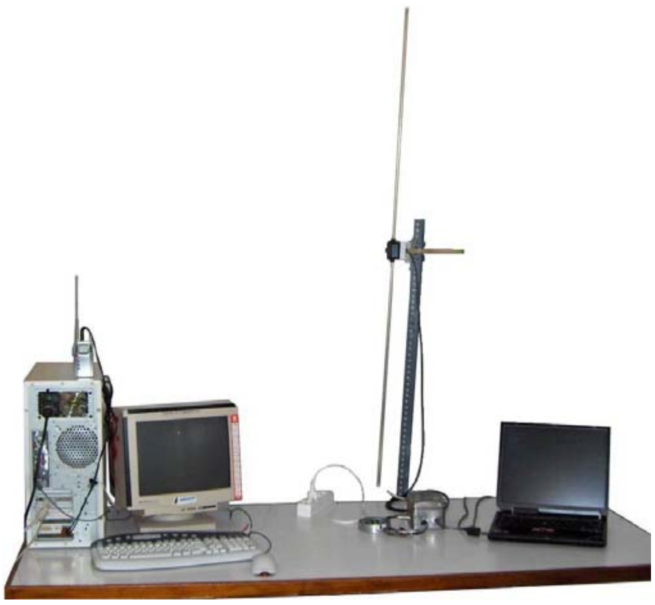


Figure 1-3 : The WFTP Communication system

⇒ FM transmitter specifications:
- Battery voltage DC4.5 V.
- Frequency Range 88 MHz ~ 108 MHz.
- Output power 1 W.
- Half wave dipole antenna

⇒ Wide Band Communications Receiver specifications:
- Frequency Range 0.1 ~ 1299.955 MHz.
- Antenna Impedance 50 Ω
- Battery voltage DC3.6V~DC6V
- Frequency Stability ± 5PPM (-10^{o}C ~ $_+$60^{o}C)

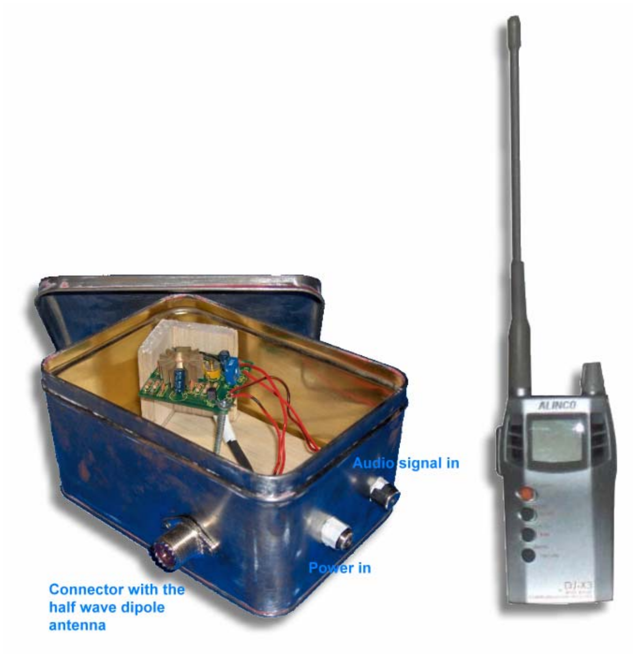
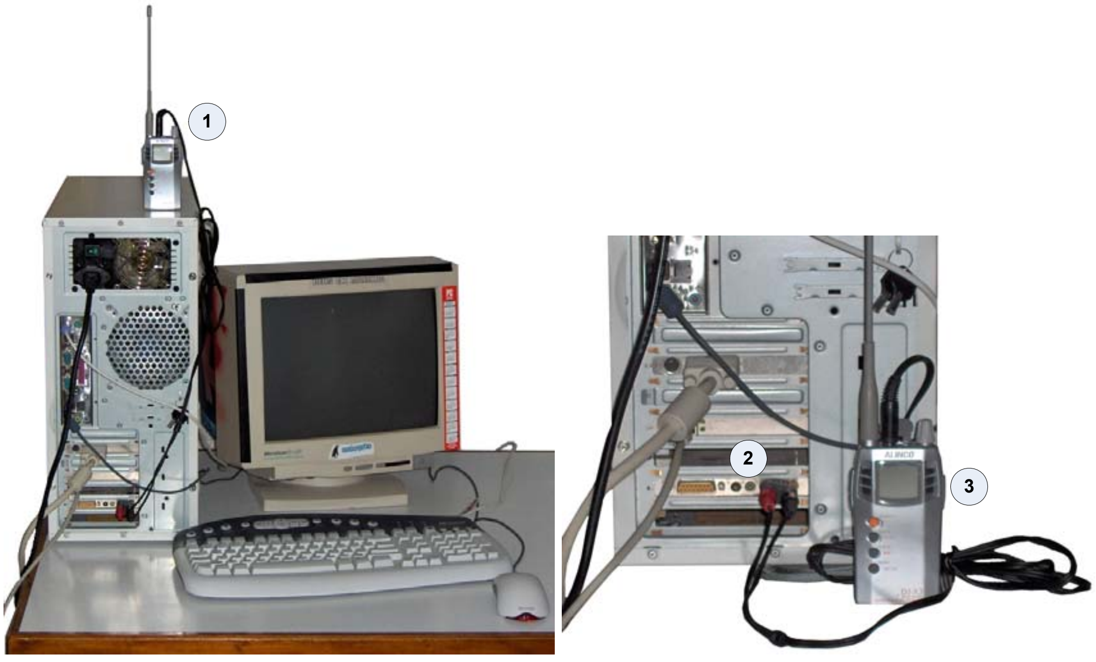


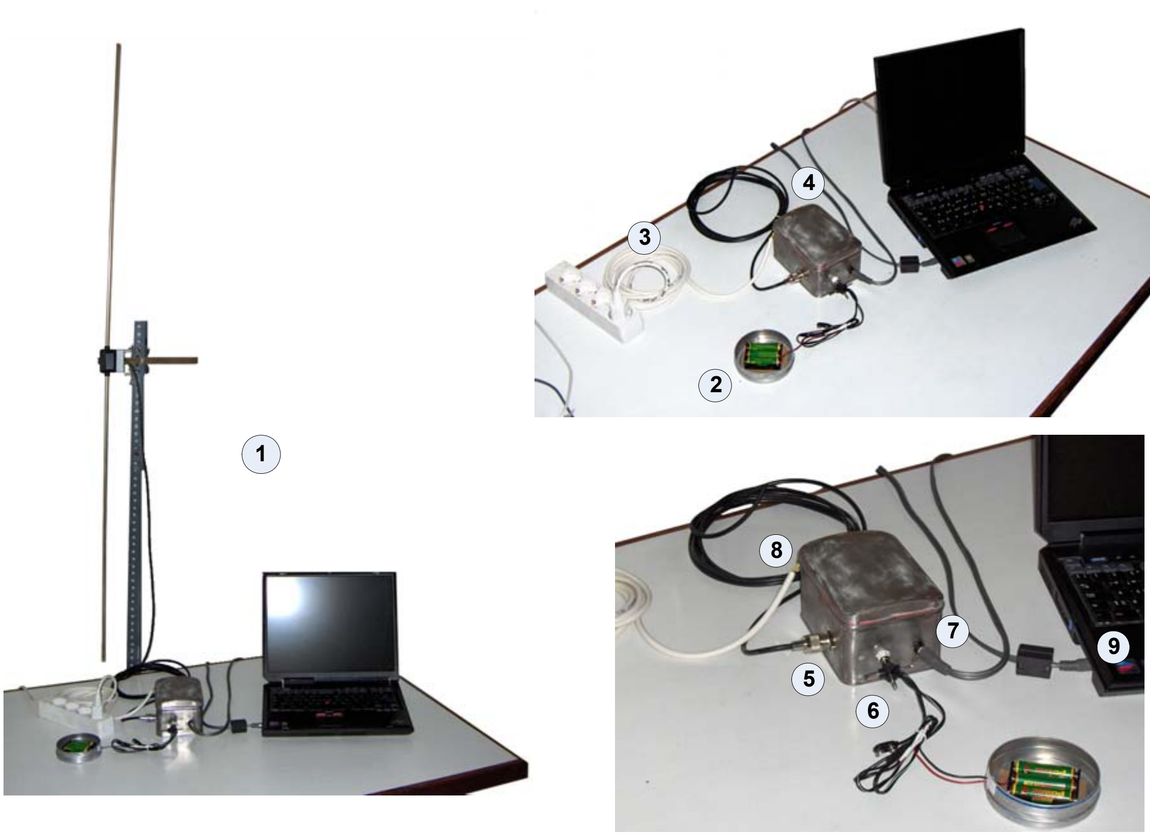
Figure 1-4: The FM transmitter and the radio receiver

The half wave dipole antenna and the FM transmitter were assembled by the members of the team while the Wide Band Communications Receiver was a choice of our advisor, *Mr. N. Sidiropoulos*. The transmitter can be supplied by an AC/DC adaptor. However, a transformer should be used followed by the appropriate filter in order to eliminate the scramble hum which appears due to the frequency (50 ~ 60 Hz) of the AC electric current.



1. Receiver : Desktop PC , Wide band communications receiver
2. Soundcard
3. Wide band communications receiver connected with the Personal Computer via the “line in” of the soundcard

Figure 1-5 : Receiver



1. Transmitter : FM transmitter , Laptop , Half wave dipole antenna , Battery case
2. Battery case : 3 AA batteries 1.5V
3. Ground cable
4. Low power FM transmitter
5. Cable to the antenna
6. Power in : DC 4.5V
7. Line in (audio signal to be transmitted)
8. Proper case grounding
9. Line out from laptop

Figure 1-6 : Transmitter

1.2 Software

The modules in the WFTP system were implemented in *Matlab*. Each member of the team was responsible for designing, developing and evaluating a number of functions which were assembled in the final form of the system. In this project we developed modules for the following subsystems:

- ⇒ Modulation, Demodulation, and Detection.
- ⇒ Error Control Coding and decoding.
- ⇒ Synchronization.
- ⇒ Phase Recovery.
- ⇒ Equalization.
- ⇒ Channel Estimation.
- ⇒ Cyclic Redundancy Check.

1.3 The Wireless FTP Communication System

The desired implementation of the WFTP system was proposed by the design illustrated in the following figure. In this case the system would consist of four basic modules, along with the appropriate hardware.

- ⇒ Desktop PC
- ⇒ Laptop PC
- ⇒ Software for the transmitter
- ⇒ FM transmitter
- ⇒ Software for the receiver
- ⇒ Radio receiver

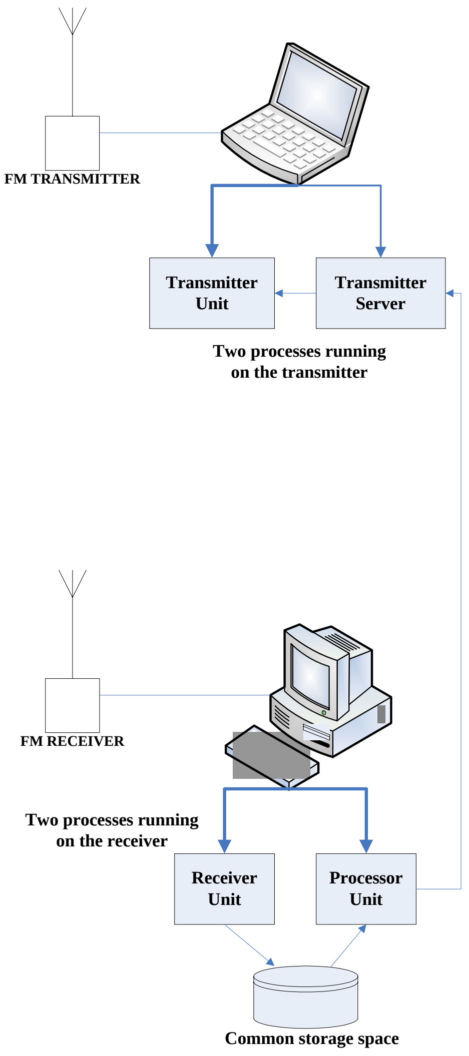


Figure 1-7: Desired design of the WFTP system

Software Modules:

⇒ *Transmitter Unit*

⇒ *Transmitter Server*

⇒ *Receiver Unit*

⇒ *Processor Unit*

Each of the preceding modules would integrate a number of functions depending on its operation. The *Transmitter Unit* and the *Receiver Unit* would work independently of the *Processor Unit* in different personal computers. The *Receiver Unit* and the *Processor Unit* would run in the same personal computer, but as different processes. The same applies for the *Transmitter Unit* and *Transmitter Server*. In order to understand the concept of this implementation we present transmission scenarios for both open and closed loop operation of the system.

In the open loop form of the system the only operating units would be the *Transmitter Unit*, the *Receiver Unit*, and the *Processor Unit*. The *Transmitter Unit* loads a packet and transmits a handshake signal in order to "wake up" the *Receiver Unit*. Once the packet is transmitted, the *Receiver Unit* stores the digital data generated by the soundcard in a specified storage space and waits for a new packet. In the meanwhile the *Processor Unit* which is running as a separate process, scans for new stored packets in the storage space of the *Receiver Unit*. The processing starts when a new packet is stored by the *Receiver Unit*.

In the closed loop form of the system with ARQ(Automatic Repeat Request), the additional unit *Transmitter Server* performs as a server for accepting negative or positive acknowledgments concerning packets' transmission, while the *Processor Unit* performs both as a processing unit and a client. A possible error in the packet's bits would force the *Processor Unit* to send a negative acknowledgement message to the *Transmitter Server* for packet retransmission. In all other cases Processor Unit should send positive acknowledgement messages to the *Transmitter Server*.
Consequently in both scenarios the *Transmitter Unit* does not wait until the processing in the *Processor Unit* is finished. Thus the transfer time is independent of the processing time spent in the receiver.

Unfortunately this design never worked in practice due to the fact that recording from the soundcard and processing data in *Matlab* cannot be done simultaneously. Moreover running the modules in the transmitter or the receiver as separate processes in the same PC, means that two *Matlab* processes must be open in the same time in each personal computer. However this scheme consumes a lot of memory and could not be implemented. In general the software implementation of a wireless data communication system using *Matlab*, limits our perspective for multiprocessing and hence achievement of high transfer rates.

Eventually the prevailing design emerged from the simplest aspect of WFTP system. The *Receiver and Processor Unit* concatenated in one process running on the PC connected with the radio receiver. Respectively the *Transmitter Unit* and the *Transmitter Server* merged in one process running on the PC connected with the FM transmitter.

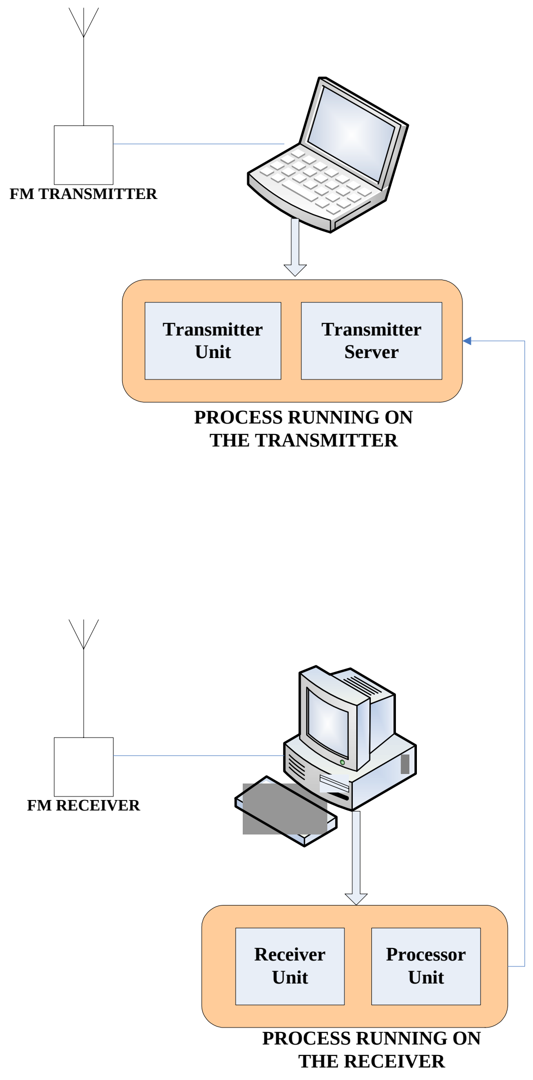


Figure 1-8: Schematic of the WFTP system

In this implementation the data transfer between the transmitter and the receiver is not independent of the processing of the packets. The processing time is included in the total transfer time of the transmitted file and hence transfer rate is reduced.

The WFTP system is mainly used for file transfer. Once a file is loaded, it is fragmented in packets of specified length. In the same time a specific sequence of bits of specified length, called *training sequence*, which is used for synchronization, equalization, and phase recovery, is generated for every packet. Each packet is shifted in the Encoder where redundant bits are added in a controlled manner. The Interleaver, which is charged with the mix up of the packet's bits, provides the resulting data bits to the modulator. The modulator transforms the packet's bits into digital waveforms of duration T_s samples. In addition the modulator transforms the training bits into the corresponding digital waveforms. The final digital modulated signal is generated by the concatenation of the digital waveforms that correspond to the training and packet bits. The interface between the radio transmitter and the digital output of the modulator is the soundcard. The audio playback feature of *Matlab* enables us to use the soundcard as a digital to analog converter. In particular the digital waveforms are play backed in a specified sampling frequency and the generated analog signal is driven to the radio transmitter.

In the receiver the soundcard performs the analog to digital conversion of the received signal at the same sampling frequency. The generated digital signal enters the synchronizer where the part of the signal carrying the actual information is isolated. In the process of synchronization the training sequence is of primary importance. The digital waveforms that correspond to the training bits enable the synchronizer to indicate the first and the last sample of the digital signal carrying the actual information. The isolated signal is filtered by the Equalizer which is responsible for inverting the effect of the channel on the transmitted signal. The demodulator processes the filtered signal and generates the corresponding data and training symbols. The phenomenon of phase shifting from the channel is eliminated by applying a linear transformation on the data symbols. The linear transformation is derived by the training sequence. The recovered symbols are transformed into the corresponding bits in the detector. Finally the deinterleaver recovers the detected bits in the correct order and the detector removes the additional redundancy.

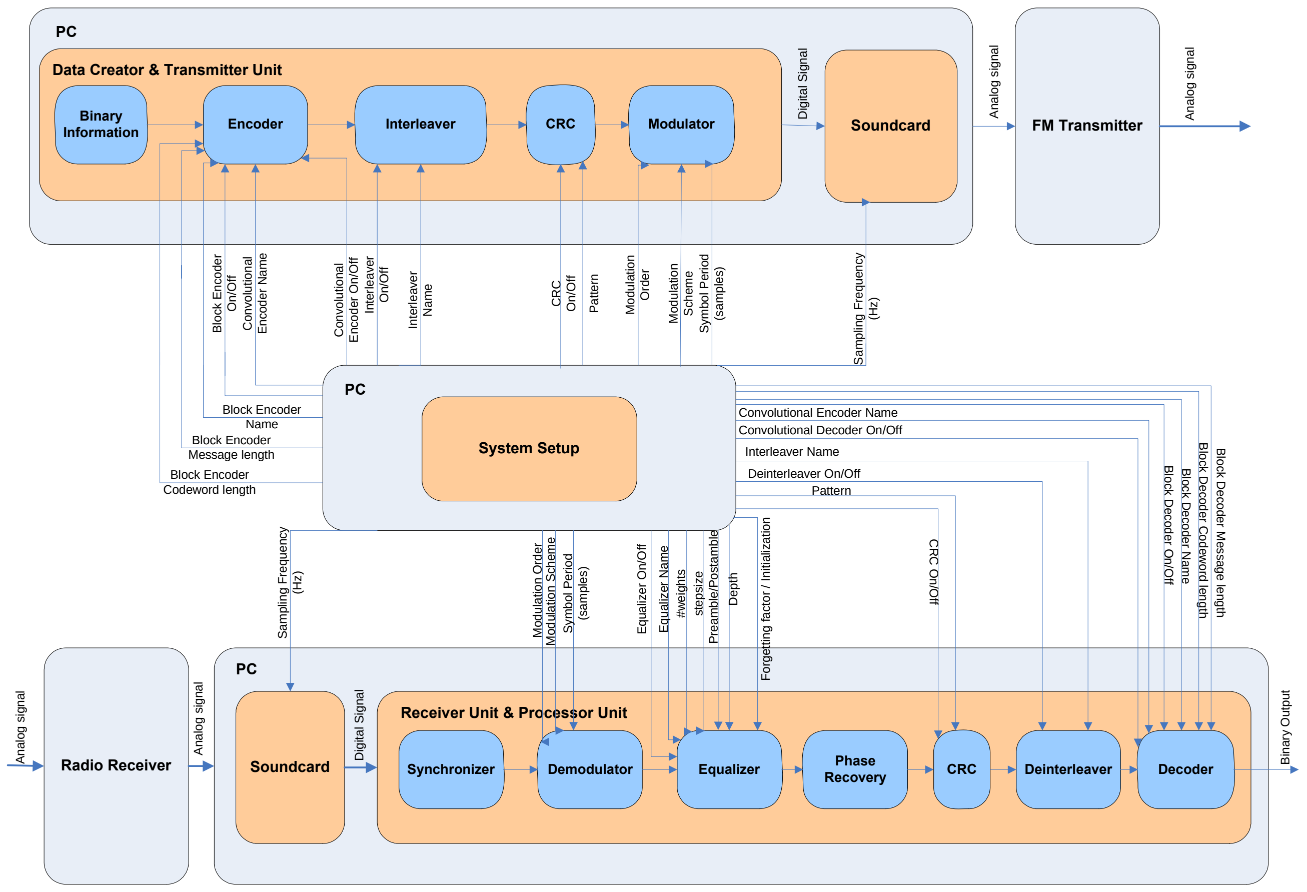


Figure 1-9: Block diagram of the WFTP system

From the above figure we can obtain that the active modules and their corresponding settings are controlled by a central module which is called the *System Setup*. The system settings are stored in a specified structure and are available in both the receiver and the transmitter. Once the desired settings have been selected in the transmitter, the *Data creator* module is responsible for generating the final samples of the modulated signal using the features specified by the *System Setup* module. The *Transmitter Unit* is responsible for transmitting the modulated data.

In the receiver the *Receiver Unit* is responsible for recording the packets whereas the final binary information is generated by the *Processor Unit*. In the closed loop form of the system if the *Processor Unit* detects errors, it transmits a negative acknowledgment to the *Transmitter Unit* for packet retransmission.

In the subsequent paragraphs we will refer to specific details concerning the operation of the WFTP Communication System.

1.3.1 Audio Playback and audio Recording

In previous sections we mentioned that the interface between the radio transmitter and the personal computer is the soundcard. The output of the *Transmitter Unit* is actually the samples of the analog signal that must be transmitted by the FM transmitter. The digital to analog conversion is achieved by using the *Matlab* function *wavplay*. The *wavplay* function playbacks an input vector using a PC-based audio device at a specified sampling rate F_s. The elements of the input vector must be in the range of $[-1,1]$.

Likewise the analog to digital conversion is achieved in the receiver by using the *Matlab* function *wavrecord*. The *wavrecord* function records a specified number of samples of an audio signal using the soundcard at a specified sampling rate F_s. The samples generated by *wavrecord* are in the range of $[-1,1]$.

In the WFTP system the recording time is adjusted dynamically according to the size of the packets. Because our system is a software implementation running on an operating system, several delays are incurred during its experimental operation. These delays are unpredictable and result from the hard disk and memory management of the operating system. Therefore in order to ensure the correct reception of the packets we add a constant $t_f = 0.5\,\text{sec}$ additional recording time. This overhead is independent of the size of the packets. Therefore in order to maintain high transfer rates, it is desired to fragment the transmitted file in a small number of packets. Consequently this results in an increase of the packets' size.

However increasing the number of bits per packet, results in an increase of the probability of packet error. Therefore the selection of the size of the packets, must ensure the achievement of the highest possible transfer rate along with the lowest possible probability of bit error.

1.3.2 Handshake and Handoff

In the WFTP system the handshake between the transmitter and the receiver is implemented using the UDP protocol. When communication is about to begin the *Receiver Unit* which is in the server mode scans a specified port of the Computer system until receiving a wake up signal from the *Transmitter Unit*.

Consider the simple scenario of a file transmission for the open loop system. The *Transmitter Unit* transmits a handshake signal followed by the first packet. The *Receiver Unit* wakes up, records the audio signal for the first packet and starts processing. In the meanwhile the *Transmitter Unit* in the server mode waits for the acknowledgement message from the *Receiver Unit*. Once the processing is finished the *Receiver Unit* transmits a neutral acknowledgement (in the open-loop system there is no meaning of positive or negative acknowledgment) and the *Transmitter Unit* transmits a new handshake signal followed by the next packet. This procedure is repeated for every packet. The acknowledgement of the last packet from the *Receiver Unit*, which is transmitted after the processing of the last packet, is called *handoff*. In general, the handoff signal informs the Transmitter Unit that the communication must be finished.

1.3.3 ARQ (Automatic Repeat Request)

In the WFTP system ARQ relies on the use of the Cyclic Redundancy Check. The error signals from the receiver to the transmitter are transferred through Ethernet using the UDP protocol.

In particular, when the transmission of a packet from the *Transmitter Unit* is completed, *Transmitter Unit* enters the server mode waiting for a positive or negative acknowledgement from the *Receiver Unit*.

In the *Receiver Unit* the *Processor Unit* controls the received packet for transmission errors using the CRC error detection code. Depending on the result the *Receiver Unit* enters the client mode and transmits a positive or negative acknowledgement to the *Transmitter Unit*.

2. EXECUTIVE SUMMARIES

Iliakis Evangelos. His primary responsibility was to develop and evaluate the modules for convolutional coding and Viterbi decoding. In addition he implemented the modules for Phase Shift Keying modulation scheme and the Cyclic Redundancy Check. Finally he was responsible for the implementation of the ARQ mechanism using the UDP protocol.

Kardaras Georgios. He was responsible for implementing and evaluating the Block encoders and decoders along with the Block interleavers and deinterleavers. Finally he was responsible for developing the Pulse Position Modulation (PPM) scheme.

Kokkinakis Chris. Chris developed and evaluated the LMS, RLS and CMA equalizers.

Mpervanakis Markos. Markos implemented the Viterbi equalizer along with the Quadrature Amplitude Modualtion scheme. Moreover he was responsible for the synchronization and the phase recovery. Finally he assembled the modules that were implemented by the members of the group, to a completely operating application.

C h a p t e r 2

ERROR CONTROL CODING

1. INTRODUCTION

Error control coding is an important and necessary step in achieving reliable communication in digital communication systems. In the model of a communication system error control coding is illustrated in Figure 2-1, and is implemented by the *channel encoder* and *decoder*.

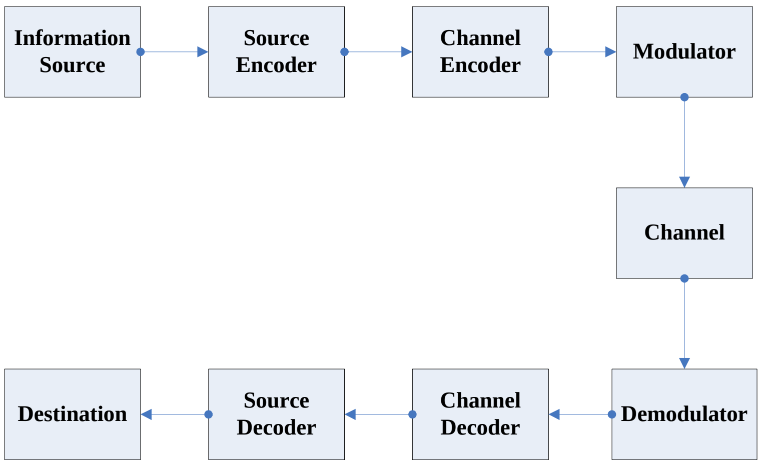


Figure 2-1: Block diagram of a typical communication system

The function of a *channel encoder* is to introduce some redundancy in the binary information sequence so that the receiver can correct errors that may have been caused by the transmission channel. The added redundancy serves to increase the reliability of the received data and aids the decoder to recover the initial information sequence.

At the receiver, the received sequence that is produced by the *demodulator*, enters the *channel decoder* and is transformed into a binary sequence called *estimated information sequence*. The decoding scheme depends on the encoding process used in the transmitter.

To focus attention on error control coding, the encoding process generally involves the mapping of a *k*-bit information sequence into an *n*-bit information sequence, called a *codeword*. The amount of redundancy introduced by the encoding of the data is measured by the ratio $\frac{n}{k}$. The reciprocal of this ratio is called the *code rate* $R = \frac{k}{n}$. In general, channel codes permit reliable communication of an information sequence over a channel that adds noise, introduces bit errors, or otherwise distorts the transmitted signal. According to the manner in which redundancy is added to the information message, Error Control Coding can be divided into two main categories; *Block and Convolutional coding*.

Block codes accept a block of k information bits and produce a block of n coded bits. By predetermined rules, n-k redundant bits are added to the k information bits to form the n coded bits. Commonly these codes are referred to as (n, k) block codes.

