**Advanced Signal Processing**
**Winter Term 2001/2002**

**Digital Subscriber Lines (xDSL):**
**Broadband Communication**
**over Twisted Wire Pairs**

# Error Correction
# and
# Trellis Coding

**Thomas Brandtner**

**brandt@sbox.tugraz.at**

# Contents

# 1 Introduction

The aim of every communication system is the transmission of information between two or more points. One may distinguish between analogue and digital communication systems depending on the type of data that is transmitted. A digital system like DSL can be divided into the transmitter, the digital channel and the receiver. A stream of bits is generated inside the transmitter and subsequently fed to the digital channel. The receiver tries to determine the right bit values out of the signal on the other end. Unfortunately the bits may be distorted while being sent over the channel. Some error sources like echos or the transfer function of the channel are deterministic, hence their influence can be cancelled. On the other hand it is not possible to correct non-deterministic errors like those originated in white noise. In this case the possibility to detect or even correct bit errors is crucial to the function of the given communication system.

## 1.1 Error Detection and Error Correction

In principle the designer of a communication system has got two choices: Either he implements a system with simple error detection with no chance to correct bits without a feedback from the receiver to the transmitter. If an error is detected, the receiver asks the transmitter to repeat the distorted bit stream by a feedback protocol. The other possibility is a system with error correction capabilities. In both cases a certain amount of transmission redundancy has to be applied to the bit stream. Otherwise the receiver would not be able to distinguish between valid and invalid data.

This redundancy can be inserted either at bit level or at bit-to-symbol level. At bit level some additional bits like parity bits are merged to the given information bits (Figure 1). If a line code with more than two symbols like quadrature amplitude modulation (QAM) is used, some of the possible symbols may be defined as valid, whereas others are invalid in order to make it possible to detect distorted signals.

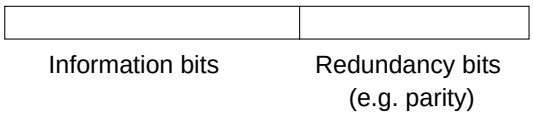


Figure 1: Transmission redundancy on bit level

However, if additional bits are inserted, the required transmission throughput is increased. It can be satisfied by either increasing the transmission rate or by expanding the number of symbols of the line code. In the first case the channel bandwidth is increased resulting in a small reduction of the signal-to-noise ratio (SNR). Hence the bit error rate (BER) rises because now it is more likely for a bit to be changed by random noise. An example of a diagram that compares the ratio between signal energy of every information bit and channel noise and the resulting BER is shown in Figure 2. If a certain BER has to be guaranteed, a certain minimum SNR must be provided. If an error correction code is used, this necessary minimum SNR decreases because it is now possible to correct some of the errors. This change in SNR is called coding gain.

## 1.2 Basic Terms

Let the whole code word consist of $N$ bits. The code word itself is generated out of $k$ information bits. If these information bits appear inside the code word without any change, the code is called to be systematic.

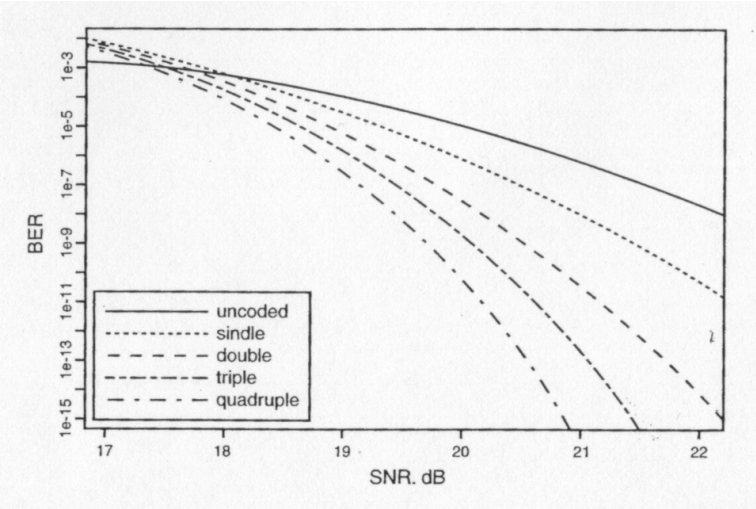


Figure 2: BER over SNR - Example of coding gain

All the other bits inside the code word are redundancy bits. The code rate $R_c$ is defined as

$$R_c = \frac{k}{N} \tag{1}$$

telling us how much redundancy has been added to the original data. But this is not enough to describe the real error detection and correction capabilities of a given code. The minimum number of different bits between two code words is called Hamming distance. The Hamming distance of a code $d^H_{min}$ is the minimum of all Hamming distances of its code words. In Figure 3 two code words with a Hamming distance of 4 are shown. If these code words are transmitted over a binary symmetrical channel with independent white noise, the likelihood that a bit is changed from 1 to 0 is equal to the probability that a 0 is altered to 1. Moreover, it is more likely that only one bit is distorted than two or more bits. Hence if an invalid code word is received, it is most likely that the nearest code word is the original one. Therefore, a given code with a hamming distance $d^H_{min}$ has the following capabilities:

- $d^H_{min} - 1$ bit errors may be detected
- $t = \lfloor \frac{d^H_{min}-1}{2} \rfloor$ bit errors may be corrected.

A short terminology is introduced calling such a code $(N, k, d^H_{min})$-Code.

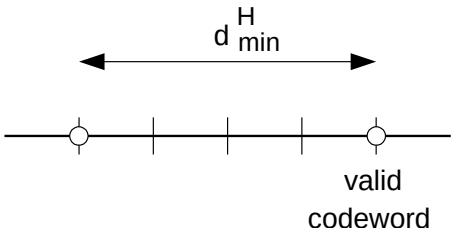


Figure 3: Hamming distance and error correction capability

# 2 Block Codes

One family of codes are block codes. A certain block of information bits are taken and redundancy bits are added. The simplest members of this family are parity codes. A more systematic approach for designing proper parity bits are cyclic codes (sometimes known as cyclic redundancy check - CRC). If one wants to generate codes which can correct more than one bit errors, the mathematical theory of Galois fields has to be used leading to BCH and Reed Solomon (RS) codes. RS codes are used in high speed DSL systems like ADSL.

