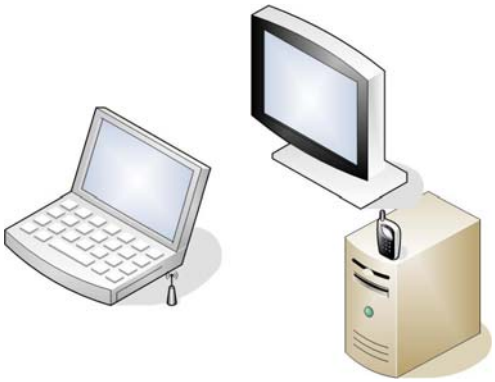
Technical University of Crete
Department of Computer and Electronic Engineering

# Block Error Correction for experimental wireless ftp radio link over analog FM



Supervisor: Νικόλαος Σιδηρόπουλος
Committee: Αθανάσιος Λιάβας
Αλέξανδρος Ποταμιάνος

Καρδαράς Γεώργιος

*Contents*

## Figures

# 1 Introduction

This thesis is part of a major project. Three other colleagues and I have formed a team for its fulfillment. The objective of the project is the connection of two computer systems via an experimental wireless link. This wireless link will be used for transferring all kind of computer files from one computer system to the other. One of the computer systems plays the role of the transmitter and the other of the receiver. In our case the transmitter unit is a laptop, whereas the receiver unit is a desktop computer. The file transferring supported is one way, which means that only files from the laptop can be send to the desktop PC.

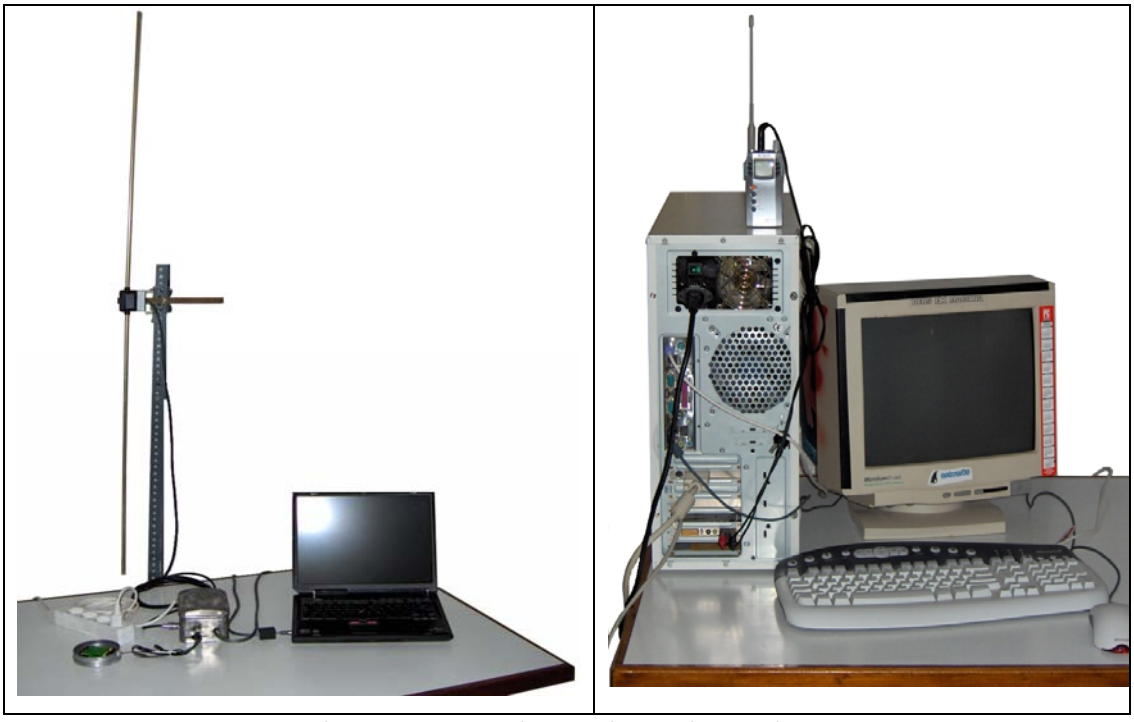


**Figure 1.1 Transmitter Unit-Receiver Unit**

Necessary prerequisite is that both computers are equipped with audio cards and Matlab. A low-power FM transmitter is connected to the audiocard's line-out RCA jack of the one computer. The FM transmitter is connected through a shield cable with a half wave dipole antenna. Likewise, a wideband communication receiver is connected to the audiocard's line-in RCA jack of the other computer. The transmitter unit generates digital data, which

are transmitted by the FM transmitter and the half wave dipole antenna over analog FM. The receiver unit receives the analog data and converts them back to digital. In this way, computer files are transferred. Both hardware and software are fully developed by the members of the team. A detailed analysis of their implementation follows.

**Figure 1.2 Complete System**

## 1.1 Hardware

The hardware used consists of the following parts:

- a low-power FM transmitter
- a half wave dipole antenna connected to the FM transmitter
- a wideband communication receiver
- 2 audiocards equipped with line-in and line-out

The FM transmitter used is a simple low-power kit. Its power is about 1 watt and the range of the signal that produces is a 1-2Km. The power is kept under or about 1 Watt,

so that there is no interference with other FM transmissions in the area. The user can adjust the frequency of the transmission between 88MHz and 108MHz by simply changing the value of a resistance with the use of a screwdriver. The input of the transmitter is the signal coming from the audiocard's line-out. The output signal of the transmitter is transferred through a shield cable to the half wave antenna where it is finally transmitted. The transmitter needs 4.5 Volt supply in order to give out power of 1 Watt. Three 1.5 Volt batteries in series are used to provide the 4.5 Volt needed power supply. Instead of using batteries, we also experimented with an AC/DC adaptor which we connected to a wall socket. The result was a constant drone caused by the frequency of the electrical AC/DC supply. For this reason, we concluded in using batteries which performed better. The transmitter is placed into a metallic box for electromagnetic shielding purposes.

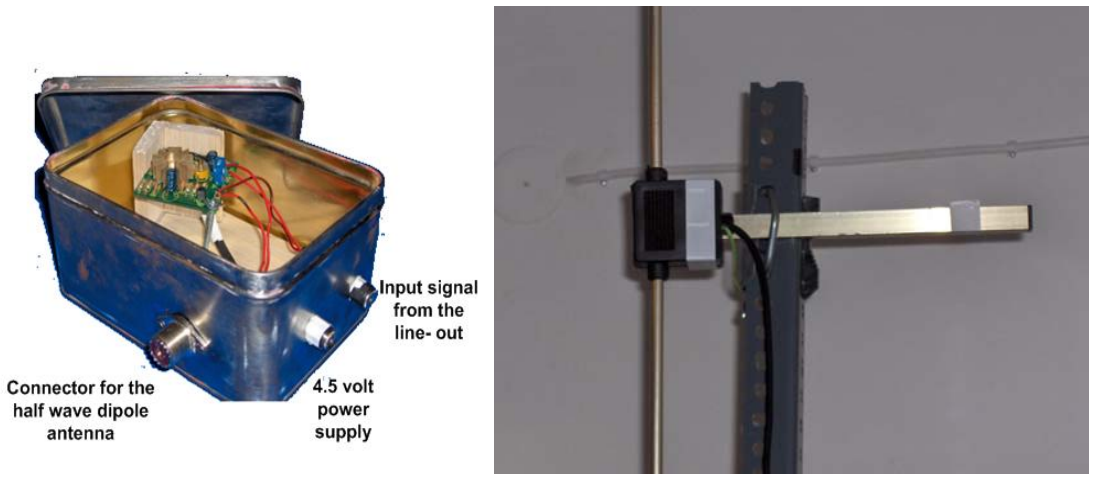



**Figure 1.3 FM transmitter and half wave dipole antenna**

The wideband communication receiver is used to detect the transmitted signal. Its specifications follow:

- Frequency range: 0.1-1300Mhz
- Antenna Impedance: 50Ωhm
- Battery Voltage: 3.6-6Volt
- Sensitivity: 30-559Mhz less than -3db

**Figure 1.4 FM transmitter, wideband communication receiver and power supply for the FM transmitter**

## 1.2 Digital Communication System

All different modules of the digital communication system are analyzed in this section. For better explanation, the system is divided in the transmitter and the receiver unit.

### 1.2.1 Transmitter Unit

The transmitter unit comprises of the following modules:

- binary “reading” of file which will be transmitted
- encoder
- interleaver
- modulator

*Binary reading of file*

First, the file we desire to send must be read. After being binary read the file is divided in groups of bits called packets. Then, the bits of each packet are imported in the encoding stage.

*Encoding stage*

In this stage the bits are encoded. By encoding, we mean that redundant bits are added and certain algorithms are implemented in order for the receiver to detect and correct the errors occurred. Our system supports both block and convolutional coding. In block coding, cyclic codes, BCH codes, Reed Solomon codes and Hamming codes are implemented. In convolutional coding, convolutional codes are implemented. Either one separately may be used.

*Interleaving stage*

In this stage the encoded bits are interleaved. By interleaving, we mean that the bits are "mixed" in order to help the encoders detect and correct as many errors as possible. Interleaving improves the system's performance especially when burst errors occur. Block interleaving and random interleaving are both supported.

*Modulating stage*

M-ary PSK, M-ary QAM and M-ary PPM are the modulation schemes our system supports. The user can define the symbol period $T$ and the order $M$ of the modulation scheme. First, the bits are converted to symbols and finally to samples. The output of this stage is the samples of the waveforms that will be transmitted. These samples are then "played" by Matlab and through the transmitter and antenna are sent to the receiver.

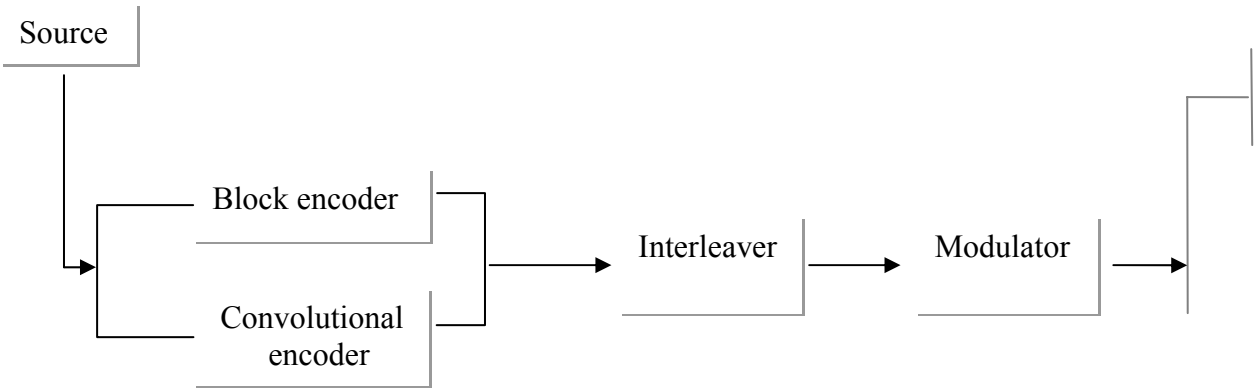



**Figure 1.5 Transmitter Unit**

### 1.2.2 Receiver Unit

The receiver unit comprises of the following modules:

- synchronization
- correlators
- amplitude and phase recovery
- equalizer
- detector
- deinterleaver
- decoder
- "write" file consisting of bits received

*Synchronizing stage*

This stage is responsible for the system's synchronization. A training sequence known by the receiver is used, so that the beginning of our signal can be detected. After detecting it the training sequence is removed and the rest of the signal is processed.

*Correlating stage*

Depending on the modulation scheme, the correspondent correlators are used. The correlators convert samples to symbols by multiplying the waveforms with the basis functions.

*Equalizing stage*

The training sequence is also used by the equalizer. In this stage, certain algorithms such as LMS, RLS and CMA process the training sequence, which is now in the form of symbols since it has exited the correlating stage. This results to a direct estimate of an appropriate equalizer. This estimation is used to equalize the rest of the data symbols. The received symbols, after equalization, are ready to enter to the correlating stage.

*Note:* We have also tried to equalize samples instead of symbols but this resulted to a great delay in the processing of each packet. This fact urged us to implement symbol-level equalization, which also proved to have a good performance.

*Amplitude and phase recovery stage*

Our system also tries to recover the amplitude and the phase of the symbols, which a very fast and efficient method.

*Detecting stage*

Depending again on the modulation scheme, the corresponding detector is used. All detectors are "minimum distance" detectors, since they calculate the distance between the received symbol coming from the correlating stage and the constellation of each modulation scheme and they detect each symbol according to the minimum of these distances.

*Deinterleaving stage*

The deinterleaver performs the inverse procedure compared to the interleaver. The bits are placed to their correct positions before being decoded.

*Decoding stage*

In this stage the bits are decoded. If errors have occurred during transmission, they are detected and then corrected. The number of errors corrected depends on the error detection and correction capability of the code used.

*"Writing" file stage*

When the bits of all packets have exited the decoding stage, they are joined together to form a file. Necessary prerequisite is that no errors occurred.

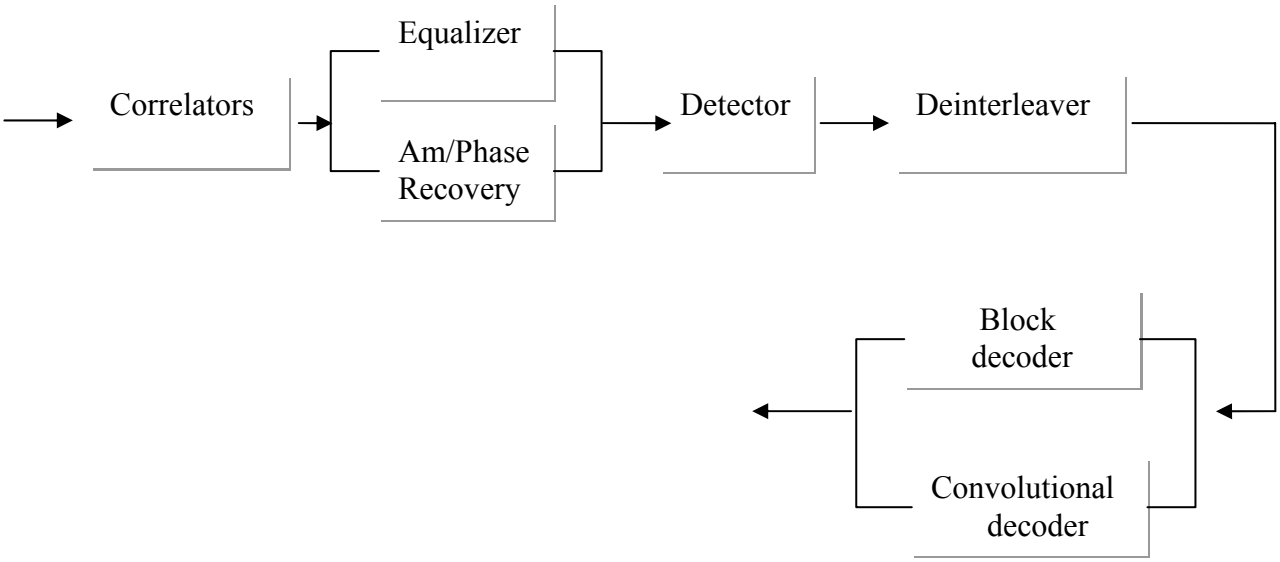



**Figure 1.6 Receiver Unit**

## 1.3 First Design

During the design of the system the existence of an Automatic Repeat Request (ARQ) scheme was predicted. In that scenario, the transmitter unit would transmit continuously packets, whereas the receiver would be divided in two sub-units. The first one would be responsible for "recording" the packets transmitted and the second unit would only process the received packets. In other words, there would be a kind of pipelining. By using a Cyclic Redundancy Code (CRC), we would know if the packet is damaged. If it is damaged or if the packet is lost, the receiver unit would demand the retransmission of this packet. This demand would reach the transmitter through the local area network (LAN) connection, using UDP objects associated with remote host, Matlab provides. However, the fact that was not predicted was the amount of memory needed to implement the above system. Indeed, the amount of memory needed to work with 2 Matlab platforms on a single computer (this refers to the receiver) is prohibitive. For this reason, the distributive system described was never implemented. However, we concluded to another system. Its description follows.

## 1.4 Final System

The system works either in open loop (no ARQ) or in a closed loop (with ARQ). The transmitter unit sends a "handshake" signal to the receiver unit through the local network

connection in order to establish a link. When the link is established, the transmitter may begin sending packets. The receiver unit has already started “recording” after the reception of the “handshake” signal. This first stage is the same both for open and close loop link.

### 1.4.1 Open loop link

In this type of connection, CRC is not added. For this reason, damaged packets or lost packets are not retransmitted. The receiver unit first “records” and then processes each packet. When the process is finished, the receiver unit asks the transmitter to transmit the next packet. The receiver unit and the transmitter unit exchange the messages “start recording” (Transmitter → Receiver) or “send next packet” (Receiver → Transmitter) through the LAN connection. This is succeeded by continuously switching roles of server and client. When the message “start recording” is sent, the transmitter unit plays the role of a server, whereas the receiver unit the role of a client. By contrast, when the message “send next packet” is sent the receiver unit is the server and the transmitter unit the client. At the end of the transmission of all packets the connection is closed using again signal through the LAN.

### 1.4.2 Closed loop link

In this type of connection CRC is added. CRC detects when a packet has not arrived correct at the receiver and alerts the system. In case a packet is damaged the receiver unit demands from the transmitter unit the retransmission of the same packet. This procedure repeats until the packet arrives correct. The rest of the procedure remains the same as in the open loop link. It is obvious that a closed loop link is a more secure way of communication, since the transmission’s result is guaranteed under normal levels of noise. However, we must also notice that by using CRC the process time of each packet increases. The cost in time will be fully analyzed in the last chapter, where the total system will be evaluated.

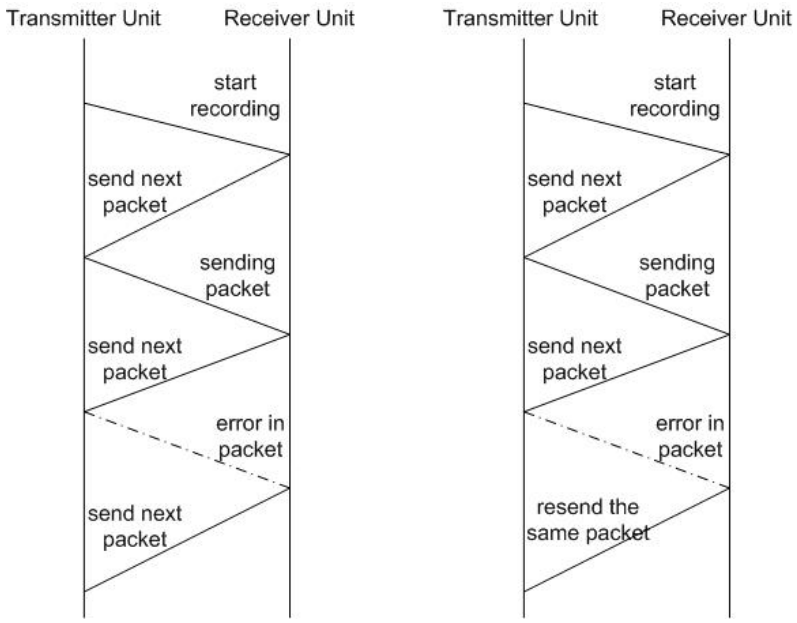



**Figure 1.7 Open loop and closed loop link**

## 1.5 Concluding Remarks

In this chapter, a general presentation of the whole system was made. Each member of the team was responsible for the implementation of certain software modules. As far as it concerns the hardware parts, all members contributed equally in their assembling. The analysis of the software modules is included in each of our thesis.

Ηλιάκης Ευάγγελος implemented: *Convolutional encoders/decoders , Phase Shift Keying (PSK) modulation scheme , Cyclic Redundancy Check (CRC) , Graphical User Interface*

Καρδαράς Γεώργιος implemented: *Block encoders/decoders , Interleavers/Deinterleavers , Pulse Position Modulation (PPM) scheme, Graphical User Interface*

Κοκκινάκης Χρήστος implemented: *Equalizers*

Μπερβανάκης Μάρκος implemented: *Viterbi decoder , Quadrature Amplitude Modulation scheme (QAM) ,Amplitude phase recovery, responsible for the assembly of all modules*

# 2 Introduction to coding

A communicational system may be characterized as reliable if no errors occur during transmission. However, regardless of the design of the transmission system there will be errors resulting in the change of one or more bits in a transmitted frame. The existence of errors depends on three important factors: the signal-to-noise-ratio (or better $E_b / N_0$ , where $E_b$ is the energy per bit and $N_0$ the noise variance), the data rate and the bandwidth. With other factors held constant the following statements are true:

- An increase in data rate increases bit error rate (BER)
- An increase in SNR decreases bit error rate
- An increase in bandwidth allows an increase in data rate

In order to cope with data transmissions errors, there are three basic approaches in common use:

- Error detection codes
- Error correction codes , also called forward error correction (FEC) codes
- Automatic repeat request (ARQ) protocols

An error detection code simply detects the presence of an error. Typically, such codes are used in conjunction with a protocol at the data link or transport level that uses an ARQ scheme. With an ARQ scheme a receiver demands the retransmission of the block of data, if an error has occurred. FEC codes are designed not just to detect but correct errors avoiding the need for retransmission. Of course the retransmission is avoided only if the code is strong enough to correct all errors. FEC schemes are frequently used in wireless transmission where retransmission scheme is highly inefficient and error rates may be high.**[10, page204]**

The goal of channel coding can be explained using the model depicted in figure 1.1. The information vector **i** is transformed into the coded bit vector **c**. Transmission over the channel introduces additive noise which results in the received bit vector **r**. **r** is decoded to produce the information bit vector $\hat{i}$ which is designed to have a reduced probability of error due the coding technique implemented. In order to correct or identify errors caused during transmission it is equivalent to calculate an estimate of the codeword or the error vector from the received vector **r**.

