ON THE GENERATING FUNCTION AND ITS APPLICATION TO TERNARY LINE CODES

K. Ouahada, H. C. Ferreira

Department of Electrical and Electronic Engineering Science, University of Johannesburg, South Africa
E-mail: $\{kouahada, hcferreira\}@uj.ac.za$

Abstract: One of the major problems researchers face is how to verify experimental or simulation results of any class of codes. Usually this can be done by deriving bounds to test the performance of codes. As they are used in binary codes and especially in linear convolutional codes, where the sum of any two codewords is another codeword and the all-zeros sequence is a codeword, the major step to derive these bounds is to calculate the generating function. Since ternary line codes are non-linear line codes, we will investigate in this paper the possibility of deriving the generating function for at least some of these codes.

Key Words: Ternary line codes, AMI, HDBn, BnZS, transfer function, generating function

1. INTRODUCTION

Ternary line codes [1] such as the High-Density Bipolar of order n (HDBn) and Binary n-Zero Substitution (BnZS) have been used in PCM metallic cable systems worldwide. They are relatively simple and satisfy tight channel input restrictions, which make them suitable for use in baseband communications systems.

Ternary line codes are considered as DC-free codes, they do not contain zero-frequency components and do have low-frequency components. The clock recovery and synchronization [2] is easy since, when using the NRZ pulses, the frequent transitions between symbols that the code sequence has, enable the encoder clock signal to be derive from the code sequence at the receiver.

All textbooks and papers use linear convolutional codes as the encoders to study the behaviour of Viterbi decoding [3]–[5] and to present the well known $\sim$ 2 dB asymptotic gain of soft decisions over hard decisions on wideband Gaussian channels.

In their paper [6], Ouahada and Ferreira investigated the performance of ternary line codes under Viterbi decoding and verified the simulation set-up to enhance the confidence in their results. They subjected their simulation set-up to several tests to verify its technical correctness and assess its numerical accuracy. They first tested their noise-generator results by writing a Matlab program to fit the Gaussian probability density function to the measured results by adjusting different parameters to obtain a better fit for better accuracy. They also double checked the 2 dB gain of the soft decision Viterbi decoding over the hard decision decoding by using a well known text book example of convolutional code [3].

Although the derivation of the upper bounds on the error probability for non-linear codes has been investigated in the context of trellis codes [7, 8], we investigate in this paper a different technique of verification which is usually used with binary convolutional codes. This technique is based on the derivation of the generating function to calculate the decoding error probability upper bounds of the soft and hard decisions, in order to be able to verify the 2 dB gain between them.

We present the concept of the generating function or the transfer function, as it is called in control systems [9], and show how it is possible to apply it to coding techniques and more specifically to ternary line codes [1, 10].

The generating function or transfer function of a line code, which was first introduced for convolutional codes [3], contains all the necessary parameters such as the value of the distances between the expected and received data, the length of the path on the trellis and the inputs of the message. It gives the properties of all the possible paths that any code can assume. The generating function will be used for the research into the upper bound for the probability of errors of Viterbi decoding. It can be derived by Mason's rule [11], which is explained as follows:

The overall transfer function of a flow graph can be obtained from the formula:

$$T = \frac{\sum_k (T_k \Delta_k)}{\Delta}, \tag{1}$$

where:

* T_k is the transfer function of each forward path between a source and a sink node.

* $\Delta = 1 - \sum L_1 + \sum L_2 - \sum L_3 + \cdots$, is the determinant [11] of the whole graph.

* $\sum L_1$ is the sum of the transfer functions of each closed path.

* $\sum L_2$ is the sum of the product of the transfer functions of all possible combinations of two non-touching loops. We are given three non-touching loops: 1, 2 and 3. Then $\sum L_2 = 1 \times 2 + 1 \times 3 + 2 \times 3$.

* $\sum L_3$ is the sum of the product of the transfer functions of all possible combinations of three non-touching loops. We are given three non-touching loops: 1, 2 and 3. Then $\sum L_3 = 1 \times 2 \times 3$.

* Δ_k is the cofactor of T_k. It is the determinant of the remaining sub-graph when the forward path, which produces T_k, is removed. Thus, it does not include any loops that touch the forward path in question. It is equal to unity when the forward path touches all the loops in the graph, or when the graph contains no loops.

Section 2 introduces the error probability of transmitted data for convolutional codes in soft and hard decision Viterbi decoding. In Section 3, a new code combining two properties of two different codes from two different classes is presented. The new code will be used to understand the behaviour of certain ternary line codes when they are subjected to Viterbi decoding algorithm. Section 4 presents a generalization of the concept of generating function. We conclude with some final remarks.

2. PROBABILITY OF ERROR IN TRANSMISSION

2.1. *Binary Convolutional Codes*

2.1.1. *Probability of error for soft-decision Viterbi decoding*

In this section we make use of the encoder of the convolutional code to study the error rate performance of Viterbi decoding on an additive white Gaussian noise channel with soft-decision decoding [3]. Many factors must be known before we can begin calculating the error probability. These include the linearity property of the convolutional encoder which will be employed to simplify the derivation, the all-zero path (reference path) to calculate the metrics, and the type of modulator used in the transmission. These are crucial factors for the calculation of any error probability.