## 2.1 Parity Codes

The simplest parity code one could use is a (?,1,1)-code: Only one additional parity bit is generated. Assuming 4 information bits, the whole code word may be written as $(i_1, i_2, i_3, i_5, p_1)$ where the parity bit is calculated as

$$p_1 = i_1 + i_2 + i_3 + i_4 \tag{2}$$

One should note that all this has to be carried out in modulo-2 arithmetic, where the addition is actually a logical exclusive-or (XOR) operation. This results in a (4,1,1)-code with a hamming distance of 2, hence it is possible to detect one-bit-errors, but no bits can be corrected at all.

If this type of code should be equipped with error correction capabilities, some additional parity bits have to be introduced. An example is the following (6,3,3)-code, which can be used to detect two bit errors and to correct one bit errors.

$$\begin{aligned} p_1 &= i_1 + i_2 && (3)\\ p_2 &= i_2 + i_3 && (4)\\ p_3 &= i_1 + i_3 && (5) \end{aligned}$$

The whole task can be written in a more general way with bit vectors and matrices. The code word c is a vector $(i_1, i_2, i_3, p_1, p_2, p_3)$ built from a generator matrix G and the original message word $m = (m_1, m_2, m_3)$:

$$c = m \cdot G \tag{6}$$

G can be divided into two parts: An identity matrix $I_k$ with k bits and the parity generator part P:

$$\begin{aligned} G &= [I_k \quad P] && (7)\\ G &= \left[\begin{array}{ccc|ccc} 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \end{array}\right] && (8) \end{aligned}$$

The identity matrix $I_k$ appears in G, since this code is systematic. Columns of P must be unique, otherwise two parity bits are always equal. This theoretical way also enables us to think about the correction procedure inside the receiver. First the parity matrix H is introduced as

$$H = [P_T \quad I_r] \tag{9}$$

where $P_T$ is the transposed parity generator part of matrix G and $I_r$ is an identity matrix with r bits. In our example this leads to

$$H = \left[\begin{array}{ccc|ccc} 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 \end{array}\right] \tag{10}$$

One important property of the two matrices P and H is

$$G \cdot H^T = [I_k \quad P] \cdot \begin{bmatrix} P \\ I_r \end{bmatrix} = P + P = 0 \tag{11}$$

Let y be the vector containing the received bits. If no error occurred, y will be equal to c. Otherwise the bit error may be described by an additional term e:

$$y = c + e \tag{12}$$

The syndrome word s is calculated as follows.

$$s = y \cdot H^T \tag{13}$$

If an error occurred, this leads to

$$s = (c+e) \cdot H^T = m \cdot G \cdot H^T + e \cdot H^T = e \cdot H^T \tag{14}$$

If the syndrome word is non-zero, a bit error has been detected. This code is capable to correct one bit errors, in such a case the error word e contains one set bit on the position where the error occurred. Hence s is one column of H. If all columns of H are unique, it is then easy to decide which bit has to be toggled in order to correct the error.

## 2.2 Cyclic Codes

In the latter section a simple example of a parity code was presented. If the number of information bits k is getting bigger, it is harder to find proper equations for the parity bit calculation. In addition to that, the hardware to generate these parity bits gets more complicated. One solution to that problem are cyclic codes. The code word of a cyclic code is generated by a shift register with certain feedback.

The theory of cyclic codes starts with a special notation for bit streams exploiting polynomials. The message word m may be written as follows:

$$m = (i_0, i_1, i_2, i_3) \tag{15}$$

$$m(x) = i_0 + i_1 x + i_2 x^2 + i_3 x^3 \tag{16}$$

It has to be noted that x is not a variable, its exponent only defines the position inside the bit stream. Now a special generator polynomial is used in order to generate valid code words. This generator polynomial has to meet several conditions, if the resulting code should have certain bit correction capabilities. The theory behind this important topic is covered in the next section dealing with Galois fields.

The code words are formed by multiplying the message word with the generator polynomial. The following table illustrates this task for a (6,3,3)-code with the generator polynomial $g(x) = 1 + x^3$.

| Message | Code-word | Code-polynomial |
|---|---|---|
| 000 | 000000 | $0 \times (1+x^3) = 0$ |
| 100 | 100100 | $1 \times (1+x^3) = 1+x^3$ |
| 010 | 010010 | $x \times (1+x^3) = x+x^4$ |
| 110 | 110110 | $(1+x) \times (1+x^3) = 1+x+x^3+x^4$ |
| 001 | 001001 | $x^2 \times (1+x^3) = x^2+x^5$ |
| 101 | 101101 | $(1+x^2) \times (1+x^3) = 1+x^2+x^3+x^5$ |
| 011 | 011011 | $(x+x^2) \times (1+x^3) = x+x^2+x^4+x^5$ |
| 111 | 111111 | $(1+x+x^2) \times (1+x^3) = 1+x+x^2+x^3+x^4+x^5$ |

Again, addition is a modulo-2 operation, hence for instance $x + x = 0$. It is clear that the code word is a multiple of g(x). The message word m(x) is shifted by d bits, where d is the number of redundancy bits and the order of g(x). The polynomial $m(x) \cdot x^d$ is usually not a multiple of g(x). But this would be the case if the remainder of the operation $x_d \cdot m(x)/g(x)$ is subtracted from $x_d \cdot m(x)$. Since all operations are carried out in modulo-2, subtraction is equal to addition. This remainder is simply copied into the d least significant bits of $x_d \cdot m(x)$.