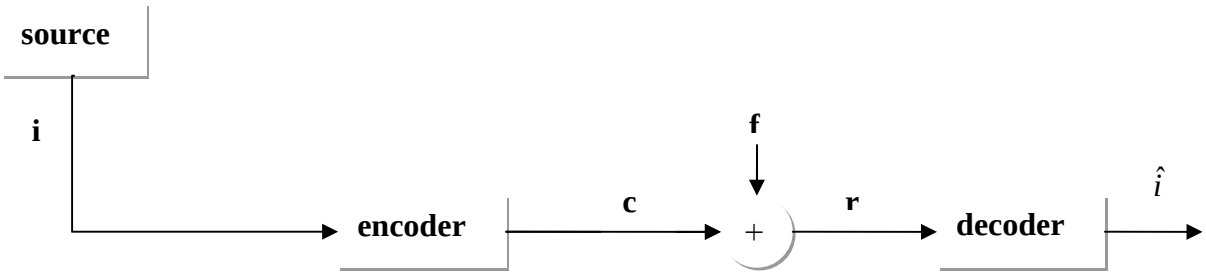
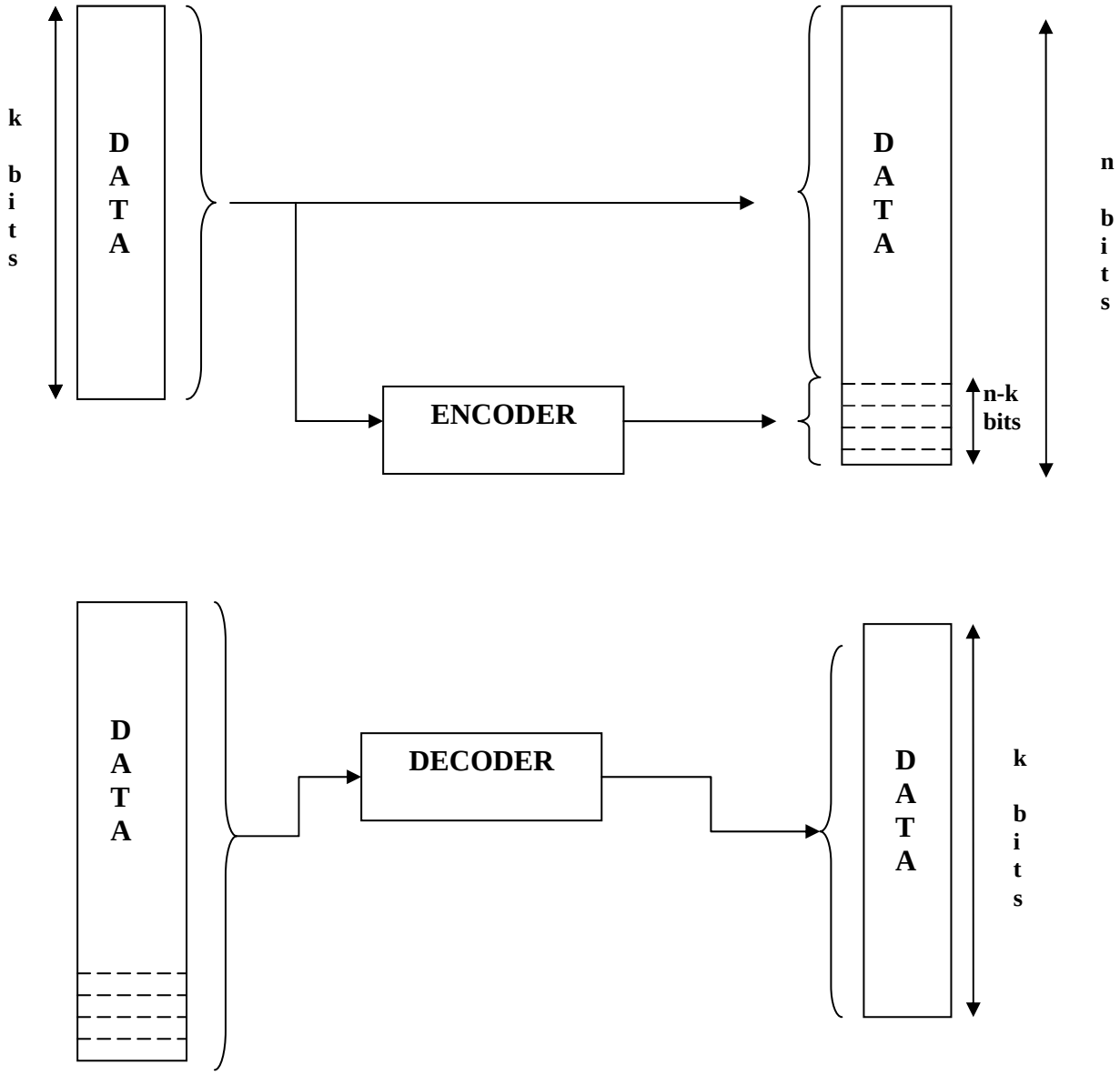



**Figure 2.1 Fundamental Structure**

All coding techniques implemented operate on the following technique. Before the transmission of each frame of bits the transmitter adds redundant bits to it. These additional bits constitute the error-detecting code. This code is calculated as a function of the other transmitted bits. Typically, for a data block of **k** bits the error detection algorithm yields an error detection code of **n - k** bits where **n-k < k**. The error detection code also referred to as the check bits is appended to the data block to produce a frame of **n** bits which is then transmitted. The incoming frame is separated by the receiver into the **k** bits of data and the rest **n-k** bits of the error correction code. Then, the same error detection calculation on the data bits as happened at the transmitter is performed at the receiver. This value is compared with the value of the incoming detection code. If there is a mismatch a detected error occurs. Generally, when a received word is decoded the output is correct or false, or no decision can be made. These outputs may be defined as correct decoding, false decoding and decoding failure respectively.

## 2.1 Possible decoding results

- Correct decoding: the transmitted codeword is equal to the decoded codeword. Errors introduced by the channel are successively removed.
- Incorrect (erroneous) decoding: the transmitted codeword differs from the decoded codeword. In this case the decoder has calculated an error which does not correspond to the true error introduced by the channel or due to excessive noise or other transmission impairments the correction code was not able to correct all the errors.
- Decoding failure: the decoder can find no solution (e.g. no valid codeword).



# 3 Galois Fields

Rational, real and complex numbers are well-known examples of infinite fields. For coding applications, a finite field is needed. Such a field is a Galois field. The following definitions are necessary for the understanding of Reed Solomon coding. They can be found in **[5,chap2]**, where there is also a more detailed analysis of the Galois field theory.

## 3.1 Group

A non-empty set A of elements is called a group under the operation * if the following axioms are satisfied:

- Closure $\forall a,b \in A : a * b \in A$
- Associativity: $\forall a,b,c \in A : a * (b * c) = (a * b) * c$
- The existence of the neutral element e: $\exists e \in A : \forall a \in A : a * e = a$
- Inverse element: $\forall a \in A : \exists b = a^{-1} \in A : a * b = e$

Commutative or Abelian groups satisfy the additional axiom

- Commutativity: $\forall a,b \in A : a * b = b * a$

## 3.2 Field

A set A with two operations (+,*) is called a field if the following axioms are satisfied:

- A is an Abelian group under addition
- A (without the null element) is an Abelian group under multiplication
- Distributive law: $\forall a,b,c \in A : a(b+c) = a \cdot b + a \cdot c$

## 3.3 Galois Field and Prime Fields

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 (+,*) mod p.

Example

| + | **0** | **1** | **2** | **3** | **4** |
|---|---|---|---|---|---|
| **0** | 0 | 1 | 2 | 3 | 4 |
| **1** | 1 | 2 | 3 | 4 | 0 |
| **2** | 2 | 3 | 4 | 0 | 1 |
| **3** | 3 | 4 | 0 | 1 | 2 |
| **4** | 4 | 0 | 1 | 2 | 3 |

| * | **0** | **1** | **2** | **3** | **4** |
|---|---|---|---|---|---|
| **0** | 0 | 0 | 0 | 0 | 0 |
| **1** | 0 | 1 | 2 | 3 | 4 |
| **2** | 0 | 2 | 4 | 1 | 3 |
| **3** | 0 | 3 | 1 | 4 | 2 |
| **4** | 0 | 4 | 3 | 2 | 1 |

***Table 3.1***

## 3.4 Primitive Element

The multiplicative group of a prime field GF(p) is a cyclic group. This means that an element $a$ exists such that any non-zero element of the field can be represented as some power of $a$.

Example

GF(5) includes the elements 0,1,2,3,4.

A primitive element of GF(5) is 2 because any non-zero element of the field can be represented as some power of 2.

$$2^1 = 2$$
$$2^2 = 4$$
$$2^3 = 8 = 3$$
$$2^4 = 16 = 1$$

## 3.5 Extension fields

There also exist fields GF(q) where q is the power of a prime p, $q = p^m$ .For other numbers no finite fields exist. The field GF(q) with $q = p^m$ m>1 , are called extension fields.

## 3.6 Irreducible polynomial

A polynomial p(x) with the coefficients from GF(p) is irreducible if there are no non-zero polynomial factors of smaller degree that also have coefficients from GF(p).

## 3.7 Primitive Polynomial

p(x) is an irreducible polynomial with **deg** p(x) =m and coefficients $p_i \in GF(p^m)$. An element $a \in GF(p^m)$ is called a primitive element if $a^i \bmod p(a)$ can produce all $p^m - 1$ elements of the extension field $GF(p^m)$ (excluding the zero element). The polynomial p(x) is called primitive if it has a primitive element as root.

# 4 Cyclic Codes

Cyclic codes form an important subclass of linear block codes. These codes are attractive for two reasons: first, encoding can be implemented easily by employing shift registers with feedback connections, and second because there are various methods for decoding them. Cyclic codes are widely used in communication systems for error control. Many classes of cyclic codes have been constructed over the years, including BCH codes and Reed Solomon codes.

## 4.1 Description of Cyclic codes

If we cyclically shift the components of an *n*-codeword $v = (u_0, u_1, ..., u_{n-1})$ one place to the right, we obtain another codeword $v' = (u_{n-1}, u_0, ..., u_{n-2})$ which is called a **cyclic shift** of $v$. If the components of $v$ are cyclically shifted $i$ places to the right, the resultant codeword is $v' = (u_{n-i}, u_{n-i+1}, ..., u_{n-1}, u_0, u_1, ..., u_{n-i-1})$. **[8 ,page 136]**

### Definition

A $(n,k)$ linear code C is called cyclic code if every cyclic shift of a codeword in C is also a codeword in C. The encoding and decoding procedure of a cyclic code will be analyzed in the next chapter, where Reed Solomon codes, which are cyclic, are thoroughly explained.

## Example

Using Matlab we will create a (7,4) cyclic code in order to verify the above definition.

*Script*

```
%codeword length
n = 7;

%message length
k = 4;

%message to be encoded in decimal values
message = 0:1:15;

%convert to binary values
message = de2bi(message.');

%cyclic encoding
codewords = encode(message,n,k,'cyclic/binary');
```

*codewords =*

| | | | | | | |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **1** | **0** | **1** | **1** | **0** | **0** | **0** |
| 1 | 1 | 1 | 0 | 1 | 0 | 0 |
| **0** | **1** | **0** | **1** | **1** | **0** | **0** |
| 1 | 1 | 0 | 0 | 0 | 1 | 0 |
| 0 | 1 | 1 | 1 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 1 | 1 | 1 | 0 |
| 0 | 1 | 1 | 0 | 0 | 0 | 1 |
| 1 | 1 | 0 | 1 | 0 | 0 | 1 |
| 1 | 0 | 0 | 0 | 1 | 0 | 1 |
| 0 | 0 | 1 | 1 | 1 | 0 | 1 |
| 1 | 0 | 1 | 0 | 0 | 1 | 1 |
| 0 | 0 | 0 | 1 | 0 | 1 | 1 |
| 0 | 1 | 0 | 0 | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 |

*cyclic shift of second row results in the fourth row*

# 5 Reed Solomon Codes

Among the non-binary cyclic codes, the most important subclass is the class of Reed Solomon codes. The RS codes were discovered by Irving S.Reed and Gustave Solomon in 1960 independently of the work by Hocquenghem, Bose and Chaudhuri **[9, introduction]**. Reed Solomon codes are a class of codes that are often used in practical systems. They are based on the Galois field theory. All arguments and algorithms are applicable not only for prime Galois fields but also for extension fields.

**Definition 5.1**

Let $a \in GF(p)$ be an element of order **n.** The Reed Solomon code of length **n** is defined by the set of polynomials A(x) of degree less than **k** such that $A(x) = A_0 + A_1 x + A_2 x^2 + ... + A_{k-1} x^{k-1}, A_i \in GF(p), k \le n.$ The codewords **a=** $(a_0, a_1, ..., a_{n-1})$ are generated by $a_i = A(a^i)$:

$$C := \{a \mid a_i = A(a_i), i = 0,1,...,n-1, \deg A(x) < k\} \qquad \textbf{[5, page 46]}$$

The minimum distance is **d = n - k + 1** and the dimension of the code is **k**.

With Reed Solomon codes, data are processed in chunk of $m$ bits, called symbols. An (n, k) Reed Solomon code has the following parameters:

**Symbol length** : $m$ bits per symbol

**Block length** : $n = 2^m - 1$ symbols = $m(2^m - 1)$ bits

**Data length** : $k$ symbols

**Size of check code** : $n - k = 2t$ symbols $= m(2t)$ bits

**Minimum distance** : $d_{\min} = 2t + 1$ symbols

where $t$ is the error correction capability of the RS code.

Example

The construction of the RS code of length 6 with minimum distance 5 over GF (7). We must first find a primitive element for this field, or, in other words an element with the order 6. We must also choose all polynomials A(x) with degree $k-1 \le n-d$ .This results in k=2.

We first verify that $a$ =5 is a primitive element from GF(7).

$5^1 = \mathbf{5}$

$5^2 = 25 = \mathbf{mod}\ (25, 7) = \mathbf{4}$

$5^3 = 25 \cdot 5 = 4 \cdot 5 = 20 = \mathbf{mod}\ (20, 7) = \mathbf{6}$

$5^4 = 30 = \mathbf{mod}\ (30, 7) = \mathbf{2}$

$5^5 = 10 = \mathbf{mod}\ (10, 7) = \mathbf{3}$

$5^6 = 15 = \mathbf{mod}\ (15, 7) = \mathbf{1}$

We notice that indeed any non-zero element of the field can be represented as some power of **5.**

We can now obtain the codewords of the RS code through the formula $a_i = A(a_i)$ , $A(x) = A_0 x + A_1 x$ with $A_0, A_1 \in GF(7)$ . The calculation of the codeword for example for the information polynomial $A(x) = 5 + 3x$ gives:

$a_0 = A(a^0) = A(1) = 5 + 3 = \mathbf{mod}(8, 7) = 1$

$a_1 = A(a^1) = A(5) = 5 + 3 \cdot 5 = \mathbf{mod}(20, 7) = 6$

$a_2 = A(a^2) = A(4) = 5 + 3 \cdot 4 = \mathbf{mod}(17, 7) = 3$

$a_3 = A(a^3) = A(6) = 5 + 3 \cdot 6 = \mathbf{mod}(23, 7) = 2$

$a_4 = A(a^4) = A(2) = 5 + 3 \cdot 2 = \mathbf{mod}(11, 7) = 4$

$a_5 = A(a^5) = A(3) = 5 + 3 \cdot 3 = \mathbf{mod}(14, 7) = 0$

$\Rightarrow a = (1, 6, 3, 2, 4, 0)$

**Definition 5.2**

The discrete transform is defined as $(i, j \in 0...n-1)$

**Transform** $a_i = A(a^i)$

**Inverse** $A_j = n^{-1}a(a^{-j})$

$a \in GF(p)$ is an element of order n and $a(x)$ and A(x) are polynomials of degree $\leq$ n-1 with coefficients from GF(p).

Let $A(x) = A_0 + ... + A_{n-1}x^{n-1}$ and $B(x) = B_0 + ... + B_{n-1}x^{n-1}$ be two polynomials from GF(p). Then the cyclic convolution A(x)*B(x) is defined as a polynomial $C(x) = C_0 + ... + C_{n-1}x^{n-1}$ where $C_j = \sum_{i=0}^{n-1} A_i B_{j-i}, j = 0,1,...,n-1$ .All indices are calculated modulo n.

## 5.1 Generator Polynomial

The Reed Solomon code of length n, dimension k and minimum distance d=n-k+1 can also be defined by all (information) polynomials i(x) of degree<k. The codewords are calculated through multiplication by the generator polynomial:

$$a(x) = \text{i(x)g(x)} \quad \deg g(x) = n-k$$

Each codeword $a(x)$ must be divisible by g(x). In each codeword $a(x)$, the elements $x = a^i$ must be roots, where $a$ is a primitive element. These correspond to the linear factors $x - a^{-i}$. The product of these linear factors results in the generator polynomial g(x) of degree n-k.

$$g(x) = \prod_{i=k}^{n-1}(x - a^{-i})$$

**Example**

We should like to construct the RS code of length n=6 with minimum distance d=5 over GF(7) like we did in the previous example. The dimension of our code is $k = n - d + 1 = 2$ and a primitive element is $a$=5. We will now calculate the generator polynomial:

$$\begin{aligned} g(x) &= \prod_{i=2}^{5} (x - a^{-i}) \\ &= (x - a^{-2})(x - a^{-3})(x - a^{-4})(x - a^{-5}) \mid \bmod 6 \\ &= (x - a^{4})(x - a^{3})(x - a^{2})(x - a\ ) \\ &= (x^2 - (a^4 + a^3)x + a^7)(x^2 - (a^2 + a)x + a^3) \\ &= (x^2 - x + 5)(x^2 - 2x + 6) \\ &= (x^4 - x^3 + 5x^2 - 2x^3 + 2x^2 - 10x + 6x^2 - 6x + 30) \mid \bmod 7 \end{aligned}$$

$$g(x) = x^4 + 4x^3 + 6x^2 + 5x + 2$$

The codeword from the previous example must be divisible by g(x). Therefore $a(x) / g(x) = i(x)$

$$(4x^4 + 2x^3 + 3x^2 + 6x + 1) / (x^4 + 4x^3 + 6x^2 + 5x + 2) = 4$$

Therefore the polynomial is $a(x) = 4g(x), i(x) = 4 + 0x$

## 5.2 Encoding

There are different methods to generate the Reed Solomon encoding scheme. All methods employ the same code but a particular information vector is mapped, in general, to a different codeword, depending on the encoding technique. Some of these methods are systematic and some of them non-systematic. The difference between systematic and non-systematic methods of encoding is that in systematic encoding, information and parity check symbols are separated. Moreover, information symbols remain the same. In our application, a systematic method is used.

### 5.3 Method 1 (non-systematic)

The **k** information digits are the coefficients of the polynomial $i(x) = i_0 + i_1 x + ... + i_{k-1} x^{k-1}$ . The codeword $a(x)$ results from the multiplication by the generator polynomial $a(x) = i(x)g(x)$

**Example**

We would like to construct the RS code of length n=7 and dimension k=3 over the extension field GF (8) $= GF(2^3)$ of GF (2).

The information digits are the coefficients of the polynomial:

$i(x) = 2x^2 + 1x + 3$ , data = { 2 1 3 }

Since we work over the extension field GF (8) we first define the primitive polynomial to be used. The primitive polynomial is $a^3 + a + 1 = 0$ and its coefficients belong to GF (2). Moreover, as a root of the primitive polynomial we will use $a = 2$ .We will now verify that the primitive polynomial and the root we selected are indeed correct.

| | **Calculate** $a^i$ **,** $i \in [0,7]$ | **Substitution with** $a = 2$ |
|---|---|---|
| $a^0$ | 1 | 1 |
| $a^1$ | $a$ | 2 |
| $a^2$ | $a^2$ | 4 |
| $a^3$ | $a+1$ * | 3 |
| $a^4$ | $a(a+1) = a^2+a$ | 6 |
| $a^5$ | $a(a^2+a) = a^3+a^2 = a^2+a+1$ | 7 |
| $a^6$ | $a(a^2+a+1) = a^3+a^2+a = a^2+a+a+1 = a^2+1$ ** | 5 |
| $a^7$ | $a(a^2+1) = a^3+a = a+1+a = 1$ | 1 |

* $a^3$ is equal to $a+1$ because its coefficients belong to GF (2).

Therefore, $a^3 + a + 1 = 0$

$\Rightarrow a^3 = -a - 1$

Because mod (-1, 2) = 1

$\Rightarrow a^3 = a + 1$

** $a^2+a+a$ is equal to $a^2+1$ because its coefficients belong to GF (2).

Therefore, $a^2+a+a+1 = a^2+2a+1 = a^2+1$ because mod (2, 2) = 0

***Table 5.1***

| **Integer** | **Representation Binary** | **Representation Element of GF(8)** |
|---|---|---|
| 0 | 000 | 0 |
| 1 | 001 | 1 |
| 2 | 010 | $a$ |
| 3 | 011 | $a+1$ |
| 4 | 100 | $a^2$ |
| 5 | 101 | $a^2+1$ |
| 6 | 110 | $a^2+a$ |
| 7 | 111 | $a^2+a+1$ |

***Table 5.2***

After substituting with the root $a = 2$ we can produce any non-zero element of the field**.** We also verify that our code is cyclic.

We now proceed by calculating the generator polynomial g(x).

$$
\begin{aligned}
g(x) &= \prod_{i=3}^{6} (x - a^{-i}) \\
&= (x - a^{-3})(x - a^{-4})(x - a^{-5})(x - a^{-6}) \mid \bmod 7 \\
&= (x - a^{4})(x - a^{3})(x - a^{2})(x - a\ ) \\
&= (x^2 - (a^4 + a^3)x + a^7)(x^2 - (a^2 + a)x + a^3) \\
&= (x^2 - (a^2 + 1)x + 1)(x^2 - (a^2 + a)x + a + 1) \mid a = 2 \\
&= (x^2 - 3x + 1)(x^2 - 6x + 3) \\
&= x^4 + 7x^3 + 6x^2 + x + 3
\end{aligned}
$$

Then we can calculate the codeword by multiplicating the information polynomial with the generator polynomial.

$$a(x) = i(x)g(x) = 2x^6 + 4x^5 + 3x^4 + 6x^3 + 6x^2 + 0x + 5$$

data { 2 1 3} $\Rightarrow$ encoded data { 2 4 3 6 6 0 5 }

*Calculate in Matlab*

We must first create the extension field GF (8).