It is assumed that the transmitted data is modulated by using the Binary Phase-Shift Keying (BPSK) [12] modulation and detected coherently at the demodulator. So the probability of error, denoted by P_2 on a binary symmetric channel (BSC), in the pair-wise comparison of two paths that differ in d (distance metric compared to the reference path) bits is [13]:

$$P_2(d) = 0.5\text{erfc}(\sqrt{\gamma_b R d}), \tag{2}$$

where γ_b is the signal to noise ratio per bit; and R is the code rate.

The time domain convolutions can be conveniently replaced by polynomial multiplications in D-transform domain and after calculating the generating function, we obtain the following form:

$$T(D, N) = \sum_{d=d_{free}}^{\infty} a_d D^d N^{f(d)} \tag{3}$$

where a_d and N are respectively the number of encoded sequences of weight d from the all-zero path and the information block length. The function $f(d)$ denotes the exponent of N and it depends on the value of d. As an example for a convolutional code with a constraint length $K = 3$ and a rate of $R = 1/2$, where its free distance is $d_{free} = 5$, we have $a_d = 2^{(d-5)}$ and $f(d) = d - 4$.

Taking the derivative of $T(D, N)$ with respect to N and setting $N = 1$, we obtain:

$$\frac{\partial T(D, N)}{\partial N} = \sum \beta_d D^d \tag{4}$$

where $\beta_d = a_d f(d)$.

The successful completion of the above calculation brings us close to our main purpose, which is the calculation of the upper-bound error probability.

Once we have determined the upper-bound error probability for the soft and hard decisions, it becomes possible to verify the well-known 2 dB gain improvement of soft decisions over hard decisions.

$$P_b < \sum \beta_d P_2(d). \tag{5}$$

Taking into consideration the error probability in (2), (5) can be written as:

$$P_b = 0.5 \sum \text{erfc}(\sqrt{\gamma_b R d}). \tag{6}$$

By using the derivative of the generating function, (6) can be expressed simply as follows:

$$P_b < 0.5 \frac{\partial T(D, N)}{\partial N}, \tag{7}$$

with $N = 1$, and $D = exp(-\gamma_b R)$ since $erfc(\sqrt{x}) < exp(-x)$.

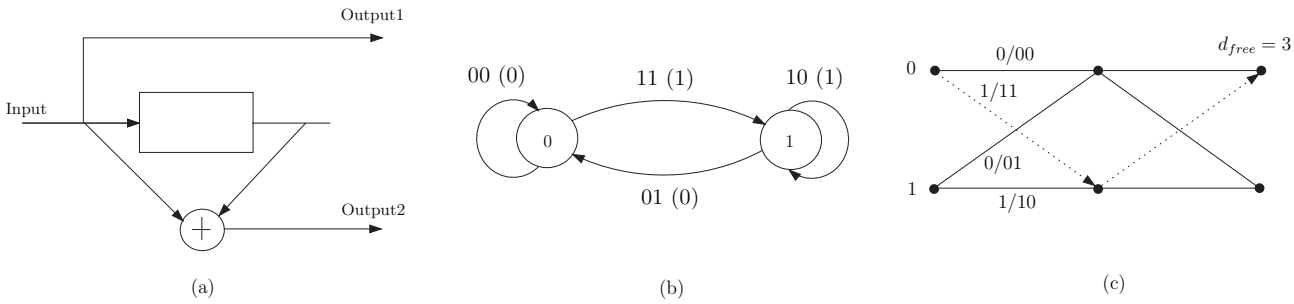

Figure 1: Convolutional encoder: $R = 1/2$, $K = 2$, $d_{free} = 3$

2.1.2. *Probability of error for hard-decision Viterbi decoding*

On a binary symmetric channel, the hard-decision decoding of the binary code will be used to study the performance of the channel's Viterbi decoding. The Hamming distance will be used for the calculation of the metrics at each node of the trellis.

Using the same notation as for soft decisions, the distance d refers to the distance from the all-zero path. It is important to know whether d is odd or even, to get an understanding of the formula of the error probability that will be used later. This principle is outlined in the formula below [13]:

If d is odd, the error probability of selecting the incorrect path is:

$$P_2(d) = \sum_{k=\frac{(d+1)}{2}}^{d} \binom{d}{k} p^k (1-p)^{d-k}, \tag{8}$$

and if d is even, the error probability of selecting the incorrect path is:

$$\begin{aligned} P_2(d) = &\sum_{k=\frac{d}{2}+1}^{d} \binom{d}{k} p^k (1-p)^{d-k} \\ &+ 0.5 \binom{d}{d/2} p^{d/2} (1-p)^{d/2}, \end{aligned} \tag{9}$$

where p is the transition probability.

By summing the pair-wise error probabilities, $P_2(d)$, over all possible paths, which merge with the all-zero path at the given node and by using the Chernoff bound [14, 15] on the pairwise error probability as $P_2(d) < [2\sqrt{p(1-p)}]^d$, we obtain the union bound:

$$P_b < \sum_{d=d_{free}}^{\infty} a_d [2\sqrt{p(1-p)}]^d. \tag{10}$$

Using the generating function directly, the error probability upper bound can be calculated as follows:

$$P_b < T(D, N), \tag{11}$$

where $D = 2\sqrt{p(1-p)}$.