Convolutional codes are one of the most widely used channel codes in practical communication systems. They convert the entire information sequence into a single codeword. The main decoding strategy for convolutional codes is based on the Viterbi algorithm. In the next sections it follows a thorough analysis of the basic properties and decoding procedures for Convolutional Codes.

2. ALGEBRAIC CODING THEORY FOR CONVOLUTIONAL CODES

2.1 Galois fields

Error control coding is based on algebraic coding theory. In this section we will introduce some basic elements of algebraic coding theory that will be used in the presentation of convolutional codes.

Definition: A set of elements G on which a binary operation $*$ is defined is called a *group* if the following conditions are satisfied [1, pg. 25]:

i. The binary operation $*$ is associative. A binary operation $*$ on G is said to be *associative* if, for any a, b, and c in G,

$$a*(b*c)=(a*b)*c$$

ii. G contains an element e such that, for any α in G,

$$a*e=e*a=a$$

where e is called an *identity element* of G.

iii. For any element a in G, there exists another element a' (inverse of a) in G such that

$$a*a'=a'*a=e$$

A group G is said to be *commutative* if its binary operation $*$ also satisfies the following condition: For any a and b in G,

$$a*b=b*a$$

Theorem: The identity element in a group G is unique. [1, pg 26]

Theorem: The inverse of a group element is unique. [1, pg 26]

Definition: The number of elements in a group is called the *order* of the group. [1, pg 26]
Definition: A group of finite order is called a *finite group*. [1, pg 26]

Example 2-1

Let G be a set of 2 elements $\{0, 1\}$. We define the binary operation modulo-2 addition, denoted by $\oplus$, on G such that

$$0 \oplus 1 = 1,\ 1 \oplus 0 = 1,\ 0 \oplus 0 = 0,\ 1 \oplus 1 = 0$$

The set of elements G is closed under the binary operation modulo-2 addition. Since the conditions in definition 2-1 are satisfied, G is a commutative group under modulo-2 addition.

Definition: Let F be a set of elements on which two binary operations that are called addition "+" and multiplication "$\cdot$", are defined. The set *F* together with the two binary operations is a field if the following conditions are satisfied [2, pg 32]:

i. *F* is a commutative group under addition +. The identity element with respect to addition is called zero element or the additive identity of *F* and is denoted by 0.

ii. The set of nonzero elements in F is a commutative group under multiplication $\cdot$. The identity element with respect to multiplication is called the unit element or the multiplicative identity of F and is denoted by 1.

iii. Multiplication is distributive over addition; that is, for any three elements *a, b,* and *c* in *F*,

$$a \cdot (b + c) = a \cdot b + a \cdot c$$

Definition: The number of elements in a field is called the *order* of the field. [2, pg 32]
Definition: A field of finite order is called a *finite field*. [2, pg 32]

Basic Properties of fields

i. For every element *a* in a field, $a \cdot 0 = 0 \cdot a = 0$.

ii. For any two nonzero elements *a* and *b* in a field, $a \cdot b \neq 0$.

iii. $a \cdot b = 0$ and $a \neq 0$ imply that *b*=0.

iv. For any two elements *a* and *b* in a field,

$$-(a \cdot b) = (-a) \cdot b = a \cdot (-b)$$

v. For $a \neq 0$, $a \cdot b = a \cdot c$ implies that *b*=*c*.

[2, pg 32, 33]

Definition: A *Galois field* is defined as any finite set satisfying the axioms of a field, and is denoted by *GF*(*q*), where $q \in \mathbb{N}$. A prime field *GF*(*p*) has the additional condition that $p \in \mathbb{N}$ is *prime*. The set of integers (0, …, *p*-1) satisfies the axioms of a field under the operations $(+, \cdot)$ mod *p*. For any positive integer m, it is possible to extend the prime field *GF*(*p*) to a field of p^m elements, which is called an *extension* field of *GF*(*p*) and is denoted by $GF(p^m)$.

[2, pg 27][1, pg 34]

2.2 Binary fields and binary arithmetic

In general convolutional codes are binary codes with symbols from the Galois field *GF*(2). The *binary field GF*(*2*), is a set of two elements {0, 1} under modulo-2 addition and modulo-2 multiplication.

+	0	1
0	0	1
1	1	0

·	0	1
0	0	0
1	0	1

Table 2-1: Modulo-2 addition and multiplication

The elements and modulo-2 operations from GF(2) are used to describe the structure, the encoding and decoding process of convolutional codes. However in later sections we will come across with a more flexible representation, in which polynomials with coefficients from the *GF*(*2*) are used in order to describe convolutional codes. A polynomial *f*(*X*) with one variable *X* and with coefficients from *GF*(*2*) is of the following form [1, pg 38]

$$f(X) = f_0 + f_1X + f_2X^2 + \ldots + f_nX^n$$

where $f_i = \{0,1\}$, $0 \le i \le n$. The degree of a polynomial is the largest power of X with a nonzero coefficient. In general, polynomials over GF(2) are commutative, associative and distributed. Moreover there are 2^n polynomials over *GF*(*2*) with degree *n*.

All the usual operations (*addition, subtraction, multiplication, division*) can be performed between polynomials over *GF*(*2*). Multiplication and addition of the coefficients are modulo-2. Consider two polynomials over GF(2), f(X) and g(X)

$$f(X) = f_0 + f_1 X + f_2 X^2 + ... + f_n X^n$$

$$g(X) = g_0 + g_1 X + g_2 X^2 + ... + g_m X^m$$

where $m \le n$

The sum and the product of the two polynomials over GF(2), denoted as $f(X) + g(X)$ and $f(X) \cdot g(X)$ respectively, are given by [1, pg 39]

$$f(X) + g(X) = (f_0 + g_0) + (f_1 + g_1)X + (f_2 + g_2)X^2 + ... + (f_m + g_m)X^m + ... + f_n X^n$$

$$f(X) \cdot g(X) = (f_0 \cdot g_0) + (f_0 g_1 + f_1 g_0)X + (f_0 g_i + f_1 g_{i-1} + ... + f_1 g_0)X^i + ... + f_n g_m X^{n+m}$$

2.3 Vector space

Another basic element of algebraic coding theory that will be mentioned in subsequent sections is *vector spaces*.

Definition: Given a field F, *a vector space V* over F is a set *V* (whose members are called members the *vectors* of *V*) equipped with two operations ⊕ (vector addition) and · (scalar multiplication), satisfying the following:

i. *V* is a commutative group under addition

ii. For any element *a* in F and any element **v** in *V*, $a \cdot \mathbf{v}$ is an element in *V*.

iii. For any elements **u** and **v** in *V* and any elements *a* and *b* in F,

$$a \cdot (\mathbf{u} + \mathbf{v}) = a \cdot \mathbf{u} + a \cdot \mathbf{v}$$

$$(a + b) \cdot \mathbf{v} = a \cdot \mathbf{v} + b \cdot \mathbf{v}$$

iv. For any **v** in *V* and any *a* and *b* in F,

$$(a \cdot b) \cdot \mathbf{v} = a \cdot (b \cdot \mathbf{v})$$

v. Let 1 be the unit element of *F*. Then, for any **v** in *V*, $1 \cdot \mathbf{v} = \mathbf{v}$.

[1, pg 57]

Consider a sequence of n components $(a_0, a_1, ..., a_{n-1})$ where each component is a member of the binary field *GF*(*2*). This sequence is called a *n-tuple* over *GF*(*2*). Because each component can take up to two different values, there are 2^n distinct n-tuples over *GF*(*2*). The set V_n of all n-tuples forms the vector space over *GF*(*2*).

3. FUNDAMENTALS OF CONVOLUTIONAL CODES

Convolutional codes are based on a linear mapping over the *GF*(2) of a set of information words to a set of codewords. A rate $R = \frac{k}{n}$ convolutional encoder with memory *m* can be realized as a *k*-input and *n*-output linear sequential circuit with input memory *m*. This means that at any given time unit, encoder outputs depend not only on the inputs but also on some number of previous inputs. The *information sequence* is divided into overlapping blocks of length *k*

$$\mathbf{u}=(\mathbf{u}_0,\mathbf{u}_1,...,\mathbf{u}_t,...,\mathbf{u}_{h-1})=((u_0^{(1)}\,u_0^{(2)}...u_0^{(k)}),(u_1^{(1)}\,u_1^{(2)}...u_1^{(k)}),...,(u_{h-1}^{(1)}\,u_{h-1}^{(2)}...u_{h-1}^{(k)}))$$

and the *codeword* is divided into blocks of length *n*.

$$\mathbf{v}=(\mathbf{v}_0,...,\mathbf{v}_t,...)=((v_0^{(0)}\,v_0^{(1)}...v_0^{(n-1)}),(v_1^{(0)}\,v_1^{(1)}...v_1^{(n-1)}),....)$$

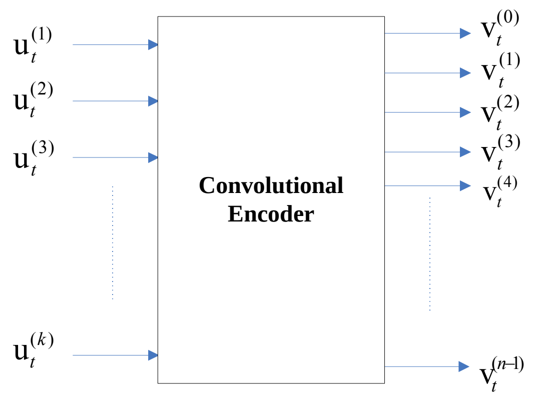


Figure 2-2: Block diagram of a Convolutional encoder

Convolutional encoders contains *k* shift registers (one for each input) , not all of which must have the same length.

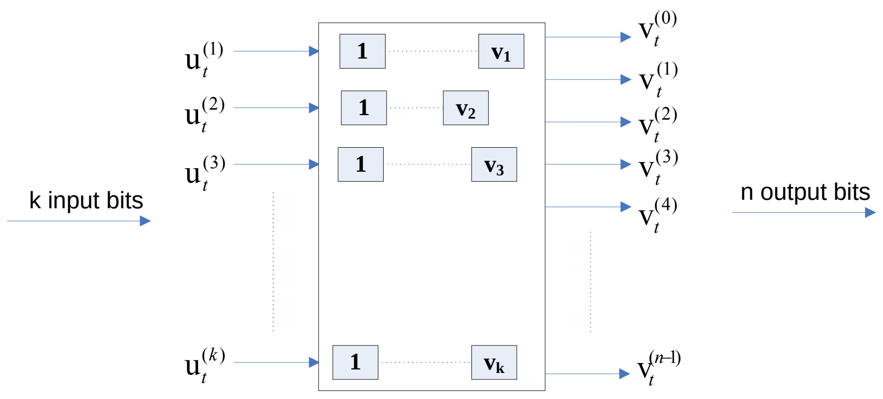


Figure 2-3: Memory elements in the encoder

As illustrated in Figure 2-3, each shift register *i* contains $\mathbf{v_i}$ delay elements , $i \in [1,k]$

Definition: *Constraint length* $\mathbf{v_i}$ is called the length of the *ith* shift register which corresponds to the *ith* input sequence, $i \in [1,k]$. [1, pg 459]

Definition: The encoder *memory m* is the maximum length of all *k* shift registers [1, pg 459]

$$m = \max_{1 \le i \le k}(v_i)$$

Definition: The *overall constraint length v* of the encoder is the sum of the lengths of all *k* shift registers. [1, pg 459]

In the special case where k=1, it follows that v_i=m and v=m.

A convolutional encoder with *k* inputs, *n* outputs and overall constraint length ***v*** is denoted as (n,k,v).

In convolutional codes and encoders the elements of the information and encoded sequences, may be drawn from the binary field, GF(2). Therefore the operations performed are modulo2-addition and modulo-2 multiplication. The result for each output is produced by a modulo-2 adder which can be implemented as XOR gate.

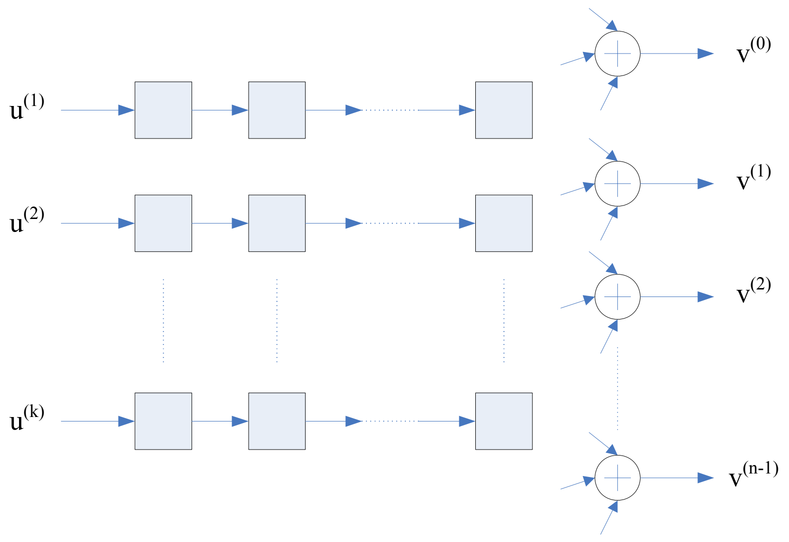


Figure 2-4: Block diagram of a feedforward convolutional encoder

Encoders for convolutional codes fall into the following categories:

- **i.** Nonsystematic Feedforward Convolutional Encoders
- **ii.** Systematic Feedforward Convolutional Encoders
- **iii.** Systematic Feedback Convolutional Encoders
- **iv.** Nonsystematic Feedback Convolutional Encoders

In this thesis we are mainly concerned on terminated convolutional codes. In order to terminate a convolutional code $k \cdot m$ zero bits are appended onto the information sequence in a way that all the storage elements in the encoder return to the zero state at the end of the input sequence.
In the following sections we will focus on the foremost three of the four categories that are used most in error control applications.

4. NONSYSTEMATIC FEEDFORWARD CONVOLUTIONAL ENCODERS

As mentioned above convolutional encoders can be realized as *Linear Time – Invariant* systems over the *GF*(2) with *k* inputs and *n* outputs. The *jth* of the *n* output sequences is denoted by

$$\mathbf{v}^{(j)} = (v_0^{(j)}, v_1^{(j)}, v_2^{(j)}, \ldots) \, , j \in [0, n-1]$$

At time *t, n* output bits are produced by the encoder

$$\begin{aligned}\mathbf{v}^{(0)} &= (\mathrm{v}_0^{(0)}, \mathrm{v}_1^{(0)}, \mathrm{v}_2^{(0)}, \ldots, \mathrm{v}_t^{(0)}, \ldots)\\ \mathbf{v}^{(1)} &= (\mathrm{v}_0^{(1)}, \mathrm{v}_1^{(1)}, \mathrm{v}_2^{(1)}, \ldots, \mathrm{v}_t^{(1)}, \ldots)\\ &\vdots\\ \mathbf{v}^{(\text{n-1})} &= (\mathrm{v}_0^{(\text{n-1})}, \mathrm{v}_1^{(\text{n-1})}, \mathrm{v}_2^{(\text{n-1})}, \ldots, \mathrm{v}_t^{(\text{n-1})}, \ldots)\end{aligned}$$

The *n* output sequences are multiplexed into a single sequence, called the *code sequence (codeword).*

$$\mathbf{v}=(\mathbf{v}_0,\ldots,\mathbf{v}_t,\ldots)=((\mathrm{v}_0^{(0)}\ \mathrm{v}_0^{(1)}\ldots\mathrm{v}_0^{(\text{n-1})}),(\mathrm{v}_1^{(0)}\ \mathrm{v}_1^{(1)}\ldots\mathrm{v}_1^{(\text{n-1})}),\ldots.)$$

where $\mathbf{v}_t = (\mathrm{v}_t^{(0)}\ldots\mathrm{v}_t^{(n-1)})$ is the encoded *n*-tuple at time unit *t*. Since the elements of the encoded sequences may be drawn form the *GF*(2), it follows that $\mathbf{v}_t \in GF(2)^n$.

The *jth* of the *n* output sequences $\mathbf{v}^{(j)}$ is obtained by convolving the input sequence with the corresponding system *impulse response*: [2, pg 204]

$$\mathbf{v}^{(j)} = \mathbf{u}^{(1)} * \mathbf{g}_1^{(j)} + \mathbf{u}^{(2)} * \mathbf{g}_2^{(j)} + \mathbf{u}^{(3)} * \mathbf{g}_3^{(j)} + \ldots + \mathbf{u}^{(k)} * \mathbf{g}_k^{(j)} = \sum_{i=1}^{k} \mathbf{u}^{(i)} * \mathbf{g}_i^{(j)} \quad , i \in [1,k] \quad j \in [0, n-1]$$

where $\mathbf{u}^{(i)}$ is the input sequence that corresponds to input *i*

$$\mathbf{u}^{(i)} = (\mathrm{u}_0^{(i)}, \mathrm{u}_1^{(i)}, \mathrm{u}_2^{(i)}, \ldots)\ , i \in [1,k]$$

and $\mathbf{g}_i^{(j)}$ are the impulse responses that correspond to output sequence *j*.

At time unit *t* the information *k*-tuple is denoted by $\mathbf{u}_t = (\mathrm{u}_t^{(1)}\ldots\mathrm{u}_t^{(k)})$ where $\mathbf{u}_t \in GF(2)^k$.

For each output *j* there are *i* corresponding impulse responses.

$$\mathbf{g}_i^{(j)} = (\mathrm{g}_{i,0}^{(j)}, \mathrm{g}_{i,1}^{(j)}, \mathrm{g}_{i,2}^{(i)}, \ldots, \mathrm{g}_{i,m}^{(i)})\ , i \in [1,k], j \in [0, n-1]$$

Generator sequences (impulse responses) describe the connections of the inputs and the delay elements with the modulo-2 adders. Every impulse response $\mathbf{g}_i^{(j)}$ has finite length $v_i + 1$. Impulse responses are called *generator sequences*.

At arbitrary time *t* the output bit of *jth* output sequence is computed by the difference equations:

$$\mathrm{v}_t^{(j)} = \sum_{i=1}^{k}\sum_{l=0}^{m} \mathrm{u}^{(i)} * \mathrm{g}_i^{(j)} = \sum_{i=1}^{k} (\mathrm{u}_t^i \mathrm{g}_{i,0}^{(j)} + \mathrm{u}_{t-1}^i \mathrm{g}_{i,1}^{(j)} + \ldots + \mathrm{u}_{t-m}^i \mathrm{g}_{i,m}^{(j)})$$

where $\mathrm{u}_t^{(i)}$ is the *ith* input bit at time *t* and $\mathrm{u}_{t-1}^{(i)}\ldots\mathrm{u}_{t-m}^{(i)}$ are the *m* previous input bits which are stored in the *ith* shift register. [2, pg 220]

4.1 Generator matrix in Time domain

The generator sequences are organized into a semi-infinite matrix G which is called the (time domain) *Generator Matrix*[1]. [1, pg 460]

$$\mathbf{G} = \begin{bmatrix} G_0\, G_1\, G_2 \ldots G_m & & \\ & G_0\, G_1\, G_2 \ldots G_m & \\ & & G_0\, G_1\, G_2 \ldots G_m \\ & & \ddots \end{bmatrix}$$

where G_l is a k x n submatrix whose entries are

$$G_l = \begin{bmatrix} g_{1,l}^{(0)} & g_{1,l}^{(1)} & \cdots & g_{1,l}^{(n-1)} \\ g_{2,l}^{(0)} & g_{1,l}^{(1)} & \cdots & g_{2,l}^{(n-1)} \\ \vdots & \vdots & & \vdots \\ g_{k,l}^{(0)} & g_{k,l}^{(1)} & \cdots & g_{k,l}^{(n-1)} \end{bmatrix}, l \in [0, m]$$

Consider the composite information sequence $\mathbf{u}$ with a finite length of blocks *h,* which is obtained by interleaving the *k* information sequences

$$\mathbf{u}=(\mathbf{u}_0,\mathbf{u}_1,\ldots,\mathbf{u}_t,\ldots,\mathbf{u}_{h-1})=((\mathrm{u}_0^{(1)}\,\mathrm{u}_0^{(2)}\ldots\mathrm{u}_0^{(k)}),(\mathrm{u}_1^{(1)}\,\mathrm{u}_1^{(2)}\ldots\mathrm{u}_1^{(k)}),\ldots,(\mathrm{u}_{h-1}^{(1)}\,\mathrm{u}_{h-1}^{(2)}\ldots\mathrm{u}_{h-1}^{(k)}))$$

where $\mathbf{u}_t = (u_t^{(1)}\ldots u_t^{(k)})$ is the information *k*-tuple at time unit *t*. Thus the Generator Matrix will have *h* rows and *2(m+h)* columns.

The encoding equations can be expressed in matrix form as

$$\mathbf{v} = \mathbf{u} \cdot \mathbf{G}$$

where the code sequence (codeword) is

$$\mathbf{v}=(\mathbf{v}_0,\mathbf{v}_1,\ldots,\mathbf{v}_t,\ldots)$$

Definition: An *(n, k, v)* convolutional code is the set of all output sequences (codewords) produced by an *(n, k, v)* convolutional encoder; that is, it is the row space of the encoder generator matrix **G**. Because the codeword **v** is a linear combination of rows of the generator matrix **G**, an *(n, k, v)* convolutional code is a linear code. [1, pg 461]

Nonsystematic feedforward convolutional encoders produce *nonrecursive convolutional codes* because the response to a single nonzero input in the encoder has finite duration.

[1] Note that in Generator matrix the blank areas are all zeros

4.2 Generator sequences

Generator sequence $\mathbf{g}_i^{(j)}$ between the *ith* input and the *jth* output is found by stimulating the encoder with the discrete impulse (1, 0, 0, 0,…) at the *ith* input and by observing the *jth* output.

However a more practical method to compute the generator sequences is described in the following steps. Suppose we want to compute $\mathbf{g}_i^{(j)}$:

Place 1 in the leftmost bit of the binary representation if the *ith* input is connected with *jth* adder. Place 1 in each spot where a connection line from the shift registers feeds into the adder and a 0 elsewhere.

Example 2-2

Consider the $R=\frac{1}{2}$ Nonsystematic Feedforward convolutional encoder presented in the following block diagram.

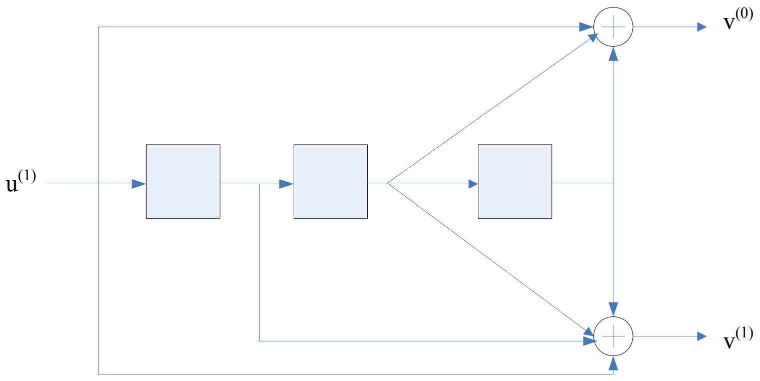


Figure 2-5: A rate R=1/2 binary nonsystematic feedforward convolutional encoder with memory order m=3.

Since *k*=1 the encoder contains one shift register. From the block diagram we can obtain that the shift register consists of three delay elements and hence its constraint length is v_1=3. Since *k*=1 and *n*=2 there will be two generator sequences:

$g_1^{(0)}=(1011), g_1^{(1)}=(1111)$. Therefore we can obtain the generator matrix G by interlacing the generator sequences.

$$g_1^{(0)}=(g_{1,0}^{0},g_{1,1}^{0},g_{1,2}^{0},g_{1,3}^{0})=(1011)$$
$$g_1^{(1)}=(g_{1,0}^{1},g_{1,1}^{1},g_{1,2}^{1},g_{1,3}^{1})=(1111)$$
$$G_0=[g_{1,0}^{(0)}\ \ g_{1,0}^{(1)}], G_1=[g_{1,1}^{(0)}\ \ g_{1,1}^{(1)}], G_2=[g_{1,2}^{(0)}\ \ g_{1,2}^{(1)}], G_3=[g_{1,3}^{(0)}\ \ g_{1,3}^{(1)}]$$

$$G=\begin{bmatrix} G_0\,G_1\,G_2\,G_3 & & \\ & G_0\,G_1\,G_2\,G_3 & \\ & \ddots & \\ & & \ddots \end{bmatrix}=\begin{bmatrix} g_{1,0}^{(0)}\,g_{1,0}^{(1)} & g_{1,1}^{(0)}\,g_{1,1}^{(1)} & g_{1,2}^{(0)}\,g_{1,2}^{(1)} & g_{1,3}^{(0)}\,g_{1,3}^{(1)} & \\ & g_{1,0}^{(0)}\,g_{1,0}^{(1)} & g_{1,1}^{(0)}\,g_{1,1}^{(1)} & g_{1,2}^{(0)}\,g_{1,2}^{(1)} & g_{1,3}^{(0)}\,g_{1,3}^{(1)} \\ & \ddots & & & \\ & & & \ddots & \end{bmatrix}$$

In the following table we describe the encoding process considering as input to the encoder the information sequence $\mathbf{u}=(\mathbf{u}_0,\mathbf{u}_1,\mathbf{u}_2,\mathbf{u}_3)=(u_0^1,u_1^1,u_2^1,u_3^1)=(1,0,1,1)$. The information sequence is divided into four blocks. Thus the generator matrix will have four rows.

$$G=\begin{bmatrix} G_0 & G_1 & G_2 & G_3 & & & \\ & G_0 & G_1 & G_2 & G_3 & & \\ & & G_0 & G_1 & G_2 & G_3 & \\ & & & G_0 & G_1 & G_2 & G_3 \end{bmatrix}$$

$\mathbf{v}_t$	**Encoding equations**
$\mathbf{v}_0=(v_0^0,v_0^1)$	$\mathbf{u}_0G_0=u_0^1(g_{1,0}^0\ g_{1,0}^1)=(1\bullet 1\ 1\bullet 1)=(1\,1)$
$\mathbf{v}_1=(v_1^0,v_1^1)$	$\mathbf{u}_0G_1+\mathbf{u}_1G_0=u_0^1(g_{1,1}^0\ g_{1,1}^1)+u_1^1(g_{1,0}^0\ g_{1,0}^1)=(1\bullet 0\ 1\bullet 1)\oplus(0\bullet 1\ 0\bullet 1)=(0\,1)\oplus(0\,0)=(0\,1)$
$\mathbf{v}_2=(v_2^0,v_2^1)$	$\mathbf{u}_0G_2+\mathbf{u}_1G_1+\mathbf{u}_2G_0=u_0^1(g_{1,2}^0\ g_{1,2}^1)+u_1^1(g_{1,1}^0\ g_{1,1}^1)+u_2^1(g_{1,0}^0\ g_{1,0}^1)=\ldots=(0\,0)$
$\mathbf{v}_3=(v_3^0,v_3^1)$	$\mathbf{u}_0G_3+\mathbf{u}_1G_2+\mathbf{u}_2G_1+\mathbf{u}_3G_0=u_0^1(g_{1,3}^0\ g_{1,3}^1)+u_1^1(g_{1,2}^0\ g_{1,2}^1)+u_2^1(g_{1,1}^0\ g_{1,1}^1)+u_2^1(g_{1,0}^0\ g_{1,0}^1)=(0\,1)$
$\mathbf{v}_4=(v_4^0,v_4^1)$	$G_3+G_2+G_1=(g_{1,3}^0\ g_{1,3}^1)+(g_{1,2}^0\ g_{1,2}^1)+(g_{1,1}^0\ g_{1,1}^1)=(0\,1)$
$\mathbf{v}_5=(v_5^0,v_5^1)$	$G_3+G_2=(g_{1,3}^0\ g_{1,3}^1)+(g_{1,2}^0\ g_{1,2}^1)=(0\,0)$
$\mathbf{v}_6=(v_6^0,v_6^1)$	$G_3=(g_{1,3}^0\ g_{1,3}^1)=(1\,1)$

Table 2-2: The encoding process

The resulting codeword **v** is (11, 01, 00, 01, 01, 00, 11).

4.3 Polynomial representation of the Generator matrix

In the section referring to the "Algebraic Coding Theory For Convolutional Codes" we mentioned that we can use a specific polynomial form, in order to describe convolutional codes.

Usually we introduce the delay operator D as the variable of polynomial. The power of D denotes the number of time units a bit is delayed with respect to the initial bit of sequence.