The remainder of the division operation can be calculated easily by a shift register with feedback. First consider how a division in modulo-2 arithmetic looks like:

```
11010 000
1011
-----
01100
 1011
 ----
 0111 0
  101 1
  ---- -
  010 10
   10 11
   -- --
   00 010
      010
```

Figure 4: Division in modulo-2 arithmetic

The original message word 11011 is shifted left by 3 digits. The generator word 1011 ($1 + x^2 + x^3$) is subtracted from the most significant bits of the message word. The result is taken and the next bit of the message word is put at the end. Again one tries to subtract the generator word. If this does not work, the next message word bit is taken into account as well. This procedure is repeated until all message bits have been processed. The result of the last subtraction is the remainder of the division 11011000 / 1011.

This procedure can be implemented in a shift register as well (Figure 5). First all the message bits are transmitted to the output and shifted into the shift register starting with the most significant bit (MSB). Each time when the MSB of the shift register is a 1, the generator polynomial must be subtracted from the bit stream. This is done by feeding back this 1-bit to some stages of the shift register. For instance, if $g(x) = 1 + x^2 + x^3$, $g_0$ and $g_1$ in the feedback are 1. After all message bits were shifted through the shift register, it contains the remainder of the division $x_d \cdot m(x)/g(x)$. Now the gate is opened and the switch at the output is altered. The bits of the shift register (and hence the remainder of the division) are transmitted to the output.

The decoder inside the receiver contains the same shift register. The received code word is shifted through. At the end the remainder of the division is stored inside the register stages. If it is zero, the received code word was a multiple of the generator polynomial, hence no error has been occurred. If it is non-zero, the remainder is the syndrome word. Usually, the position of the error bit is determined by exploiting a lookup table.

## 2.3 Galois Fields

The next step towards Reed Solomon codes is the introduction of Galois fields. Reed Solomon codes are able to correct more than one bit error. Galois fields are part of the basic theory of how to design such codes. A Galois field GF(p) is an abstract mathematical object consisting of a set of p numbers and two

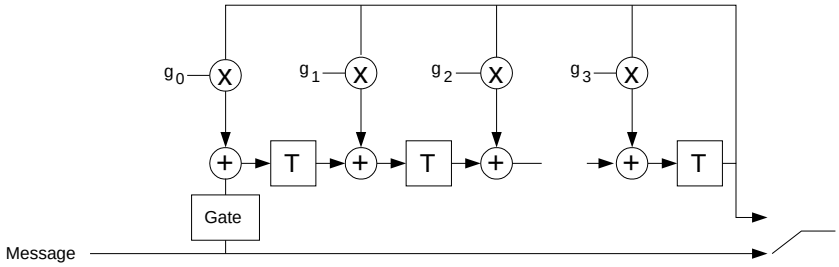


Figure 5: A shift register with feedback as CRC encoder

operations: An addition and a multiplication. An additive inverse element $-a$ and a multiplicative inverse element $a^{-1}$ exist. The zero element 0 is the neutral element for the addition, the one element 1 is the neutral element for the multiplication.

It can be shown that if p is a prime number, a primitive element $\alpha$ exists. All numbers from 1 to p-2 may be generated by $\alpha^x$. Here is one example for GF(7). It turns out that 3 is the primitive element:

$$\begin{aligned} 3^0 &= 1 \\ 3^1 &= 3 \\ 3^2 &= 2 \\ 3^3 &= 6 \\ 3^4 &= 4 \\ 3^5 &= 5 \end{aligned}$$

All operations are carried out in modulo-p, therefore there is a wrap around at 7=0. The addition is simply a modulo-7 operation like 3+5=1 without a carry to the next digit. The inverse element $-a$ is found by a mirror operation around 0 or 7, e.g. $5 \leftrightarrow 2$.

The multiplication is a modulo-7 operation as well, for instance $3 \times 5 = 3^1 \times 3^5 = 3^6 = 1$. The inverse element $a^{-1}$ may be found by considering that $3^6 = 1$. Hence $3^n \Rightarrow 3^{-n} = 3^{6-n}$.

## 2.4 Extended Galois Fields

A special family are Galois fields with $p = 2^m$ because they represent a code word with m bits. As shown in the previous section, the following conditions must be met:

$$\alpha^{p-1} = \alpha^{2^m-1} = 1 \quad (17)$$

$$\alpha^{2^m-1} + 1 = 0 \quad (18)$$

The latter polynomial may be separated into smaller polynomials. One of these polynomials has to be set to zero in order to meet the condition above. If this polynomial is irreducible and if it does not divide $a^n + 1$ for $n < 2^m - 1$, it is called primitive polynomial. Some of them are listed below.

| m | Primitive-polynomial |
|---|---|
| 3 | $1 + x + x^3$ |
| 4 | $1 + x + x^4$ |
| 5 | $1 + x^2 + x^5$ |

A complete GF($2^m$) may be generated by using such a primitive polynomial. Consider a GF(8) with a primitive polynomial $1+x+x^3$. Hence $\alpha^3$ can be reduced to $1+\alpha$ leading to the following binary representation:

| Power | Polynomial | Binary |
|---|---|---|
| 0 | 0 | 000 |
| 1 | 1 | 100 |
| $\alpha$ | $\alpha$ | 010 |
| $\alpha^2$ | $\alpha^2$ | 001 |
| $\alpha^3$ | $1+\alpha$ | 110 |
| $\alpha^4$ | $\alpha+\alpha^2$ | 011 |
| $\alpha^5$ | $1+\alpha+\alpha^2$ | 111 |
| $\alpha^6$ | $1+\alpha^2$ | 101 |

Now the abstract value $\alpha$ and its exponentials can be realised with 3 bits.