Example: In this example we present a binary convolutional code with only two states and a rate of $R = 1/2$ [13]. This code was chosen because of its simplicity and the close resemblance to the Alternate Mark Inversion (AMI) [13] line code, which will be studied further as a simple study case for ternary line codes.

Fig. 1(a) shows the block diagram of our convolutional encoder with two states.

As can be seen from the model of the convolutional encoder presented in Fig. 1(a), the block diagram has one shift-register bit that lead us to a constraint length, $K = 2$, with two outputs. Thus the encoder has a code rate of $R = 1/2$. The state diagram shown in Fig. 1(b) explains the method to obtain the distance properties of the convolutional code that we have. It is clear from the trellis diagram presented in Fig. 1(c) that the minimum distance (free distance) is $d_{free} = 3$.

To obtain the generating function, it is necessary to label the branches of the state diagram as D^0, D^1, D^2, D^3, where the exponent of D denotes the Hamming distance of the sequence of output bits compared to the sequence of output bits on the all-zero branch [13]. As mentioned earlier, the generating function is obtained by Mason's rule, so it is necessary that an input and an output be defined in the state diagram to be able to calculate the gain, which is the transfer function divided by that input to that output. From here, the self-loop, which the connection between

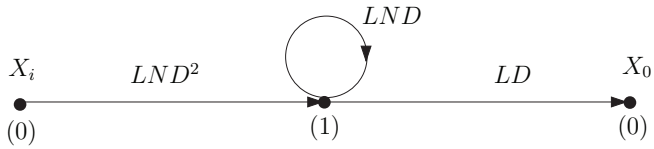

Figure 2: Flow graph of convolutional encoder: $R = 1/2$, $K = 2$, $d_{free} = 3$

the same state, at node (0) can be eliminated, since it makes no contribution to the distance properties of the code sequence relative to the all-zero code sequence. Node (0) is split into two nodes: one represents the input; the other represents the output of the state diagram.

Definition 1 The all-zero path is all branches in the trellis that correspond to the self-loop of state 00. □

Definition 2 The term L in the generating function determines the length of a given path. The exponent of L, reflects the number of branches in the path. □

Definition 3 The term N is included in the generating function only if the branch transition is caused by an input data of "one" for the all-zero path. □

Fig. 2 shows the state diagram labelled according to the distance from the all-zero paths, lengths and numbers of input ones. The terms X_0 and X_i represent respectively the output and the input of the state diagram. We denote by $T_{\text{conv}}(D, N, L)$ the generating function corresponding to all codewords beginning with a non-zero branch for the convolutional encoder and defined as:

$$T_{\text{conv}}(D, N, L) = \frac{X_0}{X_i}, \tag{12}$$

From Fig. 2, we see that we have:

$$X_{(1)} = LND^2X_i + LNDX_{(1)}, \tag{13}$$

$$X_0 = LDX_{(1)}, \tag{14}$$

$$T_{\text{conv}}(D, N, L) = \frac{L^2ND^3}{(1 - DNL)}, \tag{15}$$

From (15) we can expand the denominator using Maclaurin [16] series, which leads to the following:

$$\begin{aligned} T_{\text{conv}}(D, N, L) =& D^3L^2N + D^4L^3N^2 + D^5L^4N^3 \\ &+ \cdots + D^{3+n}L^{2+n}N^{1+n} + \cdots \end{aligned} \tag{16}$$

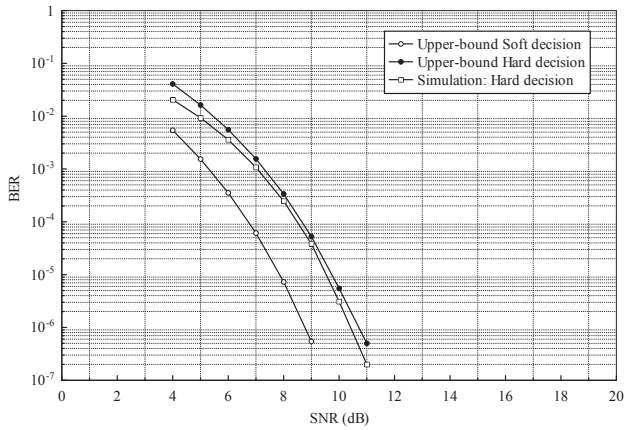

Figure 3: Upper bounds for convolutional encoder: $R = 1/2$, $K = 2$, $d_{free} = 3$