The polynomial representation of the information and encoded sequences are given by [2, pg 221]

$$\mathbf{u}^{(\mathrm{i})} \xleftrightarrow{\mathrm{Z}} \mathbf{u}^{(\mathrm{i})}(D) = \sum_{t=0}^{+\infty} \mathrm{u}_{\mathrm{t}}^{(\mathrm{i})} D^{t}$$

$$\mathbf{v}^{(j)} \xleftrightarrow{\mathrm{Z}} \mathbf{v}^{(j)}(D) = \sum_{t=0}^{+\infty} \mathrm{v}_{\mathrm{t}}^{(j)} D^{t}$$

The corresponding polynomial representation of the generator sequence $\mathbf{g}_{\mathrm{i}}^{(\mathrm{j})}$ is called *generator polynomial* and is given by [2, pg 221]

$$\mathbf{g}^{(\mathrm{i})} \xleftrightarrow{\mathrm{Z}} \mathbf{g}^{(\mathrm{i})}(D) = \sum_{t=0}^{+\infty} \mathrm{g}_{\mathrm{i,t}}^{(j)} D^{t}$$

Therefore $\mathbf{v}^{(\mathrm{j})}(D) = \sum_{i=1}^{k} \mathbf{u}^{(i)}(D) \mathbf{g}_{\mathrm{i}}^{(\mathrm{j})}(D)$

For an *(n,k,v)* convolutional encoder there are a total of $k \times n$ system functions which can be represented by the $k \times n$ Generator matrix.

$$\mathbf{G}(D) = \begin{bmatrix} \mathbf{g}_{1}^{(0)} & \mathbf{g}_{1}^{(1)} & \cdots & \mathbf{g}_{1}^{(n-1)} \\ \mathbf{g}_{2}^{(0)} & \mathbf{g}_{2}^{(1)} & \cdots & \mathbf{g}_{2}^{(n-1)} \\ \vdots & \vdots & & \vdots \\ \mathbf{g}_{\mathrm{k}}^{(0)} & \mathbf{g}_{\mathrm{k}}^{(1)} & \cdots & \mathbf{g}_{\mathrm{k}}^{(n-1)} \end{bmatrix}$$

We can express the encoding equations of a *(n,k,v)* feedforward encoder in matrix form as

$$\mathbf{V}(D) = \mathbf{U}(D)\mathbf{G}(D),$$

where $\mathbf{U}(D) = [\mathbf{u}^{(1)}(D), \mathbf{u}^{(2)}(D), ..., \mathbf{u}^{(k)}(D)]$ is the *k*-tuple of input sequences and $\mathbf{V}(D) = [\mathbf{v}^{(0)}(D), \mathbf{v}^{(1)}(D), ..., \mathbf{v}^{(n-1)}(D)]$ is the *n*-tuple of output sequences. [1, pg 263]

The final code sequence (codeword) which is produced by multiplexing the n output sequences can be expressed as

$$\mathbf{v}(D) = \mathbf{v}^{(0)}(D^{n}) + D\mathbf{v}^{(1)}(D^{n}) + ... + D^{n-1}\mathbf{v}^{(n-1)}(D^{n-1})$$

The codeword can be also derived by the expression [1, pg 264]

$$\mathbf{v}(D) = \sum_{i=1}^{k} \mathbf{u}^{(i)}(D^n)\mathbf{g}_i^{(j)}(D),$$

where $\mathbf{g}_{\mathrm{i}}(D) = \mathbf{g}_{\mathrm{i}}^{(0)}(D^n) + D\mathbf{g}_{\mathrm{i}}^{(1)}(D^n) + ... + D^{n-1}\mathbf{g}_{\mathrm{i}}^{(n-1)}(D^{n-1})$ is the composite generator polynomial relating the ith input sequence to the codeword $\mathbf{v}(D)$.

In convolutional codes there are two general realization methods that can be applied to all convolutional encoders; *Controller canonical form* and *Observer canonical form.*

In *Controller Canonical Form* there are *k* shift registers corresponding to each of the *k* input sequences. The *k* input sequences enter the left end of each shift register and the *n* output sequences are produced by modulo-2 adders external to the shift registers. The lowest degree (constant[2]) terms in the generator polynomials correspond to the connections at the left ends of the shift registers. The highest degree terms correspond to the connections at the right ends of the shift registers. The length of the *ith* shift register v_i is given by the expression:

$$v_i = \max_{1 \le j \le n-1} [\deg \mathbf{g}_i^{(j)}(D)] \quad ,i \in [1,k]$$

Moreover the memory order of the Convolutional Encoder is defined as:

$$m = \max_{1 \le j \le n-1} [\deg \mathbf{g}_i^{(j)}(D)] \quad ,i \in [1,k]$$

The overall constraint length of the encoder is defined as:

$$v = \sum_{i=1}^{k} v_i$$

In the *Observer Canonical form* there is one shift register corresponding to each of the *n* output sequences. The *k* input sequences enter modulo-2 adders internal to the shift registers and the outputs at the right of each shift register form the *n* output sequences. The lowest degree (constant) terms in the generator polynomials represent the connections at the right ends of the shift registers. The highest degree terms represent the connections at the left end of the shift registers. For this reason when an encoder is realized in observer canonical form, it is common to write the generator polynomials in the opposite of the usual order (from the highest to the lowest degree).

The length of the jth of the n shift registers is defined as:

$$v_j = \max_{1 \le i \le k} [\deg \mathbf{g}_i^{(j)}(D)] \quad ,j \in [0,n-1]$$

The memory order of the encoder is given by the expression:

$$m = \max_{1 \le j \le n-1} [v_j]$$

[2] The constant term in the generator polynomial $\mathbf{g}_i^{(j)}$ denotes the connection of the *ith* input with the modulo-2 adder that produces the *jth* output sequence

The overall constraint length of the encoder in observer canonical form is defined as:

$$v = \sum_{j=1}^{n-1} v_j$$

Consider the generator matrix G(D) of a nonsystematic feedforward encoder in controller canonical form. We can obtain the generator matrix of the encoder in the observer canonical form by reversing the order of the polynomials of the generator matrix G(D).

Example 2-3

Consider the $R = \frac{2}{3}$ nonsystematic Convolutional Encoder with the generator matrix in polynomial representation:

$$\mathbf{G}(D) = \begin{bmatrix} \mathbf{g}_1^{(0)} & \mathbf{g}_1^{(1)} & \mathbf{g}_1^{(2)} \\ \mathbf{g}_2^{(0)} & \mathbf{g}_2^{(1)} & \mathbf{g}_2^{(2)} \end{bmatrix} = \begin{bmatrix} 1+D & D & 1+D \\ D & 1 & 1 \end{bmatrix}$$

where

$$v_1 = \max(\deg \mathbf{g}_1^{(0)}, \mathbf{g}_1^{(1)}, \mathbf{g}_1^{(2)}) = 1 \quad , \quad v_2 = \max(\deg \mathbf{g}_2^{(0)}, \mathbf{g}_2^{(1)}, \mathbf{g}_2^{(2)}) = 1$$

are the constraint lengths of the two shift registers.

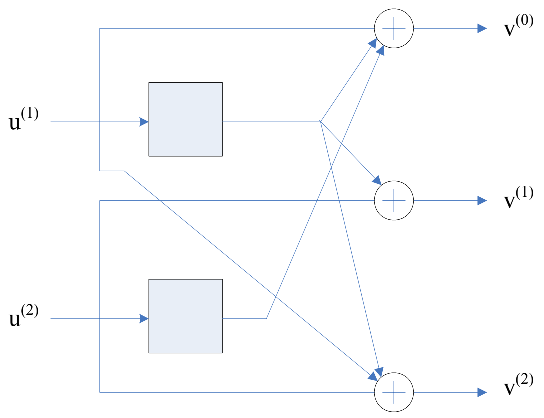


Figure 2-6: A rate R=2/3 binary feedforward convolutional encoder with memory order *m*=1.

The corresponding observer canonical form realization of the Nonsystematic Feedforward Convolutional Encoder is obtained by reversing the order of the generator polynomials in the Generator Matrix.

$$\mathbf{G}(D) = \begin{bmatrix} \mathbf{g}_1^{(0)} & \mathbf{g}_1^{(1)} & \mathbf{g}_1^{(2)} \\ \mathbf{g}_2^{(0)} & \mathbf{g}_2^{(1)} & \mathbf{g}_2^{(2)} \end{bmatrix} = \begin{bmatrix} D+1 & D & D+1 \\ D & 1 & 1 \end{bmatrix}$$

Because *n*=3 the encoder in the observer canonical form realization will contain three shift registers with corresponding constraint lengths:

$\mathbf{v}_0 = \max(\deg \mathbf{g}_1^{(0)}, \mathbf{g}_2^{(0)}) = 1, \mathbf{v}_1 = \max(\deg \mathbf{g}_1^{(1)}, \mathbf{g}_2^{(1)}) = 1, \mathbf{v}_2 = \max(\deg \mathbf{g}_1^{(2)}, \mathbf{g}_2^{(2)}) = 1$

Moreover the memory order is $m = \max\limits_{0 \le j \le 2}(\mathbf{v}_j) = \max(\mathbf{v}_0, \mathbf{v}_1, \mathbf{v}_2) = 1$.

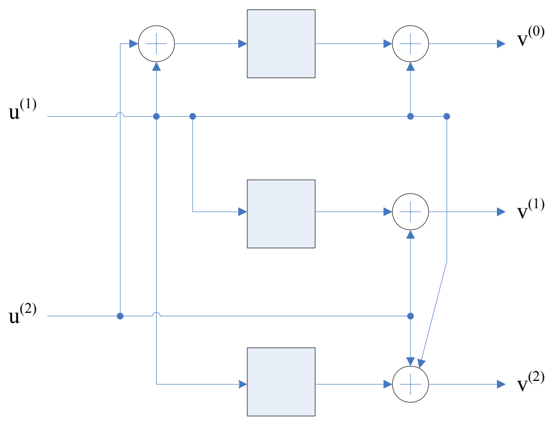


Figure 2-7: Observer canonical form realization of the encoder illustrated in Figure 2-6.

5. SYSTEMATIC FEEDFORWARD CONVOLUTIONAL ENCODERS

In a Systematic Feedforward convolutional encoder k output sequences, called *systematic output sequences*, are exact replicas of the input sequences. [1, pg 464]

$$\mathbf{v}^{(j-1)} = \mathbf{u}^i \quad ,i \in [1,k]$$
$$\mathbf{g}_i^{(j)} = \begin{cases} 1 & j=i-1 \\ 0 & j \neq i-1 \end{cases}$$

A convolutional generator matrix is systematic if the information sequence appears unchanged in the corresponding code sequence.

The polynomial representation of the generator matrix of a systematic Convolutional Encoder is a $k \times n$ matrix of the form [1, pg 465]

$$\mathbf{G}(D) = \begin{bmatrix} 1\,0 \cdots 0\, \mathbf{g}_1^{(k)}(D) \cdots \mathbf{g}_1^{(n-1)}(D) \\ 0\,1 \cdots 0\, \mathbf{g}_2^{(k)}(D) \cdots \mathbf{g}_2^{(n-1)}(D) \\ \vdots\,\vdots \quad \vdots \quad \vdots \qquad\quad \vdots \\ 0\,0 \cdots 1\, \mathbf{g}_k^{(k)}(D) \cdots \mathbf{g}_k^{(n-1)}(D) \end{bmatrix}$$

Note that the first k output sequences equal the k input sequences and are called *output information sequences*. The last *n-k* sequences are called *output parity sequences*. Thus Systematic Feedforward Convolutional encoders are defined only by the $k \times (n-k)$ last generator polynomials. The polynomial representation of the parity check matrix for the Systematic Feedforward convolutional encoder is [1, pg 466]

$$\mathbf{H}(D)=\begin{bmatrix} \mathbf{g}_1^{(k)}(D) & \mathbf{g}_2^{(k)}(D) & \cdots & \mathbf{g}_k^{(k)}(D) & 1 & 0 & \cdots & 0 \\ \mathbf{g}_1^{(k+1)}(D) & \mathbf{g}_2^{(k+1)}(D) & \cdots & \mathbf{g}_k^{(k+1)}(D) & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & & \vdots & \vdots & \vdots & & \vdots \\ \mathbf{g}_1^{(n-1)}(D) & \mathbf{g}_2^{(n-1)}(D) & \cdots & \mathbf{g}_k^{(n-1)}(D) & 0 & 0 & \cdots & 1 \end{bmatrix}$$

where the last (*n-k*) columns of **H**(D) form the $(n-k)\times(n-k)$ identity matrix. The parity check matrix can be rewritten as

$$\mathbf{H}(D)=\begin{bmatrix} \mathbf{h}_1^{(0)}(D) & \mathbf{h}_1^{(1)}(D) & \cdots & \mathbf{h}_1^{(k-1)}(D) & 1 & 0 & \cdots & 0 \\ \mathbf{h}_2^{(0)}(D) & \mathbf{h}_2^{(1)}(D) & \cdots & \mathbf{h}_2^{(k-1)}(D) & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & & \vdots & \vdots & \vdots & & \vdots \\ \mathbf{h}_{n-k}^{(0)}(D) & \mathbf{h}_{n-k}^{(1)}(D) & \cdots & \mathbf{h}_{n-k}^{(k-1)}(D) & 0 & 0 & \cdots & 1 \end{bmatrix}$$

Any codeword **V**(D) must satisfy the parity-check equations

$$\mathbf{V}(D)\mathbf{H}^{\mathbf{T}}(D)=\mathbf{0}(D)$$

where **0**(D) represents the $1\times(n-k)$ matrix of all-zero sequences. [1, pg 466]

In Observer Canonical realization of Systematic Feeforward encoders there are only *n-k* shift registers. The length of each shift register is defined as

$$v_j = \max_{1\le i\le n-1}[\deg \mathbf{g}_i^{(j)}(D)] \qquad ,j\in[0,n-1]$$

and the memory order of the encoder is given by the expression

$$m=\max_{1\le j\le n-1}[v_j]$$

Example 2-4

Consider the $R=\frac{2}{3}$ Systematic Feedforward Convolutional encoder with generator matrix

$$\mathbf{G}(D)=\begin{bmatrix} 1 & 0 & \mathbf{g}_1^{(2)} \\ 0 & 1 & \mathbf{g}_2^{(2)} \end{bmatrix}=\begin{bmatrix} 1 & 0 & 1+D+D^2 \\ 0 & 1 & 1+D \end{bmatrix}$$

where $\mathbf{g}_1^{(2)}$ corresponds to the generator polynomial of the first input with response to the second output and $\mathbf{g}_2^{(2)}$ corresponds to the generator polynomial of the first input with response to the second output. The leading order of the generator polynomial $\mathbf{g}_1^{(2)}$ is 2. Therefore the length of the first shift register is v_1 = 2. The leading order of the generator polynomial $\mathbf{g}_2^{(2)}$ is 1 and hence the length of the second shift register is $v_1 = 1$. The overall constraint length is $v = 3$. Thus we can obtain the block diagram of the systematic C(3,2,3) convolutional encoder in the controller canonical form.

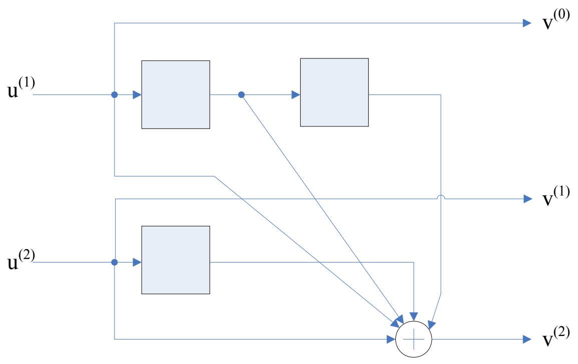


Figure 2-8: A rate R=2/3 systematic feedforward convolutional encoder in controller canonical form.

The parity-check matrix is given by

$$H(D) = [\mathbf{h}_1^{(0)}(D)\ \mathbf{h}_1^{(1)}(D)\ 1] = [1 + D + D^2\ \ 1 + D\ \ 1]$$

In order to obtain the observer canonical form of the encoder we reverse the order of generator polynomials in the generator matrix.

$$\mathbf{G}(D) = \begin{bmatrix} 1 & 0 & D^2 + D + 1 \\ 0 & 1 & D + 1 \end{bmatrix}$$

The encoder contains n-k=1 shift register. The constraint length of the shift register is

$$v_1 = \max(\deg g_1^{(2)}\ g_2^{(2)}) = 2$$

Therefore the systematic feedforward encoder can be realized in observer canonical form as shown in Figure 2-9.

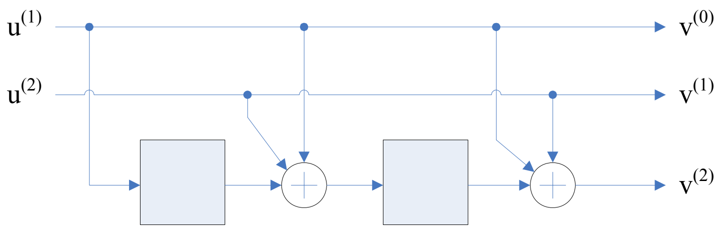


Figure 2-9: Observer canonical form realization of the encoder illustrated in Figure 2-8.

6. SYSTEMATIC FEEDBACK CONVOLUTIONAL ENCODERS

Systematic feedback encoders generate the same codes as the corresponding feedforward encoders but exhibit a different mapping between information sequences and codewords. Usually we prefer to transform a nonsystematic convolutional encoder to a systematic feedback convolutional encoder rather than create it from scratch. Therefore the main idea is to manipulate the $k \times n$ polynomial generator matrix **G**(D) so that the first k columns of the systematic generator matrix $\mathbf{G}^{'}$(D) form an $k \times k$ identity matrix. This is achieved by performing polynomial row operations on the generator matrix **G**(D). The entries of the systematic polynomial generator matrix $\mathbf{G}^{'}$(D) are rational functions in the delay operator D. Because elementary row operations do not change the row space of a matrix, the matrices **G**(D) and $\mathbf{G}^{'}$(D) generate the same code and since $\mathbf{G}^{'}$(D) contains rational functions it results in a feedback encoder realization.

Since $\mathbf{G}^{'}$(D) is in systematic form it can be used to determine a $(n-k)\times n$ systematic parity check matrix $\mathbf{H}^{'}$(D). The procedure is the same as described in the systematic feedforward encoders. In this case, $\mathbf{H}^{'}$(D) contains rational functions and its last $(n-k)$ columns form the $(n-k)\times(n-k)$ identity matrix.

Because matrices $\mathbf{G}^{'}$(D) and $\mathbf{H}^{'}$(D) contain rational functions, the impulse response of the encoder has infinite duration. Therefore the feedback shift register realization of $\mathbf{G}^{'}$(D) is an infinite impulse response linear system (IIR) and the generated code is *recursive (SRCC)*[3]. The time domain generator matrix $\mathbf{G}^{'}$ contains sequences of infinite length. For this reason systematic feedback encoders are more easily described using the polynomial representation. [1, pg 471]

Example 2-5

Consider the $R = \frac{2}{3}$ nonsystematic feedforward generator matrix given by

$$\mathbf{G}(D) = \begin{bmatrix} 1+D & D & 1+D \\ D & 1 & 1 \end{bmatrix}$$

The controller canonical form realization of Figure 2-10 contains two shift registers with the corresponding lengths

$$v_1 = \max_{0\le j\le 2}\left[\deg \mathbf{g}_1^j\right] = \max_{0\le j\le 2}\left[\deg \mathbf{g}_1^0\,\mathbf{g}_1^1\,\mathbf{g}_1^2\right] = 1$$
$$v_2 = \max_{0\le j\le 2}\left[\deg \mathbf{g}_2^j\right] = \max_{0\le j\le 2}\left[\deg \mathbf{g}_2^0\,\mathbf{g}_2^1\,\mathbf{g}_2^2\right] = 1$$

[3] SRCC= systematic recursive convolutional code

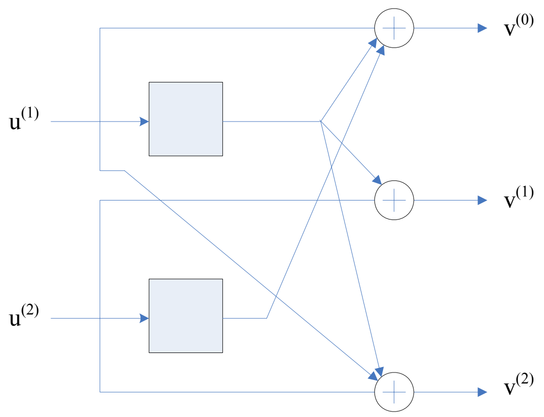


Figure 2-10: A rate R=2/3 nonsystematic feedforward convolutional encoder in controller canonical form.

To convert **G**(D) to an equivalent systematic feedback encoder, we apply the following sequence of elementary row operations.

$$step1:\ row1=\frac{1}{1+D}row1 \Rightarrow \mathbf{G}(D)=\begin{bmatrix}1 & D/(1+D) & 1\\ D & 1 & 1\end{bmatrix}$$

$$step2:\ row2=row2+Drow1 \Rightarrow \mathbf{G}(D)=\begin{bmatrix}1 & D/(1+D) & 1\\ 0 & (1+D+D^2)/(1+D) & 1+D\end{bmatrix}$$

$$step3:\ row2=\frac{1+D}{1+D+D^2}row2 \Rightarrow \mathbf{G}(D)=\begin{bmatrix}1 & D/(1+D) & 1\\ 0 & 1 & (1+D)^2/(1+D+D^2)\end{bmatrix}$$

$$step4:\ row1=row1+\frac{D}{1+D}row2 \Rightarrow \mathbf{G}(D)=\begin{bmatrix}1 & 0 & 1/(1+D+D^2)\\ 0 & 1 & (1+D^2)/(1+D+D^2)\end{bmatrix}$$

The systematic parity check matrix is given by

$$\mathbf{H}'(D)=\begin{bmatrix}\frac{1}{1+D+D^2} & \frac{1+D^2}{1+D+D^2} & 1\end{bmatrix}$$

The equivalent nonsystematic polynomial parity-check matrix is given by

$$\mathbf{H}(D)=\begin{bmatrix}1 & 1+D^2 & 1+D+D^2\end{bmatrix}$$

where the $\mathbf{h}^{(j)}(D),\ 0\le j\le n-1$ represents the parity check polynomial associated with the *jth* output sequence. **H**(D) and **H'**(D) correspond to the controller canonical form realization of the encoder. We can obtain the observer canonical form realization by reversing the order of the polynomials. The generator matrix and the systematic parity check matrix in observer canonical form are given by

$$\mathbf{G}(D)=\begin{bmatrix}1 & 0 & 1/(D^2+D+1)\\ 0 & 1 & (D^2+1)/(D^2+D+1)\end{bmatrix},\quad \mathbf{H}'(D)=\begin{bmatrix}\frac{1}{D^2+D+1} & \frac{D^2+1}{D^2+D+1} & 1\end{bmatrix}$$

7.1 State Diagram

The operation of convolutional encoders can be described by a state diagram. State diagram is a graph where nodes correspond to the encoder's possible states and branches denote the transitions between states. The state transitions are labeled with the appropriate input / output binary tuples $\mathbf{u}_t / \mathbf{v}_t$. The state of an encoder is defined as the contents of its *k* shift registers. For a (n,k,v) convolutional encoder in controller canonical form there are a total of 2^v possible states. The *ith* shift register of the encoder at time unit t contains v_i bits denoted as $s_{t-1}^{(i)},\dots,s_{t-v_i}^{(i)}, i \in [1,k]$ where $s_{t-1}^{(i)}$ represents the contents of the rightmost delay element and $s_{t-v_i}^{(i)}$ represents the contents of the leftmost delay element.

Definition: The encoder state at time unit t is the binary v-tuple

$$\sigma_t = (s_{t-1}^{(1)}\, s_{t-2}^{(1)} \dots s_{t-v_1}^{(1)} s_{t-1}^{(2)}\, s_{t-2}^{(2)} \dots s_{t-v_2}^{(2)} s_{t-1}^{(3)}\, s_{t-2}^{(3)} \dots s_{t-v_3}^{(3)} \dots s_{t-1}^{(k)}\, s_{t-2}^{(k)} \dots s_{t-v_k}^{(k)})$$

The *ith* shift register at time unit *t* contains the v_i previous input bits $u_{t-1}^{(i)} \dots u_{t-v_i}^{(i)}$. Therefore the encoder state at time unit *t* can be expressed as a *v-tuple* of the memory values.

$$\sigma_t = (u_{t-1}^{(1)}\, u_{t-2}^{(1)} \dots u_{t-v_1}^{(1)} \dots u_{t-1}^{(k)}\, u_{t-2}^{(k)} \dots u_{t-v_k}^{(k)})$$

A (n,k,v) convolutional encoder in observer canonical form contains n shift registers. In this case there are a total of 2^v possible states and the encoder state at time unit t is the binary v-tuple

$$\sigma_t = (s_{t-1}^{(1)}\, s_{t-2}^{(1)} \dots s_{t-v_1}^{(1)} s_{t-1}^{(2)}\, s_{t-2}^{(2)} \dots s_{t-v_2}^{(2)} s_{t-1}^{(3)}\, s_{t-2}^{(3)} \dots s_{t-v_3}^{(3)} \dots s_{t-1}^{(n)}\, s_{t-2}^{(n)} \dots s_{t-v_n}^{(n)})$$

The states are labeled S_0, S_1, …, S_{2^v-1} where S_i represents the state whose binary v-tuple representation is $b_0\, b_1 \dots b_{v-1}$. The exponent *i* is given by the expression $i = b_0 + 2^1 b_1 + \dots + 2^{v-1} b_{v-1}$ [1, pg 487].

7.2 Trellis Diagram

The state diagram can be represented as it evolves in time with a trellis diagram. Trellis diagram is constructed by reproducing the states horizontally and showing the state transitions going from left to right corresponding to time and data input. At time t_i, 2^v nodes that correspond to the possible 2^v states are placed vertically. At time t_{i+1}, the same structure is repeated. Then we denote with branches the possible 2^k transitions from each state to another. The branches are labeled with the corresponding output.

Consider a (n,k,v) convolutional encoder with memory m. For an information sequence of length $K^* = kh$ the trellis diagram contains $h+m$ time units. The final codeword is obtained from the labels of the branches that determine the path that corresponds to the information sequence.

In steady state the trellis diagram denotes the possible transitions between states and the corresponding outputs.

Example 2-6

Consider the $R = \frac{2}{3}$ nonsystematic Convolutional Encoder presented in Figure 2-10. Since the overall constraint length v=2 there are 2^2=4 possible states with binary representation 00, 01, 10, 11. The label of each state i is S_i, where the indicator i is given by the expression $i = b_0 + 2^1 b_1 + \ldots + 2^{v-1} b_{v-1}$. The mapping between labels and states is provided in the following table

State	***Label***
00	S_0
01	S_2
10	S_1
11	S_3

Table 2-3: States labeling

In order to determine the state diagram of the encoder we must construct the following table

State	***Input bits***	***Output bits***	***Next State***
S_0	00	000	S_0
S_0	01	011	S_2
S_0	10	101	S_1
S_0	11	110	S_3
S_1	00	111	S_0
S_1	01	100	S_2
S_1	10	010	S_1
S_1	11	001	S_3
S_2	00	100	S_0
S_2	01	111	S_2
S_2	10	001	S_1
S_2	11	010	S_3
S_3	00	011	S_0
S_3	01	000	S_2
S_3	10	110	S_1
S_3	11	101	S_3

Table 2-4: Input/Output bits for every possible state transition

for the rate R=2/3 convolutional encoder of Figure 2-10.

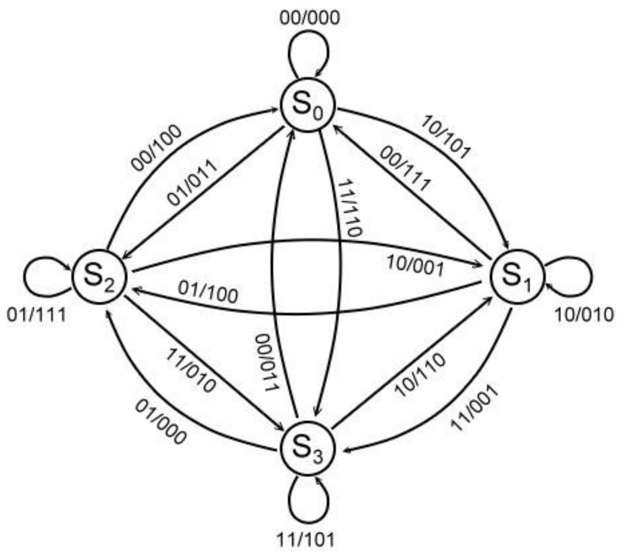


Figure 2-11: State diagram of the convolutional encoder of Figure 2-10

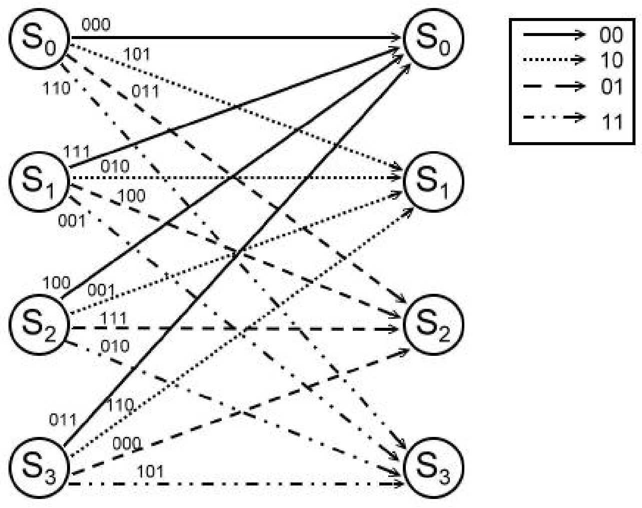


Figure 2-12: The corresponding trellis diagram in steady state of the encoder of Figure 2-10

7.3 Catastrophic Encoders

This class of convolutional encoders should be avoided when developing an error control system. The foremost disadvantage is that a finite number of channel errors can generate an infinite number of decoding errors.