Every polynomial has got certain roots $\beta$ in a Galois field. They may be found by separating the polynomial into several smaller ones. For instance,

$$1+x+x^3 = (x+\alpha)(x+\alpha^2)(x+\alpha^4) = \sum(x+\beta_i) \tag{19}$$

in GF(8), where $\alpha, \alpha^2$ and $\alpha^4$ are the roots of $1+x+x^3$. The minimum polynomial of a is the polynomial with root a, which coefficients are only certain values, for example 0 or 1 - hence binary.

| Roots | Minimum-Polynomial |
|---|---|
| $\alpha, \alpha^2, \alpha^4, \ldots$ | $1+x+x^4$ |
| $\alpha^3, \alpha^6, \ldots$ | $1+x+x^2+x^3+x^4$ |
| $\alpha^5, \ldots 1+x+x^2$ | |

The same strange things can be found in our well known decimal system: The minimum polynomial of 5 is (x-5), but the minimum polynomial of the complex number 1+j is $(x^2-2x+2)$ if the coefficients should be real values.

It now turns out that valid code words are binary polynomials with certain roots in GF($2^m$). The fact that the code word $(c_0, c_1, c_2, \ldots, c_{n-1})$ has got the root $\beta$ might be rewritten as

$$c(\beta) = c_{n-1}\beta^{n-1} + \ldots + c_1\beta^1 + c_0\beta^0 = 0 \tag{20}$$

$$c \cdot \begin{bmatrix} \beta^{n-1} \\ \vdots \\ \beta^1 \\ \beta^0 \end{bmatrix} = c \cdot H^T = 0 \tag{21}$$

$\beta$ is a number of the Galois field. Hence all elements $\beta^i$ are defined and may be represented by bits as shown above. Therefore the parity matrix H is defined leading to an already known block code. This is the case if the code should be a one-bit-correcting code.

## 2.5 BCH-Codes

BCH is short for Bose, Chaudhure and Hocquenghem. BCH-codes offer a systematic approach to a t-error-correcting code with a desired code size N. The code is built inside a GF(N+1). The generator polynomial is the least common multiple of minimum polynomials for t independent roots. Assume a

code with a desired Hamming distance of 3, therefore it will be capable of correcting 2 bit errors. Hence the two independent roots have to be found, one choice would be $\alpha$ and $\alpha^2$. Unfortunately $\alpha^2$ is not independent to $\alpha$ since they both share the same minimum polynomial. Thus $\alpha^3$ is chosen to be the second root. For every code word c(x)

$$c(\alpha) = 0 \tag{22}$$
$$c(\alpha^3) = 0 \tag{23}$$

have to be met. The generator polynomial yields

$$g(x) = (1+x+x^4)(1+x+x^2+x^3+x^4) = 1+x^4+x^6+x^7+x^8 \tag{24}$$

It is the polynomial of smallest order with the two roots $\alpha$ and $\alpha^3$. Since all code words are multiples of the generator polynomial, there all will have these two roots as well. The parity matrix H becomes

$$H^T = \begin{bmatrix} \alpha^{12} & \alpha^{14} \\ \vdots & \vdots \\ \alpha^6 & \alpha^2 \\ \alpha^3 & \alpha^1 \\ \alpha^0 & \alpha^0 \end{bmatrix} \tag{25}$$

The encoder may be realised by a shift register with feedback with the given generator polynomial. The syndrome computation is done like in simpler block codes.

$$s = y \cdot H^T = e \cdot H^T = e \cdot \begin{bmatrix} \alpha^{12} & \alpha^{14} \\ \vdots & \vdots \\ \alpha^6 & \alpha^2 \\ \alpha^3 & \alpha^1 \\ \alpha^0 & \alpha^0 \end{bmatrix} \tag{26}$$

Two errors at positions i and j lead to a syndrome word $s = (s_1, s_2)$ with

$$s_1 = \alpha^i + \alpha^j \tag{27}$$
$$s_2 = \alpha^{3i} + \alpha^{3j} \tag{28}$$

These are two equations with two unknown variables i and j. The two positions i and j are determined usually by exploiting lookup tables, the appropriate bits are toggled afterwards in order to correct the error.

## 2.6 Reed Solomon Codes

Reed Solomon Codes (RS codes) are non-binary BCH codes. All the symbols processed inside the polynomials and shift registers are not bits any longer, they are now m bits wide. $2^m - 1$ symbols are combined to one block. Since all variables inside the polynomials are symbols with m bits, they can represent values between 0 and $\alpha^{2^m-2}$. Hence coefficients of minimum polynomials are located in this range as well. As an example a byte-based RS code for 2-bit-error-correction is shown here - two roots are necessary in this case. The generator polynomial would be

$$g(x) = (x+\alpha^7)(x+\alpha^{14}) = x^2 + \alpha^{119}x + \alpha^{21} \tag{29}$$

This leads to a feedback shift register consisting of 2 stages with $g_0 = \alpha^{21}$ and $g_1 = \alpha^{119}$. Of course bytes are processed during each cycle, therefore the stages in the shift register store whole bytes and the additions and multiplications operate with bytes as well.

The coding gain of an RS code based on a 16 QAM line code with 10% redundancy and different error correction capabilities is shown in Figure 6. Significant coding gain of several dBs may be achieved.

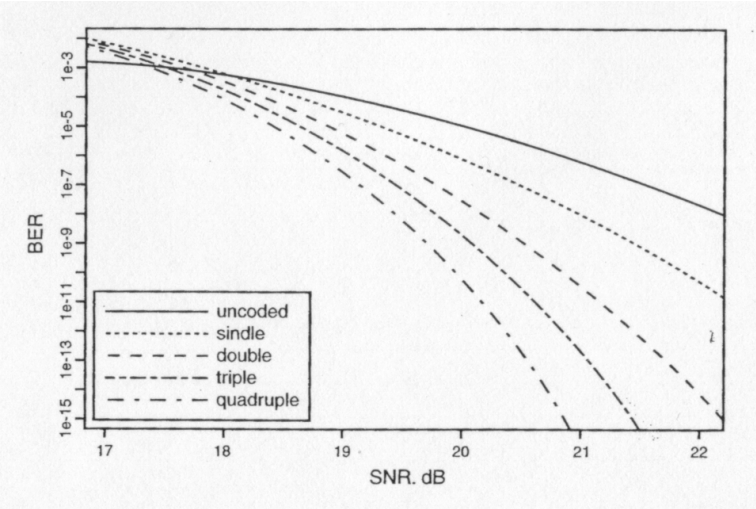


Figure 6: Coding gain of a Reed Solomon code with different bit error correction capabilities

# 3 Convolution Codes

The second big family of codes beside block codes are convolution codes. Their name comes from the fact that the output data stream of an encoder is the result of a convolution operation of the input data stream and the impulse response of the code.