*x_gf = gf(x, m)* creates a Galois field array from the matrix x. The Galois field has *2^m* elements, where m is an integer between 1 and 16. In our case *m=3*. The elements of x must be integers between 0 and *2^m-1*. The output *x_gf* is a variable that Matlab recognizes as a Galois field array, rather than an array of integers. As a result, when you manipulate *x_gf* using operators or functions such as + or **t*, Matlab works within the Galois field you have specified.

*Script*

```
%create information polynomial
data = [ 2 1 3];

%create Galois Field , work over GF(8)
data_gf = gf([ 2 1 3 ],3);
```

Then we must create the generator polynomial.

*genpoly = rsgenpoly(n,k)* returns the narrow-sense generator polynomial of a Reed-Solomon code with codeword length *n* and message length *k*. The output *genpoly* is a Galois row vector that represents the coefficients of the generator polynomial in order of descending powers. The narrow-sense generator polynomial is $(x-a^1)(x-a^2)...(x-a^{2t})$ where $a$ is a root of the default primitive polynomial for the field GF (n+1) and t is the code's error-correction capability, *(n-k)/2*.

*Script*

```
%create generator polynomial using the default primitive polynomial
generator_polynomial = rsgenpoly(7,3);
```

Finally, we multiply the information polynomial with the generator polynomial using the conv on Galois vectors that represent the polynomials. Multiplication of polynomials is equivalent to convolution of vectors.

*Script*

```
%calculate the encoded data
codeword = conv(data_gf,generator_polynomial);
```

### 5.4 Method 2 (systematic)

The **k** information coordinates are $a_{n-k}, a_{n-k+1}, ..., a_{n-1}$. A codeword can be obtained in systematic form by adding n-k parity check symbols to the data symbols. The n-k parity check coordinates are calculated in the following manner:

$$(a_{n-1}x^{n-1}+\ldots+a_{n-k}x^{n-k}):g(x)=i(x),remainder(x)$$

$$a(x)=a_{n-1}x^{n-1}+\ldots+a_{n-k}x^{n-k}-remainder(x)$$

*Calculate in Matlab*

We will encode the data of the above example in a systematic way.

data { 2 1 3} $\Rightarrow$ $a_{n-1}x^{n-1}+\ldots+a_{n-k}x^{n-k}=2x^6+x^5+3x^4$

*Script*

```
%calculate over Galois Field
data=gf([2 1 3 0 0 0 0],3);
```

We will use the default generator polynomial Matlab produces:

*Script*

```
%default generator polynomial
g=rsgenpoly (7, 3)
```

*g = GF (2^3) array. Primitive polynomial = D^3+D+1 (11 decimal)*

*Array elements =*

*1 3 1 2 3*

Therefore, $g(x)=x^4+3x^3+x^2+2x+3$

Finally, we calculate the parity symbols by division.

$(a_{n-1}x^{n-1}+\ldots+a_{n-k}x^{n-k}):g(x)=i(x),remainder(x)$

*Script*

```
%calculate remainder of division
[result,remainder]=deconv(data,g);
remainder
```

*remainder = GF (2^3) array. Primitive polynomial = D^3+D+1 (11 decimal)*

*Array elements =*

*0 0 0* ***6 0 4 5***

Numbers in bold are the parity symbols. So, the encoded data are:

encoded data { 2 1 3 6 0 4 5 }

We can verify our result by using Matlab's *rsenc* function.

*Script*

```
%work over Galois Field
data = gf([2 1 3],3);

%rsenc uses the default generator polynomial
encoded_data = rsenc(data,7,3);
```

*encoded_data = GF(2^3) array. Primitive polynomial = D^3+D+1 (11 decimal)*

*Array elements =*

2 1 3 **6 0 4 5**

## 5.5 Decoding

We will assume that a Reed Solomon code whose codewords are $a(x)$ are defined according to definition 1.1 as: $a(x) \leftrightarrow A(x), A_0 = A_1 = A_2 = ... = A_{d-2} = 0$

A(x) has exactly *d-1* successive coefficients equal to zero; hence the RS code has distance *d* and can correct $\left[\frac{d-1}{2}\right]$ errors. We assume that we receive the vector $r(x) = a(x) + f(x)$ where the coefficients $f_i \in GF(q)$. The error polynomial $f(x)$ is defined as having $f_i \neq 0$ for every error location i. We shall also assume that fewer than $\left[\frac{d-1}{2}\right]$ errors have occurred during the transmission.

To determine whether r(x) is a codeword, r(x) is transformed and one checks if $R_0 = ... = R_{d-2} = 0$ since

$$a(x) + f(x) = r(x) \leftrightarrow R(x) = A(x) + f(x)$$

If f(x) =0 then R(x) =A(x) and the coefficients $R_0 = ... = R_{d-2} = 0$ are equal to zero. On the other hand if $f(x) \neq 0$ then we define the syndrome polynomial $S(x) = S_0 + S_1 x + ... + S_{d-2} x^{d-2}$ where $R_i = F_i = S_i, i = 0,1,...,d-2$ .The syndrome polynomial depends only on the transformed error polynomial F(x) because by definition $A_0 = A_1 = A_2 = ... = A_{d-2} = 0$

## 5.6 Error locator polynomial

The error locator polynomial is defined such that:

$$c_{i=0} \Leftarrow f_i \neq 0 \quad \text{i.e.} \quad c_i f_i = 0$$

The coefficients of c(x) are equal to zero at the error locations and arbitrary at the non-error locations. This defines a large class of polynomials. From this class of polynomials we select only those for which deg C(x) $c(x) \leftrightarrow C(x)$ is equal to the number of errors. This implies that $c_i \neq 0 \Leftarrow f_i = 0$. With the transformation according to definition 5.2 we can map c(x) to C(x) as:

$$C(x) := \prod_{\{i | f_i \neq 0\}} (x - a^i)$$

The degree of C(x) is the number of coefficients from c(x)that are equal to zero or in other words the number of errors in the vector f(x).

$a(x)$ sent codeword

$f(x)$ error X marks an error location

$r(x) = a(x) + f(x)$ transmitted word

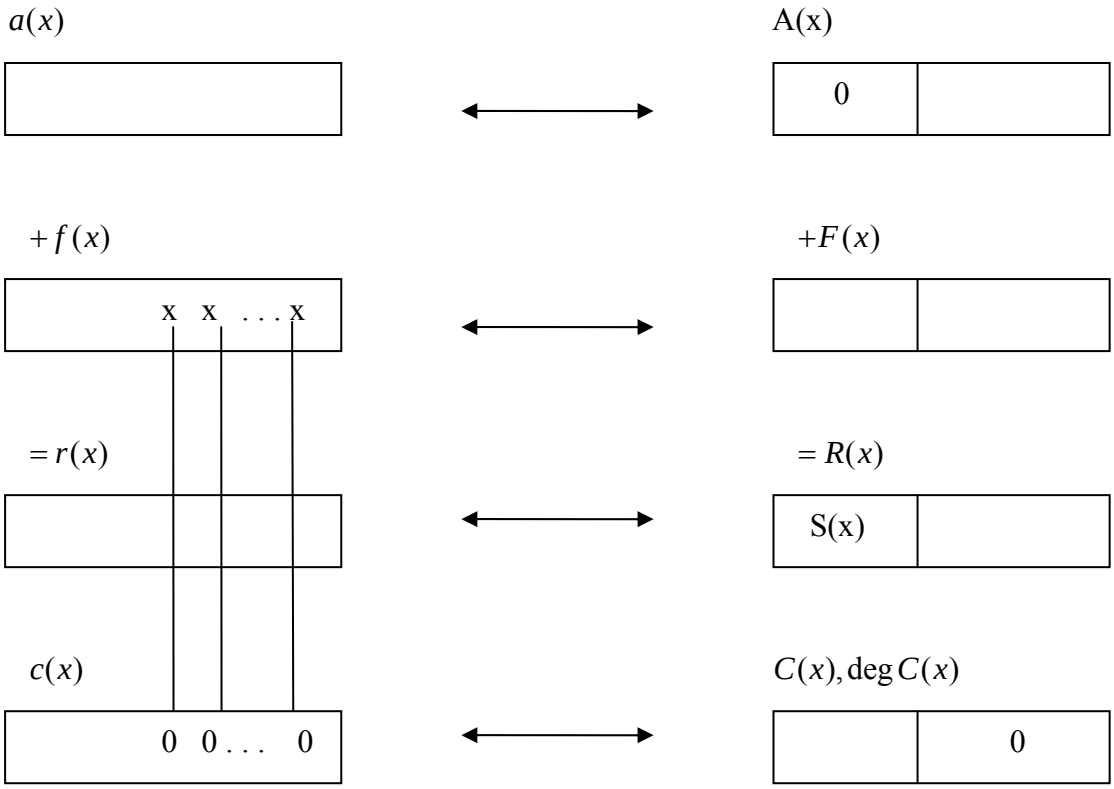



**Figure 5.1 Error correction concept**

it then applies

$$c_i f_i = 0 \leftrightarrow C(x)F(x) = 0 \bmod(x^n - 1)$$

## 5.7 Berlekamp-Massey Algorithm

The Berlekamp-Massey algorithm is a very efficient method of calculating the error location polynomial C(x). "The BMA solution can also be formulated as the shortest feedback shift register with feedback coefficients $C_i$ that can generate all syndrome coefficients". The algorithm is iterative and begins with the error locator polynomial C(x) with degree 1. At every iteration, the degree of C(x) is increased by at most one. The new $C_i(x)$ is calculated from the preceding polynomial $C_{i-1}(x)$. The flow chart of Berlekamp-Massey algorithm follows **[5, page 62]**. The complexity of Berlekamp-Massey algorithm is $O(n^2)$.

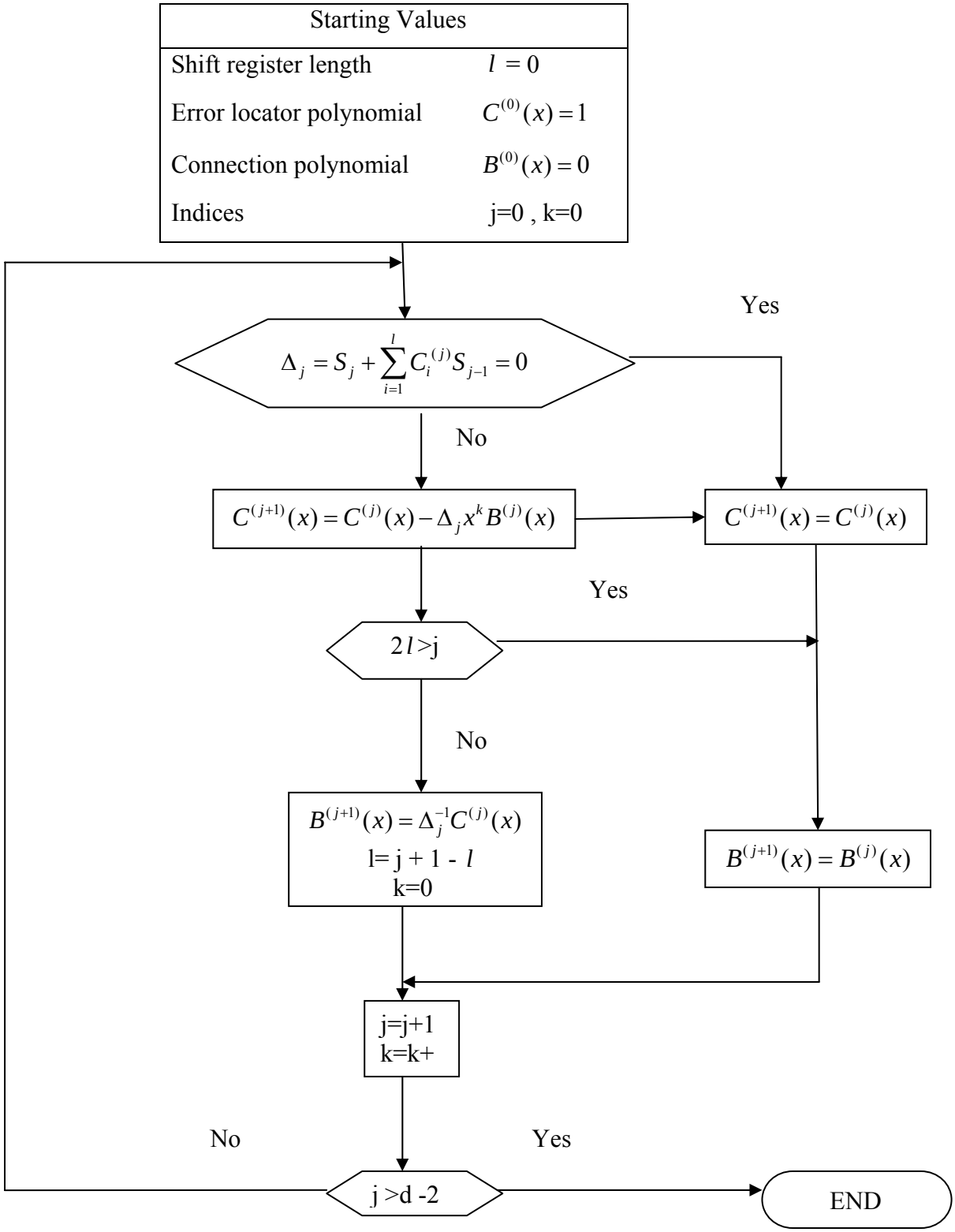



**Figure5.2 Berlekamp-Massey algorithm flowchart**

Example

We created the RS code of length 6, with minimum distance 5 over GF (7).

Our data were: data {5, 3}

After encoding the result was: encoded_data {1, 6, 3, 2, 4, 0}

Therefore, the received polynomial should have been: $a(x) = 4x^4 + 2x^3 + 3x^2 + 6x + 1$

However, we assume that during transmission two errors occurred. The error polynomial is $f(x) = 5x^4 + 3x$ .The received polynomial in this case is:

$r(x) = a(x) + f(x) = 2x^4 + 2x^3 + 3x^2 + 2x + 1$

At the receiver we know only r(x) and the RS code used. We calculate the syndrome S(x) of the received vector ( $a = 5$ is the primitive element)

$$\begin{aligned} S_0 = R_2 = 6r(a^4) &= 6(2a^{16} + 2a^{12} + 3a^8 + 2a^4 + 1) \\ &= 6(2a^4 + 2a^0 + 3a^2 + 2a^4 + 1) \\ &= 6(2\cdot 2 + 2\cdot 1 + 3\cdot 4 + 2\cdot 2 + 1) \\ &= 5 \end{aligned}$$

$$\begin{aligned} S_1 = R_3 = 6r(a^3) &= 6(2a^{12} + 2a^9 + 3a^6 + 2a^3 + 1) \\ &= 6(2a^0 + 2a^3 + 3a^0 + 2a^3 + 1) \\ &= 6(2 + 12 + 3 + 12 + 1) \\ &= 5 \end{aligned}$$

$S_2 = R_4 = 3$

$S_3 = R_5 = 3$

$\Rightarrow S(x) = 5 + 5x + 3x^2 + 3x^3$

We will now use the Berlekamp-Massey algorithm in order to calculate the error locator polynomial C(x).

| $j$ | $k$ | $l$ | $\Delta_j$ | $C^{(j+1)}(x)$ | $2l > j$ | $B^{(j+1)}(x)$ |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | $\Delta_0 = S_0 = 5$ | $C^{(1)} = 1 - 5x^0 0 = 1$ | No | $B^{(1)} = 5^{-1} = 3$ |
| 1 | 1 | 1 | $\Delta_1 = S_1 + C_1^{(1)} S_0$<br>$= 5$ | $C^{(2)} = 1 - 5x \cdot 3$<br>$= 1 + 6x$ | Yes | $B^{(2)} = B^{(1)} = 3$ |
| 2 | 2 | 1 | $\Delta_2 = S_2 + C_1^{(2)} S_1$<br>$=3 + 6 \cdot 5$<br>$= 5$ | $C^{(3)} = C^{(2)} - \Delta_2 x^2 B^{(2)}$<br>$= 1 + 6x - 5x^2 \cdot 3$<br>$= 1 + 6x + 6x^2$ | No | $B^{(3)} = 3 + (1 + 6x)$<br>$= 3 + 4x$ |
| 3 | 1 | 2 | $\Delta_3 = S_3 + C_1^{(3)} S_2 + C_2^{(3)} S_1$<br>$= 3 + 6 \cdot 3 + 6 \cdot 5$<br>$= 2$ | $C^{(4)} = C^{(3)} - 2xB^{(3)}$<br>$= 1 + 6x + 6x^2$<br>$-2x(3 + 4x)$<br>$= 1 + 5x^2$ | Yes | $B^{(4)} = B^{(3)}$ |

We calculated the error locator polynomial: $C(x) = 1 + 5x^2$

The degree of the error locator polynomial is equal to the number of errors. In our case the degree is 2, so there are two errors. In order to find the location of these errors we must calculate c(x). The coefficients of c(x) are equal to zero at the error locations and arbitrary at the non-error locations.

$$c_0 = C(a^0) = C(1) = 1 + 5 \cdot 1 = \mathrm{mod}(6, 7) = 6$$
$$c_1 = C(a^1) = C(5) = 1 + 5 \cdot 5^2 = \mathrm{mod}(126, 7) = 0$$
$$c_2 = C(a^2) = C(4) = 1 + 5 \cdot 4^2 = \mathrm{mod}(81, 7) = 4$$
$$c_3 = C(a^3) = C(6) = 1 + 5 \cdot 6^2 = \mathrm{mod}(181, 7) = 6$$
$$c_4 = C(a^4) = C(2) = 1 + 5 \cdot 2^2 = \mathrm{mod}(21, 7) = 0$$
$$\Rightarrow c = (6, 0, 4, 6, 0)$$

According to Berlekamp-Massey algorithm the error coordinates are 1 and 4 which is true since the error polynomial is $f(x) = 5x^4 + 3x$

## 5.8 Calculation of the error values

By finding C(x) we know the error locations in the received codeword. In a case of a binary code we also know the error value .In the case of a non-binary code we must calculate the error values. For this reason we use the Forney algorithm. We need to introduce the variable $l$ which corresponds to a cyclic shift of F(x), where $f(x)$ the error polynomial.

$$F_0^{(l)} = S_0, F_1^{(l)} = S_1, ..., F_{2t-1}{}^{(l)} = S_{2t-1}$$

## 5.9 Error value computation

$T^{(l)}(x)$ is called the error evaluator polynomial and is calculated by multiplication of the error locator polynomial C(x) and the syndrome S(x). The error values $f_i$ can be calculated:

$$f_i = x^{-l} n x^{-1} \frac{T^{(l)}(x)}{C'(x)} \Big| x = a^i$$

The integer $l$ is defined by

$$l: \qquad F_0^{(l)} = S_0, F_1^{(l)} = S_1, ..., F_{2t-1}^{(l)} = S_{2t-1}$$

therefore $F^{(l)}(x) = x^{(l)} F(x) \mod (x^n - 1)$

and $T^{(l)}(x)$ is defined by

$$T_j^{(l)} = -\sum_{i=0}^{j} S_{j-1} C_i, \; j = 0,1,...,e-1 \quad \text{with degC(x) = e}$$

**Example**

We will continue with the same example. We have already calculated the syndrome to be $S(x) = 5 + 5x + 3x^2 + 3x^3$. The error locator polynomial was also calculated to be $C(x) = 1 + 5x^2$ using the Berlekamp Massey algorithm.

We must calculate now the error evaluator polynomial $T^{(-2)}(x)$

$$\left.\begin{aligned} T_0^{(-2)} &= -C_0 S_0 = 2 \\ T_1^{(-2)} &= -C_0 S_1 - C_1 S_0 = 2 \end{aligned}\right\} \; T^{(-2)}(x) = 2x + 2$$

Then, we calculate the error values for $f_1$ and $f_2$.

$$C'(x) = 3x, T^{(-2)}(x) = 2x + 2$$

$$f_{1,4} = x^{2-1} n \frac{T^{(-2)}(x)}{C'(x)}\Big|_{\substack{x=5=a^1 \\ x=2=a^4}} = \begin{cases} 5 \cdot 6 \cdot \frac{5}{1} = 3 \\ 2 \cdot 6 \cdot \frac{6}{6} = 5 \end{cases}$$

## 5.10 Implementation of Reed Solomon coding on our system

On our system we chose to implement a non-binary Reed Solomon coding using a systematic method. In this chapter we will explain the procedure that was followed.

We must first define the parameters of the Reed Solomon code we will use:

- **symbol length** : $m$ bits per symbol
- **block length** : $n = 2^m - 1$ symbols
- **error correction capability** : $t = (n-k)/2$
- **data length** : $k$ symbols

Thus, the encoding algorithm expands a block of $k$ symbols to $n$ symbols by adding $n-k$ redundant check symbols. The input of our encoder is a bit stream. First, we define the symbol length in order to convert bits into symbols. Every symbol contains $m$ bits. Each symbol is stored in a row of a table. In order to implement a non-binary RS code we must convert the binary values of each row into decimal.