Definition: An encoder is catastrophic if and only if the state diagram contains a cycle with zero output weight other than the zero-weight cycle around the state S_0. [1, pg 485]

Any encoder for which a feedforward exists is noncatastrophic. Therefore systematic encoders are always noncatastrophic, since a trivial feedforward inverse exists. Minimal nonsystematic encoders are also noncatastrophic.

7.4 Distance Properties of convolutional codes

The most important distance measure for convolutional codes is the minimum free distance d_{free}. Free distance determines the error-correcting capabilities of a convolutional code.

Definition: The free distance of a convolutional code is the minimum Hamming distance[4] between any two code sequences of the code

$$d_{free} = \min_{u',u''} d(\mathbf{v}', \mathbf{v}'')$$

where $\mathbf{v}'$ and $\mathbf{v}''$ are the codewords corresponding to the information sequences $\mathbf{u}'$, $\mathbf{u}''$. [2, pg 213] Codewords $\mathbf{v}'$ and $\mathbf{v}''$ have finite length and start and end in the zero state S_0. Because a convolutional code is a linear code the minimum hamming distance of $\mathbf{v}'$ and $\mathbf{v}''$ is equal to the minimum hamming weight[5] of the sum $\mathbf{v}'$ and $\mathbf{v}''$.

$$d_{free} = \min_{u',u''}\{w(\mathbf{v}' + \mathbf{v}'')\} = \min\{w(\mathbf{v})\} = \min\{w(\mathbf{uG})\}$$

where $\mathbf{v}$ is the codeword corresponding to the information sequence $\mathbf{u}$. Therefore d_{free} is the min-weight codeword produced by any finite nonzero length information sequence. Moreover it is the minimum weight of all finite length nonzero paths in the state diagram that diverge from and remerge with the all-zero state S_0. The free distance is a code property and hence it is independent of the encoder realization. A minimum distance encoder can always correct an error sequence $\mathbf{e}$, if

$$w(\mathbf{e}) < \frac{d_{free}}{2}$$

Definition: The maximum error-correcting capability of an encoder t_{free} is given by

$$t_{free} = \frac{d_{free} - 1}{2}$$

[3, pg 49]

[4] Hamming distance between two n-tuples **v** and **w** denoted d(**v**, **w**) is defined as the number of places where they differ.
[5] The hamming weight of a n-tuple **v** denoted w(**v**) is defined as the number of nonzero components of **v.**

We can obtain the free distance from the state diagram by finding the path that corresponds to the codeword with the minimum weight. Only a subset of all the codewords specified by the state diagram is necessary in order to evaluate the minimum free distance.

- Only codewords with finite length that leave zero state at time *t=0* and end at zero state again, have to be considered.

- Codewords that leave zero state more than once can be excluded since a codeword with a smaller weight always exist.

- Codewords that are not in the zero state after $t > 2^{v}$ state transitions can be excluded. In such a case the corresponding path would require passing at least twice from the zero state and would form a loop in the state diagram. Therefore a codeword with a smaller weight always exist.

An encoder is an *optimum free distance* (OFD) encoder if its free distance is equal or superior to that of any other encoder of the same rate and constraint length. [3, pg 49]

The process of finding the free distance of a convolutional code from the state diagram is illustrated in the following example.

Example 2-7

Consider the state diagram of the Example 2-5. The transitions are labeled with an operator W where its power corresponds to the Hamming weight of the output associated with the transition/branch. Figure 2-13 shows the modified state diagram.

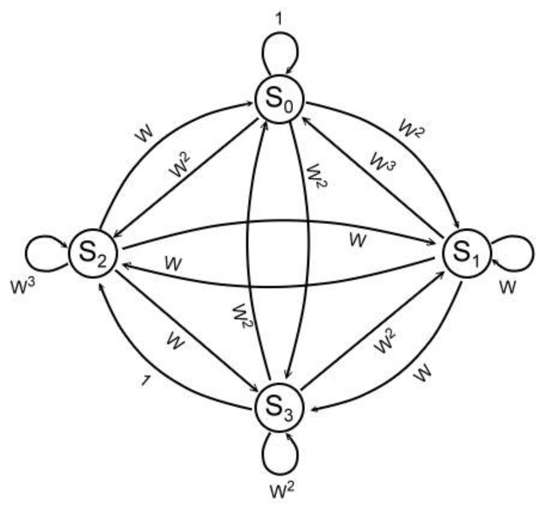


Figure 2-13: Modified encoder state diagram for the encoder of Figure 2-10.

The free distance of the C(3,2,2) convolutional code can be obtained by the codewords that leave S_0 at time *t=0* and return to it for the first time at the latest when *t=4*. The codewords that satisfy these requirements and the corresponding weight of each code sequence are given in the following table.

Codeword	Labels multiplication	Codeword weight
S_0, S_1, S_0	$W^2 W^{3=} W^5$	5
S_0, S_1, S_2, S_0	$W^2 WW^{=} W^4$	4
S_0, S_1, S_3, S_2, S_0	$W^2 WW^{=} W^4$	4
S_0, S_3, S_0	$W^2 W^{2\ =} W^4$	4
S_0, S_3, S_1, S_0	$W^2 W^2 W^{2\ =} W^6$	6
S_0, S_3, S_2, S_0	$W^2 W^{=} W^3$	3
S_0, S_2, S_0	$W^2 W^{=} W^3$	3

Table 2-5: Possible paths and the corresponding codewords weight in order to obtain the free distance.

Therefore the free distance of the code is $d_{free}=3$.

8.1 Maximum Likelihood Decoding

The Viterbi algorithm is a maximum likelihood decoding algorithm. In maximum likelihood decoding the goal is to produce an estimate $\hat{\mathbf{u}}$ of the information sequence **u** based on the received sequence **r** in order to achieve the minimum probability of decoding error assuming equiprobable input symbols. Equivalently in ML decoding for convolutional codes the decoder produces an estimate $\hat{\mathbf{v}}$ of the codeword **v** that maximizes the conditional probability of the received sequence **r**, $P(\mathbf{r}\,|\,\mathbf{v})$.

Definition: A decoder that chooses its estimate to maximize the conditional probability of the received sequence **r** is called a maximum likelihood decoder. [1, pg 11]

For a (n,k,v) encoder and an information sequence of length $K^* = kh$ there are 2^{k} branches leaving and entering the state and 2^{K^*} distinct paths through the trellis corresponding to the 2^{K^*} codewords. Assume that an information sequence $\mathbf{u} = (\mathbf{u}_0, \mathbf{u}_1, ...)$ of length $K^* = kh$ is encoded into a codeword $\mathbf{v} = (\mathbf{v}_0, \mathbf{v}_1, ...)$ of length $N = n(h+m)$ and $\mathbf{r} = (\mathbf{r}_0, \mathbf{r}_1, ...)$ is the received sequence of length *N*. A decoding error occurs if and only if $\hat{\mathbf{v}} \neq \mathbf{v}$. An optimum decoding rule must *minimize* the error probability of the decoder which is given by [1, pg 11]

$$P(E) = \sum_{\mathbf{r}} P(E\,|\,\mathbf{r}) P(\mathbf{r})$$

Consequently the optimum decoding rule must *minimize* the conditional probability of the decoder which is defined as [1, pg 11]

$$P(E\,|\,\mathbf{r}) \triangleq P(\hat{\mathbf{v}} \neq \mathbf{v}\,|\,\mathbf{r})$$

or equivalently must *maximize* the conditional probability [1, pg 11]

$$P(\hat{\mathbf{v}} = \mathbf{v}\,|\,\mathbf{r})$$

where $P(\mathbf{r})$ is the probability of the received sequence **r** and is independent of the decoding rule. Therefore the decoding rule that minimizes $P(E)$ must minimize $P(E\,|\,\mathbf{r})$. The conditional probability $P(E\,|\,\mathbf{r})$ is minimized for a given **r** by selecting $\hat{\mathbf{v}}$ as the codeword **v** that maximizes

$$P(\mathbf{v}\,|\,\mathbf{r}) = \frac{P(\mathbf{r}\,|\,\mathbf{v})}{P(\mathbf{r})}$$

If all information sequence and hence all codewords are equally likely then $P(E \mid \mathbf{r})$ is minimized for a given **r** by choosing $\hat{\mathbf{v}}$ as the codeword **v** that maximizes $P(\mathbf{r} \mid \mathbf{v})$. For a memoryless channel

$$P(\mathbf{r} \mid \mathbf{v}) = \prod_{l=0}^{h+m-1} P(\mathbf{r}_l \mid \mathbf{v}_l) = \prod_{l=0}^{N-1} P(r_l \mid v_l)$$

$$\log P(\mathbf{r} \mid \mathbf{v}) = \sum_{l=0}^{h+m-1} \log P(\mathbf{r}_l \mid \mathbf{v}_l) = \sum_{l=0}^{N-1} \log P(r_l \mid v_l)$$

where the $\log P(r_l \mid v_l)$ is a channel transition probability. The conditional probability $P(\mathbf{r} \mid \mathbf{v})$ is called the path metric [1, pg 517]. The terms $\log P(\mathbf{r}_l \mid \mathbf{v}_l)$ are called branch metrics and denoted as $M(\mathbf{r}_l \mid \mathbf{v}_l)$ whereas the terms $\log P(r_l \mid v_l)$ are called bit metrics and denoted as $M(r_l \mid v_l)$. The path metric $M(\mathbf{r} \mid \mathbf{v})$ can be written as [1, pg 517]

$$M(\mathbf{r} \mid \mathbf{v}) = \sum_{l=0}^{h+m-1} M(\mathbf{r}_l \mid \mathbf{v}_l) = \sum_{l=0}^{N-1} M(r_l \mid v_l)$$

The partial path metric for the first t branches of a path can be expressed as [1, pg 517]

$$M([\mathbf{r} \mid \mathbf{v}]_t) = \sum_{l=0}^{t-1} M(\mathbf{r}_l \mid \mathbf{v}_l) = \sum_{l=0}^{nt-1} M(r_l \mid v_l)$$

This is a minimum error probability rule when all codewords are equally likely. If the codewords are not equally likely then an maximum likelihood decoder is not necessarily optimum, since the conditional probabilities $P(\mathbf{r} \mid \mathbf{v})$ must be weighted by the codeword probabilities $P(\mathbf{v})$ to determine which codeword maximizes $P(\mathbf{v} \mid \mathbf{r})$. In some cases where the codeword probabilities are not known at the receiver a maximum likelihood decoder becomes the best feasible decoding rule. [1 pg 517]

In general there are two main categories of decoding, *Hard-decision decoding* and *Soft-decision decoding*. In *hard-decision decoding* the decoder processes the received sequence which is in binary form whereas in soft-decision decoding the decoder processes a received sequence which is unquantized or quantized in more than two levels.

The metric used in *hard-decision decoding* is the Hamming distance. The objective is to decode the hard-decision received sequence to the closest codeword in the Hamming distance. In this case a maximum likelihood decoder chooses **v** as the codeword that *minimizes* the Hamming distance

$$d(\mathbf{r} \mid \mathbf{v}) = \sum_{l=0}^{h+m-1} d(\mathbf{r}_l \mid \mathbf{v}_l) = \sum_{l=0}^{N-1} d(r_l \mid v_l)$$

In this case the terms $d(\mathbf{r}_l \mid \mathbf{v}_l)$ become the branch metrics whereas the terms $d(r_l \mid v_l)$ become the bit metrics.

8.2 The Viterbi algorithm

The decoding process of the Viterbi algorithm is based on the trellis diagram. The algorithm when applied to the received sequence **r** finds the path through the trellis with the largest metric (*maximum likelihood path*). For a terminated convolutional code (n,k,v), in the first and the last m time units not all the possible states can be reached. In the center portion of the trellis, all states are possible and each time unit contains a replica of the state diagram. There are 2^k branches leaving and entering each state.

8.2.1 Basic Algorithm

Consider a (n,k,v) convolutional encoder with memory m. An information sequence $\mathbf{u} = (\mathbf{u}_0, \mathbf{u}_1, ...)$ of length $K^* = kh$ is encoded into a codeword $\mathbf{v} = (\mathbf{v}_0, \mathbf{v}_1, ...)$ of length $N = n(h+m)$ and $\mathbf{r} = (\mathbf{r}_0, \mathbf{r}_1, ...)$ is the received sequence of length N. Assume that we apply the Viterbi algorithm in order to obtain the information sequence from the received sequence. Before describing the steps of the algorithm we will define some additional elements that will be useful in the following analysis. In the trellis diagram, for every branch j entering state S_i at time unit t there is a predecessor state $S_{predecessor}$.

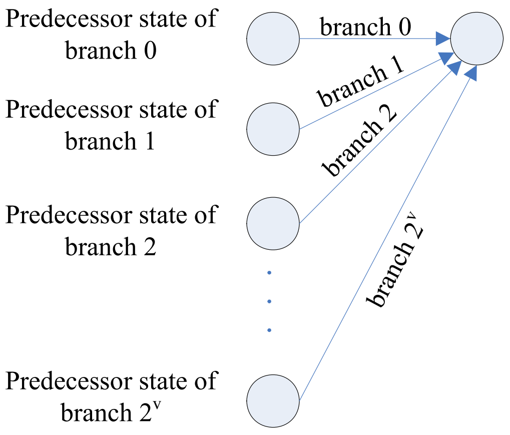

Predecessor state of branch 0
Predecessor state of branch 1
Predecessor state of branch 2
Predecessor state of branch 2^v
branch 0
branch 1
branch 2
branch 2^v

Figure 2-14: Branches and predecessor states

The path with the largest metric is called the *survivor path*.

The predecessor state of the survivor path is called the *predecessor-successor state*.

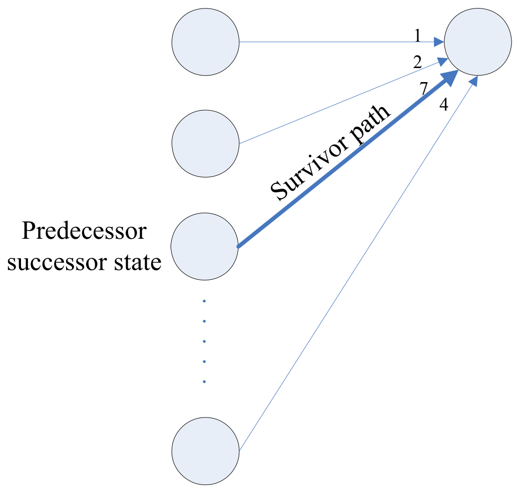


Figure 2-15: Survivor path and the Predecessor successor state

The first step of the Viterbi algorithm is called *Branch Metric Generator.* For every time interval $t_i \rightarrow t_{i+1}$ in the trellis diagram, compute the branch metrics for all the branches entering each state at time unit t_{i+1}, where $i \in [0, h+m-1]$. [1, pg 518][5, pg 87]

The next step is called *ACS* (*add, compare and select*). For every time unit t, where $0 < t < h+m$, for each state compute the partial metric of each path entering that state. The accumulated partial metric of *jth* path entering state S_i is given by the sum of the *jth* branch metric and the metric of the state $S_{predecessor}$. Afterwards, for each state, compare the partial metrics of all the paths entering that state, select the path with the largest metric (*survivor path*), store its path along with its metric and eliminate all other paths. [1, pg 518][5, pg 87]

In practice the information sequence corresponding to the survivor path at each state is stored instead of the surviving path itself. In this case there is no need to invert the estimated codeword $\hat{\mathbf{v}}$ in order to obtain the estimated information sequence $\hat{\mathbf{u}}$.

Moreover the performance of Viterbi algorithm is affected by several additional factors such as *Decoder Memory, Computational Complexity.*

i. *Decoder Memory.* At time unit *t* in the trellis diagram there are 2^v states. Therefore decoder must be able to reserve 2^v words in order to store 2^v survivor paths and their metrics. Memory space increases exponentially with the overall constraint length and hence in practice, it is not feasible to implement the Viterbi decoder for large *v*.

ii. *Computational Complexity*. At time unit t in the trellis diagram 2^k binary additions and 2^k-1 binary comparisons are performed for every state. Therefore the Viterbi computational complexity is proportional to the *branch complexity* $2^v 2^k$. Consider an information sequence of length $K^* = kh$. The trellis diagram will contain h+m time units (stages). Therefore the complexity of the Viterbi algorithm is on the order of $O\left[2^k 2^v (h+m)\right]$. The number of time units in the trellis diagram h, is linear factor in the complexity. However there will be an exponential increase of the computational complexity if the number of inputs or the overall constraint length of the encoder increases. Because of the exponential dependence of the computational complexity on the overall constraint length v and the number of inputs of the encoder k, in practical applications Viterbi algorithm is used for codes with low code-rate and relatively small overall constraint length.

There are two general methods for implementing Viterbi decoding:

$\Rightarrow$ *Hard decision output Viterbi algorithm*

$\Rightarrow$ *Soft-output Viterbi decoding algorithm*

Hard decision output Viterbi algorithm is based on the basic Viterbi algorithm and is implemented by the use of the hamming distance as the partial metric.

In Soft-output Viterbi algorithm the unquantized received sequence is processed by the use of the Euclidean distance or a correlation metric. The real valued inputs and the use of the above metrics in the Soft-output Viterbi algorithm, increases the computational complexity and the required storage memory compared with the hard decision output Viterbi algorithm. For this reason in WFTP system we implemented a software version of a hard decision output Viterbi algorithm.

9. EVALUATION OF CONVOLUTIONAL CODES

The most important metrics that can be used as guidance in order to evaluate convolutional codes are the free distance d_{free}, and the overall constraint length v. An encoder can always correct an error sequence **e,** if $w(\mathbf{e}) < \frac{d_{free}}{2}$. Because of the dependence of the error correcting capability of a convolutional code on the free distance of the code, for a given code rate and overall constraint length the best convolutional code is the one with the maximum free distance. However, free distance depends on the overall constraint length. Therefore maximizing free distance results in an increase of the constraint length and hence in an increase of the computational complexity in the decoder. In general d_{free} is of primary importance in determining the performance at high SNR's. Another quantity that is used as a performance measure for convolutional codes is the asymptotic coding gain. In general coding gain is defined as the reduction in the $\frac{E_b}{N_0}$ required to achieve a specific error probability for a coded communication system compared with an uncoded communication system. The asymptotic coding gain is the coding gain for large SNR and depends only on the code rate and the free distance of the code. For a hard decision decoder the asymptotic coding gain γ is defined as [1, pg 18]

$$\gamma \triangleq 10\log_{10}\left(\frac{Rd_{free}}{2}\right)dB,$$

where R is the code rate $\frac{k}{n}$ and d_{free} the free distance of the (*n, k, v*) convolutional code.

For a soft decision decoder the asymptotic coding gain γ is defined as

$$\gamma \triangleq 10\log_{10}\left(Rd_{free}\right)dB$$

Notice that there is an increase of 3dB over the hard decision. However in soft decision decoding, the decoding complexity increases owing to the need to accept real-valued inputs.

In general when designing a coding system for error control in a communication system, it is desired to minimize the SNR required to achieve a specific error rate. This is equivalent to maximizing the coding gain of the system compared to an uncoded system, using the same modulation signal set.

The most practically important encoders are the *nonsystematic feedforward* and *systematic feedback* convolutional encoders. Since free distance is the most important criterion for evaluating convolutional codes the two categories of the encoders will be compared on the basis of their bounds on free distance.

The lower bounds for nonsystematic convolutional codes are shown to lie above the upper bounds for systematic codes and it is concluded that more free distance is available with nonsystematic convolutional codes. In particular in systematic encoder realizations there is a reduced number of modulo-2 adders compared with the nonsystematic encoders. This results in reduced free distance in systematic encoders. For asymptotically large constraint length K the performance of a systematic code of overall constraint length K is approximately the same as that of a nonsystematic code of constraint length K(1-R). [4, pg 763]

Consequently the best nonsystematic codes achieve lower error probabilities than the best systematic codes when used with maximum likelihood or sequential decoding.
In general systematic feedback form of encoder realization is preferred in cases where decoding is done offline, decoder is subject to temporary failures or the channel is known to be noiseless during certain time intervals and decoding becomes unnecessary.
On the other hand, nonsystematic feedforward encoders may be preferred when using terminated convolutional codes. In nonsystematic feedforward encoders the termination sequence is $k \times m$ zeros appended in the end of the information sequence. Though in systematic feedback encoders the termination sequence depends on the information sequence and cannot be chosen arbitrarily. This results in additional complexity in the encoder.

In the WFTP system we implemented a nonsystematic feedforward convolutional encoder with the corresponding Viterbi decoder.

The selection of the most suitable convolutional codes for our system was based on the optimum convolutional codes for code rates $\frac{1}{2}, \frac{1}{3}, \frac{1}{4}, \frac{2}{3}, \frac{3}{4}$ which are listed in the following tables.

v	$\mathbf{g}^{(0)}$	$\mathbf{g}^{(1)}$	$\mathbf{g}^{(2)}$	d_{free}	γ(dB)
1	1	3	3	5	-0.79
2	5	7	7	8	1.25
3	13	15	17	10	2.22
4	25	33	37	12	3.01
5	47	53	75	13	3.36
6	117	127	155	15	3.98
7	225	331	367	16	4.25
8	575	623	727	18	4.77
9	1167	1375	1545	20	5.23

Table 2-6: Optimum rate R=1/3 convolutional codes [1, pg 539]

v	$\mathbf{g}^{(0)}$	$\mathbf{g}^{(1)}$	$\mathbf{g}^{(2)}$	$\mathbf{g}^{(3)}$	d_{free}	γ(dB)
1	1	1	3	3	6	-1.25
2	5	5	7	7	10	0.97
3	13	13	15	17	13	2.11
4	25	27	33	37	16	3.01
5	45	53	67	77	18	3.52
6	117	127	155	171	20	3.98
7	257	311	337	355	22	4.39
8	533	575	647	711	24	4.77
9	1173	1325	1467	1751	27	5.28

Table 2-7: Optimum rate R=1/4 convolutional codes [1, pg 539]

v	$\mathbf{g}^{(0)}$	$\mathbf{g}^{(1)}$	d_{free}	γ(dB)
1	3	1	3	-1.25
2	5	7	5	0.97
3	13	17	6	1.76
4	27	31	7	2.43
5	53	75	8	3.01
6	117	155	10	3.98
7	247	371	10	3.98
8	561	753	12	4.77
9	1131	1537	12	4.77

Table 2-8: Optimum rate R=1/2 convolutional codes [1, pg 540]

v	v_1	v_2	$\mathbf{g}_1^{(0)}$	$\mathbf{g}_1^{(1)}$	$\mathbf{g}_1^{(2)}$	$\mathbf{g}_2^{(0)}$	$\mathbf{g}_2^{(1)}$	$\mathbf{g}_2^{(2)}$	d_{free}	γ(dB)
2	1	1	3	1	0	2	3	3	3	0
3	1	2	3	2	1	4	1	7	4	1.25
4	2	2	6	5	1	7	2	5	5	2.22
5	2	3	7	6	3	12	1	13	6	3.01
6	3	3	6	13	13	13	06	17	7	3.67
7	3	4	16	13	3	25	5	34	8	4.26
8	4	4	37	31	16	23	14	35	8	4.26
9	4	5	27	23	16	46	17	41	9	4.77

Table 2-9: Optimum rate R=2/3 convolutional codes [6]

v	v_1	v_2	v_3	$\mathbf{g}_1^{(0)}$	$\mathbf{g}_1^{(1)}$	$\mathbf{g}_1^{(2)}$	$\mathbf{g}_1^{(3)}$	$\mathbf{g}_2^{(0)}$	$\mathbf{g}_2^{(1)}$	$\mathbf{g}_2^{(2)}$	$\mathbf{g}_2^{(3)}$	$\mathbf{g}_3^{(0)}$	$\mathbf{g}_3^{(1)}$	$\mathbf{g}_3^{(2)}$	$\mathbf{g}_3^{(3)}$	d_{free}	γ(dB)
2	0	1	1	1	1	1	0	3	0	0	1	3	2	0	2	3	-1.25
3	0	1	2	1	1	1	1	0	3	1	2	0	2	5	5	4	0.51
4	1	1	2	0	1	2	3	3	0	1	2	2	4	1	5	4	1.76
5	1	2	2	3	3	2	2	5	2	7	0	4	7	0	1	5	2.73
6	2	2	2	5	4	3	2	4	6	5	5	6	1	4	3	6	3.52
7	2	2	3	02	03	04	07	03	07	03	05	15	02	02	17	6	4.19
8	2	3	3	04	06	07	07	01	12	05	14	00	07	14	11	7	4.77
9	3	3	3	03	06	10	15	00	16	03	13	16	05	02	17	8	5.28

Table 2-10: Optimum rate R=3/4 convolutional codes [6]

The codes that are listed in the preceding tables, are generated by nonsystematic feedforward encoders in controller canonical form and are optimum in the sense that for a specific code rate $R=\frac{k}{n}$ and constraint length v the code listed in the appropriate table has the maximum free distance of all the (n,k,v) codes.

For a code rate $R=\frac{k}{n}$ and overall constraint length v, the generator sequences are provided in octal form. Consider the optimum convolutional encoder (2, 1, 3). The generator sequences $\mathbf{g}^{(0)}$, $\mathbf{g}^{(1)}$ in binary form are given by (001011) and (001111). However v_1=m=v=3 and hence only the rightmost $m+1$ bits form the binary representation of each generator sequence. In general, the binary representation of each generator sequence $\mathbf{g}_i^{(j)}$ is formed by the leftmost v_i+1 bits. In order to obtain the octal form of a generator sequence $\mathbf{g}_i^{(j)}$, we consider consecutive triplets of bits by starting from the rightmost bit. If the number of bits is not multiple of three, pad at the left with the appropriate zero bits.

Notice from the above tables that for large overall constraint length, the asymptotic coding gain and the free distance increases. Therefore we would like to select the convolutional code with maximum free distance and asymptotic coding gain. However because of the exponential dependence of the decoding complexity on the overall constraint length our choices are limited.

The upper bound on the BER of the optimum codes with hard-decision decoding and coherent BPSK as a function of the bit SNR in dB, are plotted in the following figures.

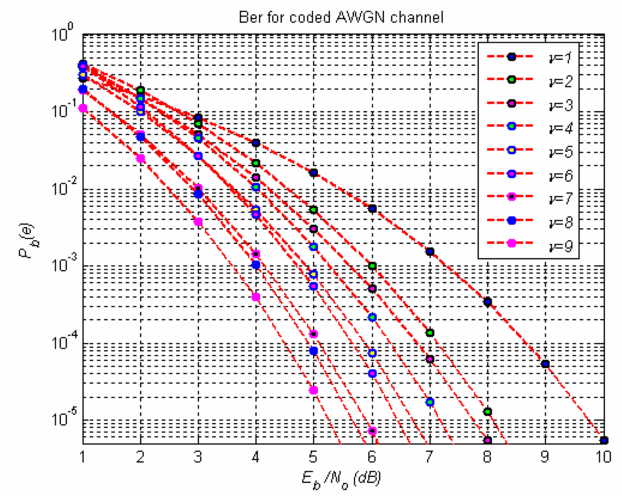


Figure 2-16: Upper bound on the BER for R=1/2 codes listed in Table 2-5.

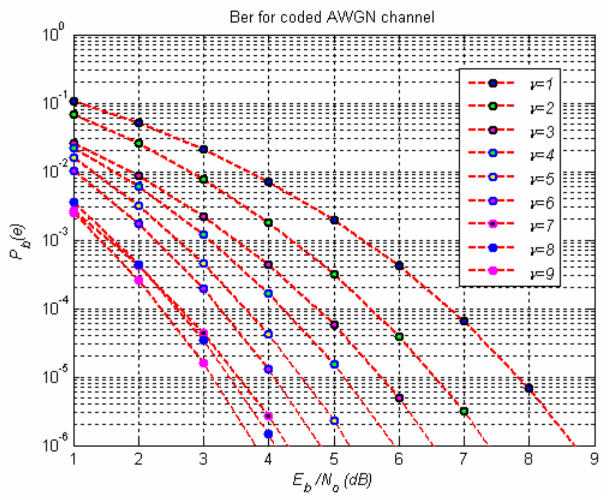


Figure 2-17: Upper bound on the BER for R=1/3 codes listed in Table 2-4.

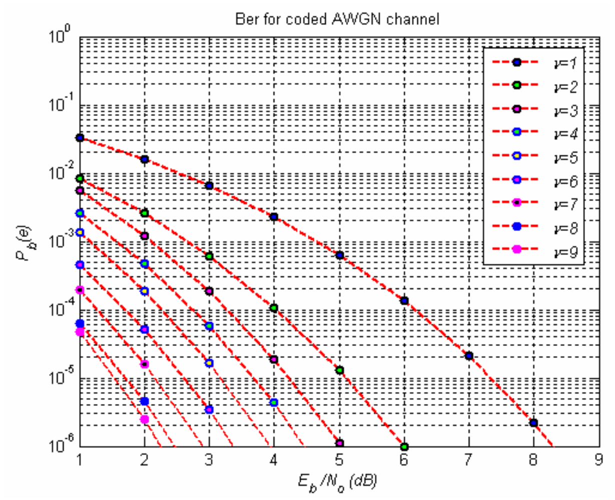


Figure 2-18: Upper bound on the BER for R=1/4 codes listed in Table 2-4.