## 3.1 Principles

Figure 7 shows the principle of an encoder for a convolution code. k input bits are fed into a shift register with M-1 stages. Each of the n output bits are generated from the input bits and their delayed values are stored in the shift register. If only one 1-bit is input to this encoder, a certain bit stream will appear on its output. This bit stream is the impulse response of the code. In the more general scenario with more than one 1-bit inside the input stream, this shift register simply performs a convolution operation between the input and the impulse response mentioned before. A short terminology for convolution codes is introduced: Such a code is called (n,k,M)-code; very similar to block code terminology.

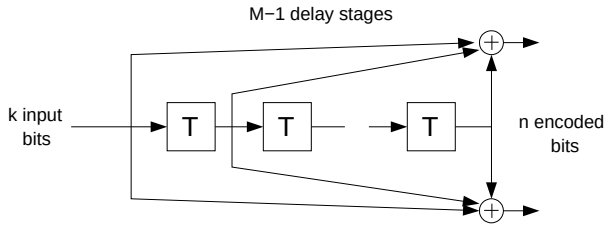


Figure 7: Encoder for a convolution code

As an example an encoder for a (3,2,2) convolution code is presented in Figure 8. Three encoded bits (n=3) are generated out of two information bits (k=2) resulting in a code rate 2/3. There is no clear boundary between information bits and redundancy bits at the output as it is in a block code. The redundancy is mixed into the output data stream permanently.

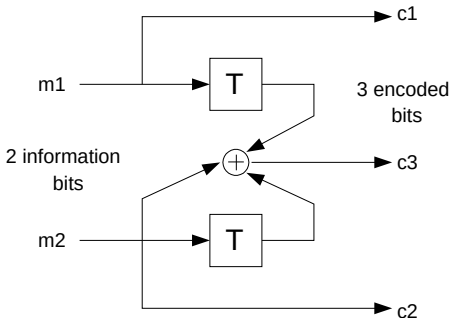


Figure 8: Encoder for a (3,2,2) convolution code

Similar to block codes a generator matrix may be introduced:

$$g(D) = \begin{bmatrix} 1 & 0 & D \\ 0 & 1 & 1+D \end{bmatrix} \tag{30}$$

where D denotes a delay of one clock cycle. However, other possibilities of a code description exist which are very useful for developing the appropriate decoder for such a code. First, a state transition diagram may be derived from the generator matrix (Figure 9). The state of the encoder is defined by the bits stored inside the shift register. The encoder in Figure 8 contains two 1-bit registers, hence there are 4 possible states $S_0$ to $S_3$. During each cycle two new input bits are fed into the encoder leading to 4 possible transitions from an old state to the new one. Each transition is directly linked to a certain combination of 3 bits at the output. This is denoted as xx/yyy in Figure 9 where xx are the 2 input bits and yyy are the 3 output bits, respectively.

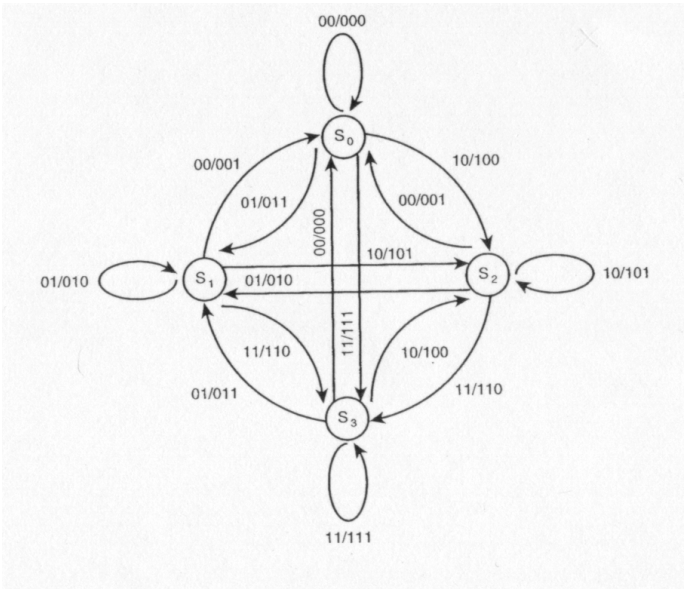


Figure 9: State transition diagram

The state transition diagram may be unfolded for each clock cycle resulting in a so called trellis diagram as shown in Figure 10. Again, there are 4 possible states. But this time the transitions point

towards 4 new states for the next clock cycle. A trellis diagram is well suited for the explanation of the Viterbi decoding algorithm.