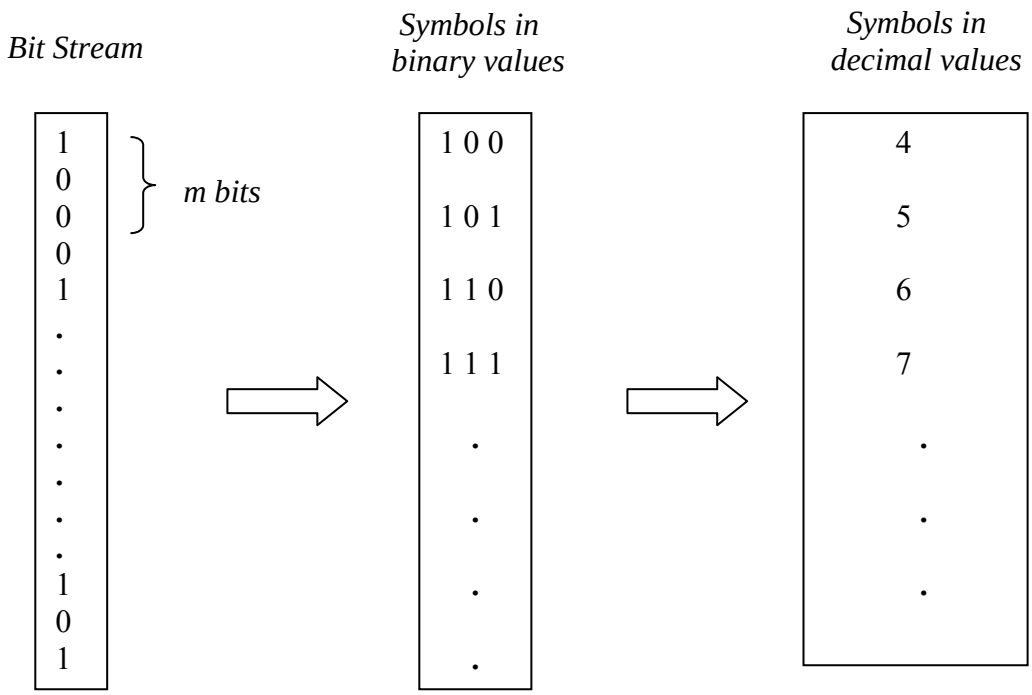



**Figure 5.3 Construction of information vector**

Then, we must create groups of $k$ symbols. Each group is called a "message" and is stored in a row of a table which is called message table.

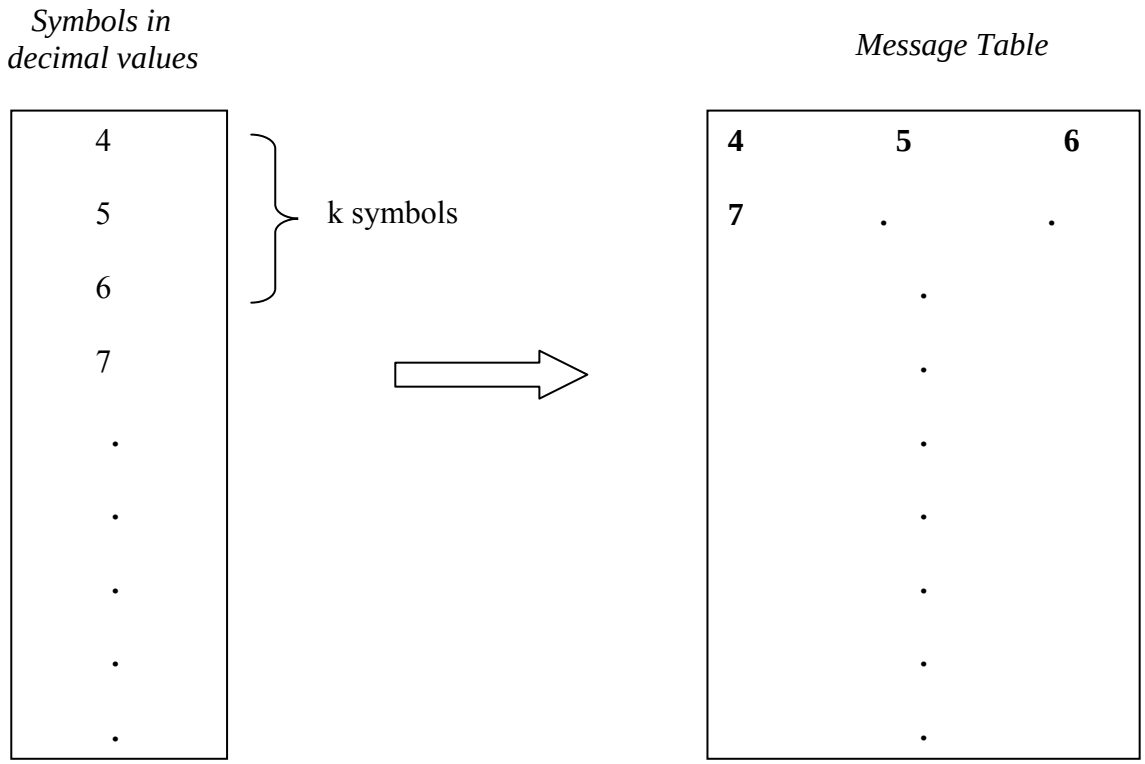



**Figure 5.4 Construction of message table**

Since we use a systematic method of encoding, each row of the message table contains the **k** coordinates $a_{n-k}, a_{n-k+1}, ..., a_{n-1}$ of the information vector $i(x)$.The result of the implementation of the Reed Solomon code on each row of the message table is also a table containing codewords in each row. The codewords, in binary form, are the output of the encoder.

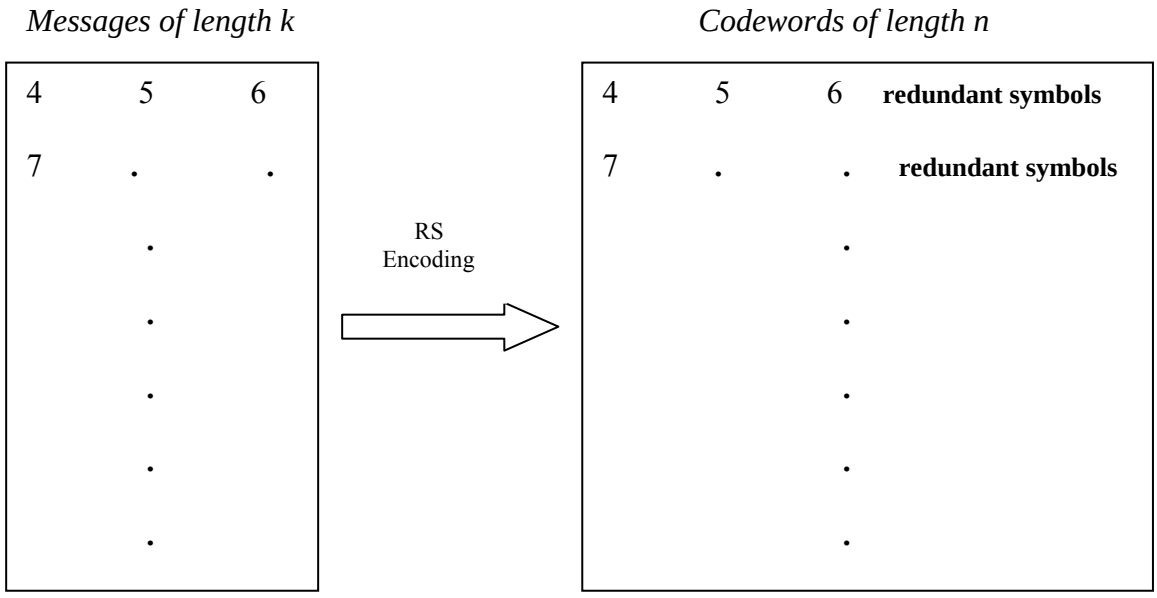



**Figure 5.5 Codeword table**

The decoder follows a similar, but inverse process in order to produce the decoded data from the codewords. Both encoder and decoder are familiar with the block length **n** and data length **k**. The decoder "translates" the codewords back to messages and then implements the Berlekamp-Massey algorithm in order to detect the error locations and then if it is possible to correct them. Reed Solomon codes are well suited for burst error correction. Burst errors are called those that occur in groups that cover hundreds or even thousands of bits. In our system burst errors are a common phenomenon since we use an FM channel for transmission and reception, where there are multiple sources of noise due to other FM transmissions in the area.

## 5.11 Why RS Codes Perform Well Against Burst Noise

Consider an ($n$, $k$) = (255, 247) R-S code, where each symbol is made up of $m$ = 8 bits (such symbols are typically referred to as *bytes*). Since $n$ - $k$ = 8, this code can correct any four symbol errors in a block of 255. For example, supposing there is a noise burst lasting for 25 bit durations and disturbing one block of data during transmission, as illustrated in Figure 5.

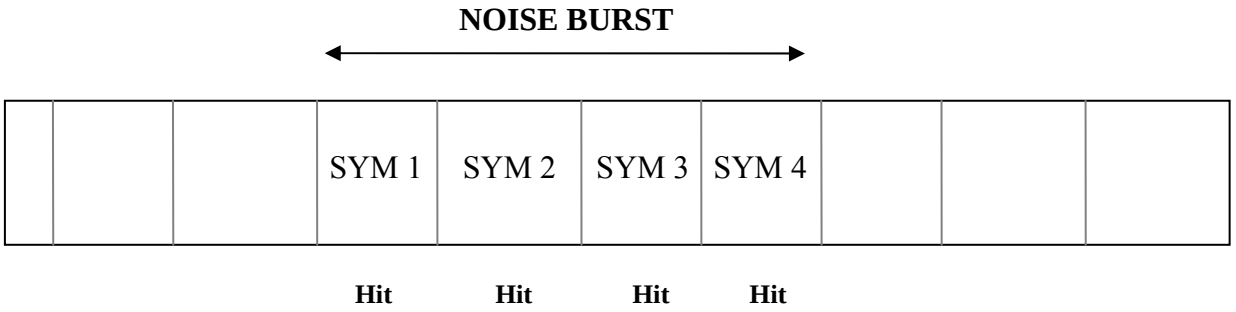



**Figure 5.6 Reed Solomon against burst noise**

In this case, we notice that a burst of noise that lasts for duration of 25 contiguous bits must disturb exactly four symbols. The RS decoder for the (255, 247) code will correct *any* four-symbol errors without regard to the type of damage suffered by the symbol. In other words, when a decoder corrects a byte, it replaces the incorrect byte with the correct one, whether the error was caused by one bit being corrupted or all eight bits being corrupted. Thus if a symbol is wrong, it might as well be wrong in all of its bit positions. This gives an RS code a tremendous burst-noise advantage over binary codes. In this example, if the 25 bit noise disturbance had occurred in a random way rather than as a contiguous burst, it should be clear that many more than four symbols would be affected (as many as 25 symbols might be disturbed). Of course, that would be beyond the error capability of the (255, 247) code.

## 5.12 Selection of RS length and dimension

The most basic criteria in order to select Reed Solomon's length **n** and dimension **k** are:

- the error correction capability of the code which we desire to be high
- the length of redundant symbols used which we desire to be small

Obviously, we must combine these two parameters in order to achieve a satisfying result. The error correction capability of a (n, k) Reed Solomon code is: $t = (n-k)/2$. We also know by definition that $n = 2^m - 1$, where $m$ is the number of bits per symbol. The value for $m$ in applications usually is between 3 and 8. Therefore, we know the possible block lengths. Since we know the block length **n** we can easily define the dimension of the code **k** so that the error correction capability **t** of the code becomes as high as possible. However, when **t** increases, the number of redundant symbols also increases. This results to a big amount of extra data to be transmitted, which we try to avoid. The amount of extra data is equal to $(n-k)$ symbols. Another metrics is the ratio $n/k$ (encoded data / data) which represents a measure of the code's redundancy. The code rate is defined as its inverse that is $k/n$.

The following tables contain some block lengths, data lengths, the corresponding error capability, the code rate and the code's redundancy metrics.

| **m=3** | **n** | **k** | **t** | **k/n** | **n/k** |
|---|---|---|---|---|---|
| | 7 | 3 | 2 | 0.42 | 2.33 |
| | 7 | 5 | 1 | 0.71 | 1.4 |

| **m=4** | **n** | **k** | **t** | **k/n** | **n/k** |
|---|---|---|---|---|---|
| | **15** | **5** | **5** | **0.33** | **3** |
| | 15 | 7 | 4 | 0.46 | 2.14 |
| | 15 | 9 | 3 | 0.60 | 1.66 |
| | 15 | 11 | 2 | 0.73 | 1.36 |
| | 15 | 13 | 1 | 0.86 | 1.15 |

| m=5 | n | k | t | k/n | n/k |
|---|---|---|---|---|---|
| | **31** | **11** | **10** | **0.35** | **2.81** |
| | 31 | 13 | 9 | 0.42 | 2.38 |
| | 31 | 15 | 8 | 0.48 | 2.06 |
| | 31 | 17 | 7 | 0.54 | 1.82 |
| | 31 | 19 | 6 | 0.61 | 1.63 |
| | **31** | **21** | **5** | **0.68** | **1.47** |

We can already observe that if for example we want to correct up to **5** symbol errors (**20** bit errors) using a RS(15,5) code the ratio **n/k** is equal to **3** , whereas using a RS(31,21) code (**25** bit errors) the ratio **n/k** is equal to **1.47** (marked with red color on the tables).Practically, this means that if we try to encode 10000bits using a RS(15,5) code the output would be 30000bits. On the contrary if we use a RS(31,21) code the output would be 14700bits.
We continue the same procedure.

| m=6 | n | k | t | k/n | n/k |
|---|---|---|---|---|---|
| | **63** | **33** | **15** | **0.52** | **1.90** |
| | 63 | 35 | 14 | 0.55 | 1.80 |
| | 63 | 37 | 13 | 0.58 | 1.70 |
| | 63 | 39 | 12 | 0.62 | 1.61 |
| | 63 | 41 | 11 | 0.65 | 1.53 |
| | **63** | **43** | **10** | **0.68** | **1.46** |

We can observe the same if we want **t = 10**. (marked with blue color on tables)

| m=7 | n | k | t | k/n | n/k |
|---|---|---|---|---|---|
| | 127 | 91 | 18 | 0.71 | 1.39 |
| | 127 | 93 | 17 | 0.73 | 1.36 |
| | 127 | 95 | 16 | 0.75 | 1.33 |
| | **127** | **97** | **15** | **0.76** | **1.30** |

It is obvious that as the number of bits per symbol increases, we can set a higher value for the error correction capability **t,** while the ratio **n/k** decreases and the code rate increases.

## 5.13 Reed Solomon performance as a function of size, redundancy and code rate

In general, the performance of a coded communication system can be measured by two factors:

- Its probability of bit error, also called bit error rate (BER), which is defined as the probability that a decoded information bit at the output of the decoder is in error.

- Its coding gain over an uncoded system that transmits information at the same rate. Coding gain is defined as the reduction in the $E_b / N_0$ required to achieve a specific error probability for a coded system compared to an uncoded system.

Although RS codes can be designed to have any redundancy, the complexity of a high-speed implementation increases with redundancy. Thus, the most attractive RS codes have high code rates (low redundancy). For Reed Solomon codes error probability is an exponentially decreasing function of block length $n$ and decoding complexity is proportional to a small power of the block length **[1]**. Moreover, a Reed Solomon code has a good performance when the noise duration represents a small percentage of the codeword. Hence, error correcting codes become more efficient (error performance improves) as the block size increases. This is clearly seen by the family of curves in figure 5.7 , where the rate of the code is held at a constant $7/8$ , while its block size increases from *n=31 symbols* (with $m = 5$ bits per symbol) to *n=256 symbols* (with $m = 8$ bits per symbol). Thus, the block size increases from *160 bits* to *2048 bits*. The performance curves are plotted for BPSK modulation over an AWGN channel.

| **Block length** ***n*** | **Data length** ***k*** | **Error correction capability** | **Code rate** | **Code gain for BER=$10^{-6}$** |
|---|---|---|---|---|
| 31 | 27 | 2 | 0.8730 | 2.96db |
| 63 | 55 | 4 | 0.8710 | 4.30db |
| 127 | 111 | 8 | 0.8740 | 5.51db |
| 127 | 111 | 8 | 0.8740 | 5.51db |
| 255 | 223 | 16 | 0.8745 | 6.51db |

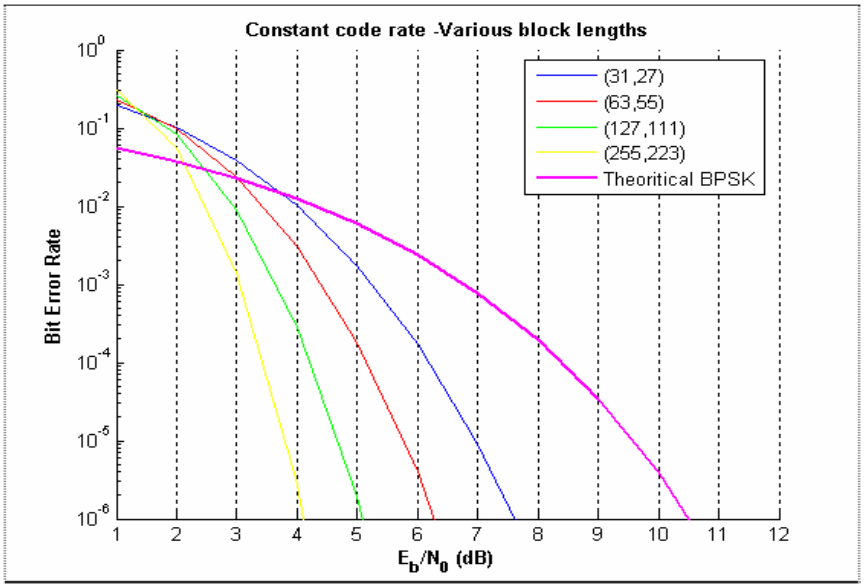



**Figure 5.7 Constant code rate-various block lengths**

As the redundancy of an RS code increases (lower code rate), its implementation grows in complexity since the number of symbols that need processing grows. However, the benefit of increased redundancy, just like the benefit of increased symbol size, is the improvement in bit error performance. This is made clear in figure 5.8 where the block length *n* is held at a constant *31*, while the number of data symbols decreases from *k=29 to k=21* ( redundancy increases from 2 to 10 symbols)

| **Block length *n*** | **Data length *k*** | **Error correction capability** | **Code rate** | **Code gain for BER=$10^{-6}$** |
|---|---|---|---|---|
| 31 | 29 | 1 | 0.935 | 1.66db |
| 31 | 27 | 2 | 0.871 | 3.01db |
| 31 | 25 | 3 | 0.806 | 3.78db |
| 31 | 23 | 4 | 0.741 | 4.41db |
| 31 | 21 | 5 | 0.677 | 4.91db |

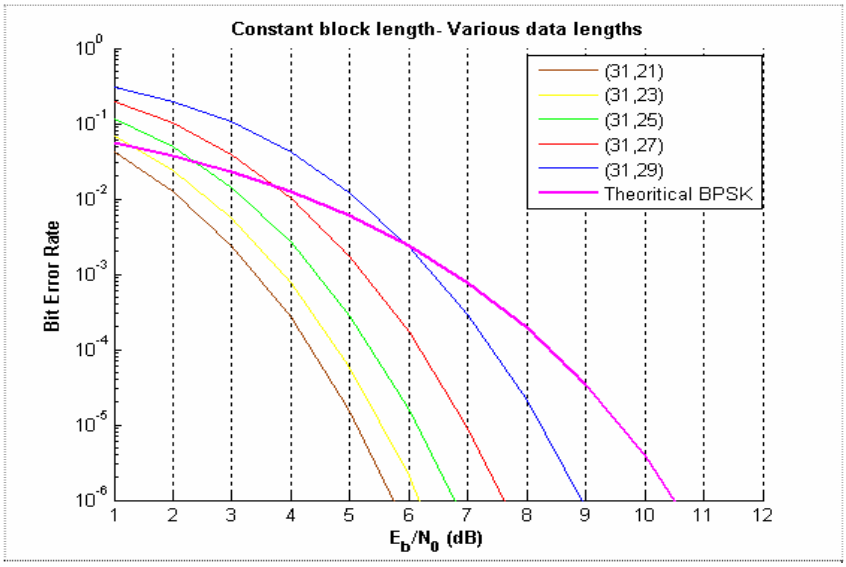



**Figure 5.8 Constant block length-various data lengths**

By studying figure 5.8 one may suggest that the improved error performance versus increased redundancy is a monotonic function that will continually provide system improvement even as the code rate approaches zero. However, this is not happening in a real-time communication system. As the rate of a code varies from minimum to maximum (0 to 1) we can observe the effects shown in figure 5.9. In the following table, the properties of all (31,k) codes are presented:

| **Block length *n*** | **Data length *k*** | **Code Rate** | **$E_b/N_o$ necessary for BER=$10^{-6}$** |
|---|---|---|---|
| 31 | 29 | 0.935 | 9.08 db |
| 31 | 27 | 0.871 | 7.84 db |
| 31 | 25 | 0.806 | 6.90 db |
| 31 | 23 | 0.741 | 6.25 db |
| 31 | 21 | 0.677 | 5.99 db |
| 31 | 19 | 0.612 | 5.52 db |
| 31 | 17 | 0.548 | 5.34 db |
| 31 | 15 | 0.483 | 5.42 db |
| 31 | 13 | 0.419 | 5.38 db |
| 31 | 11 | 0.354 | 5.40 db |
| 31 | 9 | 0.290 | 5.99 db |
| 31 | 7 | 0.225 | 6.18 db |
| 31 | 5 | 0.161 | 7.09 db |
| 31 | 3 | 0.096 | 8.57 db |

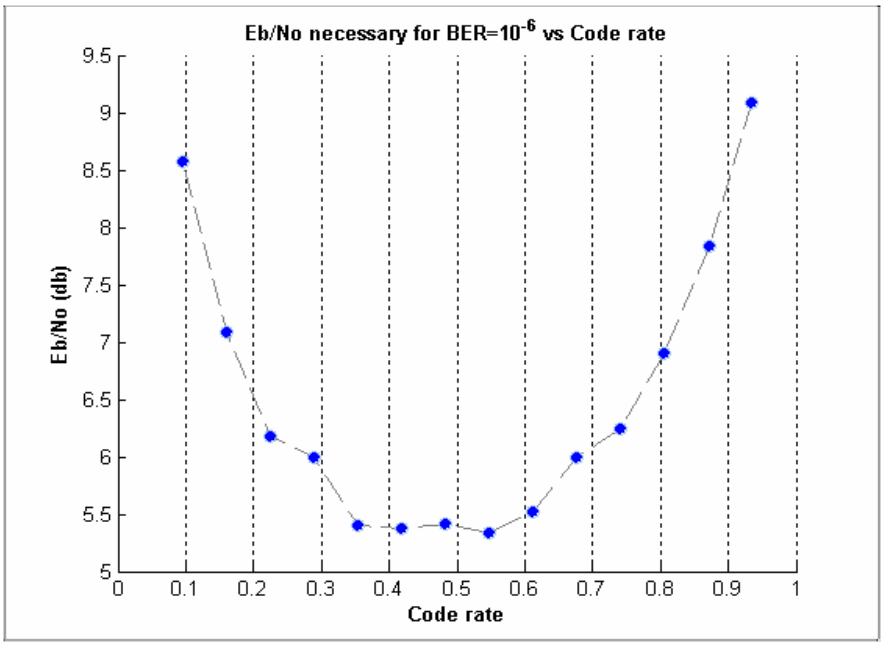



**Figure 5.9 Behavior of codes with different code rates**

In figure 5.9, we plotted the necessary SNR in db to achieve a certain bit error rate versus the code rate of each code. It is shown clearly that the optimum code that minimizes the required $E_b / N_0$ is *(31,17)* with code rate *0.548*. For a Gaussian channel, it seems that the optimum code rate is about *0.5* to *0.7*. If we were to plot error performance versus code rate for other block lengths different than 31, the curve would have the general "shape" as it does in figure 5.9.We observe that for both high and low code rates there is $E_b / N_0$ degradation. For high rates the degradation compared to the optimum rate can be explained in the following way. Codes with high code rate use a small number of redundant symbols and for this reason their error correction capability is low. As the code rate approaches unity (no coding), the system suffers worse error performance. On the other hand, the degradation at low code rates is due to the demodulator. Codes with low code rates use a large number of redundant symbols (high error correction capability). For this reason the demodulator has to process a larger amount of data which results to an increase of the bit error rate. *"For error-performance improvement due to coding, the decoder must provide enough error correction to more than compensate for the poor performance of the demodulator"* **[2]**. In this case, coding proves unable to improve system's performance.