This generating function will provide the properties of all the paths in the convolutional code which start from state 0 with a non-zero branch and end in state 0. The first term in (16) indicates that the minimum distance is $d = 3$, which is represented by the exponent of D, the corresponding path is of length 2, represented by the exponent of L; and out of two information bits, one is equal to 1 hence the exponent of N is 1. The second term in the expansion of $T_{\text{conv}}(D, N, L)$ indicates that when the minimum observed distance is $d = 4$, the corresponding path is of length 3 and two of three information bits in the path have the value 1. In the case where the sequence in the transmission is extremely long, essentially an infinite-length sequence, it is recommended that the generating function $T_{\text{conv}}(D, N, L)$ in (16) will be simplified by setting $L = 1$, which leads to:

$$\begin{aligned} T_{\text{conv}}(D, N, L) =& D^3N + D^4N^2 + D^5N^3 + \cdots \\ &+ D^{3+n}N^{1+n} + \cdots, \end{aligned} \tag{17}$$

and by writing the terms in (17) in a sigma form, we obtain:

$$T_{\text{conv}}(D, N) = \sum_{d=3}^{\infty} N^{d-2}D^d. \tag{18}$$

A Matlab program is used to plot the curves of the upper bounds for hard and soft decisions and to check the 2 dB gain improvement between them. On the same graph the simulation results are presented in order to be compared to theoretical upper bounds. We have just limited our results to hard decisions since soft decisions are presented theoretically.

Fig. 3 shows the 2 dB gain between soft decisions and hard decisions for the convolutional encoder $R = 1/2$,

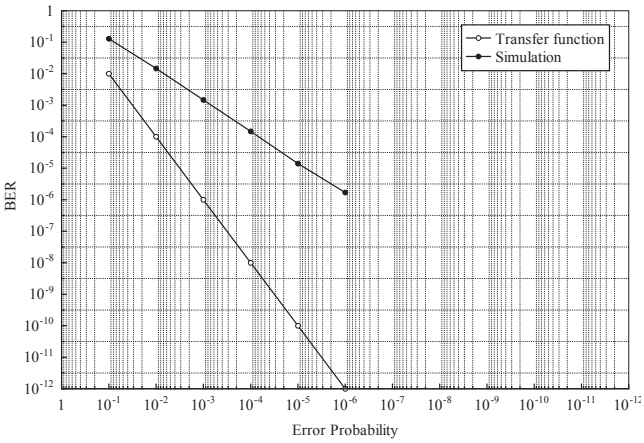

Figure 4: Error curves for the AMI code

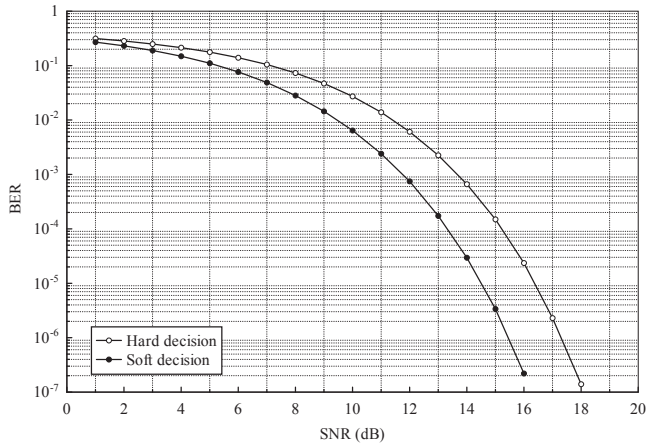

Figure 5: Soft and Hard decision Viterbi decoding for AMI

$K = 2$. We observe the closeness of our simulation for hard decisions to the theoretical hard-decision upper bound. These results make us positive about the way in which we use the same approach to calculate the error probability upper bound for our line codes.

2.2. *Ternary Line Codes*

This class of codes differs from the binary convolutional codes in terms of linearity and negative outputs that the encoder generates. Thus more complexity appears in calculations to determine the distances for Viterbi algorithm.

Example: We take the example of AMI, because of its simplicity.

Since the outputs of the AMI are positive and negative values [17], the Hamming distance will be problematic for calculating the weight of distances. Thus Euclidean distance is the appropriate choice for this class of codes.

AMI is a line code with 1Mb/s as a bandwidth and it does not need a modulator to send the message through the channel, which is not the case with convolutional codes where we use PSK modulation. Also, it must be noted that AMI is a non-linear code, while the generating function approach was developed for linear convolutional codes. And this makes it difficult for us to double check our simulation results with a theoretical background, as was done in Example 2.1.2 with the convolutional code.

From the above discussion, it can be predicted that a perfect match between the simulation and the error probability upper bound for the AMI results cannot be achieved.

Since AMI code is the simplest ternary line code to understand and to design in view of its close state machine structure as the case for convolutional codes, we assume that the derivation of its corresponding generating function is achievable. Therefore, we plotted the approximated curves, comparing the simulation results of AMI [18] to its generating function as shown in Fig. 4. It is clear from Fig. 4, as it was predicted, that the difference between the two plotted lines can be clearly seen. Thus, the assumption that a ternary line code, as the case of AMI, can be considered or treated as line code is actually proved to be wrong.

Using only simulation results [6], Fig. 5 shows the gain between soft decisions and hard decisions for AMI [19], which is almost equal to 2 dB at the bit error rate (BER) value of 10^{-6}. This result is very important and shows the accuracy of our simulation set up. To check the theoretical upper-bounds for soft and hard decisions, we need to look at another technique based on certain approaches that can verify and back up our simulation results. This will be described in the following section.