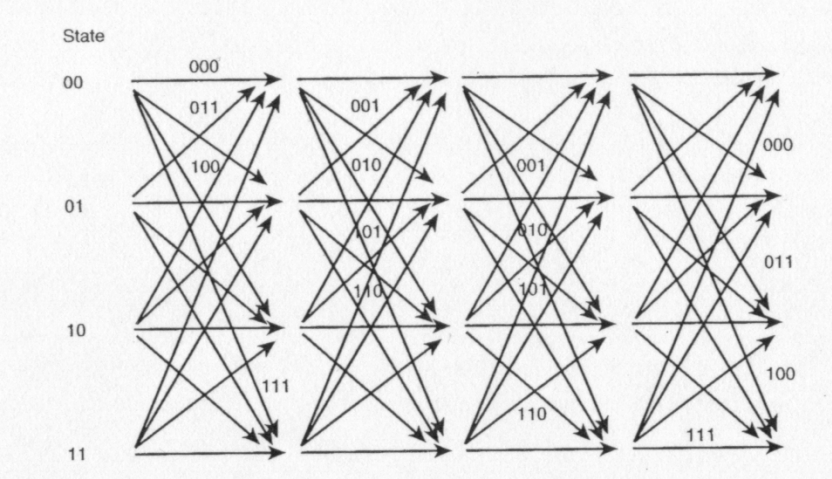


Figure 10: Trellis diagram

## 3.2 Viterbi Decoding

First consider the trellis diagram of the example code. Let $S_0$ be the initial state of the encoder and decoder. Assuming that the first three received bits are 011, the branch metrics for each transition starting at $S_0$ may be calculated. There are two major principles of how to determine the branch metrics. The easiest way is to quantise the input into 0 or 1. This solution is presented in Figure 11. The branch metrics is simply the number of different bits between the real input data and the data bits that would be necessary for the given transition. In our example the transition from $S_0$ to $S_1$ fits perfectly, whereas the transition from $S_0$ to $S_3$, for instance, has a metric of 1.

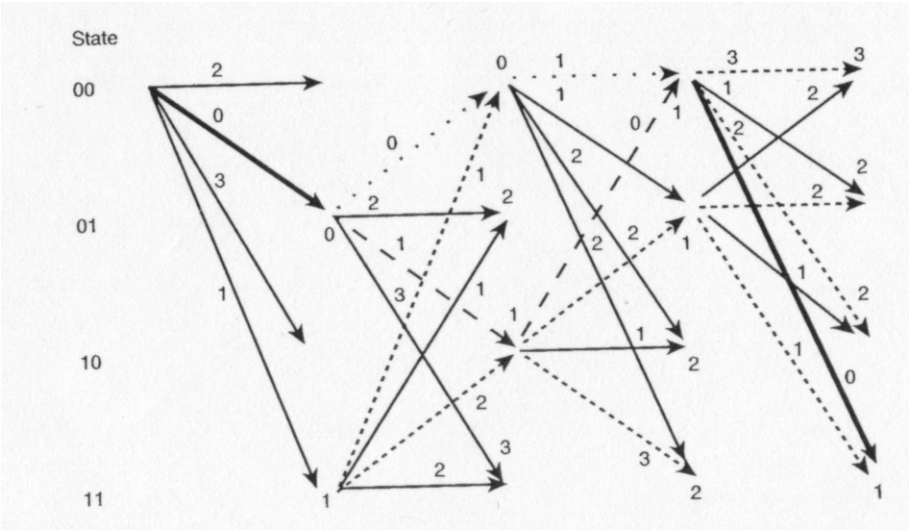


Figure 11: Viterbi decoding algorithm for quantised input: r=(011 001 001 111)

In the next clock cycle another 3 bits are received. Again the branch metrics can be determined by adding the metrics of every original state to the metrics of the transition. If one looks at a certain new state, there are many possible branches that end at this new state. However, there is always a branch with a minimum branch metrics. Only this branch has to be stored and its branch metrics is linked to the state. In the next clock cycle only this branch has to be considered for the new calculation of the branch metrics. The idea behind this so called Viterbi decoding algorithm is that it is only necessary to store the branch with the minimum branch metrics that ends at a certain end state. All the other possible branches do not survive because they will never lead to a smaller branch metrics in the future. After a while only one branch will survive defining the decoded bit stream.

The code gain may be further increased by introducing soft inputs (Figure 12). In this case the Viterbi decoder also gets the information how good a received bit was a 1 or 0. The input is not only (0 1 1) but for instance (0.2 1.4 0.7). This might be realised by an ADC with more than 1 bit resolution. Here the branch metrics is calculated as $B_M = \sum(r_i - \beta_i)^2$ where $r_i$ is the soft value of the received bit and $\beta_i$ is the necessary bit value for the transition. The information about the received bits is finer, hence it is easier for the decoder to determine the right bit stream.

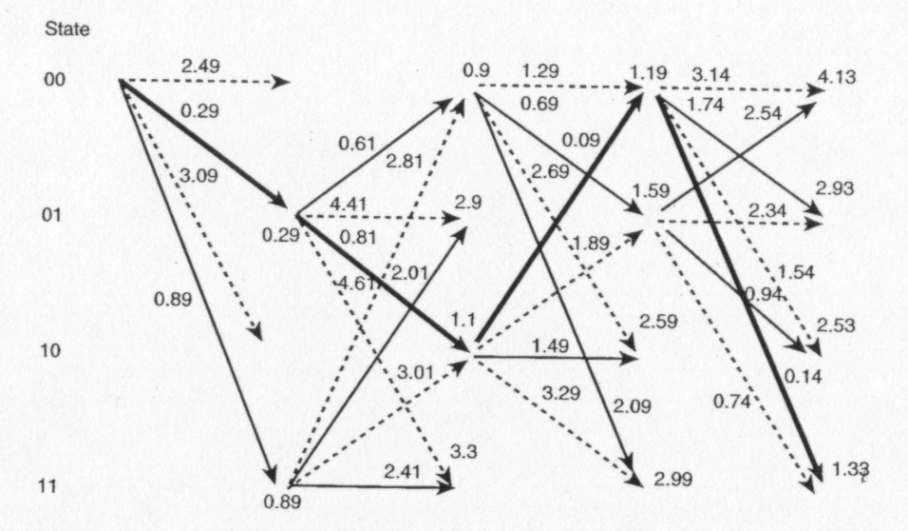


Figure 12: Viterbi decoding algorithm for soft input: r=(0.2 1.4 0.7; 0.4 -0.3 1.6; -0.2 0.2 1.1; 1.3 0.9 0.8)

## 3.3 Trellis Code Modulation

As mentioned earlier the added coding overhead or redundancy leads to an increased transmission throughput. There are two possible solutions to that problem. One could increase the transmission bandwidth if possible. Due to this higher bandwidth the SNR is decreased, but typically by less than 1.5dB. However, in many communication systems increasing the transmission bandwidth is not possible. In this case the number of symbols in the line code has to be expanded. Unfortunately this results in a SNR reduction of typically 3 or 6dB, if no additional measures like a sophisticated bit-to-signal mapping are introduced.