## 5.14 Relation between code rate and bit rate

The main reason coders are used is to improve system's performance and achieve higher bit rates. However, by using a coder redundant bits are also transmitted. There is a relation between this redundancy, which can be expressed by the code rate, and the system's bit rate. A short proof of this relation follows:

If we assume that $xbits$ will be transmitted by using a modulation scheme of order $M_1$, then if we divide them in symbols it holds that: $\#symbols = xbits / \log_2 M_1$. Every symbol is divided into $\#symbols \cdot T$ samples. The sampling of our system is $F_s$ samples/sec. In order to transmit $xbits$ we need:

$$\frac{xbits \cdot T}{\log_2 M_1 \cdot F_s} \quad \text{sec}$$

Finally, the bit rate is given by:

$$R_1 = \frac{xbits}{\frac{xbits \cdot T}{\log_2 M_1 \cdot F_s}} = \frac{\log_2 M_1 \cdot F_s}{T} \ bits/\text{sec} \quad \textbf{(1)}$$

If the order of the modulation scheme remains the same and an encoder with code rate $k/n$ is used, it is proved that: $R' = (k/n) \cdot R_1$

In case we desire a higher bit rate we can either increase the order of the modulation scheme or decrease the symbol period. We choose to increase the order of the modulation scheme to $M_2$ and use an encoder with code rate $k/n$, in order to correct the errors that will occur. We assume again that we want to transmit $xbits$. Since an encoder is used the result is that $(n/k) \cdot xbits$ will be transmitted in: $\frac{(n/k) \cdot xbits \cdot T}{\log_2 M_2 \cdot F_s}$ sec

Therefore, $$Bit\_rate_2 = \frac{xbits}{\frac{(n/k) \cdot xbits \cdot T}{\log_2 M_2 \cdot F_s}} = \frac{(k/n) \cdot \log_2 M_2 \cdot F_s}{T} \quad bits/\text{sec} \quad \textbf{(2)}$$

Then we divide the relation **(2)** with **(1).** Since we desire that $Bit\_rate_2 > Bit\_rate_1$ the result of the division must be greater than unity. It will be equal to unity if $Bit\_rate_2 = Bit\_rate_1$

$$(k/n)\cdot\frac{\log_2 M_2}{\log_2 M_1}\geq 1 \qquad \textbf{(3)}$$

For example, if the code rate was $10/15$ and $M_1 = 4, M_2 = 8$ then:

$(10/15)\cdot(\log_2 8/\log_2 4) = 1$ . This means that we achieve the same bit rate.

By contrast if $M_1 = 4, M_2 = 16$ with the same code rate then, $(10/15)\cdot(\log_2 16/\log_2 4) = 1.33 > 1$. This means that we achieve a higher bit rate by a percentage of **33%**.

## 5.15 Conlusions

Reliability and a high bit rate are the two main factors that define which Reed Solomon code will be used. For a given bit rate, we must consider the result of relation **(3),** proved in the previous section. When we conclude to a value for the code rate, we must select a value for the block length and the data length. It is already shown that for a constant code rate, longer Reed Solomon's codes perform better than those with a short block length. However, if the code rate is close to unity, it is very possible that the usage of an encoder will deteriorate system's performance. Moreover, as mentioned already, decoding complexity is proportional to a *small* power of the block length and for this reason we are not restricted to use codes with small block lengths. Therefore, it is preferable to compromise with a code rate not very close to unity and select an average or long block. In many applications blocks of 255 symbols are used because in this case the symbol length is **8**, which corresponds to **1** byte.

## 5.16 Simulations of Reed Solomon coding over AWGN channel

In order to study the performance of Reed Solomon coding we run several Monte-Carlo simulations of various modulation schemes over an additive white Gaussian noise (AWGN) channel. Reed Solomon coding was tested for Phase Shift Keying (PSK), Quadrature Amplitude Modulation (QAM) and Pulse Position Modulation (PPM). The codes selected for the Monte Carlo simulations were **RS(31,21)** and **RS(127,97)** with the following properties:

| n | k | t | code rate |
|---|---|---|---|
| 31 | 21 | 5 | 0.68 |
| 127 | 97 | 15 | 0.76 |

We observe that the two codes have different block lengths, different error correction capability and different code rate. One could assume that RS(127,97) would have a better performance than RS(31,21) since its error correction capability is higher. However, this is not true because:

RS(31,21) is capable of correcting 5 symbols in each codeword. Percentage: **16.13%**
RS(127,97) is capable of correcting 15 symbols in each codeword. Percentage: **11.81%**

The simulations will show that RS(31, 21) will perform better than RS(127,97) in most cases or that both will not be capable of improving the system's performance and in fact they will cause degradation. If we wanted to achieve a better performance with similar code rate using block length of 127 symbols, we would select RS(127,85) with code rate 0.67.

RS(127,85) is capable of correcting 21 symbols in each codeword. Percentage: **16.54%**

## 5.17 Performance of RS coding using Phase Shift Keying

We performed Monte-Carlo simulations for 4-8-16-32 PSK. For each simulation $10^5$ bits were used as input. The two coders were tested for 100 values of noise variance. The minimum value was 0.01 and the maximum value was 1 . The bit error performance of **RS(31,21)** and **RS(127,97)** over an AWGN channel using PSK is depicted in the following figures. For comparison, the bit error performance of an uncoded PSK system is also included.

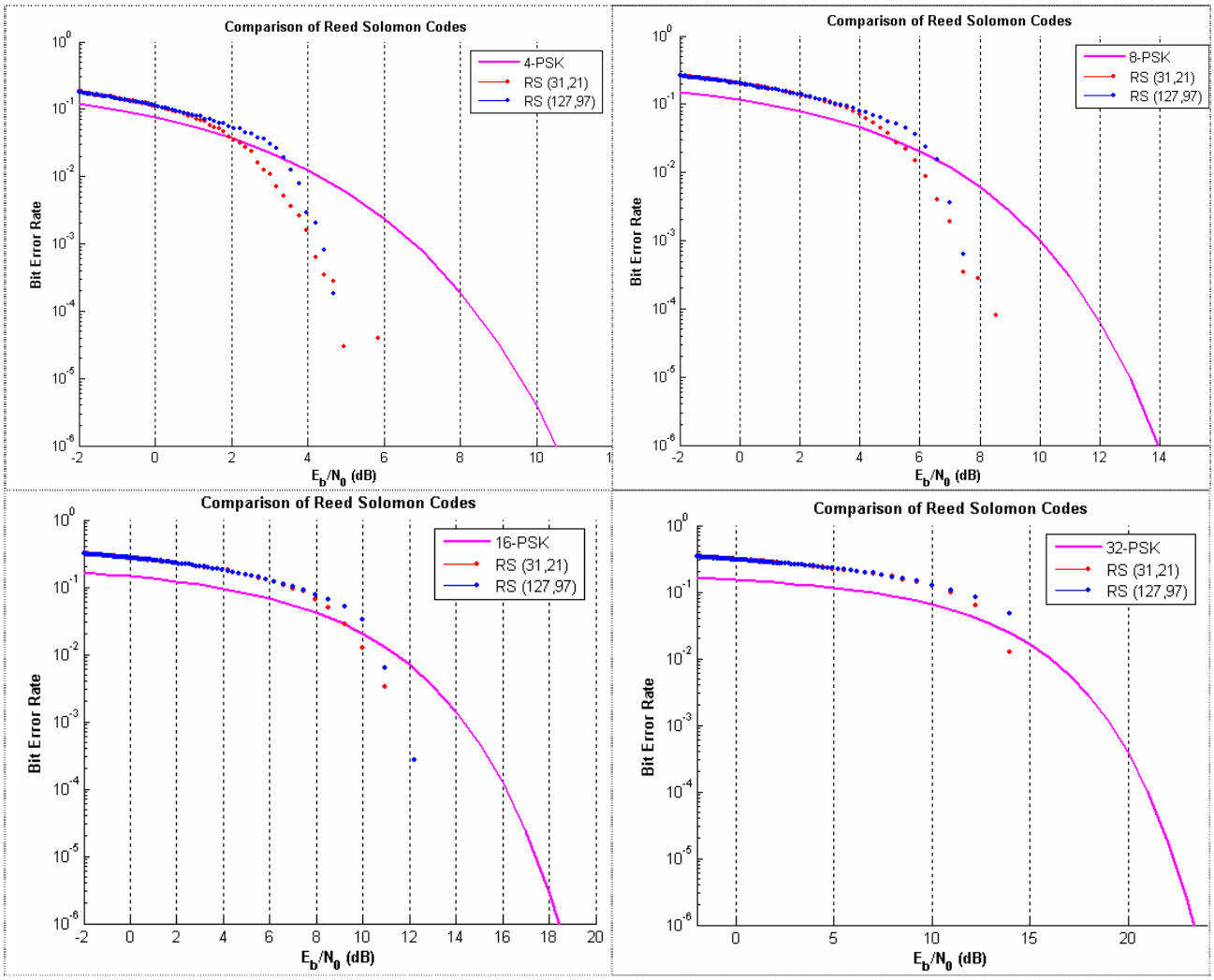



**Figure 5.10 PSK simulations**

In general, we see that the coded system provides a lower bit error probability than the uncoded system for the same SNR when the SNR is above a certain threshold. This threshold is called **coding threshold**. Below the coding threshold the codes lose their effectiveness, the coding gain becomes negative and actually they make the situation worse. The existence of the coding threshold is obvious in the figures for 4-8 PSK.

**4-PSK**

| | RS(31,21) | RS (127,97) |
|---|---|---|
| Coding Threshold | 2.4db | 3.6db |

| | BER RS(31,21) | BER RS(127,97) | Uncoded system |
|---|---|---|---|
| SNR=3.98db | $10^{-2.8}$ | $10^{-2.5}$ | $10^{-1.9}$ |

For a $BER \approx 10^{-4}$ RS(31,21) has a coding gain of almost 3.5db over the uncoded PSK.

**8-PSK**

| | RS(31,21) | RS (127,97) |
|---|---|---|
| Coding Threshold | 5.52db | 6.57db |

| | BER RS(31,21) | BER RS(127,97) | Uncoded system |
|---|---|---|---|
| SNR=7.45db | $10^{-3.47}$ | $10^{-3.19}$ | $10^{-2.05}$ |

For a $BER \approx 10^{-4}$ RS(31,21) has a coding gain of almost 3.2db over the uncoded PSK.

**16-PSK**

| | RS(31,21) | RS (127,97) |
|---|---|---|
| Coding Threshold | 9.2db | 10db |

| | BER RS(31,21) | BER RS(127,97) | Uncoded system |
|---|---|---|---|
| SNR=10.97db | $10^{-2.49}$ | $10^{-2.2}$ | $10^{-1.89}$ |

For a $BER \approx 10^{-3.6}$ RS(127,97) has a coding gain of 3.2db over the uncoded PSK.

**32-PSK**

For 32-PSK the coding threshold is very high and for that reason the coding gain for both codes remain negative.

For 4-8 PSK, **RS(31,21)** performs better for a greater range of $\frac{E_b}{N_0}$ than **RS(127,97).** However the performance of both is poor for 16-32 PSK.

## 5.18 Performance of RS coding using Quadrature Amplitude Modulation

We performed Monte-Carlo simulations for 4-8-16-32 QAM. The simulations settings are the same as mentioned above in PSK. The theoretical bit error performance for an uncoded QAM system is also presented for comparison.

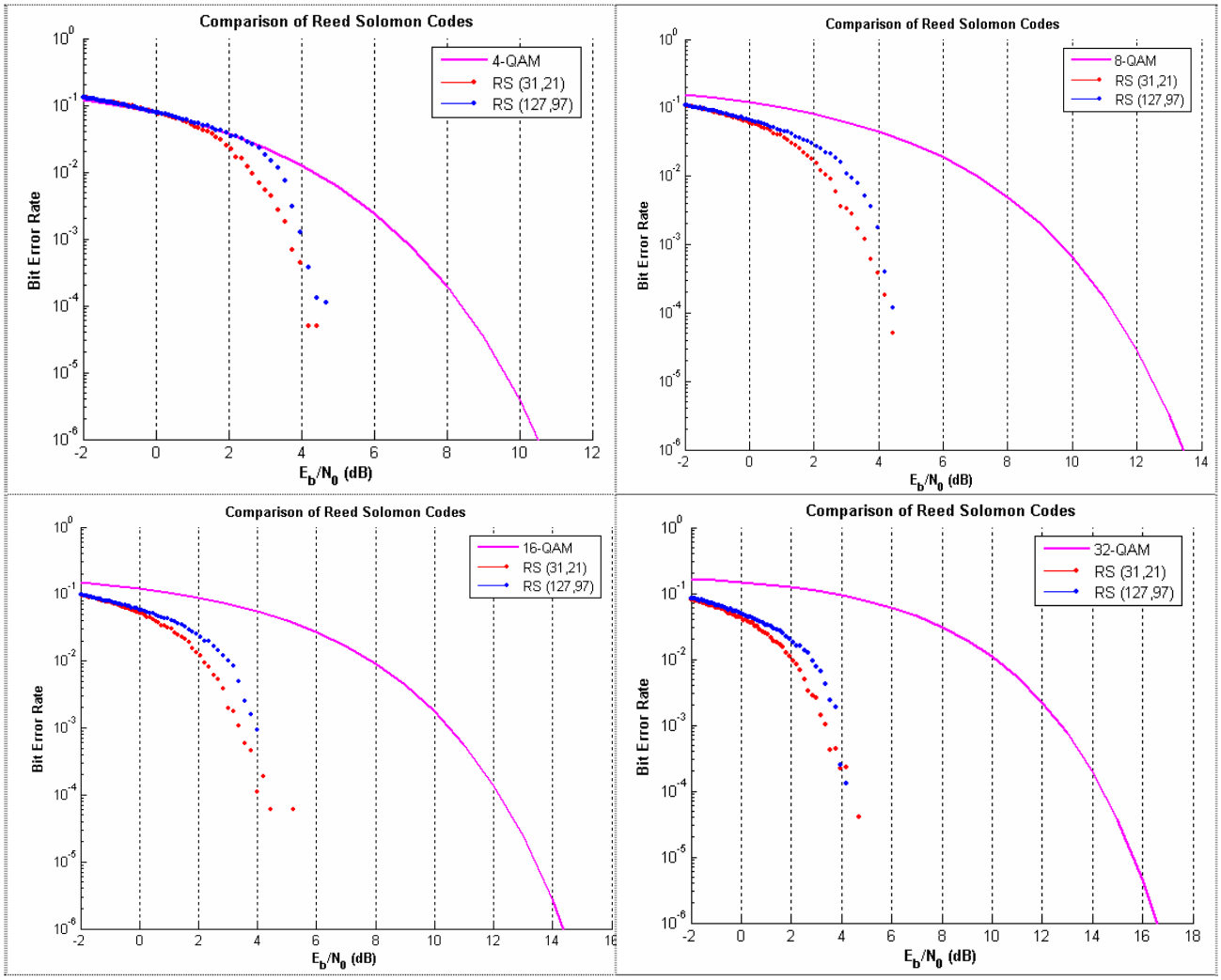



**Figure 5.11 QAM simulations**

It is obvious that for all orders of QAM, RS(31,21) code has a better performance than RS(127,97). We also notice that by using the encoders the necessary SNR for all orders of QAM is always lower than 6db.

**4-QAM**

| | RS(31,21) | RS (127,97) |
|---|---|---|
| Coding Threshold | 1.3db | 3db |

| | BER RS(31,21) | BER RS(127,97) | Uncoded system |
|---|---|---|---|
| SNR=3.98db | $10^{-3.35}$ | $10^{-2.9}$ | $10^{-1.9}$ |

For a $BER \approx 10^{-4.3}$ RS(31,21) has a coding gain of 4.3db over the uncoded QAM.

**8-QAM**

| | RS(31,21) | RS (127,97) |
|---|---|---|
| Coding Threshold | - | - |

| | BER RS(31,21) | BER RS(127,97) | Uncoded system |
|---|---|---|---|
| SNR=3.98db | $10^{-3.42}$ | $10^{-2.75}$ | $10^{-1.35}$ |

For a $BER \approx 10^{-4.3}$ RS(31,21) has a coding gain almost 7.25db over the uncoded QAM.

**16-QAM**

| | RS(31,21) | RS (127,97) |
|---|---|---|
| Coding Threshold | - | - |

| | BER RS(31,21) | BER RS(127,97) | Uncoded system |
|---|---|---|---|
| SNR=3.98db | $10^{-3.96}$ | $10^{-3.02}$ | $10^{-1.2}$ |

For a $BER \approx 10^{-4.3}$ RS(31,21) has a coding gain of 7.3db over the uncoded QAM.

**32-QAM**

| | RS(31,21) | RS (127,97) |
|---|---|---|
| Coding Threshold | - | - |

| | BER RS(31,21) | BER RS(127,97) | Uncoded system |
|---|---|---|---|
| SNR=4.2db | $10^{-3.64}$ | $10^{-3.89}$ | $10^{-1.04}$ |

For a $BER \approx 10^{-4.3}$ RS(31,21) has a coding gain 10.2db over the uncoded QAM.

## 5.19 Performance of RS coding using Pulse Position Modulation

We performed Monte-Carlo simulations for 4-8-16-32 PPM. The implementation of PPM, especially for our system, is explained in chapter 7. The symbol period remains the same for all cases, $T = 64$. The results are presented in the following figures.

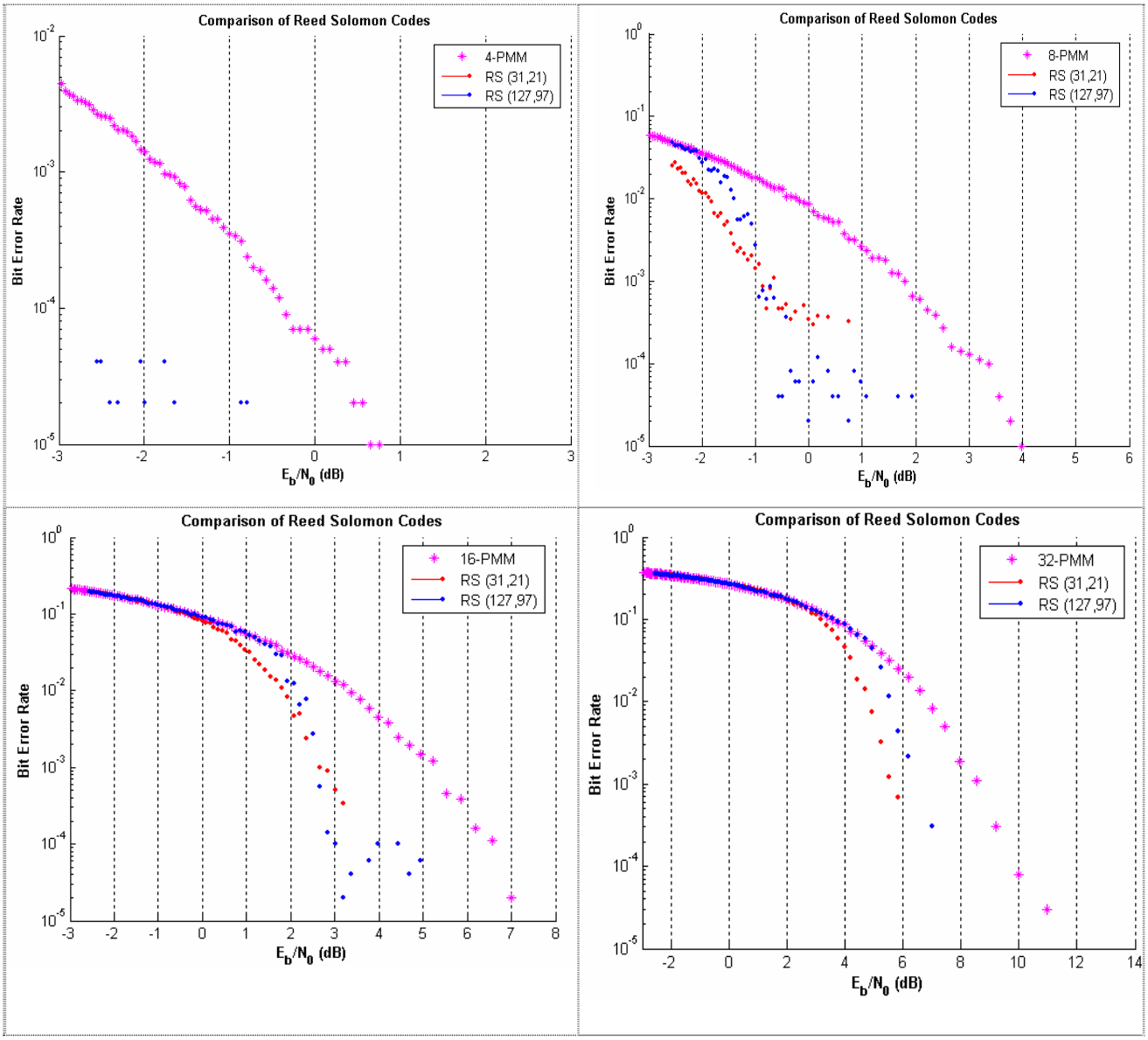



**Figure 5.12 PPM simulations**

RS(31,21) code had a better performance than RS(127,97) for 4-PPM and 32-PPM. In the cases, of 8-PPM and 16-PPM, it is not clear which code performs better. However, it is clear that RS(31,21) has a lower coding threshold.