3. CONVO–AMI: A NEW "LINEARISED" LINE CODE

The generating function is often used to obtain the upper bound for the error probability of Viterbi decoding, and is readily obtained for linear convolutional codes. However, this is not the case for the non-linear ternary line codes that we present in this paper. A new line code called Convo-AMI was therefore developed in order to apply existing theory to these ternary line codes.

To obtain the corresponding Convo-AMI code, the negative symbols of the AMI are converted to binary using the 2-bit two's complement rule. The obtained code is now considered as a binary code with two bit output corresponding to each one input bit. The new binary

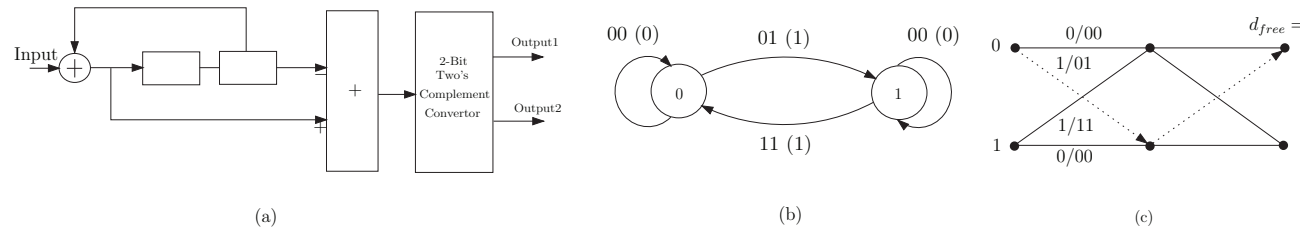

Figure 6: Convo-AMI encoder: $R = 1/2$, $K = 2$, $d_{free} = 3$

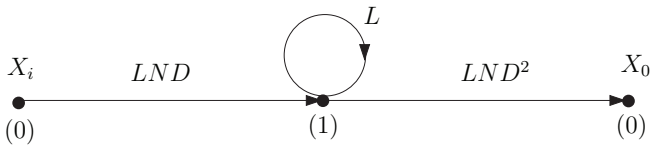

Figure 7: Flow graph of the Convo-AMI code, $d_{free} = 3$

code with two states is considered as a new convolutional code with rate $R = 1/2$ and constraint length $K = 2$. The new Convo-AMI code, with block digram presented in Fig. 6(a) has a similar state machine structure to the previously studied convolutional code as shown in Fig. 6(b).

Using the Convo-AMI, we were able to obtain a tight upper bound on the error probability using the generating function. Our new code retained certain characteristics of the AMI as the number of states and the same outputs in a binary form and also has a minimum distance, $d_{free} = 3$, as depicted in Fig. 6(c). This makes the new code similar to the convolutional code of constraint length $K = 2$ presented in Fig. 1. The behaviour of the new code however, imitates a binary convolutional code.

$T_{\text{Convo-AMI}}(D, N, L)$ denotes the generating function of the new code, Convo-AMI encoder, defined as follows:

$$T_{\text{Convo-AMI}}(D, N, L) = \frac{X_0}{X_i}, \tag{19}$$

As was done with the convolutional encoder and from Fig. 7, we have,

$$T_{\text{Convo-AMI}}(D, N, L) = \frac{L^2 N^2 D^3}{(1 - L)}. \tag{20}$$

Using Maclaurin series, the denominator in (20) can be expanded and we can obtain the following:

$$T_{\text{Convo-AMI}}(D, N, L) = D^3 N^2 (L^2 + L^3 + L^4 + \cdots). \tag{21}$$

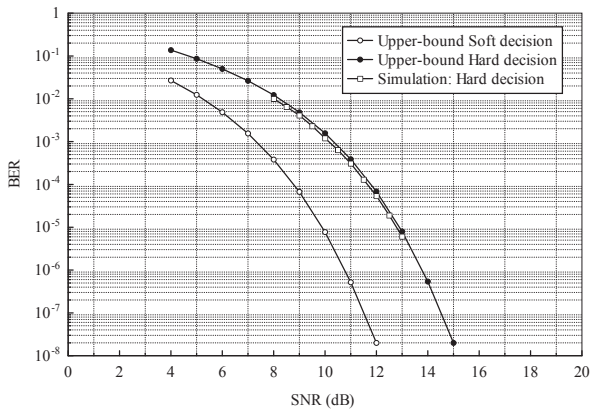

Figure 8: Upper bounds for Convo-AMI: $R = 1/2$, $K = 2$, $d_{free} = 3$

It is clear from (21), that there is an infinite number of paths with the same distance equal to three which differ by two input bits from the all-zero path. There is only one path with different lengths, varying from 2 to infinity.

The name given to this new code is Convo-AMI. The name reflects the link between both codes: AMI and convolutional codes. While we retained the characteristics of the AMI, we could apply Mason's rule to obtain the generating function of Convo-AMI presented in (21).

Fig. 8 shows the 2 dB gain between soft- and hard-decision Viterbi decoding for the Convo-AMI code.