In general the Hamming distance between two code words do not determine the error possibility. But there is a close relationship between the Euclidean distance and the likelihood of false detection (Figure 13). The big dots denote the positions of the valid symbols of a given line code with more than 2 symbols. This is called the constellation structure of the line code. The transmitter outputs one symbol per cycle with its exact position. On the channel noise is added to the original symbol value. If it is a Gaussian channel, the noise amplitude is Gaussian distributed. The variance of this distribution is related to the noise power. Somewhere in the middle between two symbols a threshold value $V_{diff}$ is defined. Hence the part of the Gaussian distribution that excesses this threshold leads to an error in detection. The error possibility is related to the distance between the two symbols - this distance is called Euclidean distance.

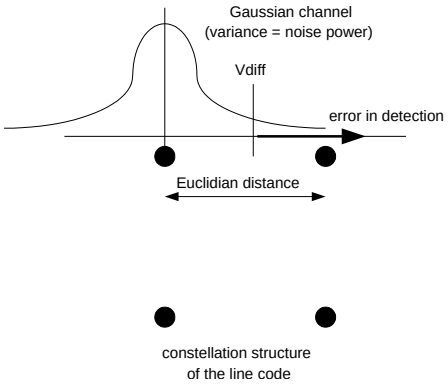


Figure 13: Euclidean distance of symbols in a constellation structure of a line code

In Figure 14 a (3,2,2) convolution encoder is shown. Three output bits $y_0$ to $y_2$ are generated from two input bits $x_1$ and $x_2$. $y_2$ is always equal to $x_2$, $y_1$ is the delayed $x_1$. But $y_0$ is the output of a convolution encoder, hence it contains more “security” against bit errors.

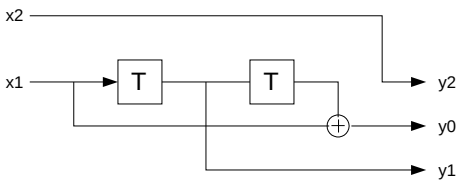


Figure 14: (3,2,2)-encoder for a trellis code

Assume that a line code with pulse amplitude modulation (PAM) with 8 different signal levels is used (Figure 15. Each level is associated to a unique stream of 3 bits. It is most likely that $y_0$ is distorted by noise since adjacent signal levels or symbols only differ in the value of $y_0$. However, bit errors in $y_0$ may be corrected leading to better coding gain.

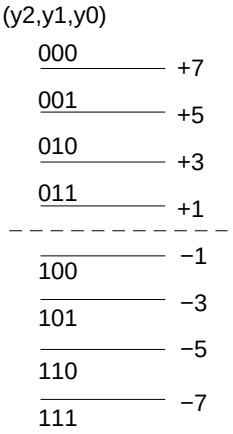


Figure 15: PAM bit-to-signal mapping

A more complicated constellation set mapping for a 16 symbol line code with quadrature amplitude modulation (QAM) is shown in Figure 16. First the 16 symbols are divided into two sets of 8 symbols. Each set is related to $y_0$ equal to 0 or 1, respectively. The minimum Euclidian distance between all the 16 symbols is $\Delta_0$, which is increased to $\sqrt{2}\Delta_0$ inside each set. Hence bit errors inside each set are more unlikely. The effective BER of $y_0$ may be improved if $y_0$ is secured by a convolution code.

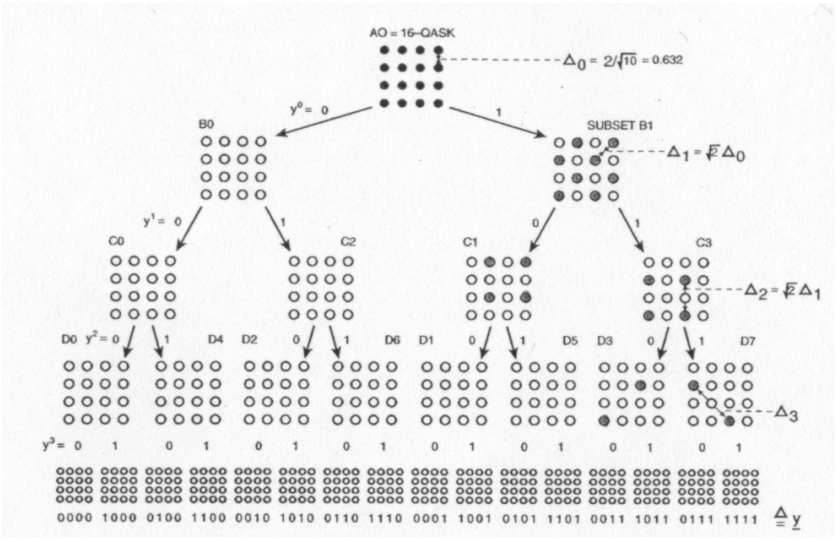


Figure 16: 16 QAM constellation set partitioning for a trellis code

## 3.4 Phase Invariant Trellis Encoder

The symbol clock in DSL systems is not transmitted via a special line. Hence it must be recovered inside the receiver out of the data stream (Figure 17). Unfortunately it cannot be ensured that the input signal contains a peak at the carrier frequency $f_c$. However, if the input signal is squared, the resulting signal has got a peak at $2\,f_c$. A phase locked loop (PLL) may be used to lock onto this peak. Its output is divided by two and subsequently sourced to the QAM demodulator.