**4-PPM**

In case of 4-PPM, RS(31,21) code managed to detect and correct all errors that occurred. For this reason, there are no red dots on the first figure. RS(127,97) code also corrected a large number of errors but not all of them.

**8-PPM**

| | RS(31,21) | RS (127,97) |
|---|---|---|
| Coding Threshold | - | -2db |

For a $BER \approx 10^{-4}$ RS(127,97) has a coding gain of about 3db over the uncoded PPM.

**16-PPM**

| | RS(31,21) | RS (127,97) |
|---|---|---|
| Coding Threshold | 0.26db | 1.8db |

| | BER RS(31,21) | BER RS(127,97) | Uncoded syster |
|---|---|---|---|
| SNR=2.67db | $10^{-3}$ | $10^{-3.25}$ | $10^{-1.74}$ |

For a $BER \approx 10^{-4}$ RS(127,97) has a coding gain of 2.5db over the uncoded PPM.

**32-PPM**

| | RS(31,21) | RS (127,97) |
|---|---|---|
| Coding Threshold | 3db | 5db |

This is only case where it is obvious that RS(31,21) code has a greater code gain than RS(127,97) code.

.

## 5.20 Simulations' Conclusions

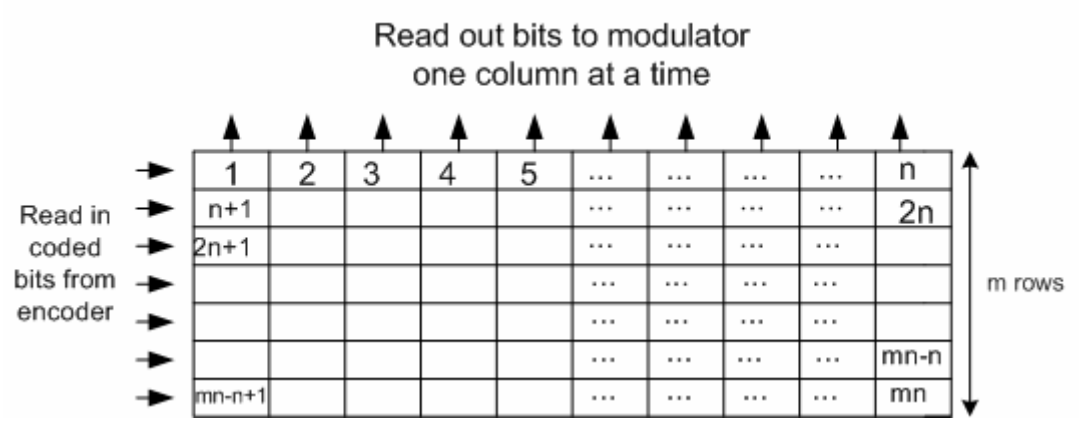
- The decrease of SNR for a given BER was greater for the code with the shorter block length, even if its error correction capability was lower than that of the code with long block length
- Reed Solomon coding apparently works better with QAM
- In PSK for high values of noise variance Reed Solomon coding had negative coding gain, so coding actually deteriorate the system's performance

# 6 Interleaving

An interleaver permutes symbols according to a mapping. A corresponding deinterleaver uses the inverse mapping to restore the original sequence of symbols. Interleaving and deinterleaving can be useful for reducing burst errors in a communication system. There are a number of different ways in which interleaving can be performed. The simplest interleaving method is termed block interleaving.

## 6.1 Block interleaving

Block interleaving is a common technique used with block codes in wireless systems. The advantage of interleaving is that a burst error that affects a sequence of bits is spread out over a number of separate blocks at the receiver so that error correction is possible. Interleaving is accomplished by reading and writing data from memory in different orders. In this case the data to be transmitted are stored in a rectangular array in which each row consists of *n* bits equal to the block size. Data are then read out one column at a time. The result is that the *k* data bits and their corresponding $(n-k)$ check bits , which form a single *n* bit block , are spread out and interspersed with bits from other blocks. At the receiver the data are deinterleaved to recover the original order. If, during transmission, a burst of noise affects a consecutive sequence of bits, those bits belong to different blocks and hence only a fraction of the bits in error need to be corrected by any one set of check bits. In other words, a burst of length l=*mb* is broken up into *m* bursts of length *b.* Therefore, in case we have an (n,k) code than can correct all combinations of *t* or fewer errors, where $t = \lfloor (n-k)/2 \rfloor$. If we use an interleaver of degree *m*, then the result is an *(mn, mk)* code that can correct burst errors of up to *mt* bits. An example of block interleaving follows: **[10, page 231]**




*Note: The numbers in the matrix indicate the order in which bits are read in.Interleaver output sequence: 1,n+1,2n+1,…*

**Figure 6.1 Block Interleaver**

Constructing a block interleaver is quite simple. The bits are first converted to symbols and grouped in order to form the codeword table. Then, the elements of each column of the table are first converted back to bits and finally read out to the modulator.

## 6.1 Random interleaving

Random interleaving is a technique where symbols are reordered using random permutation before the modulation stage. In order to construct a random interleaver we used Matlab's function *rand ('state')*. This function generates arrays of random numbers whose elements are uniformly distributed in the interval (0, 1). The state parameter initializes the random number generator that the function uses to determine the permutation. Since the value of *'state'* is known to the receiver unit, we are in position to invert the random permutation which has taken place in the transmitter unit. The value of variable *'state'* is defined by the user of our system and different states produce different permutations.

# 7 Pulse Position Modulation (PPM)

## 7.1 Multidimensional Signal Waveforms

Besides the two dimensional QAM and PSK modulation methods, we also implemented on our system a multidimensional modulation method and more specifically the Pulse Position Modulation (PPM). We begin by designing a set of $M = 2^k$ signal waveforms, which are mutually orthogonal. The dimension $N$ is equal to the number of waveforms $M$ . The special characteristic about these waveforms is that their orthogonality is due to the fact that there is no overlapping in the domain of time.

The M waveforms can be expressed as:

$$s_m(t) = A \cdot g_T(t-(m-1)\cdot T/M), \quad \begin{matrix} m = 1,2...,M \\ (m-1)\cdot T/M \le t \le m\cdot(T/M) \end{matrix}$$

where $g_T(t)$ is a pulse of duration $T/M$ and arbitrary shape. We also note that the whole set of our waveforms has the same energy since they have the same amplitude $A$. The calculation of this energy follows:

$$\int_0^T s_m^2(t)dt = A^2 \int_{(m-1)T/M}^{mT/M} g_T^2(t-(m-1)T/M)dt$$

$$= A^2 \int_0^{T/M} g_T^2(t)dt$$

$$= A^2 \cdot E_g = E_s$$

In order to represent our waveforms geometrically we must construct the orthonormal basis functions using the Gram-Schmidt procedure. In our case we can define the M basis functions as:

$$\psi_m(t) = \begin{cases} \frac{1}{\sqrt{E_g}} g_T(t-(m-1)T/M), & (m-1)T/M \le t \le mT/M \\ 0 \quad , & otherwise \end{cases}$$

for $m = 1, 2, ..., M$

Since we have constructed the set of N of orthonormal waveforms $\psi_m(t)$ we are in position to express our M signals $s_m(t)$ as linear combinations of $\psi_m(t)$. We can therefore write:

$$s_m(t) = \sum_{n=1}^{N} s_{mn}\psi_n(t), \quad m = 1, 2..., M$$

where

$$s_{mn} = \int_{-\infty}^{+\infty} s_m(t)\psi_n(t)dt$$

The notation and parts of theory were borrowed from **[4, pages 402-406]**.

In our case, the M-ary PPM waveforms are represented by the M-dimension vectors:

$$s_1 = (\sqrt{E_s}, 0, 0, ..., 0)$$

$$s_2 = (0, \sqrt{E_s}, 0, ..., 0)$$

$$\vdots\ \vdots$$

$$s_M = (0, 0, 0, ..., \sqrt{E_s})$$

These vectors are orthogonal and the distance between any pair of vectors is equal to:

$$d_{mn} = \sqrt{\|s_m - s_n\|^2} = \sqrt{2E_s}, \quad for\ \ all\ \ m \neq n$$

For example, if we desire three orthogonal signals we follow the procedure below. Since we want to construct three orthogonal signals we also have three dimensions $M = N = 3$. We begin by constructing three orthonormal basis functions. We define pulse $g_T(t)$=1 with duration T/3.

$m = 1, 2, 3$

$$\text{For m=1} \quad \Rightarrow \psi_1(t) = \begin{cases} \frac{1}{\sqrt{Eg}} g_T(t) & 0 \le t \le T/3 \\ 0 & otherwise \end{cases}$$

$$\text{For m=2} \quad \Rightarrow \psi_2(t) = \begin{cases} \frac{1}{\sqrt{Eg}} g_T(t - T/3) & T/3 \le t \le 2T/3 \\ 0 & otherwise \end{cases}$$

$$\text{For m=3} \quad \Rightarrow \psi_3(t) = \begin{cases} \dfrac{1}{\sqrt{Eg}} g_T(t-2T/3) & 2T/3 \le t \le 3T/3 \\ 0 & \textit{otherwise} \end{cases}$$

We continue by multiplying the M-dimensional vectors with the basis functions. In our example the M-dimensional vectors are:

$$s_1 = (\sqrt{E_s}, 0, 0)$$

$$s_2 = (0, \sqrt{E_s}, 0)$$

$$s_3 = (0, 0, \sqrt{E_s})$$

$$\begin{aligned} s_1(t) &= s_{11} \cdot \psi_1(t) + s_{12} \cdot \psi_2(t) + s_{13} \cdot \psi_3(t) \\ &= s_{11} \cdot \psi_1(t) \\ &= \sqrt{E_s} \cdot \psi_1(t) \\ &= \sqrt{E_s} \cdot \frac{1}{\sqrt{E_g}} g_T(t) \\ &= A \cdot g_T(t) \end{aligned}$$

In the same way we calculate:

$$s_2(t) = A \cdot g_T(t - T/3)$$

$$s_3(t) = A \cdot g_T(t - 2T/3)$$

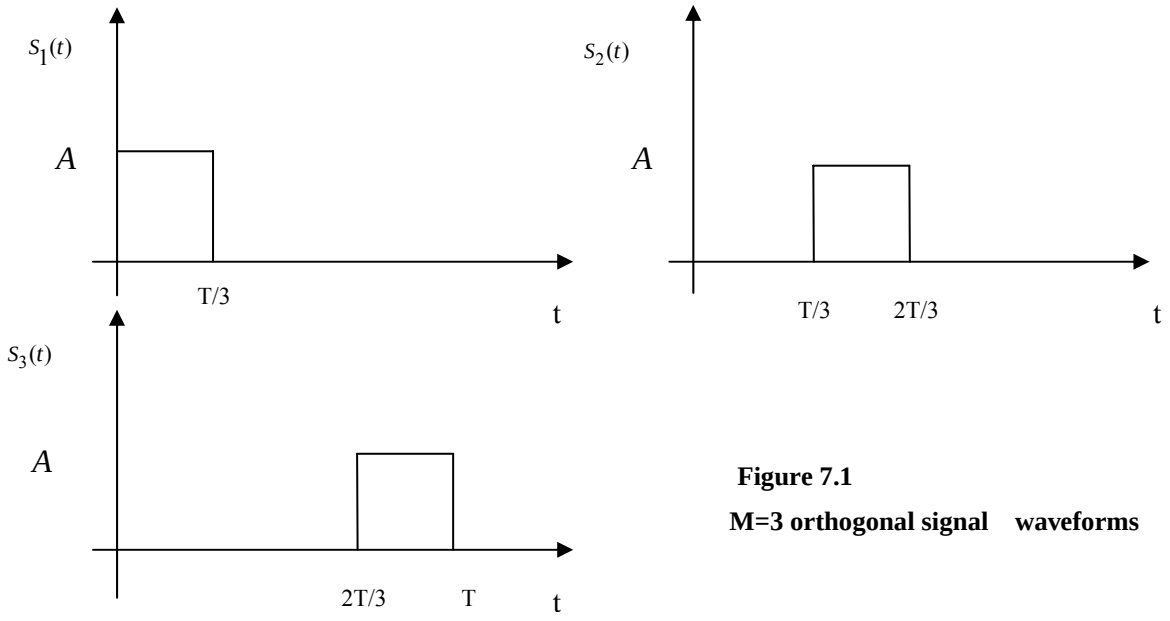



**Figure 7.1**
**M=3 orthogonal signal waveforms**

## 7.2 Modulator

The pulse position modulator has been implemented in a module called ppm_modulator. The bits coming out of the encoder, if we use one, are the input for the ppm_modulator. Otherwise the input of the ppm_modulator is the bits of the file we will transmit. The user defines the order $M$ of PPM and the symbol period $T$. The bit rate of our system depends exclusively on these two factors if a coder is not used.

$$Bit\_rate = \frac{\#bits}{time} \quad \text{(bits/sec)}$$

Since the order of PPM is M the input bits are divided into symbols according to the relation:

$$\#symbols = \frac{\#bits}{\log_2 M}$$

Moreover, every symbol has a symbol period T. Therefore, every symbol is divided into $\#symbols \cdot T$ samples. The sampling rate of our system by default is $F_s$. In order to transmit #bits we need:

$$\frac{\#bits \cdot T}{\log_2 M \cdot F_s} \quad \text{sec}$$

Finally, the $Bit\_rate$ can now easily be calculated:

$$Bit\_rate = \frac{\#bits}{\frac{\#bits \cdot T}{\log_2 M \cdot F_s}}$$

$$Bit\_rate = \frac{\log_2 M \cdot F_s}{T} \qquad bits/\text{sec}$$

We notice that by increasing the order *M* of PPM or by decreasing the symbol period *T* we can achieve higher bit rates. Certainly, upper and lower bounds exist for *M* and *T* respectively for a reliable transmission.

The ppm_modulator module first constructs the *M* orthonormal basis functions. The arbitrary pulse $g_T(t)$ is set to be equal to 1 for a period *T/M*. The symbol energy $E_S$ is defined to be equal to the energy of pulse $g_T(t)$ $E_g$. In this way the amplitude of the final signal waveform $s_m(t)$ is equal to $\sqrt{E_S} \cdot \frac{1}{\sqrt{E_g}} = 1$. We forced the amplitude of $s_m(t)$ to be 1 because Matlab's functions '*wavplay*' and '*wavrecord*', which we use to transmit and receive, work with amplitudes in the range $[-1\ ,\ 1]$. Any higher or lower amplitudes are cut off. The bit stream that will be modulated is divided into symbols. Each symbol is mapped to a unique waveform from the set of waveforms $s_m(t)$. The transform from symbols to waveforms, or to be exact the transform from symbols to samples, is the final stage of the modulation. The samples are ready to be transmitted by the '*wavplay*' function.

For example, the following waveform shows the waveforms for 4-PPM with symbol period $T = 8$. The correspondent symbols of each waveform are presented on the top of the figure.

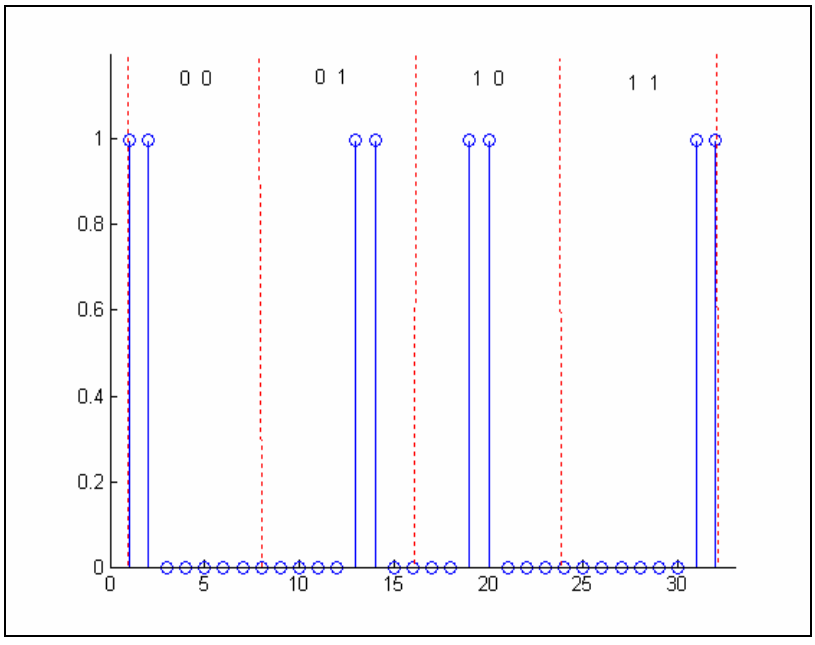



**Figure 7.2 4-PPM, T=8**

## 7.3 Demodulator

The demodulation stage is divided in two modules:

- the ppm_correlators module
- the ppm_detector module

### 7.3.1 Correlators

In the ppm_correlators module the same orthonormal basis functions as in the ppm_modulator module are constructed. By multiplying the received signal with the basis functions, we calculate the vector $s_m$ which represents the $M$ dimensions. The value of vector $s_m$ in one of the $M$ dimensions is equal to $\sqrt{E_S}$, whereas in all other dimensions is zero. The vector $s_m$ is the output of ppm_correlators module.

### 7.3.2 Detector

The detector used is called 'minimum distance detector'. The input of the ppm_detector module is the output of the ppm_correlator module, which is vector $s_m$. This vector is used in order to detect the symbol received. This is done by calculating the Euclidean distance between vector $s_m$ and each vector of the set:

$$s_1 = (\sqrt{E_s}, 0, 0, ..., 0)$$

$$s_2 = (0, \sqrt{E_s}, 0, ..., 0)$$

$$\vdots \quad \vdots$$

$$s_M = (0, 0, 0, ..., \sqrt{E_s})$$

The vector from the set, whose distance from $s_m$ is minimum, is selected. According to this vector we retrieve the symbol transmitted. Since the symbols are known, it is easy to convert them back to a bit stream.

# 8 System's Evaluation

## 8.1 General

In this chapter, the performance of the real time digital communication system will be analyzed thoroughly. The evaluation that follows is based on a large number of trial transmissions conducted by the members of the team. All experiments were repeated several times using the same computer systems in order to draw safe conclusions. The main objective, which was the connection of two computer systems via a wireless link, is achieved. However, new objectives arise now concerning the performance of the digital communication system.

## 8.2 New objectives

It is necessary for the system to combine the following characteristics that nowadays all digital communication systems do:

- reliability
- as low as possible bit error rate
- as high as possible transmission and transfer rates

Reliability is ensured through the use of the Automatic Repeat Request (ARQ) scheme. As explained in the chapter1, a "Stop and Wait" ARQ scheme was implemented. Unfortunately, the loss in time, due to the delay for the acknowledgement of each packet, is great. However, it is certain that every packet will reach its destination containing no errors. The delays caused by ARQ may be avoided, if all bits arrive correct at the receiver. For this reason, we would prefer the probability of bit error, or in other words the bit error rate, to be as low as possible and in ideal conditions equal to zero. Finally, besides reliability, speed is also an important issue.

The definitions of transmission rate and transfer rate are given. Transmission rate is defined as the rate with which bits are transmitted. It is equal to the following ratio:

$$Transmission\_rate = \frac{\#bits}{time\_of\_transmission}$$

For our system, as already explained in *chapter7* it holds that:

$$Transmission\_rate = \frac{\log_2 M \cdot F_S}{T} \qquad bits/\sec,$$

where $M$ is the order of the modulation scheme, $F_S$ the sampling frequency of the audiocard and $T$ is the symbol period. We observe that transmission rate depends on the modulation scheme selected (software) and also on the audiocard's capacity of sampling. (hardware)

On the other hand, transfer rate is defined as the rate with which bits are transferred to the other computer system and create a file. The time, from the moment the wireless link is established and the "handshake" takes place, until the moment the transmitter informs the receiver that there is no other packet to send and the link closes, is defined as the transfer time. This means that transmission time and processing time of each packet are added to form the denominator of the ratio.

$$Tranfer\_rate = \frac{\#bits}{(transmission\_time + processing\_time)} \qquad bits/sec$$

## 8.3 Hardware and software constraints

The new standards are met only if both hardware and software cooperate properly. Certain features of hardware and software are responsible for the deterioration of the performance of our system. More specifically, as far as hardware is concerned, in order to succeed high transfer rates the need of a high power process computer system is necessary. Moreover, in order to succeed high transmission rates, the sampling frequency the audiocard supports must also be

high. As far as software is concerned, three major constraints appear. The first is already mentioned in the introductory chapter and concerns the fact that Matlab does not support multitasking. Thus, it is difficult to implement a distributive system. The second constraint concerns the communication using the local network through Matlab. The communication with the UDP objects is not reliable, since often these objects are lost and the retransmission is inevitable. Finally, although the modules of all stages are programmed and measured to work as fast as possible, small delays still appear.