4. GENERALIZATION

After we have resolved the problem of linearisation of AMI, we wish to generalize this method to the rest of the ternary line codes. The two's complement technique and the simplicity of AMI structure helped us to linearise this line code and work through its look-alike linear line code to study the behaviour of its error probability upper bound . The question that still confronted us was if we could linearise the rest of our ternary line codes, would we be able to implement all Mason's rule steps to calculate the generating function?

Table I: Decoding gain and state machine structure of different line codes

Ternary Line Codes	Gain at BER = 10^{-6} for 3-bit quantisation	Number of states	Self loop
AMI	~ 2.00	2	yes
HDB1	~ 2.00	8	no
B4ZS	2.05	18	no
B6ZS (0VB0VB)	2.00	70	no

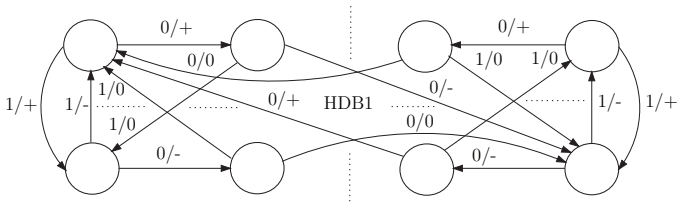

Figure 9: HDB1 State diagram

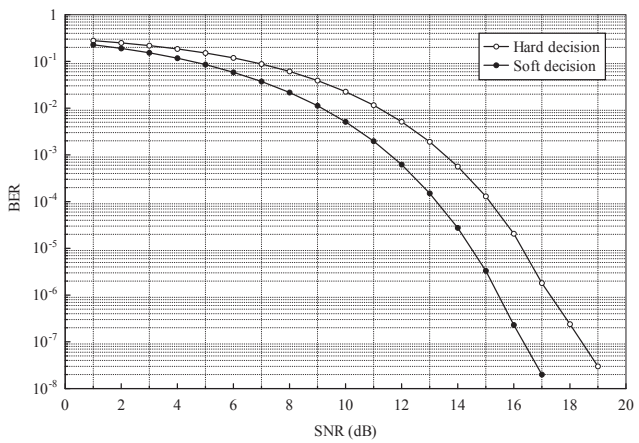

Figure 10: Soft and Hard decision Viterbi decoding for HDB1

It is clear from the previous sections that the calculation of the generating function of any code needs to involve some basic steps, which are presented as follows:

1. The use of Hamming distance to calculate the exponent of the metric distances.
2. The choice of an all-zero path to be used as a reference to calculate the metric distances of all other paths by comparing the received data to that of the all-zero path.
3. The self-loop is necessary for the generating function to apply Mason's rule.

The AMI was not difficult to implement because of the similarity of its state diagram to that of the convolutional encoder. Difficulties arose when the code did not have all, or even some, of the steps mentioned before. The challenge is to identify which steps are necessary and which are not.

Most of ternary line codes [20], have no self loops in their state machines. If we take the example of the state machine of HDB1 line code presented in Fig. 9, we can clearly see that it has no self loop. Fig. 10 shows the simulation results and emphasises the 2 dB gain between the hard- and soft decision Viterbi decoding of the HDB1 line code.

Table I presents the structure of few ternary line codes and the gain achieved between soft and hard decision Viterbi decoding. It can be seen that AMI is the only ternary line code that has a self loop.

As was the case with AMI, the problem of the Hamming distance is always solved by changing the negative outputs of the encoder to their two's complement.

The all-zero path presents a question: what will happen if the encoder does not have zero outputs? Does this change the generating function? To find the answers, we will use a more extended and complicated convolutional encoder to calculate the generating function. We use a convolutional encoder with four states to calculate the generating function not only based on the all-zero path but also based on other different reference paths.

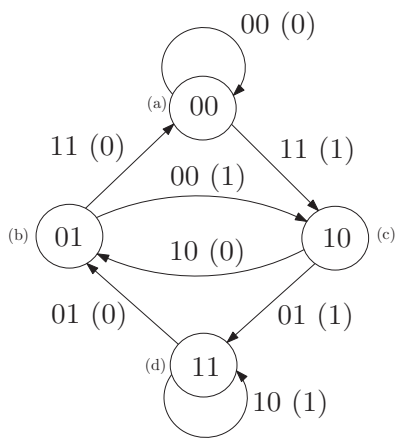

Figure 11: State Diagram of convolutional encoder: $R = 1/2$, $K = 3$, $d_{free} = 5$

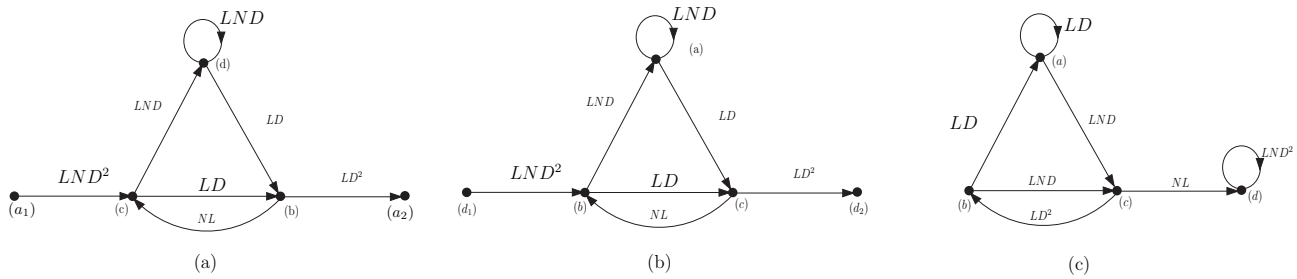

Figure 13: Flow graph of the convolutional encoder $R = 1/2$, $K = 3$, $d_{free} = 5$ with different reference states