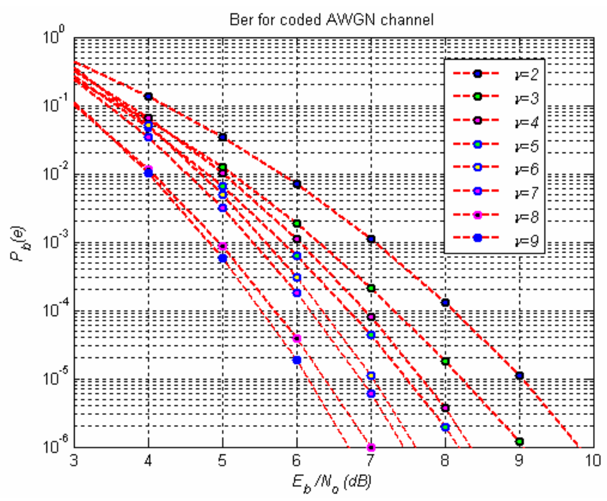


Figure 2-19: Upper bound on the BER for R=2/3 codes listed in Table 2-6.

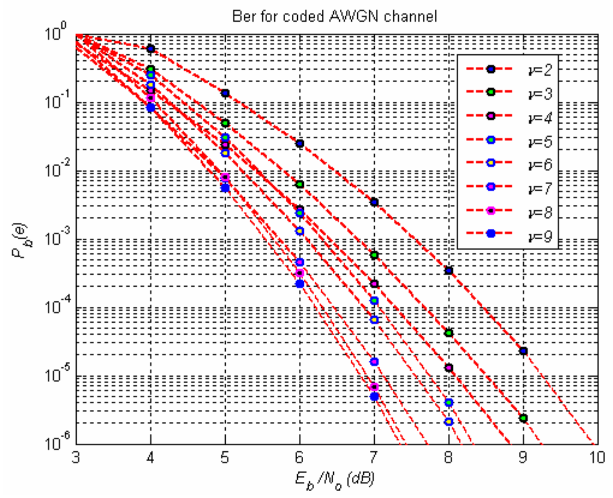


Figure 2-20: Upper bound on the BER for R=3/4 codes listed in Table 2-7.

The preceding figures denote that by increasing the overall constraint length, a specific probability of error can be achieved in lower SNR. However there is a contemporary exponential increase in the decoding complexity.

In the following tables the optimum convolutional codes are compared on the basis of their typical decoding timings in the personal computers used for the experimental operation of the WFTP system for 100.000 bits.

v	*R=1/2*	*R=1/3*	*R=1/4*	*R=2/3*	*R=3/4*
1	0.22 sec	0.26 sec	0.30 sec	-	-
2	0.28 sec	0.32 sec	0.43 sec	0.20 sec	0.21 sec
3	0.52 sec	0.38 sec	0.53 sec	0.30 sec	0.27 sec
4	0.49 sec	0.52 sec	0.58 sec	0.39 sec	0.39 sec
5	0.78 sec	0.76 sec	1.22 sec	0.64 sec	0.63 sec
6	1.22 sec	1.23 sec	1.28 sec	1.04 sec	1.05 sec
7	2.21 sec	2.16 sec	2.20 sec	2.02 sec	2.05 sec

Table 2-11: Typical decoding timings of the optimum rates convolutional codes for 100.000 bits

In general, large free distances and low error probabilities are achieved not by increasing *k* and *n* but by increasing the memory order *m*. Thus it is desired to use a convolutional encoder with the maximum overall constraint length. However as we can obtain from the table 2-11, the decoding time increases along with the increase of the overall constraint length *v*. In the WFTP system the desired typical processing time for the different modules must be less than 1 sec in order to reduce the processing time to the minimum possible. Therefore we must select the convolutional codes that achieve the maximum possible free distance along with an acceptable decoding complexity.
Though we can obtain that for $v > 6$ the gain in SNR, as *v* increases, is less than 1dB. Moreover for $v>6$ the decoding time exceeds the 2 sec for all the code rates. Consequently the set of convolutional codes that meet our requirements and will be applied to the WFTP system are listed in the following table.

R	***v***	***m***	d_{free}	γ(dB)	***Branch Complexity***
1/4	6	6	20	3.98	2^7
1/4	4	4	16	3.01	2^5
1/3	6	6	15	3.98	2^7
1/3	5	5	13	3.36	2^6
1/2	6	6	10	3.98	2^7
1/2	5	5	8	3.01	2^6
2/3	6	3	7	3.67	2^8
3/4	6	2	6	3.52	2^9
2/3	5	3	6	3.01	2^7
3/4	5	2	5	2.73	2^8
2/3	4	2	5	2.22	2^6
3/4	4	2	4	1.76	2^7

Table 2-12: The convolutional codes that will be applied on the WFTP system.

The convolutional codes in the preceding table are listed in descending order on the basis of their free distance. The code with the largest free distance is expected to achieve better performance.

However the use of a convolutional encoder in an uncoded communication system reduces the transmission rate. The decrease of transmission rate depends on the code rate of the convolutional code. The code rate determines the redundant information that will be added to the actual information sequence. Assume that a packet of length N in the WFTP system is an input to a $R = \frac{k}{n}$ convolutional encoder. The encoded packet will be of length $\frac{N}{R}$. Consequently this results in a decrease of transmission rate, since the transmission of the same actual information requires more time units.

Consider the uncoded WFTP system where each packet contains N bits and each symbol corresponds to $\log_2(M)$ bits. If the number of samples per symbol is T$_{sym}$ and the sampling frequency is Fs, then the total transmission time per packet is

$$t_{TRANSMISSION} = \frac{N}{Fs\log_2(M)} T_{sym}$$

Therefore the transmission rate is

$$R_{TRANSMISSION} = \frac{N}{t_{TRANSMISSION}} = \frac{Fs\log_2(M)}{T_{sym}}$$

If we add an (*n, k, v*) convolutional encoder to the system, the total transmission time will be given by

$$t_{TRANSMISSION} = \frac{\frac{Nn}{k}}{Fs\log_2(M)} T_{sym} = \frac{Nn}{Fs\log_2(M)k} T_{sym},$$

and the transmission rate

$$R^{'}_{TRANSMISSION} = \frac{N}{t_{TRANSMISSION}} = \frac{kFs\log_2(M)}{nT_{sym}} = \frac{k}{n} R_{TRANSMISSION}$$

Eventually the transmission rate of the system is decreased by a factor $\frac{k}{n}$. The transmission rate loss percentage is given by $\frac{n-k}{n}100\%$.

In the following table we represent the transmission rate loss for each one of the convolutional codes that we have selected for the *WFTP* system.

	Code rates				
	1/4	*1/3*	*½*	*2/3*	*3/4*
Transmission rate loss	75%	66%	50%	33%	25%

Table 2-13: Transmission rate loss as a function of code rate

The transmission rate loss per code rate is listed in descending order. Notice that the 1/4 code, which is expected to achieve the best performance in the set of codes listed in Table 2-12, results in an 75% transmission rate loss while the 3/4 code, which is expected to achieve the worst performance, results in a 25% transmission rate loss. In general the selection of the convolutional codes that will be used in a communication system must compromise the tradeoff between performance, transmission rate loss and decoding complexity.

10. IMPLEMENTATION OF THE FECC MODULES

In *WFTP* system the transmitter contains a nonsystematic feedforward convolutional encoder module and the receiver the corresponding hard-decision Viterbi decoder module. The two modules are implemented in *MATLAB*.

10.1 Encoder Implementation

The encoder module consists of two components:

⇒ Polynomial to trellis diagram
⇒ Convolutional Encoder

10.1.1 Polynomial to trellis diagram

Encoding of convolutional codes is based on the state diagram or equivalently the trellis diagram in steady state. The *poly2trellis* function accepts as inputs the constraint lengths of the convolutional encoder and the generator sequences in octal form and constructs the trellis diagram that corresponds to the specified encoder.

The generator sequences are given as input to the function through the *kxn* matrix G_{matrix} which is of the form

$$G_{matrix} = \begin{bmatrix} g_1^{(0)} & g_1^{(1)} & \cdots & g_1^{(n-1)} \\ g_2^{(0)} & g_2^{(1)} & \cdots & g_2^{(n-1)} \\ \vdots & \vdots & \cdots & \vdots \\ g_k^{(0)} & g_k^{(1)} & \cdots & g_k^{(n-1)} \end{bmatrix},$$

where $g_i^{(j)}$ is the *ith* generator sequence with response to *jth* output in octal form[6].

The pseudocode of the function *poly2trellis* is given in Listing 2-1.

```
/*inputs*/
$v_i$, $G_{matrix}$
$v = sum(v_i)$ , $m = max(v_i)$ , $[k, n] = size(G_{matrix})$
states = $dec2bin(0{:}2^v\text{-}1)$

/*possible blocks of information containing k bits*/
u = $dec2bin(0{:}2^k\text{-}1 , k)$

for every state
    store state number
    /*add state to the delay elements*/
    start_index = 1
    end_index = 0
    for  every shift register $i$
        end_index = end_index + $v_i$
        shift_registers( $i$, $1{:}v_i$) = state(start_index:end_index)
        start_index = start_index + $v_i$
    end
    for every possible block of information
        for every jth output
            g_current = ( $g_1^{(j)} \ldots g_k^{(j)}$ )$^\mathbf{T}$
            for every element in g_current
                /* find the connection lines of  input i and its corresponding delay elements */
                /*to the adder that produces jth output*/
                ith_connections = find( oct2bin(g_current) = =1 )
                Store in xor_buffer the bit of the ith input and the bits of the connected delay elements
                output_bit = mod( sum(xor_buffer) , 2 )
                codeword=[codeword;output_bit]
            end
            store codeword in octal form
        end
        shift right by 1 all bits in the delay elements
        add in the leftmost delay element of ith shift register the ith current input bit
        calculate state number and store next state
    end
end
```

Listing 2-1: Function poly2trellis for polynomial representation to trellis diagram conversion

Remarks. The result of the function *poly2trellis* is a struct which contains the number of input symbols 2^k, the number of output symbols 2^n, the number of states 2^v, the next states and the corresponding codewords. In the nextStates matrix the *ith* column number corresponds to the binary input in the encoder *dec2bin(i-1, k)*. The *jth* row number indicates the initial state with binary representation that results from the labeling convention which was mentioned in previous section.

[6] In order to convert the binary representation of the generator sequence $\mathbf{g}_i^{(j)}$ into an octal form, consider consecutive triplets of bits, by starting from the rightmost bit. The rightmost bit in each triplet is the least significant. If the number of bits is not a multiple of three, then place zero bits at the left end as necessary.

Therefore the row with number *j* corresponds to the state with label S_t where *t=j-1* and $t = b_0 + 2^1 b_1 + ... + 2^{v-1} b_{v-1}$. Consequently nextStates(*j,i*) corresponds to the encoder's next state if input is *i-1* and previous state is S_{j-1}.

10.2 Convolutional Encoder

The convolutional encoder is implemented as a look up table. Starting at zero state and using the trellis diagram produced by poly2trellis function the information sequence is encoded into the corresponding codeword.

```
/*inputs*/
Trellis
current_state=0
for every block of k bits
    input ← calculate the decimal representation of the information block
    output ← Trellis.outputs (current_state+1, input+1)
    codeword ← [codeword; output]
    current_state ← Trellis.nextStates(current_state+1, input+1)
end
```

Listing 2-2: Function convenc encodes the information sequence into the corresponding codeword.

Remarks. The information sequence is encoded by the convolutional encoder in blocks of *k* bits. Because of the structure of the trellis struct, which is obtained by the poly2trellis function, it is desirable to express the information blocks in decimal form. In particular, consider the decimal representation "x" of a block of *k* bits. If the current state is *y,* the output of the encoder in octal form is given by trellis.outputs(y+1, x+1) while the corresponding next state in decimal form trellis.nextStates(y+1, x+1).

10.3 Decoder Implementation

The Viterbi decoder implementation is a hard-decision decoder which is based on the *Basic Viterbi algorithm* and the *traceback* technique.

Step 1. Construct the *metrics table* along with the *history table*. The 2^vx*(h+m)* metrics table contains the metric of the survivor path of each state for all (m+h) time units. History table contains the predecessor-successor state of each state for all (m+h) time units. This step combines the *Branch Metric Generator* and the *ACS* units of Viterbi algorithm. Because we are interested in terminated convolutional codes, at time unit t=0 the initial state is S_0 and its metric is 0. Increase t by 1 and for each state compute the partial metric of each path entering that state. The partial metric of each path is the sum of the Hamming distance between, the received n bits and the n bits that correspond to the path[7] and the metric of the predecessor state. Afterwards for each state, compare the partial metrics of all the paths entering that state and select the path with the largest metric (*survivor path*). The predecessor state of this path is called the *predecessor-survivor state*. Therefore for time unit t we store in the *metrics table* the metric of the survivor path for each state and in the history table the number of predecessor-survivor state for each state.

Step 2. Start from the last record in the metrics table that correspond to the time unit h+m-1. Select the state having the smallest partial metric and save the number of that state in the h+m-1 position of the *traceback path* table.

Step 3. For the state being selected in step 2 check in the history table its predecessor state. Select the new state and save its number in the *traceback path* table. Continue working backward until the beginning of trellis is reached.

Step 4. Work forward through the states that are stored in the *traceback path* table. For each transition between states *i* and *i+1* in the *traceback path* table, look up from the trellis diagram the input bit/bits that corresponds to the specified transition. The process finishes when the end of the *traceback path* table is reached. The result of Step 4 is concatenation of the information sequence and the termination sequence. In order to obtain the actual information, the last *km* bits must be discarded.

[7] Each path in the trellis diagram is labeled with the corresponding n output bits.

C h a p t e r 3

CYCLIC REDUNDANCY CHECK

1. INTRODUCTION

Cyclic Redundancy Check (CRC) is an error-checking code that is widely used in data communication systems. The CRC is a very powerful but easily implemented technique to obtain data reliability and is used to protect *k-bit* blocks of data called Frames. Using this technique, the transmitter appends an extra (*n-k*)-bit sequence to every frame called Frame Check Sequence (FCS). The resulting *n*-bit frame is exactly divisible by some predetermined number. The receiver then divides the incoming frame by that number and, if there is no remainder, assumes that there is no error. Therefore the FCS holds redundant information about the frame that helps the receiver detect errors in the frame. Since the CRC is only an error detecting code, the position of an error in the received message can not be determined. CRC codes are used in communication protocols that use automatic repeat request (ARQ).

2. FRAME CHECK SEQUENCE GENERATION

In CRC codes the FCS is obtained using modulo-2 arithmetic. In general FCS is the remainder of the *binary long division* between the *k*-bit block of data and a predetermined divisor. Assume that *D* is the *k*-bit block of data, *F* the (*n-k*)-bit frame check sequence, *T* the n-bit frame to be transmitted and *P* the predetermined divisor consisting of *n-k+1* bits. The frame to be transmitted *T* is produced by shifting left the block of data *D* by (*n-k*) bits and padding the rightmost (*n-k*) bits with zeros. Adding *F* to the rightmost (*n-k*) zeros yields to the concatenation of *D* and *F* which is $T = 2^{n-k} D + F$.[8, pg207].

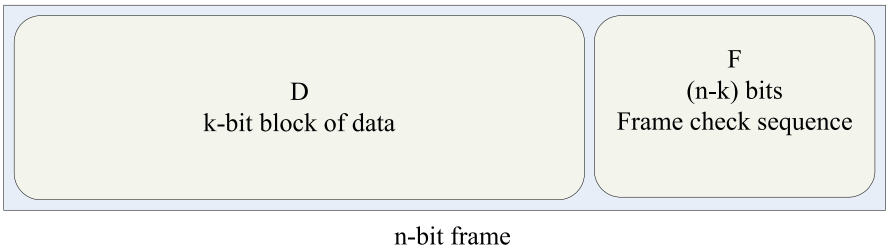


Figure 3-1: Concatenation of Frame check sequence and the data block

In order to have no transmission errors in the receiver, the remainder of the division $\frac{T}{P}$ should be zero. Suppose we divide $2^{n-k}D$ by *P*.

$$\frac{2^{n-k}D}{P} = Q + \frac{R}{P},$$

where *Q* is the quotient and *R* the remainder. Therefore the frame to be transmitted can be written as $T = 2^{n-k}D + R$. Rewrite the division of the frame *T* and the predetermined divisor.

$$\frac{T}{P} = \frac{2^{n-k}D + R}{P} = \frac{2^{n-k}}{P} + \frac{R}{P} = Q + \frac{R}{P} + \frac{R}{P}$$

However in modulo-2 addition any binary number added to itself yields zero. Thus $\frac{T}{P} = Q$. There is no remainder and therefore *T* is exactly divisible by *P*.

The frame check sequence is obtained by the (*n-k*) bits of the remainder of the division of $2^{(n-k)}D$ with *P*.

The process of frame check sequence generation will be illustrated with the following example.

Example 3-1

In order to understand the frame check sequence generation we represent a simple binary long division between the bit strings 1110 (*divisor*) and 1100011(*dividend*).

```
          1011     quotient
1110   | 1100011
         1110     quotient = (degree(1100)= = degree(1110))=1,  remainder=XOR(1100,1110)=010
          0100    quotient = (degree(1110)= = degree(0100))=0
          0000    remainder=XOR(0100,0000)=100
           1001   quotient = (degree(1110)= = degree(1001))=1, remainder=XOR(1110,1001)=0111
           1110
            1111  quotient = (degree(1110)= = degree(1111))=1, remainder=XOR(1110,1111)=0001
            1110
             001  remainder
```

Table 3-1: Binary long division

Consider *D*=1001 and *P*=1101. Since *k*=4 and *n-k+1*=4, the length of the frame to be transmitted *T* is given by *n*=4+*k*-1=7. Thus the length of the FCS is *n-k* = 3.

Step 1. Shift left block of data by (*n-k*)=3 bits and pad with zeros. Thus *D*=1001000.

Step 2. Perform binary long division between the padded block of data *D* and the predetermined divisor *P*.

```
             1111    Quotient
1101  | 1001000
        1101     quotient = (degree(1001)= = degree(1101))=1,  remainder=XOR(1001, 1101)=0100
         1000
         1101    quotient = (degree(1000)= = degree(1101))=1, remainder=XOR(1000, 1101)=0101
          1010
          1101   quotient = (degree(1010)= = degree(1101))=1, remainder=XOR(1010, 1101)=0111
           1110
           1101  quotient = (degree(1110)= = degree(1101))=1, remainder=XOR(1110, 1101)=0011
            011
```

Table 3-2: Computation of the frame check sequence

Therefore *T*=1001**011**.

At the receiver the received frame is divided with the predetermined divisor *P*.

```
             1111    Quotient
1101  | 1001011
        1101
         1000
         1101
          1011
          1101
           1101
           1101
            000
```

Table 3-3: Division of the received frame by the predetermined divisor P

Since the remainder is 000 the frame received with no errors.

CRC codes can be described using polynomial representation. Starting from the least significant (rightmost) bit of the binary representation, express all values as polynomials in a variable *X*, with binary coefficients. The coefficients correspond to the bits in the binary number. The polynomial representation of the frame *T* is given by [8, pg 210]

$$T(X) = X^{n-k}D(X) + R(X),$$

where *D(X)* is the polynomial representation of the data block and *R(X)* is the polynomial representation of FCS. Notice that the coefficients of the polynomials can be drawn from the *GF*(2) as described in Chapter 2. The operations between the binary coefficients are modulo-2 addition and multiplication.

An error *E(X)* is undetectable only if it is divisible by the generator polynomial *P(X)*. Detectable errors are the errors that are not divisible by *P(X)* and are listed below.

- All single-bit errors, if *P(X)* has more than one nonzero term.
- All double-bit errors, as long as *P(X)* has a factor with three terms.

- Any odd number of errors, as long as *P(X)* contains a factor (*X*+1).
- Any burst error for which the length of the burst is less than or equal to *n-k*.
- A fraction of error bursts of length *n-k*+1.
- A fraction of error bursts of length greater than *n-k*+1.

[9, pg 64]

The table 3-4 represents the most widely used generator polynomials.

CRC Code	***Generator Polynomial***
CRC-16	$X^{16}+X^{15}+X^{2}+1$
SDLC (IBM, CCITT)	$X^{16}+X^{15}+X^{2}+1$
CRC-16 REVERSE	$X^{16}+X^{14}+X+1$
SDLC REVERSE	$X^{16}+X^{11}+X^{4}+1$
LRCC-16	$X^{16}+1$
CRC-12	$X^{12}+X^{11}+X^{3}+X^{2}+X+1$
LRCC-8	$X^{8}+1$
ETHERNET, CRC-32	$X^{32}+X^{26}+X^{23}+X^{22}+X^{16}+X^{12}+X^{11}$ $+X^{10}+X^{8}+X^{7}+X^{5}+X^{4}+X^{2}+X+1$

Table 3-4: Commonly used generator polynomials [9, pg 64]

In the WFTP system we used the CRC-16 generator polynomial, $X^{16}+X^{15}+X^{2}+1$ with corresponding binary representation 11000000000000101.

3.1 Error detection in WFTP system

The Cyclic Redundancy Check in WFTP system consists of two modules. The function *crc_addfcs* in the transmitter adds the appropriate FCS to the data block that accepts as input. At the receiver, the function *crc_err_detect* checks the received frame for errors.

```
crc_addfcs (Message, Pattern)

k ← length(Message)
n ← length(Pattern)+k-1
pad Message with n-k zeros
initialize register with the n-k first bits of padded Message
current_element ← n-k+1
while current_element~ = n+1 do
    shift left register's contents by 1 and add current_element
    current_element ← current_element+1
    if degree(pattern) = = degree(register) then
        register ← xor (pattern, register)
    else
        register ← xor (zeros(1,n-k+1), register)
    end
end
FCS ← last n-k register's contents
add FCS to Message
```

Remarks. The function *crc_addfcs* accepts as inputs the block of data *D* and the predetermined divisor *P* in order to produce the corresponding FCS and construct the frame to be transmitted. The initial message is padded with $n-k$ zeros. The procedure that will be described in the following steps, implements the binary long division between the padded message and the predetermined divisor. Initially the register is filled with the $n-k$ rightmost bits of the padded message. In the first iteration the contents of the register are shifted left by one, and the $n-k+1$ bit of the message is entered in the rightmost position in the register. If the degree of the binary representation of the register's contents is the same as the degree of the predetermined divisor then the result of the xor operation between these two bit strings is stored in the register. In any other case the result of the xor operation between the predetermined divisor and $n-k+1$ zeros is the remainder and is stored in the register. This process continues for every bit in the padded message. The $n-k$ rightmost bits that are in the shift register after the $k-1$ iterations, represent the frame check sequence.

```
crc_err_detect (ReceivedMessage, Pattern)

k ← length(ReceivedMessage)
n ←  length(Pattern)+k-1
initialize register with the n-k first bits of ReceivedMessage
current_element ← n-k+1
while current_element~ = n+1 do
        shift left register's contents and add current_element
        current_element ← current_element+1
        if  degree(pattern) = = degree(register) then
            register ← xor (pattern, register)
        else
            register ← xor (zeros(1,n-k+1), register)
        end
end
rx ← sum (register)
if rx = = 0 then
        no error
else
        error
```

Remarks. The same procedure is followed in the function *crc_err_detect*. In order to obtain if there is an error in the received sequence, we check the remainder of the binary long division between the received sequence and the predetermined divisor.

C h a p t e r 4

PHASE SHIFT KEYING

1. MPSK

1.1 Modulation, Demodulation, Detection

In the WFTP system, along with the error correction and error detection modules we implemented the necessary modules for the MPSK modulation scheme.

In MPSK $k = \log_2(M)$ data bits are represented by a symbol of different phase and hence the bandwidth efficiency is increased *k* times. The M-ary PSK signal set is defined as [7, pg 397]

$$u_m(t) = g_T(t)\cos(2\pi \mathrm{f_c}\, t + \frac{2\pi \mathrm{m}}{M}) \quad m = 1,\dots,M \ , \ 0 \le \mathrm{t} \le \mathrm{T}$$

where $g_T(t)$ is a rectangle pulse which is given by

$$g_T(t) = \sqrt{\frac{2Es}{T}} \ , \ 0 \le t \le T$$

The above expression can be written as

$$\begin{aligned} u_m(t) &= g_T(t)\cos(2\pi \mathrm{f_c}\, t + \frac{2\pi \mathrm{m}}{M}) \\ &= \sqrt{\frac{2Es}{T}}\cos(2\pi\, \mathrm{fc\, t})\cos(\frac{2\pi\mathrm{m}}{\mathrm{M}}) - \sin(2\pi\, \mathrm{fc\, t})\sin(\frac{2\pi\mathrm{m}}{\mathrm{M}}) \\ &= \sqrt{\frac{2}{T}}\sqrt{Es}\ Amc\cos(2\pi\, \mathrm{fc\, t}) - \sqrt{\frac{2}{T}}\sqrt{Es}\ Ams\sin(2\pi\, \mathrm{fc\, t}) \end{aligned}$$

where

$$A_{mc} = \cos\left(\frac{2\pi m}{M}\right) \quad m = 1,\dots,M$$
$$A_{ms} = \sin\left(\frac{2\pi m}{M}\right) \quad m = 1,\dots,M$$

The orthogonal basis functions are given by

$$y_1(t) = \sqrt{\frac{2}{T}}\cos\left(2\pi f_c t\right), \quad 0 \le t \le T$$

$$y_2(t) = -\sqrt{\frac{2}{T}}\sin\left(2\pi f_c t\right), \quad 0 \le t \le T$$

Therefore the signal set of MPSK can be written as

$$u_m(t) = \sqrt{E_s}A_{mc}\, y_1(t) + \sqrt{E_s}A_{ms}\, y_2(t), \quad m = 1,\ldots,M, \quad 0 \le t \le T$$

where E_s is the energy per symbol.

The phase of each symbol is given by

$$\theta_m = \frac{2\pi m}{M}$$

The MPSK signal constellation is two-dimensional and hence each signal is represented by a two-dimensional vector of the form [7, pg 398]

$$\mathbf{s_m} = (s_{m1}\ s_{m2}) = \left(\sqrt{E_s}\cos\left(\frac{2\pi m}{M}\right)\ \sqrt{E_s}\sin\left(\frac{2\pi m}{M}\right)\right)$$

The polar coordinates of the signal are $\left(\sqrt{E_s}, \theta_m\right)$ where $\sqrt{E_s}$ is its magnitude and θ_m is its angle with respect to the horizontal axis. The signal points are equally spaced on a circle of radius $\sqrt{E_s}$ and centered at the origin.