## 8.4 Evaluation plan and metrics

In the following sections, there is a detailed evaluation of the system's performance. The evaluation and several conclusions are based on two metrics: the transfer rate and the bit error rate. Both metrics are calculated for all experiments including open and closed loop links. The results of the experiments are grouped according to the modulation scheme used each time. Finally, we underline that issues such as Reed Solomon encoding, interleaving and Pulse Position Modulation scheme, which are included in the particular thesis, are processed with additional detail.

## 8.5 General presentation

We experimented in transmitting a file of size 108Kbytes. The selection of the file size was not random. The goal was to succeed a bit error rate of order $10^{-6}$ in every file transfer. Such a bit error rate is satisfying. In this case, the need for retransmission of the same packet (ARQ operation) will be rare. As far the sampling frequency Fs is concerned, its value depends on the specific audio hardware installed. Typical values supported by most sound cards are *8.000, 11025, 22050,* and *44100 Hz*. The experiments were performed using as sampling frequency *44100samples/sec* and *88200samples/sec,* since the audiocard available could support it. It is expected that transmission and transfer rates will be higher using *88200samples/sec.*The length of the training sequence used is 500bits. In the following table, all parameters of all stages of our system are shown. These parameters were combined during the experiments in order to understand the channel behavior and of course achieve the objectives set.

| | | | | |
|---|---|---|---|---|
| **Encoders/Decoders** | Convolutional | Reed Solomon | | |
| | *type of encoder* | *message length* | | |
| | *method* | *codeword length* | | |
| | | | | |
| **Interleavers** | *block interleaver* | *random interleaver* | | |
| | | | | |
| **Modulation Schemes** | *PSK* | *QAM* | *PPM* | *symbol period* |
| | | | | |
| **Equalizers** | LMS | RLS | CMA | Viterbi Equalizer |
| | *# taps* | *# taps* | *# taps* | *# taps* |
| | *LMS step* | *RLS initialization* | *CMA step* | *depth* |
| | | *RLS forget factor* | | |
| | | | | |
| LS Amp/Phase Recovery | *On/Off* | | | |
| CRC | *On/Off* | | | |
| Fs | *44100,88200samples/sec* | | | |

## 8.6 Definition of packet size

Before beginning the experiments, it was essential to define the size of the packet in bits. Several trial transmissions with different packet sizes and different file sizes were made. We already knew that the functions of Matlab for playback and recording begin their operations with random delays. This complicates a lot our "Stop and Wait" system. For this reason and to be sure that each packet is completely recorded, we were forced to record for an extra amount of time, 0.5 sec, for each packet. This means that for every packet 0.5 sec are lost and added to our transfer time. After taking under consideration this fact, we concluded that as the size of the file increases the packet size should also increase in order to avoid sending a large number of packets. If the number of packets for transmission is large $(\#number\_of\_packets \cdot 0.5)$ sec of delay are inevitable. For this reason, it is preferable to use large packets in order to avoid these delays. This is clearly shown in the following figure, where size of packet vs transfer time is plotted. Three different packet sizes were tested (15000, 50000, 100000 bits) for three different file sizes (880848, 143952, 44288 bits) and we measured the corresponding transfer times. The modulation scheme was 4PSK, the sampling frequency 44100samples/sec and CRC was off.

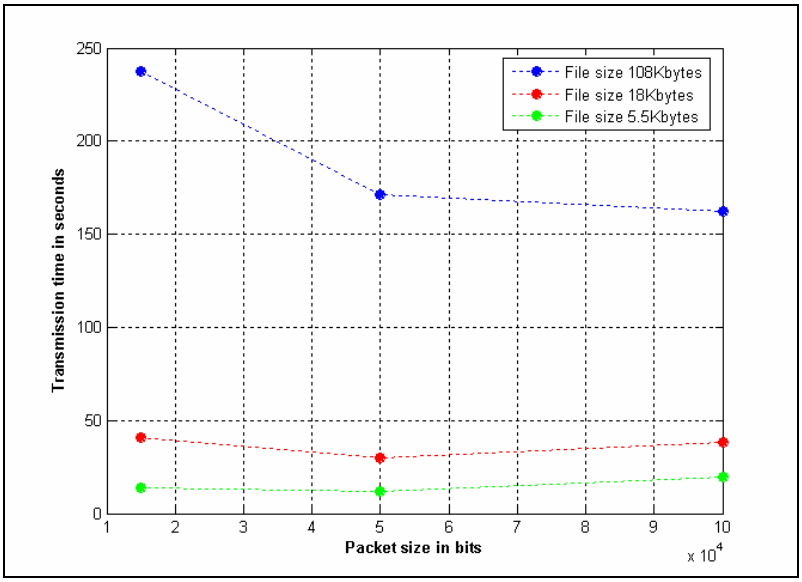



**Figure 8.1 Packet size simulation**

For the large size file (blue color), as the packet size increases the transfer time diminishes. For the medium size file (red color), the minimum transfer time is succeeded for 50000 packet size. Finally, for the small file size (green color), as the packet size increases, the transfer time increases too.

## 8.7 Evaluation of PSK modulation scheme

As already mentioned the maximum values for the sampling frequency $F_s$ are: *44100 and 88200 samples/sec*. The value for the symbol period used was $T = 10$ .The system was first tested without equalizers and encoders. The only module working on the receiver unit, besides the demodulator was the amplitude and phase recovery stage. In this way, we succeeded the least processing time and therefore the maximum transfer rate. PSK of order 4 and 8 had a good performance, with bit error rate constantly equal to zero, for both sampling frequencies and both for open and closed loop. We continued by adding an equalizer to the system. The transfer rate decreased, due to the extra processing time for equalizing. However, this deterioration was small and varied from 0.1–0.4 Kbps. The bit error rate remained at the same levels. We may conclude that the cost of using an equalizer is not so great and there is no doubt that equalization is always for our benefit. Using any coders was not judged necessary, since in no case errors occurred to correct. In the closed loop, at least during the experiments performed, we did not notice any retransmissions of the same packet.

*4 PSK Results*

| Settings | Fs | Total Bits | BER | Transmission Rate | Transfer Rate |
|---|---|---|---|---|---|
| LMS Off | 44100 Hz | $9 \cdot 10^5$ bits | 0 | 8820 bps | 5518 bps |
| | 88200Hz | $9 \cdot 10^5$ bits | 0 | 17640 bps | 8281bps |
| LMS On | 44100 Hz | $9 \cdot 10^5$ bits | 0 | 8820 bps | 5480 bps |
| | 88200Hz | $9 \cdot 10^5$ bits | 0 | 17640 bps | 7808 bps |

**Open Loop**

| Settings | Fs | Total Bits | BER | Transmission Rate | Transfer Rate |
|---|---|---|---|---|---|
| LMS Off | 44100 Hz | $9 \cdot 10^5$ bits | 0 | 8820 bps | 4888 bps |
| | 88200Hz | $9 \cdot 10^5$ bits | 0 | 17640 bps | 6921bps |
| LMS On | 44100 Hz | $9 \cdot 10^5$ bits | 0 | 8820 bps | 4851bps |
| | 88200Hz | $9 \cdot 10^5$ bits | 0 | 17640 bps | 6587bps |

**Closed Loop**

---

*8 PSK Results*

| Settings | Fs | Total Bits | BER | Transmission Rate | Transfer Rate |
|---|---|---|---|---|---|
| LMS Off | 44100 Hz | $9 \cdot 10^5$ bits | 0 | 13230 bps | 8635 bps |
| | 88200Hz | $9 \cdot 10^5$ bits | 0 | 26460 bps | 12525bps |
| LMS On | 44100 Hz | $9 \cdot 10^5$ bits | 0 | 13230 bps | 8378bps |
| | 88200Hz | $9 \cdot 10^5$ bits | 0 | 26460 bps | 12403bps |

**Open Loop**

| Settings | Fs | Total Bits | BER | Transmission Rate | Transfer Rate |
|---|---|---|---|---|---|
| LMS Off | 44100 Hz | $9 \cdot 10^5$ bits | 0 | 13230 bps | 7168bps |
| | 88200Hz | $9 \cdot 10^5$ bits | 0 | 26460 bps | 9656bps |
| LMS On | 44100 Hz | $9 \cdot 10^5$ bits | 0 | 13230 bps | 6977bps |
| | 88200Hz | $9 \cdot 10^5$ bits | 0 | 26460 bps | 9574bps |

**Closed Loop**

Then, we tested 16PSK without using any equalizers or coders. The average bit error rate was $1.24 \cdot 10^{-4}$ for sampling frequency 44100 Hz and $5.6 \cdot 10^{-2}$ for sampling frequency 88200 Hz. The need of equalization was in that case essential. By using an equalizer we managed to achieve again a lower bit error rate equal to $1.03 \cdot 10^{-4}$ for $F_S = 44100\,\text{Hz}$. However, for $F_S = 88200\,\text{Hz}$ the bit error rate only decreased to $2 \cdot 10^{-2}$. Instead of using only an equalizer, we tried correcting

the bit errors with coders for both values of $F_S$. Both Reed Solomon and convolutional coders achieved a sufficient low BER for $F_S = 44100$ Hz. However, the cost was a significant loss in transfer rate. For $F_S = 88.200$ Hz, their error correction capability was not proved sufficient. In the closed loop no retransmissions of the same packet occurred.

*16 PSK Results*

| Settings | Fs | Total Bits | BER | Transmission Rate | Transfer Rate |
|---|---|---|---|---|---|
| LMS Off | 44100Hz | $9 \cdot 10^5$ bits | $1.24 \cdot 10^{-4}$ | 17640 bps | 11370 bps |
| | 88200Hz | $9 \cdot 10^5$ bits | $5.6 \cdot 10^{-2}$ | 35280 bps | 16636 bps |
| LMS On | 44100Hz | $9 \cdot 10^5$ bits | $1.03 \cdot 10^{-4}$ | 17640 bps | 11296 bps |
| | 88200Hz | $9 \cdot 10^5$ bits | $2 \cdot 10^{-2}$ | 35280 bps | 15914 bps |
| **RS(255,231)** & LMS | 44100Hz | $9 \cdot 10^5$ bits | 0 | 15876 bps | 9940 bps |
| | 88200Hz | $9 \cdot 10^5$ bits | $1.1 \cdot 10^{-2}$ | 26460 bps | 7006 bps |
| **Conv(3,2,6)** & LMS | 44100Hz | $9 \cdot 10^5$ bits | 0 | 7033bps | 7329 bps |
| | 88200Hz | $9 \cdot 10^5$ bits | $6 \cdot 10^{-3}$ | 8881 bps | 10857 bps |

**Open Loop**

| Settings | Fs | Total Bits | BER | Transmission Rate | Transfer Rate |
|---|---|---|---|---|---|
| **RS(255,231)** & LMS | 44100 Hz | $9 \cdot 10^5$ bits | 0 | 15876 bps | 7154bps |
| **Conv(4,3,6)** LMS | 44100 Hz | $9 \cdot 10^5$ bits | 0 | 13230 bps | 6912bps |

**Closed Loop**

The same procedure continued for 32PSK. The system was tested only for sampling frequency 44100Hz, because its failure at 88200Hz was certain since neither PSK16 at 88200Hz was capable of providing us with a BER under the desired threshold.. The results for 32PSK even at 44100Hz were disappointing. The only thing worth noticing in the following table is that BER is constantly decreasing by using an equalizer or a combination of an equalizer and a coder but never reaches a low probability, whereas in parallel the transfer rate also decreases.

*32 PSK Results*

| Settings | Fs | Total Bits | BER | Transmission Rate | Transfer Rate |
|---|---|---|---|---|---|
| LMS Off | 44100 Hz | $9 \cdot 10^5$ bits | $15 \cdot 10^{-2}$ | 22050 bps | 13722 bps |
| LMS On | 44100 Hz | $9 \cdot 10^5$ bits | $12.8 \cdot 10^{-2}$ | 22050 bps | 13447 bps |
| **RS(255,201)** & LMS | 44100 Hz | $9 \cdot 10^5$ bits | $11.1 \cdot 10^{-2}$ | 17199 bps | 3467 bps |
| **Conv(2,1,5)** & LMS | 44100 Hz | $9 \cdot 10^5$ bits | $10.8 \cdot 10^{-2}$ | 11025 bps | 4973 bps |

**Open Loop**

## 8.8 Evaluation of Reed Solomon coders for 16-32PSK

For sampling frequency equal to 44100Hz, the quantity of errors Reed Solomon should correct was not so large. For this reason, coders with high codes rates were preferred. In this way, the loss in transmission and transfer rate would not be so great. At first randomly a high code rate was selected, 0.96, and the coders with this code rate were tested. Particularly, RS(255,247) never achieved a serious decrease of BER . Since this result was not satisfying, we tested coders with lower code rates. We concluded to a code rate of 0.9. Coders of this code rate are: RS(255,231) , RS(127,115) and RS(63,57). These three coders decreased, almost in all cases BER to zero but with a loss of 12.6% in transfer rate. According to theory, when the code rate is held constant coders with longer block lengths perform better than coders with short block lengths as far as error correcting is concerned. For this reason, we could use RS(255,231) or RS(127,115) to keep up with theory.

| Coders | Code rate | BER | Transmission Rate | Transfer Rate |
|---|---|---|---|---|
| RS(255,231) | 0.905 | 0 | 15876 bps | 9940 bps |
| RS(127,115) | 0.905 | 0 | 15876 bps | 10035 bps |
| RS(63,57) | 0.904 | 0 | 15876 bps | 9850 bps |

Moreover, we observe that the transfer rates the coders achieved were similar. According to theory, we would expect that the coder with the higher complexity that is RS(255,231) would correspond to the lower transfer rate. However, that part of theory is not verified totally. This is probably due to the fact that the results are experimental and their values are very close. Then, we continued by trying increasing a little the code rate to 0.92 and use an interleaver. This resulted also to a low BER near zero, for the case of RS(127,117) but there was no serious gain in transfer rate. Since the modules of interleaving do not demand a great amount of processing

time and seem to improve performance rather than deteriorating it, we concluded to use them in combination with the coders. Block interleavers are specially implemented for RS coders, whereas random interleavers are used with convolutional coders. Finally, coders with lower code rate than 0.9 were also tested, RS(255,223) , RS(127,111) and RS(63,55) with code rate 0.87 . This resulted again to a BER, almost in all cases equal to zero, but also to a reasonable decrease of transfer rate.

| Coders | Code rate | BER | Transmission Rate | Transfer Rate |
|---|---|---|---|---|
| RS(255,223) | 0.874 | 0 | 15346 bps | 9036 bps |
| RS(127,111) | 0.874 | 0 | 15346 bps | 9331 bps |
| RS(63,55) | 0.873 | 0 | 15346 bps | 8850 bps |

For sampling frequency 88200Hz Reed Solomon coders are used to decrease a BER=$1.1 \cdot 10^{-2}$. The coders tested had a greater error correction capability of those for 44100Hz and therefore a lower code rate. More specifically, RS coders with code rates 0.82, 0.78, 0.76 were tested. The result was that the BER decreased, which is totally meaningless for our system, since the transfer rate loss was near 60%. We also checked if by using a longer training sequence the equalizer performed better. After several trials, even when the training sequence became 7 times longer, the bit error rate remained at high levels. Correspondingly, in case of 32PSK, the situation was similar or even worse since the transfer rate reached 3467bps, performance even lower compared to 4PSK at 44100Hz.

## 8.9 Conclusions for PSK

Some useful conclusions were drawn by the experiments performed for PSK. The first conclusion was that the use of an equalizer does not cause a serious decrease of the transfer rate. For this reason, its use is suggested for all orders of all modulation schemes. The same also holds for the interleavers, which as explained can be used, whenever coding is necessary. Using PSK the transfer rates achieved are between 7329bps and 12525bps for an open loop. For a closed loop, the transfer rates are between 4851bps and 9656bps. The difference between open and

closed loop is due to possible retransmissions that occurred and to the processing time needed for CRC. The higher transfer rate reached was by 8PSK at 88200Hz. We also note that the encoded 16PSK with RS at 44100Hz performed better than 8PSK at 44100Hz, which proves that the use of encoders was not useless.

The following diagrams present the order of M vs the higher transfer rates of each order for open and close loop for the two sampling frequencies.

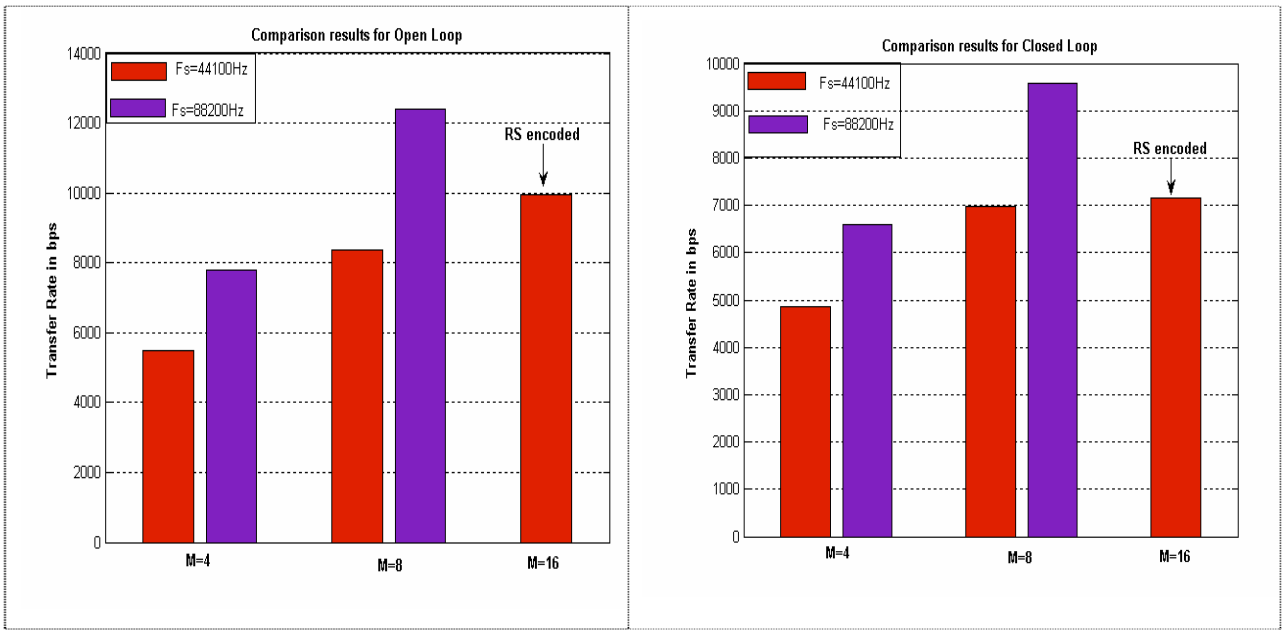



**Figure 8.2 PSK open-closed loop comparison**

## 8.10Evaluation of QAM scheme

Based on the conclusions for PSK, all the experiments for QAM were performed keeping always the equalizing unit on. The symbol period was $T = 10$ for all experiments and the sampling frequencies 44100Hz and 88200Hz. Similarly to 4PSK the bit error rate was zero for both sampling frequencies. In the closed loop, no retransmissions of the same packet were observed.

*4 QAM Results*

| Settings | Fs | Total Bits | BER | Transmission Rate | Transfer Rate |
|---|---|---|---|---|---|
| LMS On | 44100 Hz | $9 \cdot 10^5$ bits | 0 | 8820bps | 5483bps |
| | 88200Hz | $9 \cdot 10^5$ bits | 0 | 13230bps | 7343bps |

**Open Loop**

| Settings | Fs | Total Bits | BER | Transmission Rate | Transfer Rate |
|---|---|---|---|---|---|
| LMS On | 44100 Hz | $9 \cdot 10^5$ bits | 0 | 8820bps | 4853bps |
| | 88200Hz | $9 \cdot 10^5$ bits | 0 | 13230bps | 6250bps |

**Closed Loop**

However, for 8QAM the system did not behave the same way it did for 8PSK. For sampling rate 44100Hz the bit error rate was sufficiently low but for 88200Hz BER was $3.6 \cdot 10^{-3}$. The next step was to use coders but they did not manage to guarantee error free transmission. Certainly, BER was decreased and that was very helpful for the closed loop, which with repeated retransmissions was capable of sending correctly the file, but with a serious decrease of transfer rate.

*8 QAM Results*

| Settings | Fs | Total Bits | BER | Transmission Rate | Transfer Rate |
|---|---|---|---|---|---|
| LMS On | 44100Hz | $9 \cdot 10^5$ bits | 0 | 13230 bps | 8569bps |
| | 88200Hz | $9 \cdot 10^5$ bits | $3.6 \cdot 10^{-3}$ | 26640 bps | 12308bps |
| RS(127,111) & LMS | 88200Hz | $9 \cdot 10^5$ bits | $1.4 \cdot 10^{-3}$ | 23126bps | 4557bps |
| Conv(2,1,6) & LMS | 88200Hz | $9 \cdot 10^5$ bits | $3.7 \cdot 10^{-4}$ | 13230bps | 6077bps |

**Open Loop**

| Settings | Fs | Total Bits | BER | Transmission Rate | Transfer Rate |
|---|---|---|---|---|---|
| LMS On | 44100Hz | $9 \cdot 10^5$ bits | 0 | 13230 bps | 7143bps |
| RS(127,111) & LMS | 88200Hz | $9 \cdot 10^5$ bits | 0 | 22725 bps | 4001bps |
| Conv(2,1,6) & LMS | 88200Hz | $9 \cdot 10^5$ bits | 0 | 13230 bps | 4580bps |

**Closed Loop**

For sampling frequency 44100Hz and QAM16 the system achieved a low bit error rate, only by using a coder. On the contrary, when the sampling frequency was 88200Hz neither of the coders could assure that the bit error rate would be zero but only an expected improvement.