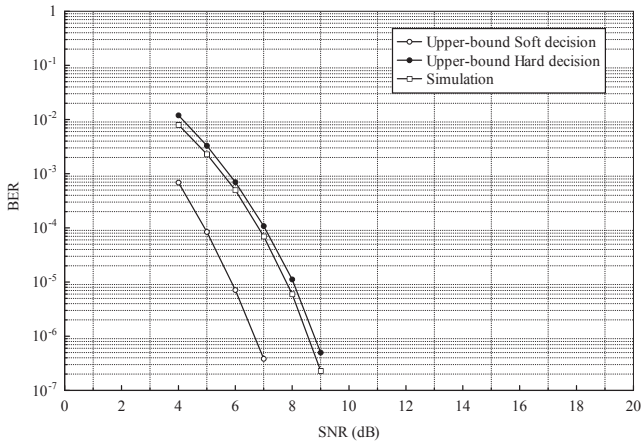

Figure 12: Upper bounds for convolutional encoder: $R = 1/2$, $K = 3$, $d_{free} = 5$

Example:

We take the example of a more general convolutional encoder with 4 states and a rate of $R = 1/2$ [13]. In addition to the calculation of upper bound curves, we will compare our simulation to these curves.

Fig. 11 shows the state diagram of the convolutional encoder with $R = 1/2$, $d_{free} = 5$. All states are labelled alphabetically.

Fig. 12 shows how close the hard-decision simulation is to the hard-decision error probability upper bound.

4.1. *Generating function with state (00) as the reference path*

The state (00), labelled in Fig. 11 as state (a), is taken here to be the reference for the rest of the states to calculate the distance metric for all other paths compared to the all-zero path, which will be called the reference path. The corresponding flow graph of the state machine in Fig. 11 is presented in Fig. 13(a) and the corresponding relations between all states input and outputs are presented as follows:

$$X_{a_2} = LD^2 X_b, \tag{22}$$

$$X_b = LDX_b + LDX_c, \tag{23}$$

$$X_c = NLX_b + LND^2 X_{a_1}, \tag{24}$$

$$X_d = LNDX_c + LNDX_d, \tag{25}$$

Using the substitution method to solve the system to calculate the generating function

$$T_{(a)}(D, N, L) = \frac{X_{a_2}}{X_{a_1}} \tag{26}$$

we get

$$T_{(a)}(D, N, L) = \frac{L^3 ND^5}{(1 - ND(L + L^2))}. \tag{27}$$

It is clear here that the minimum distance or the free distance is equal to 5.

With $L = 1$, we get

$$\begin{aligned} T_{(00)}(D, N) =& T_{(a)}(D, N) \\ =& \sum_{d=5}^{\infty} 2^{(d-5)} N(d-4) D^d. \end{aligned} \tag{28}$$

4.2. *Generating function with state (11) as the reference path*

The state (11), labelled in Fig. 11 as state (d), is taken here to be the reference for the rest of the states to calculate the distance metric for all other paths previously

compared to the reference path.

The Hamming distance, as used before to calculate the distance metric for each path and symbolized in the generating function as the exponent of the distances, will be calculated referring to the output (10) of the state (11) instead of (00) of the state (00). The same procedure will be followed to search for the weight of each distance corresponding to each path.

Definition 4 The term N is included in the generating function only if the branch transition is a result of an input data of the value "zero". □

The term L is used as it is defined in Definition 2. From Definition 4, we can see that the value of N is defined differently to the one in Definition 3 since it depends on the input of the reference path.

In the case of the all-zero path, where the self loop has the input of 0, the term N was included in the generating function for any input of 1. This value is always considered to be the complement to the value of the input of the reference path.

In the case of the (11) state as a reference path, we are dealing with a reference path with an input of 1. Thus we include the term N in the generating function when the input data is "zero".

In implementing these changes to the calculating procedure, we try here to calculate the generating function of the same convolutional encoder introduced in Fig. 11 and presented by its flow graph in Fig. 13(b), corresponding to (11) as the reference path.

$$X_{d_2} = LD^2X_c, \tag{29}$$

$$X_c = LDX_a + LDX_b, \tag{30}$$

$$X_b = NLX_c + LND^2X_{d_1}, \tag{31}$$

$$X_a = LNDX_b + LNDX_a. \tag{32}$$

Using the substitution method to solve the system to calculate the generating function,

$$T_{(d)}(D, N, L) = \frac{X_{d2}}{X_{d1}}, \tag{33}$$

we get

$$T_{(d)}(D, N, L) = \frac{L^3ND^5}{(1 - ND(L + L^2))}, \tag{34}$$

It is clear here that the minimum distance or the free distance is equal to 5.