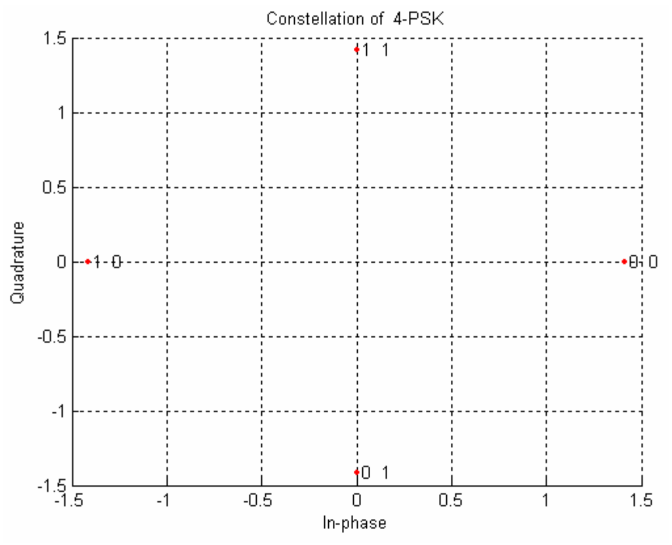


Figure 4-1: 4-PSK contellation

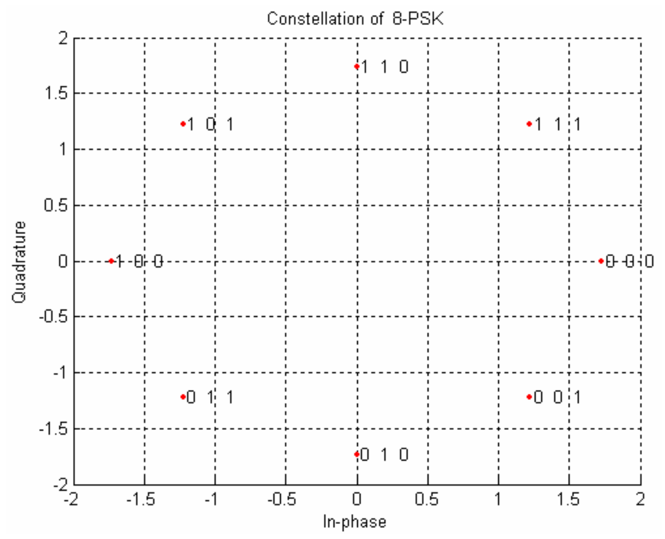


Figure 4-2: 8-PSK contellation

In PSK modulation technique the main process in done by the mapping of k bits to the corresponding symbols. Since every k bits are represented by a symbol there are 2^k possible combinations of bits and hence 2^k symbols.

m	Bits	Phase	s_m	
1	000	π/8	$0.92\sqrt{E_S}$	$0.38\sqrt{E_S}$
2	001	3π/8	$0.38\sqrt{E_S}$	$0.92\sqrt{E_S}$
3	010	5π/8	$-0.38\sqrt{E_S}$	$0.92\sqrt{E_S}$
4	011	7π/8	$-0.92\sqrt{E_S}$	$0.38\sqrt{E_S}$
5	100	9π/8	$-0.92\sqrt{E_S}$	$-0.38\sqrt{E_S}$
6	101	11π/8	$-0.38\sqrt{E_S}$	$-0.92\sqrt{E_S}$
7	110	13π/8	$0.38\sqrt{E_S}$	$-0.92\sqrt{E_S}$
8	111	15π/8	$0.92\sqrt{E_S}$	$-0.38\sqrt{E_S}$

Table 4-1: Mapping of bits into symbols for 8-PSK

m	Bits	Phase	s_m	
1	00	π/4	$\sqrt{E_S}$	$\sqrt{E_S}$
2	01	3π/4	$-\sqrt{E_S}$	$\sqrt{E_S}$
3	10	5π/4	$-\sqrt{E_S}$	$-\sqrt{E_S}$
4	11	7π/4	$\sqrt{E_S}$	$-\sqrt{E_S}$

Table 4-2: Mapping of bits into symbols for 4-PSK

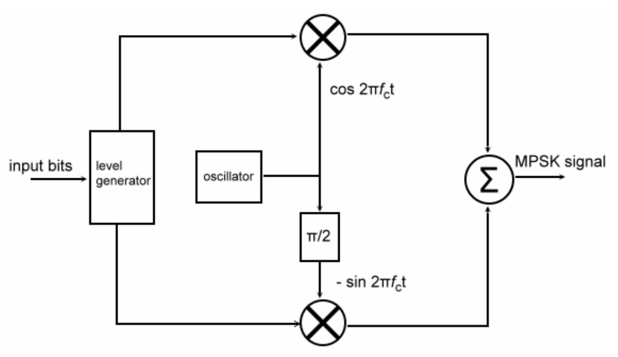


Figure 4-3: Block diagram of MPSK modulator

The MPSK modulator is presented in the above figure. The level generator unit, performs the mapping of bits to the corresponding symbols. The oscillator produces the carrier $\cos(2\pi f_c t)$. Shifting the phase of $\cos(2\pi f_c t)$ by π/2 we can obtain the carrier $-\sin(2\pi f_c t)$, since $\cos(2\pi f_c t + \pi/2) = -\sin(2\pi f_c t)$.

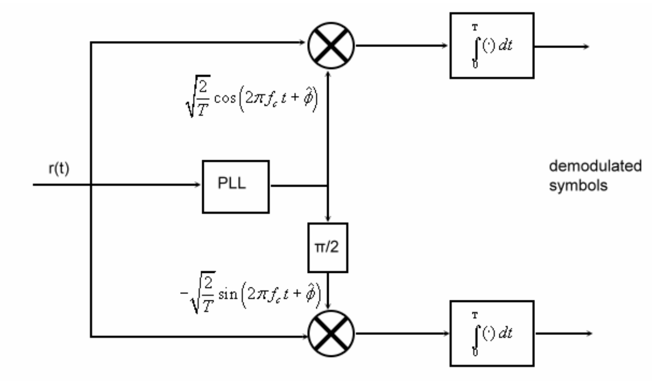
The Euclidean distance between to symbols on the constellation is given by [7, pg 399]

$$d_{mn} = \sqrt{\| \mathbf{s_m} - \mathbf{s_n} \|^2} = \sqrt{2E_s\left(1 - \cos\frac{2\pi(m-n)}{M}\right)}$$

The minimum Euclidean distance betweens two symbols on the constellation is given by

$$d_{min} = \sqrt{2E_s\left(1 - \cos\frac{2\pi}{M}\right)}$$

The corresponding MPSK correlation demodulator is presented in the following figure.



r(t)
PLL
π/2
$\sqrt{\frac{2}{T}}\cos(2\pi f_c t + \hat{\phi})$
$-\sqrt{\frac{2}{T}}\sin(2\pi f_c t + \hat{\phi})$
$\int_0^T (\cdot)\,dt$
$\int_0^T (\cdot)\,dt$
demodulated symbols

Figure 4-4: Block diagram of MPSK demodulator

Since the MPSK signal set has only two basis functions the receiver uses two correlators.

The optimum detector for MPSK signals finds the symbol $\boldsymbol{s_m}$ that minimizes the Euclidean distance $\boldsymbol{D}(\mathbf{r},\mathbf{s}_{\boldsymbol{m}}) = \left(\sum_{k=1}^{N} (r_k - s_{mk})^2 \right)^{1/2}, m = 1...M$. Equivalently the detector selects the symbol $\boldsymbol{s_m}$ that corresponds to the maximum projection of $\mathbf{r}$ on $\boldsymbol{s_m}$

$$C(\mathbf{r},\mathbf{s}_{\boldsymbol{m}}) = \mathbf{r}\,\boldsymbol{s_m}$$

Since all the symbols have the same energy, the optimum detector can be implemented in order to select the vector $\boldsymbol{s_m}$ whose phase is closest to $\theta_r = \tan^{-1}\frac{r_2}{r_1}$.

1.2 Error Probability for MPSK

Consider the transmission of digital information by use of M PSK waveforms through an AWGN channel. Each waveform of duration T *sec* is corrupted by additive white Gaussian noise, with power spectral density $\Phi_{nm} = \frac{N_0}{2} W/Hz$. Thus the received signal in the interval $0 \le t \le T$ can be expressed as $r(t) = s_m(t) + n(t), \quad 0 \le t \le T$. The function of the demodulator is to convert the received signal into a two-dimensional vector $\mathbf{r} = \begin{bmatrix} r_1 \\ r_2 \end{bmatrix}$. Given that the transmitted symbol is $\mathbf{s}_{\boldsymbol{m}} = \begin{bmatrix} s_{m1} \\ s_{m2} \end{bmatrix}$, an error occurs if $\mathbf{r}$ falls outside the decision region of $\mathbf{s}_{\boldsymbol{m}}$, Z_m. Thus [7, pg 461]

$$P_s = 1 - \int_{Z_m} p(\mathbf{r}\,|\,\mathbf{s}_{\boldsymbol{m}}) d\mathbf{r}$$

where $p(\mathbf{r}\,|\,\mathbf{s}_{\boldsymbol{m}}) = \frac{1}{\pi N_0} \exp\left\{ -\frac{1}{N_0}\left[\left(r_1 - \sqrt{E_s}\cos\theta_m\right)^2 + \left(r_2 - \sqrt{E_s}\cos\theta_m\right)^2 \right] \right\}$ is the two-dimensional joint probability density function of the received vector $\mathbf{r}$ [7, pg 461]. Eventually the symbol error probability for MPSK is given by

$$P_s = \frac{M-1}{M} - \frac{1}{2} erf\left[\sqrt{\frac{E_s}{N_0}} \sin\frac{\pi}{M} \right] - \frac{1}{\sqrt{\pi}} \int_0^{\sqrt{E_s/N_0}\sin\pi/M} e^{-y^2} erf\left(y \cot\frac{\pi}{M} \right) dy \text{ [8]}$$

However for $E_s / N_0 \gg 1$ the preceding expression for symbol error probability can be obtained by the following approximation.[7, pg 463]

[8] The error function is defined as $erf(x) \triangleq \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt$. The complementary error function erfc is given by $erfc = 1 - erf = 2Q\left(\sqrt{2}x\right)$.

$$P_s \approx erfc\left(\sqrt{\frac{E_s}{N_0}}\sin\frac{\pi}{M}\right) = 2Q\left(\sqrt{\frac{2E_s}{N_0}}\sin\frac{\pi}{M}\right)$$

The bit error rate can be related to the symbol error rate by

$$P_b \approx \frac{P_s}{\log_2 M}$$

The symbol and bit error rates for $M = 2,4,8,16,32$ *and* 64 are illustrated in the following figures.

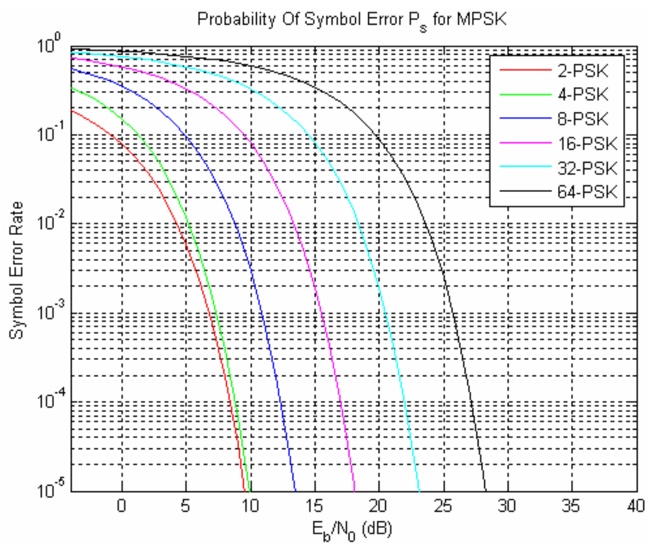


Figure 4-5: Probability of symbol error P_s for MPSK

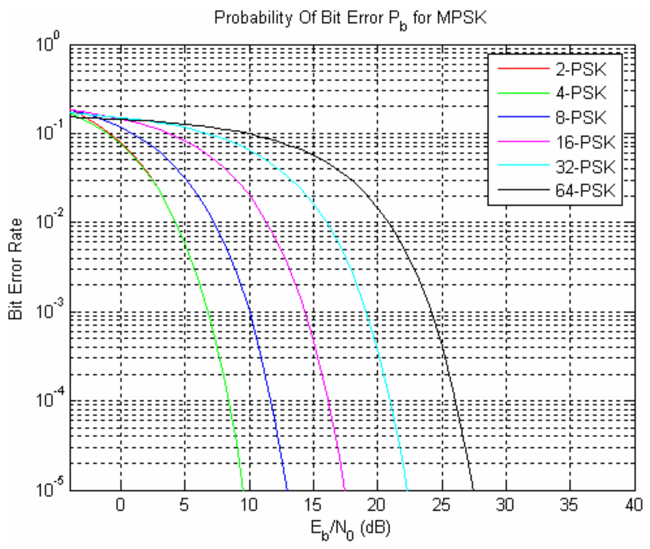


Figure 4-6: Probability of bit error P_b for MPSK

From the above figures we can obtain that beyond *M*=4, doubling the number of phases, require a substantial increase in SNR. At $P_s = 10^{-5}$, the SNR difference between *M*=4 and *M*=8 is approximately 4dB and the difference between *M*=8 and *M*=16 is approximately 5dB. In general for large values of *M*, doubling the number of phases requires an SNR increase of 6dB to maintain the same performance.

2. IMPLEMENTATION

In WFTP system we have implemented a software version of the MPSK modulator, demodulator and detector. However, modulation is performed by two units. The modulator unit which works as the level generator shown in figure 4-3 and a special unit called *pulse_shape* which is responsible to perform the multiplication with the two carriers and the construction of the signal to be transmitted. Pulse_shape is a unitary function that produces a discrete waveform, depending on the modulation technique, by using the corresponding basis waveforms. Moreover it performs demodulation by correlating the received signal with the corresponding orthogonal basis functions and produces the received symbols.

At the transmitter, the *psk_modulator* function along with the *psk_mapper* function is used to map the packet bits into symbols.

The *psk_mapper* function creates an $M \times (\log_2(M) + 4)$ table with the mappings between bits and symbols. This table is called *psk map* and each row of the table corresponds to a value of m=1…M. The first column of the table holds the decimal representation of the M possible bit strings while the next $\log_2(M)$ columns hold the M bit strings. Eventually the three last columns hold the phase, and the two-dimensional vector $\mathbf{s}_m$ that correspond to each value of m.

The mapping of packet bits into the corresponding symbols is performed by the *psk_modulator* function with the use of the *psk map* as a look up table. The symbol $\mathbf{s}_m$ that corresponds to $\log_2(M)$ bits with decimal representation *i*, is found in the i+1 position of the map. The *psk_modulator* function clusters the $N\log_2(M)$ packet bits into *N* groups of $\log_2(M)$ bits, and calculates their corresponding decimal representation. This procedure can be obtained in the following figure.

$$\underset{\textit{Packet bits}}{\begin{bmatrix} b_0^0 \\ b_0^1 \\ \vdots \\ b_0^k \\ \vdots \\ b_N^0 \\ b_N^1 \\ \vdots \\ b_N^k \end{bmatrix}} \underset{\substack{\textit{Clustering}\\ \textit{in } N \textit{ groups of}\\ \log_2(M)\textit{bits}}}{\Rightarrow} \begin{bmatrix} b_0^0 & b_0^1 & \cdots & b_0^k \\ \vdots & \vdots & & \vdots \\ b_N^0 & b_N^1 & \cdots & b_N^k \end{bmatrix} \underset{\substack{\textit{Convert in}\\ \textit{the corresponding}\\ \textit{decimal representation}}}{\Rightarrow} dec\left(\begin{bmatrix} b_0^0 & b_0^1 & \cdots & b_0^k \\ \vdots & \vdots & & \vdots \\ b_N^0 & b_N^1 & \cdots & b_N^k \end{bmatrix} \right) = \begin{bmatrix} d_0 \\ \vdots \\ d_N \end{bmatrix}$$

Figure 4-7: Creating the index vector for mapping bits to symbols

The decimal representation is used as index in the *psk map* in order to obtain the symbols that correspond to the initial packet bits.

In the next step *pulse_shape* accepts as input the symbols created by *psk_modulator* and generates the discrete-time waveform with a specified symbol period. Symbol period is defined as the number of samples of the basis discrete-time waveforms, and is denoted as T_s. Let $\mathbf{s}_{m1}$ and $\mathbf{s}_{m2}$ be the *N*-dimensional column vectors containing the symbols generated by the modulator. The basis functions in vector form can be expressed as

$$\mathbf{y}_1 = \sqrt{\frac{2}{T_s}}\cos\left(2\pi\frac{n}{T_s}\right), \quad 0 \le n \le T_s$$

$$\mathbf{y}_2 = -\sqrt{\frac{2}{T}}\sin\left(2\pi\frac{n}{T_s}\right), \quad 0 \le n \le T_s$$

where $\mathbf{y}_1$ and $\mathbf{y}_2$ are T_s x 1 vectors.

Therefore the modulation process in matrix notation can be described by

$$\mathbf{U} = \mathbf{s}_{m1} \cdot \mathbf{y}_1' + \mathbf{s}_{m2} \cdot \mathbf{y}_2'$$

where **U** is a $N \times T_s$ matrix containing the *N* discrete-time waveforms of length T_s for each symbol $\mathbf{s}_m$. Eventually the matrix **U** is transformed into a $NT_s \times 1$ vector **u**. This vector contains the samples of the *N* discrete-time waveforms of length T_s following the order of symbols in vectors $\mathbf{s}_{m1}$ and $\mathbf{s}_{m2}$. Therefore the samples of the waveform that corresponds to the symbol $\mathbf{s}_m$ are $\left[\mathbf{u}\left((m\text{-}1)T_s + 1\right) \ldots \mathbf{u}\left(mT_s\right)\right]^T$, where $1 \le m \le M$.

Modulation process in the software implementation that is used in WFTP system is based on the preceding analysis.

At the receiver, *pulse_shape* is used to demodulate the received signal. Let **v** be a $NT_s \times 1$ vector containing the samples of the *N* waveforms of the received signal. This vector is transformed into a $N \times T_s$ matrix **V**, which contains the T_s samples of the N discrete-time waveforms. Consider again the vector form $\mathbf{y}_1$ and $\mathbf{y}_2$ of the two basis functions. In order to obtain the received symbols from the matrix **V**, we define a $T_s \times 2$ matrix **Y** where the first and the second columns of **Y** correspond to $\mathbf{y}_1$ and $\mathbf{y}_2$ respectively. Therefore the demodulation process in matrix form is defined as

$$\mathbf{R} = \mathbf{V} \cdot \mathbf{Y}$$

where **R** is a $N \times 2$ matrix containing the symbols generated by the correlators. Each row of matrix R corresponds to a transmitted symbol $\mathbf{s}_m$.

Afterwards *psk_detector* along with the *psk map* created by *psk_mapper* performs the detection of the received bits. Consider the $N \times 2$ matrix R generated by the *pulse_shape* function.

$$\mathbf{R} = \begin{bmatrix} r_{11} & r_{12} \\ r_{21} & r_{22} \\ \vdots & \vdots \\ r_{N1} & r_{N2} \end{bmatrix}$$

In order to obtain the detected bits, it is essential to compute the Euclidean distance of every received symbol $(r_{i1} \quad r_{i2})$, $i = 1...N$ from the *M* possible transmitted symbols $\mathbf{s}_m$. We define the $N \times 2M$ matrix $\mathbf{R}'$, which contains *M* replicas of the matrix **R**.

$$\mathbf{R}' = \begin{bmatrix} r_{11} & r_{12} & r_{11} & r_{12} & \cdots & r_{11} & r_{12} \\ r_{21} & r_{22} & r_{21} & r_{22} & \cdots & r_{21} & r_{22} \\ \vdots & \vdots & \vdots & \vdots & & \vdots & \vdots \\ r_{N1} & r_{N2} & r_{N1} & r_{N2} & \cdots & r_{N1} & r_{N2} \end{bmatrix}$$

Equivalently we define the $N \times 2M$ matrix **S**, which contains *N* replicas of the *M* possible symbols.

$$\mathbf{S} = \begin{bmatrix} s_{11} & s_{12} & s_{21} & s_{22} & \cdots & s_{M1} & s_{M2} \\ s_{11} & s_{12} & s_{21} & s_{22} & & s_{M1} & s_{M2} \\ \vdots & \vdots & \vdots & \vdots & & \vdots & \vdots \\ s_{11} & s_{12} & s_{21} & s_{22} & \cdots & s_{M1} & s_{M2} \end{bmatrix}$$

In order to compute the Euclidean distance we define the $N \times 2M$ matrix **C** so that

$$\mathbf{C} = \left(\mathbf{R}^{'} - \mathbf{S}\right)^2 = \begin{bmatrix} (r_{11}-s_{11})^2 & (r_{12}-s_{12})^2 & (r_{11}-s_{21})^2 & (r_{12}-s_{22})^2 & \cdots & (r_{11}-s_{M1})^2 & (r_{12}-s_{M2})^2 \\ (r_{21}-s_{11})^2 & (r_{22}-s_{12})^2 & (r_{21}-s_{21})^2 & (r_{22}-s_{22})^2 & & (r_{21}-s_{M1})^2 & (r_{22}-s_{M2})^2 \\ \vdots & \vdots & \vdots & \vdots & & \vdots & \vdots \\ (r_{N1}-s_{11})^2 & (r_{N2}-s_{12})^2 & (r_{N1}-s_{21})^2 & (r_{N2}-s_{22})^2 & \cdots & (r_{N1}-s_{M1})^2 & (r_{N1}-s_{M2})^2 \end{bmatrix}$$

We split matrix **C** into two matrices **C**$_1$, **C**$_2$ of size $N \times M$, containing the elements of the odd and even columns of **C** respectively.

$$\mathbf{C}_1 = \begin{bmatrix} (r_{11}-s_{11})^2 & (r_{11}-s_{21})^2 & \cdots & (r_{11}-s_{M1})^2 \\ (r_{21}-s_{11})^2 & (r_{21}-s_{21})^2 & \cdots & (r_{21}-s_{M1})^2 \\ \vdots & \vdots & & \vdots \\ (r_{N1}-s_{11})^2 & (r_{N1}-s_{21})^2 & \cdots & (r_{N1}-s_{M1})^2 \end{bmatrix} \mathbf{C}_2 = \begin{bmatrix} (r_{12}-s_{12})^2 & (r_{12}-s_{22})^2 & \cdots & (r_{12}-s_{M2})^2 \\ (r_{22}-s_{12})^2 & (r_{22}-s_{22})^2 & \cdots & (r_{22}-s_{M2})^2 \\ \vdots & \vdots & & \vdots \\ (r_{N2}-s_{12})^2 & (r_{N2}-s_{22})^2 & \cdots & (r_{N1}-s_{M2})^2 \end{bmatrix}$$

Therefore the Euclidean distance of the received symbols from all the *M* possible symbols generated by MPSK is given by

$$\mathbf{D} = \left(\mathbf{C}_1 + \mathbf{C}_2\right)^{1/2}$$

The received symbol $\mathbf{r}_j$, where $j \in [1, N]$, is mapped to the symbol $\mathbf{s}_{\boldsymbol{m}}$ that minimizes the Euclidean distance $d_{jm} = \sqrt{\| \mathbf{r}_{\boldsymbol{j}} - \mathbf{s}_{\boldsymbol{m}} \|^2}$, where $m \in [1, M]$.

The detected bits can be obtained by using the *psk map* as a look up table indexed with the generated symbols $\mathbf{s}_{\boldsymbol{m}}$.

C h a p t e r 5

EVALUATION OF THE WFTP COMMUNICATION SYSTEM

1. INTRODUCTION

In Chapter 1, we have mentioned the three foremost objectives of the WFTP Communication system.

⇒ Reliable file transfer.

⇒ Low bit error rate on the order of 10^{-6}

⇒ Achievement of the highest possible transfer rates.

In general, based on the results that were gathered from the trial transmissions of the WFTP system, we may conclude that our primary objectives were accomplished. The reliability of the system lies on the use of an ARQ mechanism ("Stop and wait") as described in Chapter 1. The drawback from the use of an error detection mechanism is the additional delays that are introduced in the transfer time. Moreover low bit error rates achieved using different modulation schemes whereas the zero bit error rates in many transmissions were the irrefutable evidence of the success of our efforts.
Unfortunately the hardware and software constraints that emerged during the design and development of the WFTP system, limits our perspective for achieving high transfer rates.
In Chapter 1 we have underlined that several factors in the software and hardware implementation, lower the performance of the WFTP system. The inability of developing a distributed software application in Matlab, leads us in a simple and less efficient design. Furthermore the unpredictable delays of the operating system resulting from the memory and hard disk management, yield in an unnecessary additional recording duration in the Receiver Unit which increases the total transfer time.
The ARQ mechanism is implemented using the UDP protocol which is supported in Matlab. However the UDP does not ensure that the sent message will reach its destination and hence the acknowledgements may be lost. Such cases have been predicted in the design of the system, but introduce additional delay, as the Transmitter Unit waits a number of timeouts to occur before deciding to resend the packet or end the transmission.

Considering the hardware, the maximum sampling frequency that is supported from the PC audio device, is a very important factor concerning the maximum transfer and transmission rate that we can achieve. The nominative sampling rate referred in Matlab, is 44100 Hz. Usually soundcards support this nominal rate. Our experiments testified that we can attain zero bit errors in some modulation schemes using as an upper bound the 88200 Hz sampling rate.

This chapter provides a thorough analysis of the performance of the WFTP system based on the results of the trial transmissions.

1.1 Evaluation Metrics

1.1.1 Bit error rate

The reliability and performance of the WFTP system is measured by the bit error rate in packet and the average bit error rate.
The packet bit error rate in the open and closed loop WFTP system is defined as the number of corrupted bits in the packet divided by the number of the total bits in the packet.
Therefore the average bit error rate for a transmitted file in an open loop system is given by

$$ber_{average} = \frac{\sum_{i=1}^{N} ber_{packet\ i}}{N}$$

where $N = \#packets$.
In the closed loop WFTP system the average bit error rate is defined as

$$ber_{average} = \frac{\sum_{i=1}^{Np} ber_{packet\ i}}{Np}$$

where $Np = \#packets + \#retransmitted\ \ packets$.

1.1.2 Transmission and Transfer rates

Before introducing the measures of transmission and transfer rates we define the total transfer time $t_{TRANSFER}$ of a file, as the total time duration between the first handshake and the last handoff among the transmitter and the receiver. The total transfer time includes the processing time from the Receiver Unit, the total audio recording time in the receiver and the delays introduced by the handshake and acknowledgement signals.

Consider a file of size F bits, which is fragmented in N packets of length L bits. The rate $\frac{N \times L}{t_{TRANSFER}}$ is called the transfer rate $R_{TRANSFER}$ and is measured in $bits/\sec$. In the presentation of the experimental results we will refer to average transfer rates. In cases where we used encoded schemes, we will represent the results for each encoder (Block or Convolutional) separately. The transmission rate has been computed in Chapter 2 and is given by the expression

$$R_{TRANSMISSION} = \frac{Fs \log_2(M)}{T_s} \ (bps)$$

where F_s is the sampling frequency in *samples/sec*, M is the size of the modulation method and T_s is the number of samples per symbol.
The above expressions stand for the open and closed loop form of the WFTP system.

1.2 Evaluation Process

The evaluation of the performance of the WFTP system is based on the results of the experimentation on the system. The possible settings of the system are listed in the two subsequent tables.

System Settings
Filesize
Number of bits per packet
Number of training bits
Sampling frequency (Hz)

Table 5-1: General System Settings

Field	Modules	Settings			
Error Control Coding	Convolutional Encoder Viterbi Decoder	On / Off	Code rate	Operation Mode	
	Reed Solomon Encoder / Decoder	On / Off	Message length	Codeword length	
Interleaving	Random Interleaver	On / Off	Word length		
	Block Interleaver	On / Off	Word length		
Error Detection	CRC	On / Off			
Equalization	LMS Equalizer	On / Off	Number of weights	Stepsize	
	RLS Equalizer	On / Off	Number of weights	Initialization	Forgetting factor
	CMA Equalizer	On / Off	Number of weights	Stepsize	
	Viterbi Equalizer	On / Off	Preamble	Postamble	Depth
Digital Transmission	MPSK	On / Off	Size	Samples per Symbol	
	QAM	On / Off	Size	Samples per Symbol	
	PPM	On / Off	Size	Samples per Symbol	
Phase Recovery	Phase Recovery	On / Off			

Table 5-2: Settings for the modules of the WFTP system

Because of the huge number of the possible combinations, our main effort was to avoid performing meaningless transmissions. Therefore we considered performing the transmissions for a specified packet and file size that will provide us with some representative results concerning the system's performance.

In order to select the size of the packet, we transmitted three files of 6KB, 18KB, and 108KB respectively, with packets of size 15000, 50000 and 100000 bits using the following system settings:

Number of training bits	**Modulation**	T_s **(samples)**
500	4-PSK	10

The experiments were performed on the open loop system with sampling rate at 44100 Hz. The system consisted of the following modules.

- ⇒ 4-PSK modulator
- ⇒ 4-PSK demodulator
- ⇒ 4-PSK detector
- ⇒ Synchronizer
- ⇒ Phase recovery

The distance between the radio transmitter and the radio receiver was about 2m.

The question that springs to mind immediately is why we didn't use packets of size greater than 100000 bits. All the modulation methods have been tested with packets of size greater than 100000 bits, but 4-PSK was the only one that resulted in not detectable bit error rates. However transmissions with 100000 bits per packet resulted in significantly low bit error rates with the most of the modulation methods. In general for the choice of the packet size, we wanted to test packet sizes that could perform well in the majority of the modulation methods. The total transfer time for each transmission is listed in the following table.

	File size (KB)		
Packet size (*bits*)	6	18	108
15000	14 (sec)	41.09 (sec)	237.471 (sec)
50000	12 (sec)	30.7 (sec)	171.587 (sec)
100000	19.6 (sec)	38.51 (sec)	162.203 (sec)

Table 5-3: Transfer times for different size of packets and different file sizes

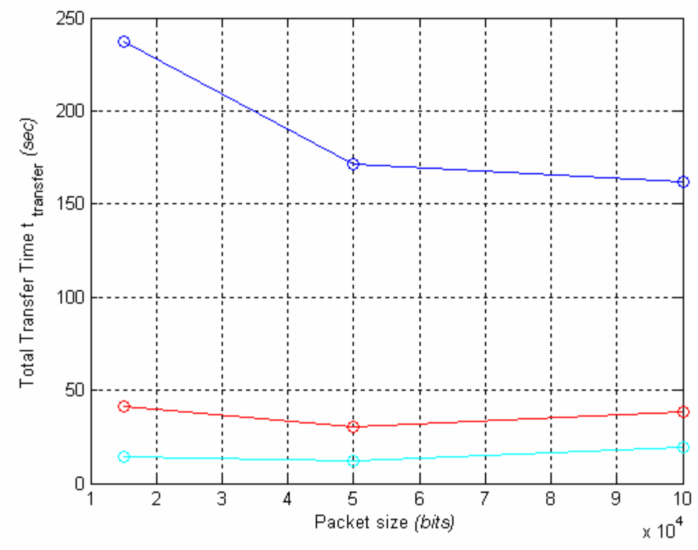


Figure 5-1: The most efficient packet size for WFTP system is 100000 bits

From the above results we can easily obtain that the best transfer time for the 6 KB and 18 KB files is achieved by using 50000 bits per packet. However for a 108 KB file the packet size that achieves the best transfer time is 100000 bits per packet.