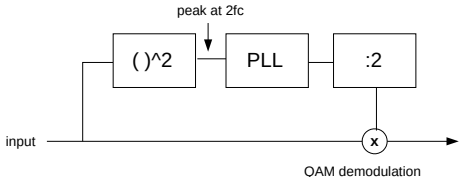


Figure 17: Block diagram of a carrier recovery

This division by two leads to an uncertainty in phase of 0 or 180 degrees of the recovered carrier. One solution is to transmit a preamble right in front of the data stream. If the bits of the preamble are inverted, the phase is wrong and has to be changed. Another possibility is to exploit a phase invariant trellis code (Figure 18). The output bits $y_2$ to $y_4$ are exact copies of 3 input bits $x_2$ to $x_4$, whereas $y_0$ and $y_1$ are derived from one input bit $x_1$. A differential encoder is situated in front of the convolution encoder. Hence $y_0$ and $y_1$ are only related to changes of $x_1$.

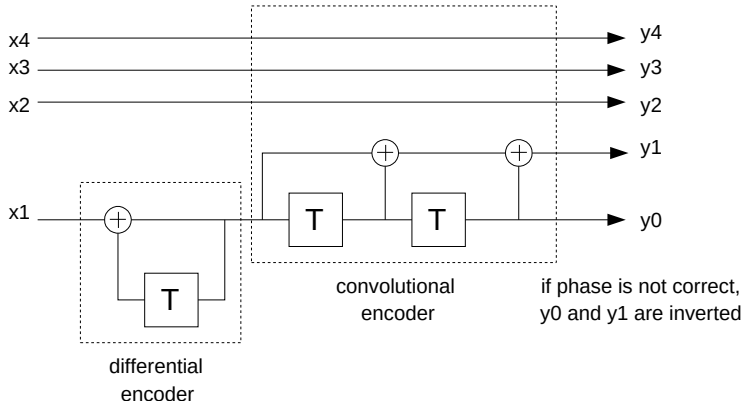


Figure 18: Phase invariant encoder with differential encoder

The four state constellation mapping of Figure 19 is 180 degrees invariant as far as the first three output bits $y_2$ to $y_4$ are concerned. If the phase is incorrect, the mirrored symbol around the origin will be detected leading to inverted bits $y_0$ and $y_1$. Since these bits only describe changes of the bit $x_1$, this inversion does not matter. Therefore, the whole communication system is invariant to a 180 phase shift of the clock carrier.

# 4 Line Codes

This short section summarises different DSL systems and the codes being used.

- DSL: Data rate of 160kbps (bits per second). Line code is a 2B1Q-code, this is short for 2 input bits are mapped to one 4-valued symbol.
- HDSL: Data rate of 800kbps. Again, a 2B1Q-code is used as line code with no error correction capabilities.
- ADSL: Data rate of 2208/276kbps. It is a discrete multi-tone (DMT) system with 512/64 channels. Trellis coding with a (3,2,4)-convolution-code is exploited. Reed Solomon codes with symbols of 1 byte are used for error correction.

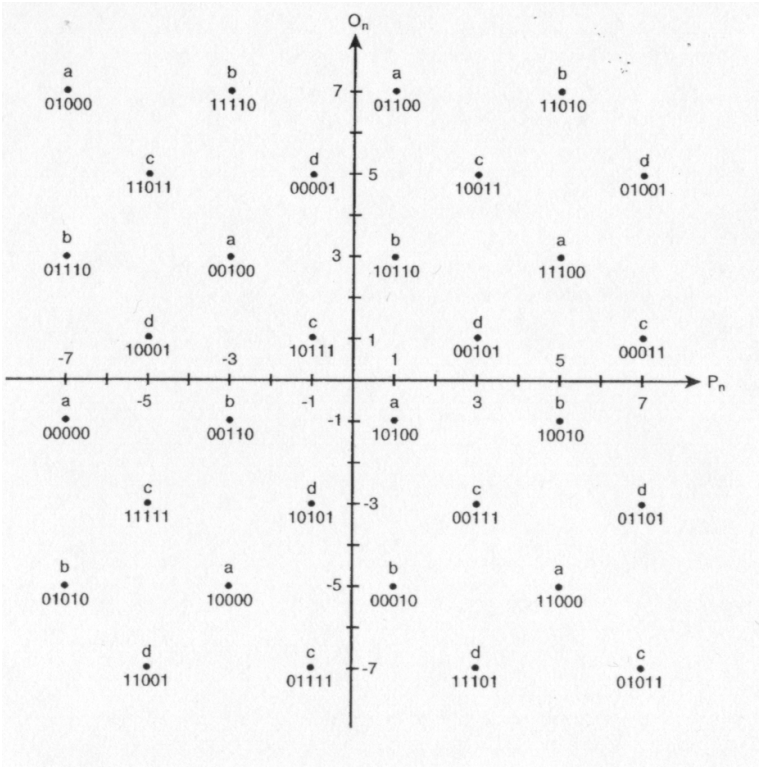


Figure 19: Four-state 180 degrees phase invariant constellation mapping

# 5 References

Walter Y. Chen. DSL Simulation Techniques and Standards Development for Digital Subscriber Line Systems. Indianapolis, MacMillan Technical Publishing, 1998.

Galois fields:

- William E. Cherowitzo. Lecture notes on Coding Theory and Cryptology.
  Online available at http://www-math.cudenver.edu/-wcherowi/courses/m5410/ctcln.html.
  Mathematics Department, University of Colorado, Denver.
- Alex Grant. Lecture notes on Error Control Coding.
  Online available at http://www.itr.unisa.edu.au/-alex/ECC/notes_contents.html.
  Institute for Telecommunications Research, University of Southern Australia, Adelaide.
- Matlab Communications Toolbox - Galois Field Computations.
  http://www.mathworks.com/access/helpdesk/help/toolbox/comm/tutor3.shtml.

Reed Solomon Codes:

- Martin Riley, Iain Richardson. Reed Solomon Codes.
  Online available at http://www.4i2i.com/reed_solomon_codes.htm.
- Adina Matache. Encoding/Decoding Reed Solomon Codes.
  Online available at http://drake.ee.washington.edu/-adina/rsc/slide/slide.html.
  Department of Electrical Engineering, University of Washington, Seattle.