***16 QAM Results***

| Settings | Fs | Total Bits | BER | Transmission Rate | Transfer Rate |
|---|---|---|---|---|---|
| LMS On | 44100Hz | $9 \cdot 10^5$ bits | $6.4 \cdot 10^{-4}$ | 17640bps | 9823bps |
| | 88200Hz | $9 \cdot 10^5$ bits | $6.8 \cdot 10^{-3}$ | 35280bps | 13868 bps |
| RS(127,111) & LMS | 44100Hz | $9 \cdot 10^5$ bits | 0 | 13759bps | 8413bps |
| RS(127,105) & LMS | 88200Hz | $9 \cdot 10^5$ bits | $0.8 \cdot 10^{-3}$ | 28929bps | 6996bps |
| Conv(3,1,6) & LMS | 44100Hz | $9 \cdot 10^5$ bits | 0 | 5880bps | 4082bps |
| | 88200Hz | $9 \cdot 10^5$ bits | $9.18 \cdot 10^{-4}$ | 11760bps | 6056bps |

**Open loop**

| Settings | Fs | Total Bits | BER | Transmission Rate | Transfer Rate |
|---|---|---|---|---|---|
| RS(127,111) & LMS | 44100Hz | $9 \cdot 10^5$ bits | 0 | 13759bps | 5157 bps |
| Conv(3,1,6) & LMS | 44100Hz | $9 \cdot 10^5$ bits | 0 | 5880bps | 2016bps |

**Closed loop**

Likewise 32PSK, 32QAM never succeeded a sufficient low BER, even though coders were used. The results following are just added to show the performance of coders.

*32 QAM Results*

| Settings | Fs | Total Bits | BER | Transmission Rate | Transfer Rate |
|---|---|---|---|---|---|
| LMS On | 44100 Hz | $9 \cdot 10^5$ bits | $14.3 \cdot 10^{-2}$ | 22050 bps | 12330 bps |
| **RS(255,201)** & LMS | 44100 Hz | $9 \cdot 10^5$ bits | $9 \cdot 10^{-2}$ | 17199bps | 3167 bps |
| Conv(2,1,5) & LMS | 44100 Hz | $9 \cdot 10^5$ bits | $7.6 \cdot 10^{-2}$ | 11025bps | 5012 bps |

**Open loop**

## 8.11 Evaluation of Reed Solomon coders for 8-16-32QAM

The need of a Reed Solomon coder was necessary in case of 8QAM at 88200Hz. There was an improvement in the value of bit error rate. There were also several transmissions with bit error rate equal to zero, which means that adding on CRC, each packet will probably arrive at its destination correctly. The price, however, was that the coder used, had a code rate 0.87, resulting in the delay of our system. More specifically, the transfer rate, with RS(127,111) on, was 4557bps. This means that the performance of 8QAM at 88200Hz was worse than that of the same order at 44100Hz.

For 16QAM at 44100Hz, coders with different code rates were tried and we concluded to those who seemed to be more stable. First, we experimented with the same coders as in 16PSK since the bit error rate was similar and in fact they had a good performance.

| Coders | Code rate | BER | Transmission Rate | Transfer Rate |
|---|---|---|---|---|
| RS(255,223) | 0.874 | 0 | 15346 bps | 8086bps |
| RS(127,111) | 0.874 | 0 | 15346 bps | 8413bps |
| RS(63,55) | 0.873 | 0 | 15346 bps | 7763bps |

Compared to the encoded 16PSK the performance in transfer rate was lower. It was also lower than 8QAM at 44100Hz. For this reason, coders with higher code rates were tested. For code rate 0.9, RS(127,115) gave the lowest average bit error rate and succeeded transfer rate 8999bps, which is a little higher than 8QAM.

As far as 32QAM is concerned, the Reed Solomon coders only managed to diminish the probability of error, but still its value is so great that the use CRC is prohibitive. It is certain that 32QAM functioning in a closed loop in our system will result to an endless loop of retransmissions.

## 8.12 Conclusions for QAM

Using QAM the transfer rates achieved are between 5483bps and 8569bps for an open loop. For a closed loop, the transfer rates are between 4853bps and 7143bps. The higher transfer rate reached was by 8QAM at 44100Hz. It is interesting to notice that the use of encoders on 16QAM at 44100Hz decreased the transfer rate of the system compared to 8QAM at 44100Hz. However, in the closed loop we observe that transfer rate decreases. This is due to the fact that the RS coders did not provide us constantly with a BER=0 and retransmissions were necessary. The same diagrams as in PSK evaluation follow:

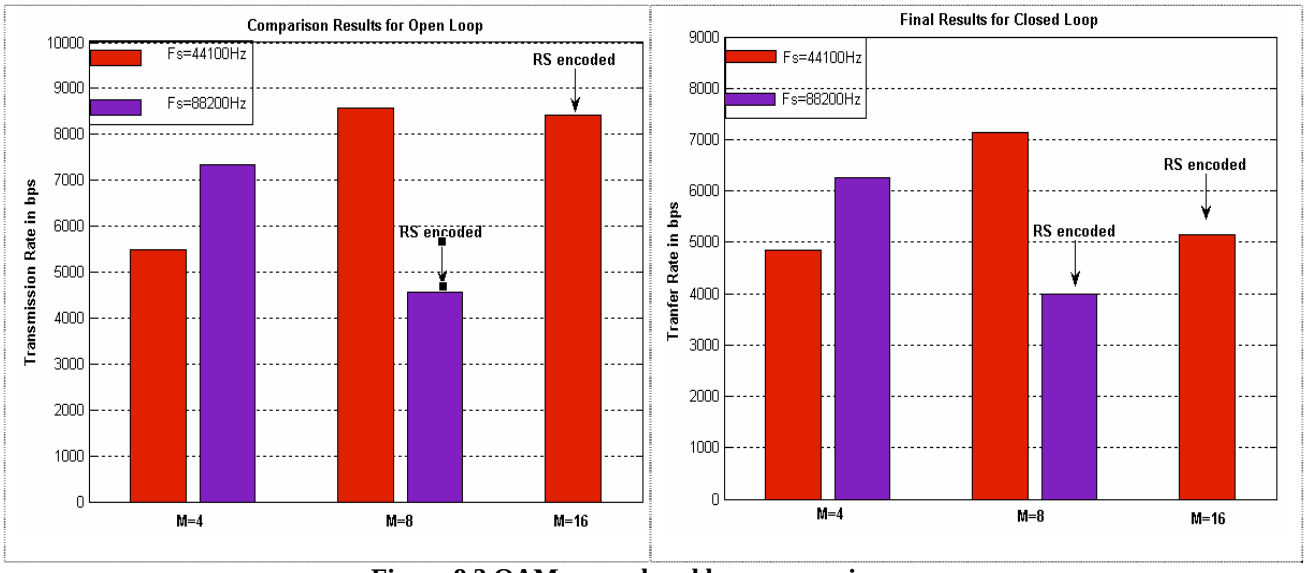



**Figure 8.3 QAM open-closed loop comparison**

## 8.13Evaluation of PPM scheme

The case of PPM differs comparing to the other two modulation schemes as far as the equalization is concerned. Since PPM is a multidimensional modulation scheme, the number of its correlators varies and each time is equal to the dimension selected. For this reason, the typical way of equalizing the symbols at the exit of the correlators was not possible. Alternatively, another method was tried. The equalization was performed before the samples entered the correlation stage. However, this method was not proven so efficient since the equalization on samples was very time-consuming. Moreover, the unit of PPM detector also consumes more time than the detector unit of the other two modulation schemes. These facts can justify to a point the performance in transfer time of PPM.

In case of 4PPM the symbol period selected was $T = 8$. We note at this point that for 4PPM the number of possible waveforms is equal to the dimensions, which are 4. The symbol period $T = 8$ corresponds to 8 samples with which 4 different waveforms must be created (see figure 7.2, chapter 7), which is possible. This clarification is done to show that the minimum value for the symbol period could have been 4 samples, which was tested unsuccessfully. For 4PPM the BER was zero for both frequencies and the transfer rates very satisfying. This seems to contradict with what was mentioned in the beginning of this section but it is reminded that the others modulation schemes were tested for symbol period $T = 10$.

***4 PPM Results***

| Settings | Fs | Total Bits | BER | Transmission Rate | Transfer Rate |
|---|---|---|---|---|---|
| Equalizers Off | 44100 Hz | $9 \cdot 10^5$ bits | 0 | 11025 bps | 5659bps |
| | 88200Hz | $9 \cdot 10^5$ bits | 0 | 22050 bps | 11399bps |

**Open Loop**

| Settings | Fs | Total Bits | BER | Transmission Rate | Transfer Rate |
|---|---|---|---|---|---|
| Equalizers Off | 44100 Hz | $9 \cdot 10^5$ bits | 0 | 11025 bps | 4988bps |
| | 88200Hz | $9 \cdot 10^5$ bits | 0 | 22050 bps | 9004bps |
| CRC On | | | | | |

**Closed Loop**

By continuing to 8PPM we tried to hold the symbol period constant. The experiments results showed that there was a bit error rate of order $10^{-1}$ for sampling frequency 44100Hz that no coder was capable of correcting. Alternatively, we increased the symbol period. Therefore we doubled the samples to 16. With the double symbol period the BER was constantly zero but the transfer rate was worse compared to 4PPM.

*8 PPM Results*

| Settings | Fs | Total Bits | BER | Transmission Rate | Transfer Rate |
|---|---|---|---|---|---|
| Equalizers Off , T=16 | 44100 Hz | $9 \cdot 10^5$ bits | 0 | 8269 bps | 4487 bps |
| | 88200Hz | $9 \cdot 10^5$ bits | 0 | 16538 bps | 5930 bps |

**Open Loop**

| Settings | Fs | Total Bits | BER | Transmission Rate | Transfer Rate |
|---|---|---|---|---|---|
| Equalizers Off , T=16 | 44100 Hz | $9 \cdot 10^5$ bits | 0 | 8269 bps | 4034bps |
| | 88200Hz | $9 \cdot 10^5$ bits | 0 | 16538 bps | 5193bps |
| CRC On | | | | | |

**Closed Loop**

The same procedure was followed for 16PPM. First $T = 16$ was tested and the average BER was $8.4 \cdot 10^{-2}$. No coders were tested for correcting this error, given that with 16samples per symbol plus the redundant symbols from the encoders, the degradation in transfer rate was guaranteed. As in 8PPM when we used higher symbol period (the double again) the bit error rate became zero.

*16 PPM Results*

| Settings | Fs | Total Bits | BER | Transmission Rate | Transfer Rate |
|---|---|---|---|---|---|
| Equalizers Off , T=32 | 44100 Hz | $9 \cdot 10^5$ bits | 0 | 5513 bps | 3097 bps |
| | 88200Hz | $9 \cdot 10^5$ bits | 0 | 11026 bps | 4189 bps |

**Open Loop**

| Settings | Fs | Total Bits | BER | Transmission Rate | Transfer Rate |
|---|---|---|---|---|---|
| Equalizers Off | 44100 Hz | $9 \cdot 10^5$ bits | 0 | 5513 bps | 2884bps |
| | 88200Hz | $9 \cdot 10^5$ bits | 0 | 11026 bps | 3800bps |
| CRC On | | | | | |

**Closed Loop**

We notice that 16PPM is even worse than 8PPM. There was no reason to continue increasing the order of PPM since that would surely deteriorate the transfer rate.

## 8.14 Conclusions for PPM

In general, the performance of PPM was restricted because of the fact that we were obliged to increase continuously the symbol period as the order of the PPM increased. However, 4PPM managed a satisfying transfer rate. Finally, we present its performance for both open and close loop, like it was done in cases of PSK and QAM. The different symbol periods are indicated on the figures.

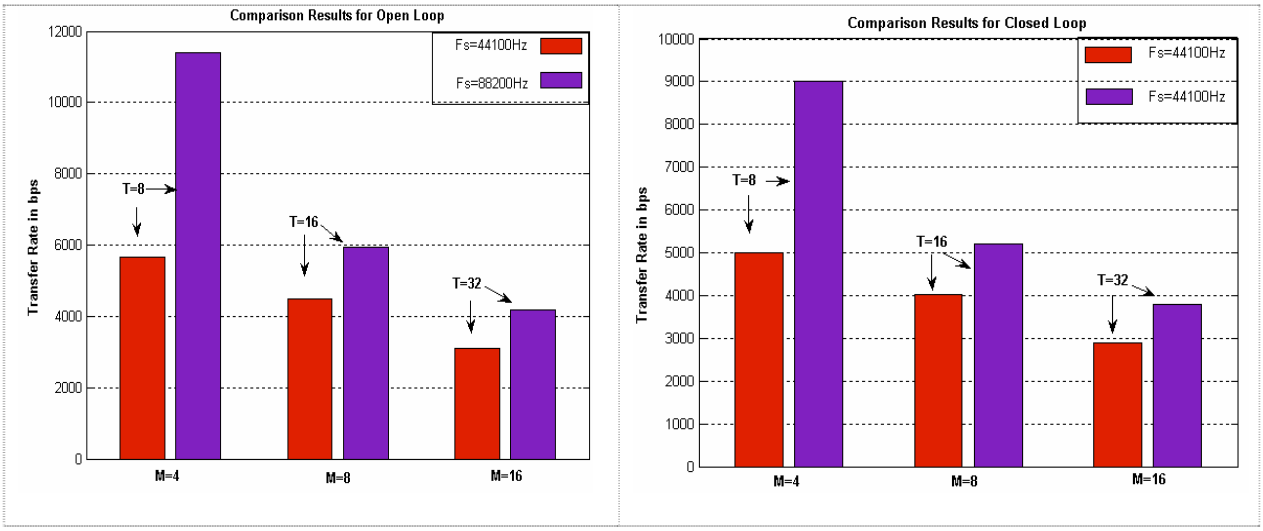



**Figure 8.4 PPM open-closed loop comparison**

## 8.15 Overall summary

M-ary PSK and M-ary QAM attain similar performances as far as bit error rate and transfer rate are concerned for orders 4 and 8 at 44100Hz, with M-ary PSK being slightly better. However, M-ary PSK performs obviously better for order 16. This is due to the encoding schemes that for 16-QAM, although they correct all bit errors, they were not capable of preserving the transfer rate at a high level. Both modulation schemes failed for M equal to 32.

From another perspective, we can also borrow from theory that for M-ary PSK and QAM, as M grows the same does the probability of bit error for a given SNR. Thus, the only way to perform in high orders of M and achieve high data rates is to increase SNR. Trying increasing the symbol power for QAM always resulted in cutting off of values from the waveforms and for this reason normalization was done. For PSK and PPM there was not a similar problem with the amplitude of the waveforms, since the amplitude of PSK was in the bounds [-1,1] and the amplitude of PPM was "fixed" to be in these bounds, as explained in chapter 7. In case of M-ary orthogonal PPM, as M increases the data rate diminishes. This is happening because as the order grows, the symbol period also grows, in order to succeed a low bit error rate.

As far as Reed Solomon coders are concerned, according to the results their application was beneficial only for PSK, where the bit error rate for order 16 was turned to zero and simultaneously the transfer rate was greater than that of order 8 (44100Hz). The same holds for 8QAM (88200Hz), where the bit error rate was zero and the transfer rate was again greater than that of order 4. In all other cases, Reed Solomon coders may have decreased or nullified the probability of bit error but the price was in transfer rate. Instead of coders CRC can be used, but since it is an error detection code and not an error correction code, there is always the probability of endless retransmissions.

In the following page, the higher transfer rates of all orders of all modulation schemes for both open and close loop are presented. In the figures, it is seen clearly which modulation scheme performed better in every order of M. It is reminded that the symbol period of PSK and QAM is 10, whereas for PPM the symbol period varies.

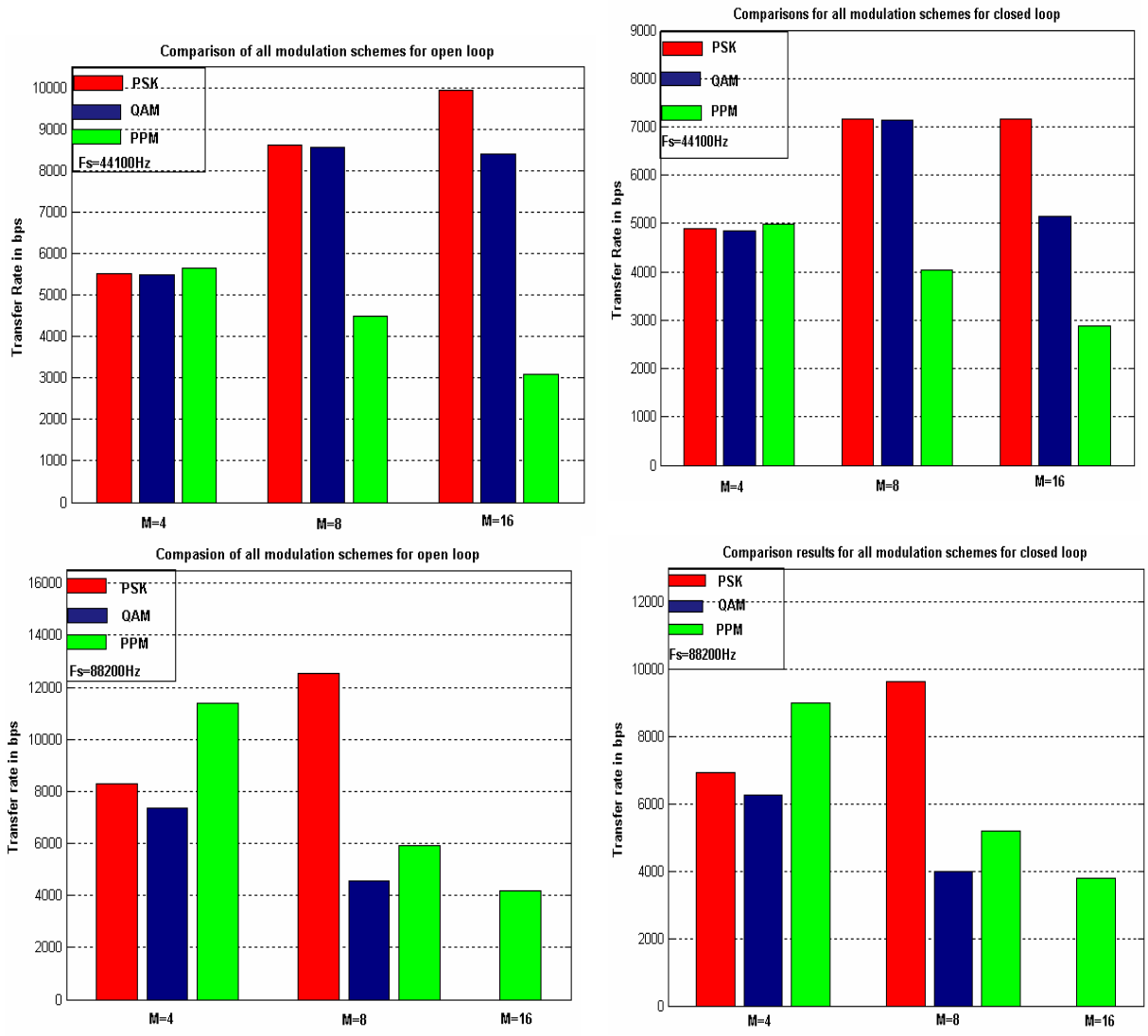



**Figure 8.5 Comparison of all modulation schemes**

## 8.15 System Proposed

Before proposing a final system we will present a table containing all transfer rates achieved with minimum bit error rate and by which configuration in an open loop. Using this table it is easy to conclude to a system to propose for an open loop. As far as the closed loop is concerned, we will prefer the configuration that resulted to the minimum number of retransmissions, at least during the experiments.

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

According to the table by selecting 8PSK with LMS equalizer or 4PPM without LMS equalizer at sampling frequency 88200Hz, we can achieve transfer rates higher than 10000bps. Furthermore, we remind that both 8PSK and 4PPM resulted to a sufficiently low bit error rate that in the majority of the experiments was constantly equal to zero. This means that for an open loop the number of bit errors is very low and therefore the file will not arrive at its destination corrupted. Finally, we can use the same configuration for a closed loop since there were rare retransmissions and the transfer rate was still over 8000bps. However, for compatibility reasons with the audiocards we will propose the value for the sampling frequency to be 44100Hz. The final proposed systems compatible with all audiocards for an open and a closed loop follow:

## Final proposed system

| **Modules for 8PSK** | |
|---|---|
| | |
| Synchronizer | **O N** |
| PSK Correlators | **O N** |
| LMS Equalizer | **O N** |
| Am/Ph Recovery | **O N** |
| PSK Detector | **O N** |
| | |
| Sampling Frequency | 44100Hz |
| | |
| Bit Error Rate | $10^{-6}$ |
| Transfer Rate | 8378bps |

**Open loop**

| **Modules for 8PSK** | |
|---|---|
| | |
| Synchronizer | **O N** |
| PSK Correlators | **O N** |
| LMS Equalizer | **O N** |
| Am/Ph Recovery | **O N** |
| PSK Detector | **O N** |
| CRC | **O N** |
| | |
| Sampling Frequency | 44100Hz |
| | |
| Bit Error Rate | 0 |
| Transfer Rate | 6977bps |

**Closed loop**

## REFERENCES


[1] Bernard Sklar: Digital *Communications Fundamentals and applications,* Prentice Hall 1988

[2] Bernard Sklar: *Defining, Designing and Evaluating Digital Communication Systems,* IEEE Communications Magazine November 1993

[3] C. Richard Johnson Jr., William A. Sethares: *Telecommunication Breakdown, Concepts of communication Transmitted via Software-Defined Radio,* Prentice Hall 2003

[4] John Proakis, Masoud Salehi: *Digital Communications,* Prentice Hall 2002

[5] Martin Bossert: *Channel Coding for Telecommunications*, WILEY 1999

[6] Nicholas Laneman: *Reed Solomon Decoding Algorithms for digital audio broadcasting in the AM band,* IEEE Transactions on Broadcasting, vol 47, No2 June 2001

[7] Robert H. Morelos-Zaragoza: *The Art of Error Correcting Coding,* WILEY 2002

[8] Shu Lin, Daniel J. Costello, Jr: *Error Control Coding (second edition),* Prentice Hall 2004

[9] Stephen B. Wicker, Vijay K. Bhargava: *Reed-Solomon codes and their applications,* IEEE Press 1993

[10] William Stallings: *Wireless Communications and Networks*, Prentice Hall 2002