In general in the WFTP system it is desired to fragment the transmitted file in a small number of packets. Several unpredictable delays have been obtained during its experimental operation and result from the operating system. Therefore in order to ensure the correct reception of the packets we added a constant 0.5 sec additional recording time. This overhead is independent of the size of the packets. Thus increasing the number of the packets, results in a proportional increase of the

additional recording time. Consequently in order to maintain high transfer rates in the open loop system, we selected the maximum possible packet size of 100000 bits.
However increasing the number of bits per packet, results in an increase of the probability of bit error in every packet. Therefore the use of this packet size in high order modulation schemes may result in high bit error rates. Consequently in the closed loop form of the system this results in a large number of retransmissions and hence in a significant decrease of the transfer rate. In such cases we must enhance a powerful encoder in the system in order to reduce the number of retransmissions and maintain high transfer rates.

In the subsequent sections we will represent the results of our experimentation on the WFTP system and evaluate the overall performance for different settings. Moreover we will focus particularly on the performance of the Convolutional Codes and the PSK modulation scheme as they constitute the main part of this thesis.

In the trial transmissions we used the nominal sampling rate of 44100 Hz that is supported from the majority of the soundcards. However because increasing the sampling rate results in an increase of the transmission and transfer rate, we experimented with the 88200 Hz sampling rate. Following the same perspective we evaluated the minimum value of the symbol period in samples that would result in the maximum possible transfer rate along with low bit error rate in the majority of the modulation schemes. Consequently the experiments were performed with the following system settings:

- ⇒ Symbol period $T_s = 10\,samples$
- ⇒ Sampling frequency $F_s = 44100, 88200\,Hz$
- ⇒ Size of packet 100.000 *bits*
- ⇒ Size of file 108 KB

The distance between the radio transmitter and the radio receiver was about 2m.
Every modulation scheme was tested on the open and closed loop system. In cases where the transmissions were not successful, we enhanced different encoders in the system and obtained the overall performance. The results will be provided in tables according to the modulation scheme used in the open and closed loop system.
Considering an encoded system (open or closed loop) the metrics average BER, $\mathbf{R_{TRANSMISSION}}$, and $\mathbf{R_{TRANSFER}}$ correspond to the average performance of the most efficient encoder resulted from the

experimentation on the specific system. In the closed loop system (uncoded or encoded) the average BER and $\mathbf{R}_{\mathbf{TRANSFER}}$ do not include the time spent in packet retransmissions.

1.3 MPSK Modulation

In this section we provide the results and the conclusions drawn from the experimentation on the PSK modulation scheme. The experiments were performed in both the open and the closed loop form of the WFTP system. Every M-PSK modulation scheme is primarily evaluated with the basic system settings consisting of the Synchronizer, the PSK modulator, the PSK demodulator, the PSK detector and the Phase recovery. This basic system is tested with the sampling rates of 44100, 88200 Hz and symbol period of 10 samples. Thereby we achieve the least processing time and hence the highest possible transfer rate. In the next step we added the LMS and RLS modules in the system and obtained the overall performance. Because of the fact that the two equalizers did not presented a substantial difference in the processing time we used the LMS equalizer along with the most of our experiments. In cases where errors occurred we used a Block or Convolutional encoder to ensure the correct reception of the packets. Despite the average adequate performance of the encoders, they could not yield in sufficiently low bit error rates in every M-PSK modulation scheme.

In the trial transmissions, our primary concern was to achieve the lowest possible bit error rate. Therefore the experiments for the M-PSK modulation scheme followed an increasing order on the basis of the modulation size, *M*.

1.3.1 4-PSK

The 4-PSK modulation scheme using the basic system on both sampling rates of 44100 and 88200 Hz, performed with an average bit error rate below the 10^{-6} threshold. In the closed loop form of the system additional delays were introduced due to the processing of the packet bits. The above transmissions were repeated with the additional modules of RLS and LMS equalizers resulting in a constant zero average bit error rate. However the average transfer rate decreased due to the extra processing time in the equalizers.

Additional Modules	**Fs (*Hz*)**	**Packet size (*bits*)**	**#Packets**	**Average BER**	**$R_{TRANSMISSION}$ (*bps*)**	**$R_{TRANSFER}$ (*bps*)**	**Transfer Rate Loss (%)**
LMS Equalizer Off	44.100	10^5	9	0	8820	5518	-
	88.200	10^5	9	0	17640	8281	-
LMS Equalizer On	44.100	10^5	9	0	8820	5480	0.69
	88.200	10^5	9	0	17640	7808	5.7

Table 5-4: 4-PSK open loop results

Additional Modules	**Fs (*Hz*)**	**Packet size (*bits*)**	**#Packets**	**Average BER**	**$R_{TRANSMISSION}$ (*bps*)**	**$R_{TRANSFER}$ (*bps*)**
LMS Equalizer Off CRC On	44.100	10^5	9	0	8820	4888
	88.200	10^5	9	0	17640	6921
LMS Equalizer On CRC On	44.100	10^5	9	0	8820	4851
	88.200	10^5	9	0	17640	6587

Table 5-5: 4-PSK closed loop results

1.3.2 8-PSK

In order to increase the transfer rate we performed the same transmissions on 8-PSK. As expected the total transfer time decreased and hence the average transfer rate improved significantly. The transmissions occurred with no errors in the closed and open loop form of the system.

Additional Modules	**Fs (*Hz*)**	**Packet size (*bits*)**	**#Packets**	**Average BER**	**$R_{TRANSMISSION}$ (*bps*)**	**$R_{TRANSFER}$ (*bps*)**	**Transfer Rate Loss (%)**
LMS Equalizer Off	44.100	10^5	9	0	13230	8635	-
	88.200	10^5	9	0	26460	12525	-
LMS Equalizer On	44.100	10^5	9	0	13230	8378	3
	88.200	10^5	9	0	26460	12403	1

Table 5-6: 8-PSK open loop results

Additional Modules	**Fs (*Hz*)**	**Packet size (*bits*)**	**#Packets**	**Average BER**	**$R_{TRANSMISSION}$ (*bps*)**	**$R_{TRANSFER}$ (*bps*)**
LMS Equalizer Off CRC On	44.100	10^5	9	0	13230	7168
	88.200	10^5	9	0	26460	9656
LMS Equalizer On CRC On	44.100	10^5	9	0	13230	6977
	88.200	10^5	9	0	26460	9574

Table 5-7: 8-PSK closed loop results

From the above results we may conclude that for 4 and 8 PSK modulation schemes the WFTP system operates with no errors. The rate loss introduced by the equalizers is negligible and hence their use is proposed. Since the system operates with no errors there is no need to use an error correction scheme as there will be an overhead in the processing time and consequently a reduction of the transfer rate. The reduced transfer rates in the closed loop form of the system for 4 and 8 PSK resulted from the extra processing time introduced in the receiver.

1.3.3 16-PSK

As increasing the order of the PSK modulation scheme we expect that the packets will be received with errors. Before applying any error correction scheme on the system we performed the appropriate experiments on the basic system. In spite of the increase of the average transfer rate, the transmissions were not successful and errors occurred and without the use of equalizers. Therefore we tested different Reed Solomon and Convolutional encoders and evaluated their error correcting capabilities over a large number of transmissions. In general both codes managed to correct the bit errors occurred in the open loop system operating at sampling rate of 44100 Hz. The Reed Solomon encoder at a sampling frequency of 44100 results in a lower decrease of the transfer rate compared with the Convolutional encoder. Nevertheless at sampling rate of 88200 Hz the transfer rate loss introduced by the Convolutional encoder is much lower than that of the Reed Solomon encoder.

Additional Modules	**Fs (*Hz*)**	**Packet size (*bits*)**	**#Packets**	**Average BER**	**$R_{TRANSMISSION}$ (*bps*)**	**$R_{TRANSFER}$ (*bps*)**	**Transfer Rate Loss (%)**
LMS Equalizer Off	44100	10^5	9	$1.24 \cdot 10^{-4}$	17640	11370	-
	88200	10^5	9	$5.6 \cdot 10^{-2}$	35280	16636	-
LMS Equalizer On	44100	10^5	9	$1.03 \cdot 10^{-4}$	17640	11296	0.65
	88200	10^5	9	$2 \cdot 10^{-2}$	35280	15914	4.4
$(255, 231)$ Reed Solomon encoder On LMS Equalizer On	44100	10^5	9	0	15786	9940	12.6
	88200	10^5	9	$1.1 \cdot 10^{-2}$	26460	7006	58
$(3,2,6)$ Convolutional encoder On LMS Equalizer On	44100	10^5	9	0	11760	7033	38
$(2,1,5)$ Convolutional encoder On LMS Equalizer On	88200	10^5	9	$6 \cdot 10^{-3}$	17640	8881	44

Table 5-8: 16-PSK open loop results

Additional Modules	**Fs (*Hz*)**	**Packet size (*bits*)**	**#Packets**	**Average BER**	**$R_{TRANSMISSION}$ (*bps*)**	**$R_{TRANSFER}$ (*bps*)**
$(255, 231)$ Reed Solomon encoder On LMS Equalizer On CRC On	44100	10^5	9	0	15876	7154
$(4,3,6)$ Convolutional encoder On LMS Equalizer On CRC On	44100	10^5	9	0	13230	6912

Table 5-9: 16-PSK closed loop results

1.3.4 Evaluation of Convolutional encoders for 16-PSK

In Chapter 2 we selected a variety of convolutional codes of different code rates to apply on the WFTP system.

R	***v***	***m***	d_{free}	γ(dB)	***Branch Complexity***
1/4	6	6	20	3.98	2^7
1/4	4	4	16	3.01	2^5
1/3	6	6	15	3.98	2^7
1/3	5	5	13	3.36	2^6
1/2	6	6	10	3.98	2^7
1/2	5	5	8	3.01	2^6
2/3	6	3	7	3.67	2^8
3/4	6	2	6	3.52	2^9
2/3	5	3	6	3.01	2^7
3/4	5	2	5	2.73	2^8
2/3	4	2	5	2.22	2^6
3/4	4	2	4	1.76	2^7

Table 5-10: The optimum convolutional codes that we selected for the WFTP system

Considering the open loop system with 16-PSK and operating sampling rate at 44100 Hz, we would like to select a convolutional encoder that would correct the transmission errors and would not deteriorate a lot the transfer rate. Therefore we tested the optimum $\frac{1}{2}$ encoder $(2,1,6)$, which resulted in zero average bit error rate. However the average transfer rate reduced at 5580 bps. This reduction is due to the additional recording and processing time introduced by the encoder. The $(2,1,6)$ encoder doubles the input bits and hence doubles the samples of the modulated signal. Therefore for sampling frequency of 44100 Hz there is an additional recording time in the receiver. Consequently the processing time in the synchronizer, is increased. The processing time in the PSK modulator, demodulator and detector due to their design is not incremented significantly. However there is an average 1.22 sec additional time per packet in the decoder. In order to reduce the transfer rate we selected an encoder of higher code rate. We expect that the $\frac{2}{3}$ convolutional code with the maximum free distance will achieve the best performance. If this encoder cannot result in zero bit error rates we must test the rest of the $\frac{1}{2}$ encoders in order to attain higher transfer rates. However the $(3,2,6)$ encoder performed with zero average bit error rate and improved the transfer rate at 7033 bps. Trying to attain the maximum transfer rate we used the $(4,3,6)$ encoder which also resulted in zero average bit error rate with a transfer rate of 7625 bps. Notice that the decoding time of the $(4,3,6)$ encoder is similar to the decoding time of the $(3,2,6)$ encoder. The $(4,3,6)$ and $(3,2,6)$ encoders increase the samples of the modulated signal by a factor of 1.3 and 1.5 respectively. Therefore the overall processing time in the receiver using the $(3,2,6)$ encoder is greater than that of the $(4,3,6)$.

In order to attain the performance of the rest $\frac{3}{4}$ encoders we experimented with the $(4,3,5)$ encoder. The trial transmissions occurred with errors and hence because of the fact that the $(4,3,5)$ encoder is expected to reach better performance than the $(4,3,4)$ we stopped experimenting.

Up to this point we may conclude that the $(4,3,6)$ and $(3,2,6)$ encoders can be used to the system over a sampling rate of 44100 Hz. Using $\frac{1}{4}$ and $\frac{1}{3}$ encoders is not essential, since zero average bit error rate can be achieved with encoders that result in a less reduction of the transfer rate.

The same procedure is applied on the open loop system with 16-PSK and sampling rate of 88200 Hz. However we tested more convolutional encoders since we could not achieve average bit error rate below the threshold accuracy of our experiments which was $\frac{1}{9}\cdot 10^{-5}$. The results are listed in the following table.

Convolutional Encoders	**Average BER**	$\mathbf{R_{TRANSFER}}$ **(*bps*)**	$\mathbf{R_{TRANSMISSION}}$ **(*bps*)**
$(3,1,6)$	$13\cdot 10^{-4}$	4581	11760
$(3,1,5)$	$3\cdot 10^{-3}$	5592	11760
$(2,1,6)$	$3.4\cdot 10^{-3}$	8049	17640
$(2,1,5)$	$6\cdot 10^{-3}$	8881	17640
$(3,2,6)$	$8.9\cdot 10^{-3}$	10844	23520
$(4,3,6)$	$9.3\cdot 10^{-3}$	11655	26460
$(3,2,5)$	$12\cdot 10^{-3}$	11429	23520
$(4,3,5)$	$23\cdot 10^{-3}$	12396	26460
$(3,2,4)$	$38\cdot 10^{-3}$	11643	23520
$(4,3,4)$	$41\cdot 10^{-3}$	12892	26460

Table 5-11: Performance of the Convolutional Codes at 16-PSK (88200 Hz)

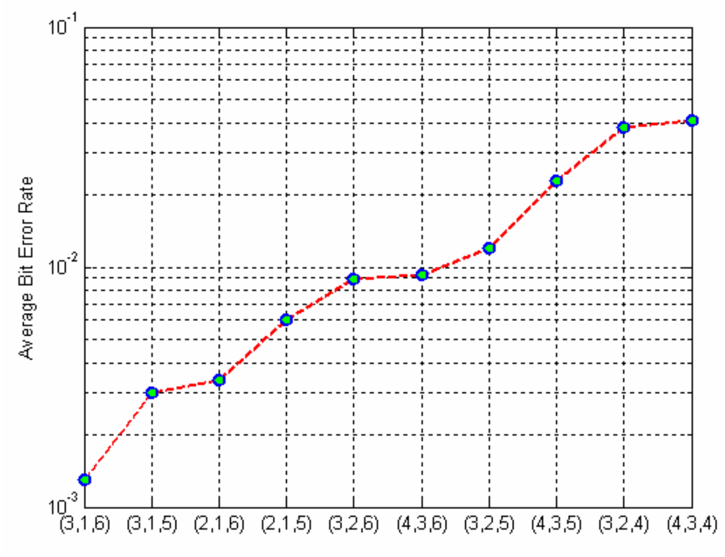


Figure 5-2: Average bit error rates for the tested convolutional codes

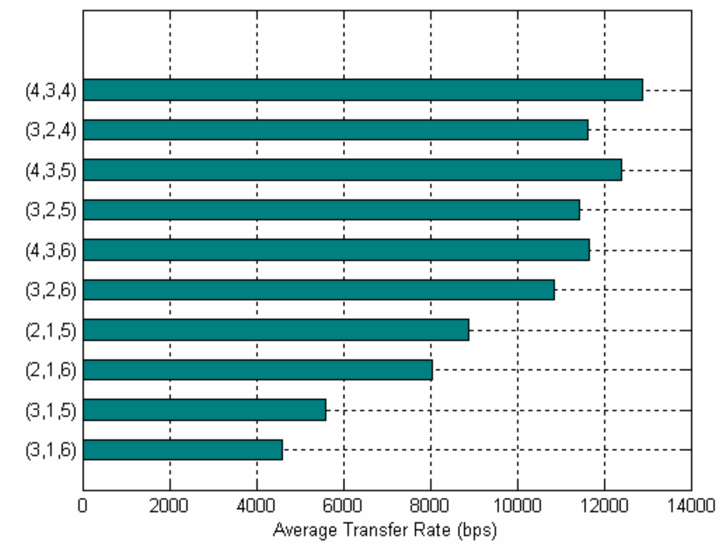


Figure 5-3: Average transfer rates for the tested convolutional codes

From the above results we obtain that the performance of the convolutional encoders verify the theoretical conclusions mentioned in chapter 2. The encoder with the maximum free distance $(3,1,6)$ achieves the lowest bit error rates. However it did not nullify the average BER of $2\cdot 10^{-2}$. Thus we should experiment with the $\frac{1}{4}$ convolutional encoders. However since the $(3,1,6)$ encoder reduced the average transfer rate at 4581 bps without correcting all the transmission errors there is no need to use an encoder which will result in lower transfer rates. A better performance can be achieved using a 4-PSK modulated scheme without encoders. To conclude with, the best transfer rate in 16-PSK is 9940 bps which has been obtained using the modules Synchronizer, PSK modulator, PSK demodulator, PSK detector, Phase Recovery and Reed Solomon encoder.

In the closed loop system with 16-PSK we tested the $(4,3,6)$ encoder for sampling rate of 44100 Hz and achieved zero average bit error rate. Since the average bit error rate for sampling rate of 88200 Hz with the use of encoders in the open loop system is on the order of 10^{-2} the use of an error detection scheme does not improve the performance.

1.3.5 32-PSK

Since in the 16-PSK the operation of the system at the sampling rate of 88200 Hz occurred with errors which could not be corrected, the 32-PSK will be tested only for the sampling frequency of 44100 Hz. Considering again the basic system as the basis of our experiments we represent the results for 32-PSK in the following table.

Additional Modules	**Fs (*Hz*)**	**Packet size (*bits*)**	**#Packets**	**Average BER**	**$R_{TRANSMISSION}$ (*bps*)**	**$R_{TRANSFER}$ (*bps*)**	**Transfer Rate Loss (%)**
LMS Equalizer Off	44100	10^5	9	$15\cdot10^{-2}$	22050	13722	-
LMS Equalizer On	44100	10^5	9	$12.8\cdot10^{-2}$	22050	13447	2%
$(255,201)$ Reed Solomon encoder On LMS Equalizer On	44100	10^5	9	$11.1\cdot10^{-2}$	17199	3467	75%
$(2,1,5)$ Convolutional encoder On LMS Equalizer On	44100	10^5	9	$10.8\cdot10^{-2}$	11025	4973	64%

Table 5-12: 32-PSK open loop results

The average bit error rate of $12.8\cdot10^{-2}$ using the basic system along with the equalizers, justify the use of Block and Convolutional encoders. In general despite the use of the encoders we did not achieve zero bit error rates. The least average bit error rate attained with the Convolutional encoder $(2,1,5)$. However the transfer rate reduced at 4973 bps. This performance is similar to the average bit error rate achieved by the $(3,1,6)$ convolutional encoder in 16-PSK and is less than the average transfer rate of the uncoded 4-PSK. Therefore the encoders with lower code rate were not tested.

1.3.6 MPSK conclusions

In the preceding presentation of the experimental results for the MPSK modulation scheme, we set as the initial objective the achievement of zero average bit error rate over a basic system consisting of the Synchronizer, the PSK modulator, the PSK demodulator, the PSK detector and the Phase Recovery. This goal was accomplished with the use of 4 and 8 PSK without the need of encoders. The cost of the use of equalizers in the average transfer rate is negligible and hence they will be embodied in the basic system.

Experimenting on the open and close loop form of the system with 16 and 32 PSK shown that it is essential to enhance an encoder in the system. Due to the resulting low transfer rate, it is not feasible to use encoders of low code rate.

In general the highest transfer rate along with the average zero bit error rate in MPSK is performed by the uncoded 8-PSK without the use of equalizers.

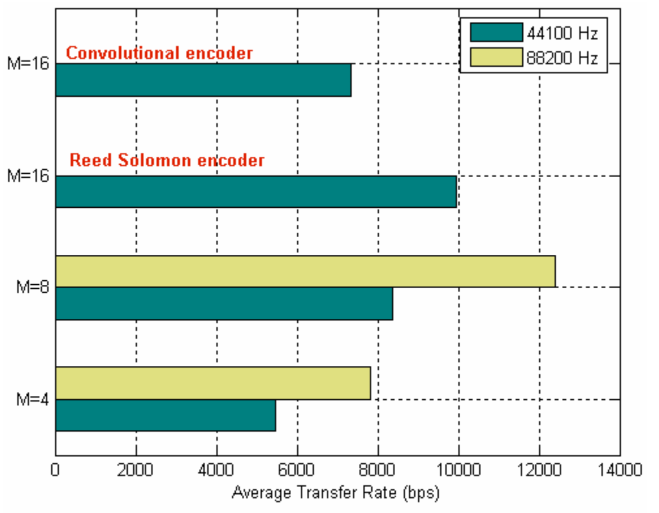


Figure 5-4: Average transfer rates of MPSK schemes that performed with zero average bit error rate in open loop.

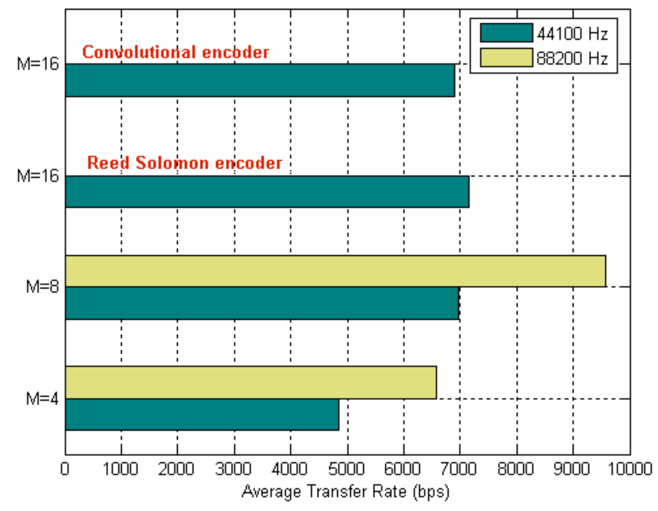


Figure 5-5: Average transfer rates of MPSK schemes that performed with zero average bit error rate in closed loop.

1.4 MQAM Modulation

The experimentation on the Quadrature Amplitude Modulation scheme was realized in a similar way as MPSK. The basic system consists of the Synchronizer, the MQAM modulator, the MQAM demodulator, the MQAM detector, the Phase recovery and the RLS, LMS equalizing modules. The trial transmissions were performed for sampling frequencies of 44100 and 88200 Hz whereas the symbol period was preserved at 10 samples per symbol.

1.4.1 4-QAM

The operation of the 4-QAM modulation scheme on the basic system, on both sampling rates of 44100 and 88200 Hz, resulted in zero average bit error rate. In the closed loop form of the system additional delays were introduced due to the processing of the packet bits.

Additional Modules	**Fs (*Hz*)**	**Packet size (*bits*)**	**#Packets**	**Average BER**	**$R_{TRANSMISSION}$ (*bps*)**	**$R_{TRANSFER}$ (*bps*)**
LMS Equalizer On	44100	10^5	9	0	8820	5483
	88200	10^5	9	0	13230	7343

Table 5-13: 4-QAM open loop results

Additional Modules	**Fs (*Hz*)**	**Packet size (*bits*)**	**#Packets**	**Average BER**	**$R_{TRANSMISSION}$ (*bps*)**	**$R_{TRANSFER}$ (*bps*)**
LMS Equalizer On CRC On	44100	10^5	9	0	8820	4853
	88200	10^5	9	0	13230	6250

Table 5-14: 4-QAM closed loop results

Notice that the average transfer rates for the 4-QAM and the 4-PSK modulation schemes are similar. Moreover the average transfer rate using CRC is reduced due to the additional processing time for error detection in the received packet.

1.4.2 8-QAM

In the 8-QAM modulation scheme the system performed with no errors operating at sampling rate of 44100 Hz. However the enhancement of a Block or Convolutional encoder became necessary at the sampling rate of 88200 Hz. In general the tested Reed Solomon encoders achieved a lower average bit error rate compared to the Convolutional encoders but reduced the average transfer rate of the basic system with the uncoded 8-QAM by 57%.

Additional Modules	Fs (*Hz*)	Packet size (*bits*)	#Packets	Average BER	$R_{TRANSMISSION}$ (*bps*)	$R_{TRANSFER}$ (*bps*)
LMS Equalizer On	44100	10^5	9	0	13230	8569
	88200	10^5	9	$3.6 \cdot 10^{-3}$	26460	12308
$(127,111)$ Reed Solomon encoder On LMS Equalizer On	88200	10^5	9	$1.4 \cdot 10^{-3}$	23126	4557
$(2,1,6)$ Convolutional encoder On LMS Equalizer On	88200	10^5	9	$37 \cdot 10^{-5}$	13230	6277

Table 5-15: 8-QAM open loop results

Additional Modules	Fs (*Hz*)	Packet size (*bits*)	#Packets	Average BER	$R_{TRANSMISSION}$ (*bps*)	$R_{TRANSFER}$ (*bps*)
LMS Equalizer On CRC On	44100	10^5	9	0	13230	7143
$(127,111)$ Reed Solomon Encoder On LMS Equalizer On CRC On	88200	10^5	9	0	23126	4001
$(2,1,6)$ Convolutional encoder On LMS Equalizer On CRC On	88200	10^5	9	0	13230	4580

Table 5-16: 8-QAM closed loop results

Notice that the average transfer rate in the closed loop form of the system using the $(2,1,6)$ encoder is reduced dramatically. Because of the fact that the encoder does not ensure the correction of the received packet many retransmissions may occur. Therefore the increase of the overall transfer time, results in the significant decrease of the transfer rate.

1.4.3 Evaluation of Convolutional encoders for 8-QAM

We experimented with Convolutional encoders of large free distance on the open and closed loop form of the basic system with 8-QAM and sampling rate at 88200 Hz. Unfortunately we could not achieve zero bit error rates.

Convolutional Encoders	Average BER	Average Transfer Rate (*bps*)	Transmission Rate (*bps*)
$(2,1,6)$	$37 \cdot 10^{-5}$	6277	13230
$(3,2,6)$	$1.5 \cdot 10^{-3}$	8489	17640

Table 5-17: Performance of the tested convolutional codes for 8-QAM on the open loop form of the system with sampling rate at 88200 Hz

At first we used the $(3,2,6)$, encoder as it reduces the transmission rate only by a factor of ~ 0.6. However the results were not satisfactory and hence we used the encoder $(2,1,6)$. We can obtain that the last encoder which achieved the best average bit error rate decreased the transfer rate at the level of the uncoded 4-PSK. The use of an encoder of code rates $\frac{1}{3},\frac{1}{4}$ is meaningless as the transfer rate will be diminished.

In the closed loop form of the basic system with 8-QAM and sampling rate of 88200 Hz, the $(2,1,6)$ encoder managed to achieve error rates below the threshold of 10^{-5} in the total of the trial transmissions. In addition the $(3,2,6)$ encoders presented a significant error correcting capability.

1.4.4 16-QAM

In 16-QAM we performed the trial transmissions over the basic system, and attained the following results for the open and closed loop form of the system.

Additional Modules	**Fs** *(Hz)*	**Packet size** *(bits)*	**#Packets**	**Average BER**	**$R_{TRANSMISSION}$** ***(bps)***	**$R_{TRANSFER}$** ***(bps)***
LMS Equalizer On	44100	10^5	9	$6.42\cdot10^{-4}$	17640	9823
	88200	10^5	9	$6.8\cdot10^{-3}$	35280	13868
$(127,111)$ Reed Solomon encoder On LMS Equalizer On	44100	10^5	9	0	13759	8413
$(127,105)$ Reed Solomon encoder On LMS Equalizer On	88200	10^5	9	$0.8\cdot10^{-3}$	28929	6996
$(3,1,6)$ Convolutional encoder On LMS Equalizer On	44100	10^5	9	0	5880	4082
$(3,1,6)$ Convolutional encoder On LMS Equalizer On	88200	10^5	9	$9.18\cdot10^{-4}$	11760	6056

Table 5-18: 16-QAM open loop results

Additional Modules	Fs (*Hz*)	Packet size (*bits*)	#Packets	Average BER	$R_{\text{TRANSMISSION}}$ (*bps*)	R_{TRANSFER} (*bps*)
Equalizer On CRC On	44100	10^5	9	0	17640	7965
$(127, 111)$ Reed Solomon encoder On LMS Equalizer On CRC On	44100	10^5	9	0	15347	5157
$(3, 1, 6)$ Convolutional encoder On LMS Equalizer On CRC On	44100	10^5	9	0	5880	2016

Table 5-19: 16-QAM closed loop results

The enhancement of the Reed Solomon and Convolutional encoders on the experiments performed at 44100 Hz, resulted in zero average bit error rate. However they did not manage to achieve average bit error rates below the threshold of 10^{-5} bit error rate at sampling frequency of 88200 Hz.

1.4.5 Evaluation of Convolutional encoders for 16-QAM

The encoders used in the experimentation on the open loop form of the basic system with 16-QAM are listed in the following table.

Convolutional Encoders	Average BER	Average Transfer Rate (*bps*)	Transmission Rate (*bps*)
$(3,1,6)$	0	4082	5880
$(2,1,6)$	$1.22 \cdot 10^{-5}$	5754	8820
$(2,1,5)$	$9.3 \cdot 10^{-4}$	5874	8820
$(3,2,6)$	$1.1 \cdot 10^{-4}$	7462	11760
$(4,3,6)$	$2.03 \cdot 10^{-4}$	8073	13230

Table 5-20: Convolutional encoders used with 16-QAM at 44100 Hz

Convolutional Encoders	Average BER	Average Transfer Rate (*bps*)	Transmission Rate (*bps*)
$(3,1,6)$	$9.18 \cdot 10^{-4}$	6056	11760
$(2,1,6)$	$6.76 \cdot 10^{-4}$	8446	17640
$(3,2,6)$	$5.6 \cdot 10^{-3}$	10451	23520

Table 5-21: Convolutional encoders used with 16-QAM at 88200 Hz

The experiments on the open loop system with 16-QAM modulation and sampling rate of 44100 Hz showed that only the $(3,1,6)$ encoder managed to perform with zero bit errors while the $(2,1,6)$ and the $(2,1,5)$ encoders achieved significant performance. On the contrary the $(3,2,6)$ and $(4,3,6)$

encoders did not lower the BER substantially. However the use of the $(3,1,6)$ encoder decreases the average transfer rate and deteriorates the overall performance.
In the trial transmissions over the basic system with 16-QAM modulation and sampling rate of 88200 Hz, the encoder with the maximum free distance attained the least average bit error rate. The first encoder tested, was the $(3,2,6)$ convolutional encoder which resulted in $5.6 \cdot 10^{-3}$ average bit error rate. In order to achieve lower bit errors we used the $(2,1,6)$ encoder which improved the average ber, but decreased the average transfer rate. Trying to nullify the bit errors we enhanced on the system the $(3,1,6)$ encoder. Although the ber decreased, the average transfer rate reached the performance level of 4-QAM and hence the use of an encoder with lower code rate was meaningless.
In the closed loop form of the system we considered only the 16-QAM scheme with sampling frequency at 44100 Hz which resulted in zero average bit error rate for $(3,1,6)$ encoder.

1.4.6 32-QAM

In the case of 32 QAM we did not achieve reliable transmission of the packets. The experimentation on the 32 QAM in general proved that there is an upper limit in the order of the modulation we are able to use in the WFTP system.

Additional Modules	**Fs (*Hz*)**	**Packet size (*bits*)**	**#Packets**	**Average BER**	**$R_{TRANSMISSION}$ (*bps*)**	**$R_{TRANSFER}$ (*bps*)**
LMS Equalizer On	44100	10^5	9	$14.3 \cdot 10^{-2}$	22050	12330
Reed Solomon encoder On LMS Equalizer On	44100	10^5	9	$9 \cdot 10^{-2}$	17199	3167
$(2,1,5)$ Convolutional encoder On LMS Equalizer On	44100	10^5	9	$7.6 \cdot 10^{-2}$	11025	5012

Table 5-22: 32 QAM open loop results

In spite of the use of a convolutional encoder with a good theoretical performance the results proved that the system cannot operate with 32 QAM. Since the average bit error rate is on the order of 10^{-2}, applying the CRC code on the system would result in a large number of retransmissions and a further deterioration of the average transfer rate.

1.4.7 MQAM conclusions

In the MQAM modulation scheme we followed the same order of experiments as in MPSK. In general the 4-QAM and the 8-QAM can operate without errors and the need of an encoder in both the open and closed loop form of the system. The use of encoders was necessary in 8-QAM at sampling rate of 88200 Hz, in 16-QAM and in 32-QAM.

In the first case despite the significant decrease in the bit error rate, the Convolutional encoders that we tested did not succeed in eliminating the transmission errors. However using a more powerful encoder would reduce the transfer rate to a minimal level. Since the basic system with 8-QAM performs with no errors at 44100 Hz with an average transfer rate of 8569 bps, the further experimentation was not essential.

In 16-QAM, the encoders eliminated the transmissions errors occurred in the uncoded scheme of the open loop system at 44100 Hz. Moreover they offered a slight improvement of the average bit error rate to the open loop system at 88200 Hz sampling rate. As expected the 32-QAM could not perform without errors. Despite the use of encoders the average bit error rate preserved in high level and hence it was not further tested. In general the highest transfer rate along with the average zero bit error rate in MQAM is performed by the uncoded 8-QAM operating at 44100 Hz.

It is worth noticing that the best average transfer time along with zero average bit error rate in the closed loop form of the system is achieved by using 16-QAM modulation scheme at sampling frequency of 44100 Hz. On the contrary we expected that 8-QAM would have the best performance as in the open loop case. However the use of encoders degraded the average transfer time in 8-QAM and hence the use of uncoded 16-QAM at sampling rate of 44100 Hz resulted in the best average transfer time.

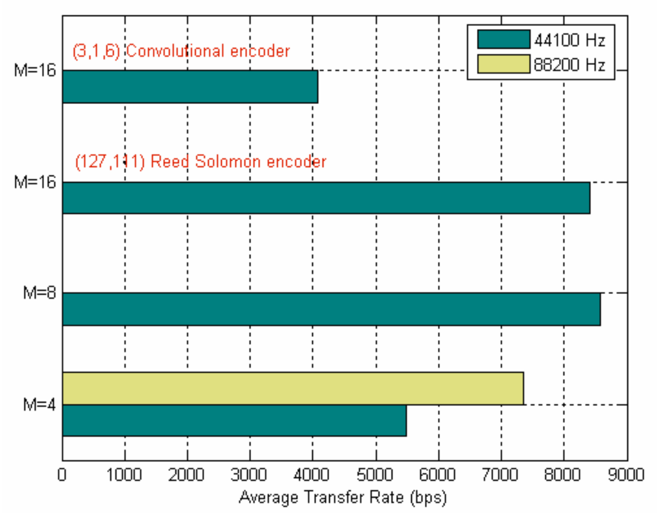


Figure 5-6: Average transfer rates of MQAM schemes that performed with zero average bit error rate in the open loop system form.

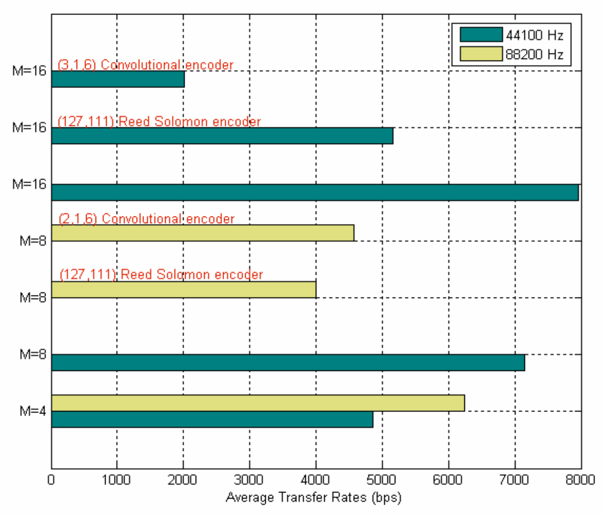


Figure 5-7: Average transfer rates of MQAM schemes that performed with zero average bit error rate in the closed loop system form.

1.5 PPM Modulation

The trial transmissions over the PPM modulation scheme were realized in a different way from the two preceding modulation schemes. The basic system consisted of the Synchronizer, the PPM modulator, the PPM correlators, the PPM detector, and the Phase recovery. The experiments performed for both sampling rates of 44100 and 88200 Hz and a varying symbol period depending on the order of the modulation.

1.5.1 4-PPM

In the 4-PPM we used symbol period $T_s = 8$ samples. The transmissions were performed for 44100 and 88200 Hz sampling rates in open and closed loop form of the system.

Additional Modules	**Fs (*Hz*)**	**Packet size (*bits*)**	**#Packets**	**Average BER**	$\mathbf{R}_{\mathbf{TRANSMISSION}}$ **(*bps*)**	$\mathbf{R}_{\mathbf{TRANSFER}}$ **(*bps*)**
LMS Equalizer Off	44100	10^5	9	0	11025	5659
	88200	10^5	9	0	22050	11399

Table 5-23: 4-PPM open loop results

Additional Modules	**Fs (*Hz*)**	**Packet size (*bits*)**	**#Packets**	**Average BER**	$\mathbf{R}_{\mathbf{TRANSMISSION}}$ **(*bps*)**	$\mathbf{R}_{\mathbf{TRANSFER}}$ **(*bps*)**
LMS Equalizer Off CRC On	44100	10^5	9	0	11025	4988
	88200	10^5	9	0	22050	9004

Table 5-24:4-PPM closed loop results

Notice that 4-PPM operating at sampling rate of 88200 Hz achieves the best transfer rate among the three modulation schemes of order *M*=4. However the 4-PSK and 4-QAM operate with symbol period of 10 samples per symbol whereas the symbol period of 4-PPM is 8 samples per symbol.

1.5.2 8-PPM

Unfortunately the increase of the order of the modulation scheme did not result in higher transfer rates. The increase in the symbol period reduced the average transfer rates at both sampling frequencies by 21% and 50.4% respectively, compared with the results of 4-PPM.

Additional Modules	**Fs** (*Hz*)	**Packet size** (*bits*)	**#Packets**	**Average BER**	$R_{TRANSMISSION}$ (*bps*)	$R_{TRANSFER}$ (*bps*)
LMS Equalizer Off	44100	10^5	9	0	8269	4487
	88200	10^5	9	0	16538	5930

Table 5-25: 8-PPM open loop results

Additional Modules	**Fs** (*Hz*)	**Packet size** (*bits*)	**#Packets**	**Average BER**	$R_{TRANSMISSION}$ (*bps*)	$R_{TRANSFER}$ (*bps*)
LMS Equalizer Off CRC On	44100	10^5	9	0	8269	4034
	88200	10^5	9	0	16538	5193

Table 5-26: 8-PPM closed loop results

1.5.3 16-PPM

In this case the symbol period was set to 32 samples per symbol and the trial transmissions occurred with no errors.

Additional Modules	**Fs** (*Hz*)	**Packet size** (*bits*)	**#Packets**	**Average BER**	$R_{TRANSMISSION}$ (*bps*)	$R_{TRANSFER}$ (*bps*)
LMS Equalizer Off	44100	10^5	9	0	5513	3097
	88200	10^5	9	0	11026	4189

Table 5-27: 16-PPM open loop results

Additional Modules	**Fs** (*Hz*)	**Packet size** (*bits*)	**#Packets**	**Average BER**	$R_{TRANSMISSION}$ (*bps*)	$R_{TRANSFER}$ (*bps*)
LMS Equalizer Off CRC On	44100	10^5	9	0	5513	2884
	88200	10^5	9	0	11026	3800

Table 5-28: 16-PPM closed loop results

1.5.4 PPM conclusions

In the PPM modulation scheme, despite the zero average bit error rate the transfer rates were not improved compared with the two preceding modulation schemes. The 4-PPM resulted with no errors while the average transfer rate at sampling rate of 88200 Hz was very promising. However from the above results we can obtain that the performance of the PPM is degraded by increasing the modulation order. The experimentation on 8 PPM with symbol period $T_s = 8\,samples$ resulted in an average bit error rate on the order of 10^{-1}. Moreover similar results were obtained by testing the 16 PPM with symbol period $T_s = 16\ samples$ which resulted in an average bit error rate of $8.4 \cdot 10^{-2}$. Even if we use a Block or Convolutional encoder, we will not achieve zero bit error rates. Consequently we increased the symbol period. However this resulted in an increase of the processing time in the receiver and an overall reduction of the average transfer rate.

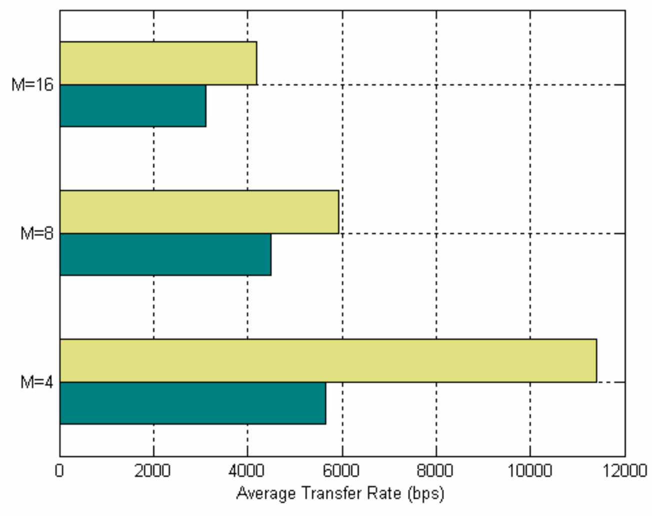


Figure 5-8: Average transfer rates of PPM schemes that performed with zero average bit error rate in the open loop system form.

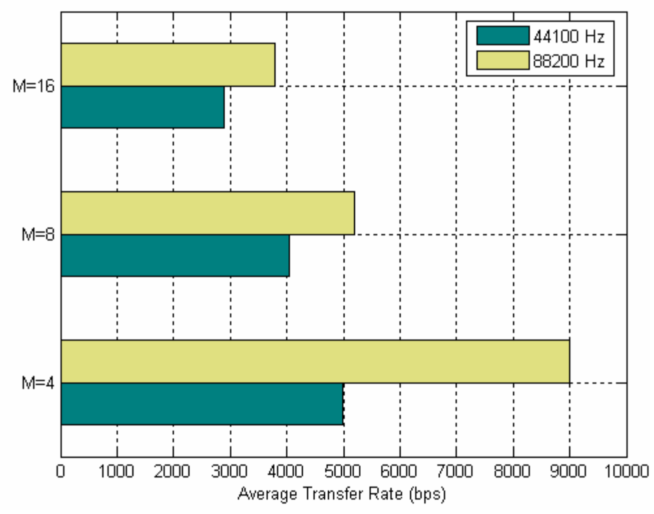


Figure 5-9: Average transfer rates of PPM schemes that performed with zero average bit error rate in the closed loop system form.

1.6 Summary

In this section we will provide an overall evaluation of the WFTP system along with the most significant conclusions that were drawn from the preceding analysis.

Average Transfer Rate	Modulation scheme	M	Fs (Hz)	Ts (samples)	Additional System Modules		
$< 4000\,bps$	PPM	16	44100	32	-	-	-
$4000 - 6000\,bps$	PSK	4	44100	10	LMS Equalizer	-	-
	PSK	4	44100	10	-	-	-
	QAM	4	44100	10	LMS Equalizer	-	-
	PPM	4	44100	8	-	-	-
	PPM	8	44100	16	-	-	-
	PPM	8	88200	16	-	-	-
	PPM	16	88200	32	-	-	-
	QAM	16	44100	10	LMS Equalizer	$(3,1,6)$ Convolutional encoder	-
$6000 - 8000\,bps$	PSK	4	88200	10	LMS Equalizer	-	-
	PSK	16	44100	10	LMS Equalizer	$(3,2,6)$ Convolutional encoder	-
	QAM	4	88200	10	LMS Equalizer	-	-
	PSK	16	44100	10	LMS Equalizer	$(4,3,6)$ Convolutional encoder	-
$8000 - 10000\,bps$	PSK	4	88200	10	-	-	-
	PSK	8	44100	10	-	-	-
	PSK	8	44100	10	LMS Equalizer	-	-
	PSK	16	44100	10	LMS Equalizer	$(255,231)$ Reed Solomon encoder	-
	QAM	8	44100	10	LMS Equalizer	-	-
	QAM	16	44100	10	LMS Equalizer	$(127,111)$ Reed Solomon encoder	-
$> 10000\,bps$	PSK	8	88200	10	-	-	-
	PSK	8	88200	10	LMS Equalizer	-	-
	PPM	4	88200	8	-	-	-

Table 5-29: Average transfer rates achieved by various open loop systems with BER below 10^{-5}

Average Transfer Rate	Modulation scheme	M	Fs (Hz)	Ts (samples)	Additional System Modules		
$< 4000\,bps$	QAM	16	44100	10	LMS Equalizer	$(3,1,6)$ Convolutional encoder	CRC
	PPM	16	88200	32	-	-	CRC
	PPM	16	44100	32	-	-	CRC
$4000-6000\,bps$	PSK	4	44100	10	LMS Equalizer	-	CRC
	PSK	4	44100	10	-	-	CRC
	QAM	4	44100	10	LMS Equalizer	-	CRC
	PPM	4	44100	8	-	-	CRC
	PPM	8	44100	16	-	-	CRC
	PPM	8	88200	16	-	-	CRC
	QAM	16	44100	10	LMS Equalizer	$(127,115)$ Reed Solomon encoder	CRC
	QAM	8	88200	10	LMS Equalizer	$(3,1,6)$ Convolutional encoder	CRC
	QAM	8	88200	10	LMS Equalizer	$(127,111)$ Reed Solomon encoder	CRC
$6000-8000\,bps$	PSK	4	88200	10	LMS Equalizer	-	CRC
	PSK	16	44100	10	LMS Equalizer	$(3,2,6)$ Convolutional encoder	CRC
	QAM	4	88200	10	LMS Equalizer	-	CRC
	PSK	16	44100	10	LMS Equalizer	$(4,3,6)$ Convolutional encoder	CRC
	PSK	8	44100	10	-	-	CRC
	PSK	8	44100	10	LMS Equalizer	-	CRC
	PSK	4	88200	10	-	-	CRC
	PSK	16	44100	10	LMS Equalizer	$(255,231)$ Reed Solomon encoder	CRC
	QAM	8	44100	10	LMS Equalizer	-	CRC
	QAM	16	44100	10	LMS Equalizer	-	CRC
$8000-10000\,bps$	PSK	8	88200	10	-	-	CRC
	PSK	8	88200	10	LMS Equalizer	-	CRC
	PPM	4	88200	8	-	-	CRC

Table 5-30: Average transfer rates achieved by various closed loop systems with BER below 10^{-5}

In the above tables we represent the average transfer rates achieved by the various open and closed loop systems with BER below the threshold of 10^{-5}.

At first we may notice that the use of the LMS equalizer did not have a negative impact on the average transfer rate of the tested systems. In general because the equalizer operates on symbols and not on the samples of the received signal, the processing time spent on equalizing is trivial. Moreover the use of the encoders managed to reduce the BER of the open loop systems with 16 QAM and 16 PSK at sampling rate of 44100 Hz. However the resulting transfer rates are similar to the transfer rates attained by the use of lower order modulation schemes. In particular the average transfer rate of the open loop system with 16 QAM and the $(3,1,6)$ convolutional encoder at sampling rate of 44100 Hz is in the same level as the average transfer rate of the open loop system with 4 PSK, 4 QAM and 4 PPM at sampling rate of 44100 Hz. In addition the average transfer rate of the open loop system with 16 PSK and the $(3,2,6)$ convolutional encoder at sampling rate of 44100 Hz is in the same level as the average transfer rate of the open loop system with 4 PSK, 4 QAM at sampling rate of 88200 Hz.

The highest transfer rates achieved in every modulation scheme that was tested with the WFTP system are listed in the following table. Notice that we consider transmissions on the open loop and closed loop form of the system where no errors occurred.

Modulation Scheme	$\mathbf{R}_{\text{TRANSFER}}$ **(*bps*)**	**Fs (*Hz*)**	$\boldsymbol{T_s}$ **(*samples*)**
8-PSK	12525	88200	10
8-QAM	8569	44100	10
4-PPM	11399	88200	8

Table 5-31: Highest transfer rates for every modulation scheme on the open loop form of the system

Modulation Scheme	$\mathbf{R}_{\text{TRANSFER}}$ **(*bps*)**	**Fs (*Hz*)**	$\boldsymbol{T_s}$ **(*samples*)**
8-PSK	9656	88200	10
16-QAM	7965	44100	10
4-PPM	9004	88200	8

Table 5-32: Highest transfer rates for every modulation scheme on the closed loop form of the system

The most efficient modulation scheme that can be used along with the WFTP system is the 8-PSK. Recall that the trial transmissions were performed with 500 bits of training sequence. In cases where we reduced the training sequence the system operated in higher transmission rates but we could not ensure that all the transmissions would occur with no errors. In order to eliminate the possible bit errors we experimented with a variety of encoders. However the resulting transfer rates were lower than 12525 bps. Modulation schemes of high order, such as 32 QAM and 32 PSK were tested with

2000, 3500 bits of training sequence. In spite of the decrease in the number of bit errors, the transfer rates were reduced at 3 – 4 kbps. The enhancement of an error correcting scheme in such a system will result in transfer rates which have no practical meaning.

In general the Convolutional encoders reduced the bit errors in the MPSK and MQAM modulation schemes. They were mainly used along with the 16-PSK, 16-QAM, 32-PSK and 32-QAM. Undoubtedly, the $(n,1,v)$ encoders, such as $(3,1,6)$ and $(2,1,6)$ presented the best error correcting capability. However due to the large amount of redundancy they introduce in the packet, the processing time in the modules of the Synchronizer, the PSK demodulator and the PSK detector is increased. On the contrary, the $(n,n-1,v)$ convolutional encoders that used in the experiments are less efficient but result in a lower rate loss. The best error correcting capability was attained by the $(2,1,6)$ convolutional encoder in the open-loop system with modulation scheme of 8-QAM and operating sampling rate of 44100 Hz. In general the performance of the Convolutional Codes is more efficient when the system is operating at sampling rate of 44100 Hz. However in order to enforce the reliability of the WFTP system without a significant decrease of the transfer rate we propose the enhancement of an $(4,3,6)$ or $(3,2,6)$ encoder as an additional module to the basic system.

The use of the CRC code improved the credibility of the system in cases where the bit error rates were on the order of 10^{-4}. However the transfer rates degraded due to the additional processing time for error detection in the receiver and the unpredictable number of retransmissions. In order to ensure the correct transmission of the packets a CRC code is proposed as an additional module in the system.

In cases where we want the system to operate with the maximum transfer rate and the least possible BER we propose the following system settings.

Fs **(*Hz*)**	$\boldsymbol{T_s}$ **(*samples*)**	**Modulation Scheme**	**Synchronizer**	**Phase Recovery**	**Equalizer**
88200	10	8-PSK	On	On	LMS

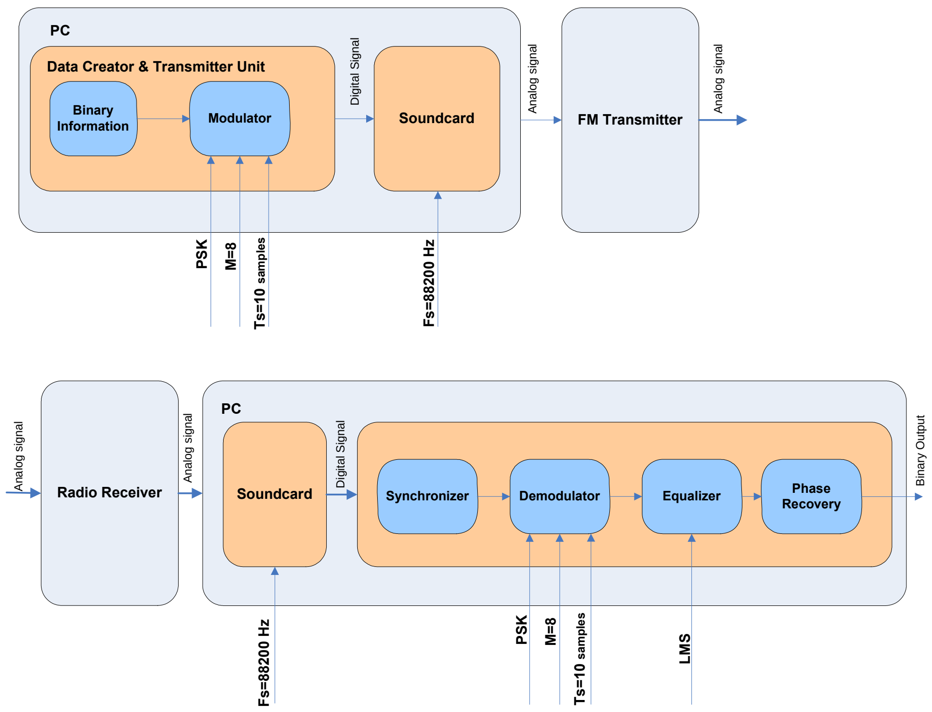


Figure 5-10: Block diagram of the system operating with the highest transfer rate

However the nominal sampling frequency supported by Matlab and the majority of the soundcards is 44100 Hz. Therefore in order to ensure the compatibility with the hardware, we suggest the system settings listed in the subsequent table.

Fs ***(Hz)***	T_s ***(samples)***	**Modulation Scheme**	**Synchronizer**	**Phase Recovery**	**Equalizer**
44100	10	8-PSK	On	On	LMS
44100	10	8-QAM	On	On	LMS

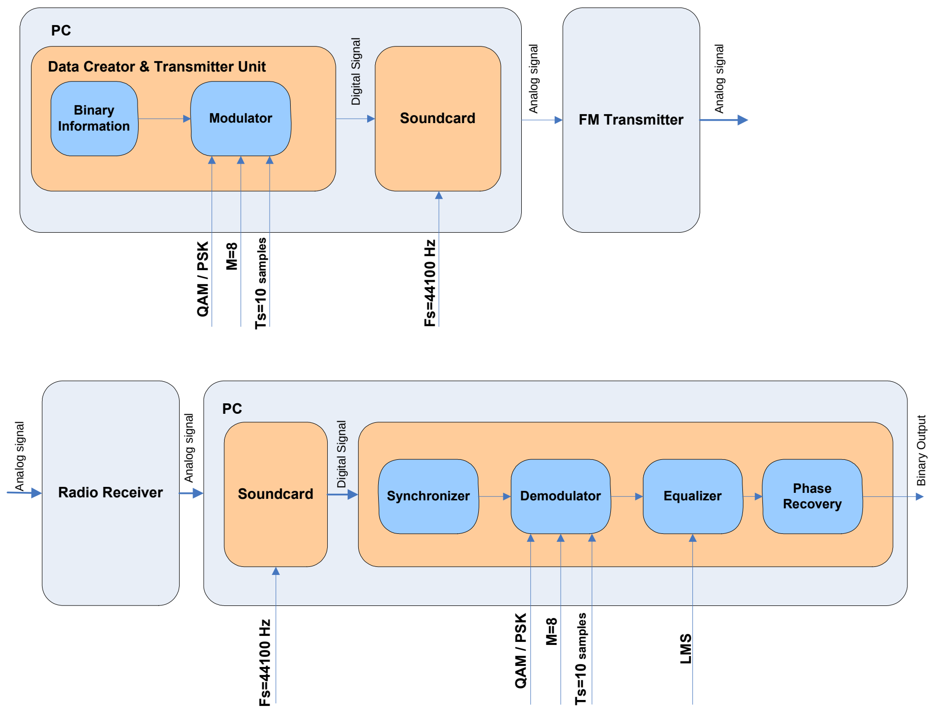


Figure 5-11: Block diagram of the system operating with the highest transfer rate, compatible with every soundcard

In cases where our primary concern is reliability, we propose the following system settings.

Fs (Hz)	T_s (samples)	Modulation Scheme	Synchronizer	Phase Recovery	Equalizer	Encoder	CRC
88200	10	8-PSK	On	On	LMS	-	On
88200	8	4-PPM	On	On	-	-	On
44100	10	16-PSK	On	On	LMS	(3, 2, 6)	-

In general the WFTP system offers the ability to experiment with the various system settings. The main idea of this project was to implement a software based communication system where we could try various system designs with different modules and observe the resulting performance. Due to hardware and software constraints the WFTP system can operate reliably up to a maximum transfer rate. The enhancement of high order modulation schemes resulted in higher transfer rates along with the need of a Block or Convolutional channel encoder for error correction. Eventually the overall results proved that in the WFTP system it is more efficient to use uncoded 8-PSK modulation scheme rather than modulation schemes of higher order with channel encoders.

REFERENCES

[1] Shu Lin, Daniel J. Costello, Jr "Error Control Coding (2nd edition)," Prentice Hall 2004

[2] Martin Bossert, "Channel Coding for Telecommunications" Wiley 1999

[3] Ajay Dholakia, "Introduction to Convolutional Codes with Applications", Kluwer Academic Publishers 1994.

[4] A. J. Viterbi, "Convolutional Codes and Their Performance in Communication Systems", IEEE Trans. Commun. Technol., COM-19:751-72, October 1971.

[5] Robert H. Morelos-Zaragoza, "The Art of Error Correcting Coding", Wiley 2002

[6] J.J. Chang, D.J. Hwang and M. C. Lin, "Some extended results on the search for Good Convolutional Codes", IEEE Trans. Inform. Theory, IT-43: 1682-97, September 1997

[7] John Proakis, Masoud Salehi, "Digital Communications", Prentice Hall 2002

[8] William Stallings, "Wireless Communications and Networks", Prentice Hall 2002

[9] Ramabadran, T., and Gaitonde,S. "A Tutorial on CRC Computations." IEEE Micro August 1988