With $L = 1$ we get

$$\begin{aligned} T_{(11)}(D, N) =& T_{(d)}(D, N) \\ =& \sum_{d=5}^{\infty} 2^{(d-5)} N(d-4)D^d. \end{aligned} \tag{35}$$

It can be seen from (28) and (35), that both generating functions have the same formula, which means that we have always:

$$T_{(00)}(D, N) = T_{(11)}(D, N). \tag{36}$$

It is clear from (36) that the reference path from a self loop does not affect the generating function if we take into consideration the changes that we have to make to the definitions of the term N, related to the inputs at the self loops.

Proposition: The generating function of any convolutional code is independent to the self-loop.

Proof: One of the major properties of convolutional codes is the symmetry of their state machines. Fig. 11 shows clearly the symmetry between the two self-loop states. This symmetry guarantees the non-changing of the minimum distance and the shortest path corresponding to it. The minimum distance and the shortest path are both the major elements for the derivation of the generating function. Thus if these two elements are unchangeable, then we can confirm that the generating function stays the same despite the selection of the self loop. ■

4.3. *Generating function and the self-loop*

Mason's rule shows that to calculate the generating function of any system it is necessary to have an input and output for the system. The generating function is then the ratio of both. Using the state diagram in telecommunications is a way to get to the application of Mason's rule by obtaining an input and an output from the states of the code. This is ensured by the self-loops.

In general the self-loops make no contributions to the distances since the distances are results of the comparison of the binary outputs of the code to the outputs of the reference path.

In the case where we choose two different states - one for the input and the other for the output of the system - the question that can be asked is: can we obtain the same result of the generating function as if we use the self-loop? In other words, can two states replace the self-loop state?

From Fig. 11, the state (01), labelled as state (b), is chosen as the input of the system and the state (11), labelled as (d), is chosen as the output of the system.

The branch between the two states will be the reference path and N will be used for the inputs of "one". The branch between the state (11) and the state (01) will be eliminated since it is used as a reference path.

Fig. 13(c) shows the new flow graph of the convolutional encoder.

$$T_{(bd)}(D, N, L) = \frac{X_d}{X_b}, \tag{37}$$

$$X_b = LD^2 X_c, \tag{38}$$

$$X_d = LND^2 Xd + LNX_c, \tag{39}$$

$$T_{(bd)}(D, N, L) = \frac{N}{(1 - NLD^2)D^2}, \tag{40}$$

Comparing this generating function with those in the previous sections, it is clear that they are totally different from each other. This leads to the conclusion that the self-loop in the state diagram for any line code is a requirement to calculate the generating function of the line code.

Proposition: The self-loop is fundamental in the calculation of the generating function of any code.

Proof: The calculation of the generating function of any code is based on the linearity of that code. In control systems, the linearity is guaranteed when the output and the input of a system are related with a linear function. The linearity is guaranteed with the independence of the input of the system to the output as the non-existence of any feedback.

The split of the self loop is to ensure that the input and the output are not connected to other states that might contribute to the generating function and play a role similar to the feedback of the system.

This fundamental property does not exist for all codes that do not have self-loops, which make all states depending and contributing to one another and thus will impact on the generating function. ■

5. CONCLUSION

It is clear from our investigation that the calculation of the generating function of any line code - used for the error probability - using Mason's rule requires certain steps to be executed. Some of these steps can be avoided or modified, like the reference path, but one of them is a necessity and can never be avoided, namely the self-loop.

The problem that we experienced with the ternary line codes studied here is that most of them do not have self loops which makes it difficult or, even impossible, to calculate the generating function.

The case of the AMI line code is an exception in view of its simplicity and the state diagram. It is necessary after this investigation to look for other tools besides Mason's rule that could be used to calculate the generating function or try to apply similar technique used for non-linear codes in the context of trellis codes to non-linear line codes.

6. AKNOWLEDGEMENT

This material is based upon work supported by National Research Foundation under Grant number (2053408) and also by AAT.

7. REFERENCES

[1] A. Croisier, "Introduction to Pseudo-ternary Transmission Codes," *IBM J. Res. Develop.* , vol.14, pp. 354–367, July 1970.

[2] B. R. Alyaei and A. Glass, "Line coded modulation," in the Proceedings of *IEEE*-Signal Processing and Communication Systems, Omaha, NE USA, pp. 1–4, Sept. 28-30, 2009.

[3] A. Viterbi and J. Omura, *Principles of Digital Communication and Coding*. McGraw-Hill Kogakusha LTD, Tokyo Japan, 1979.

[4] G. D. Jr. Forney, "The Viterbi algorithm," *Proceedings IEEE*, vol. 61, no. 3, pp. 268–278, 1967.

[5] A. J. Viterbi, "Error bounds for convolutional codes and an asymptotically optimum decoding algorithm," *IEEE Trans. Info. Theory*, vol. 13, pp. 260–269, Oct. 2004.

[6] K. Ouahada and H. C. Ferreira, "Simula-

tion study of the performance of ternary line codes under Viterbi decoding," *IEE Proceedings–Communications*, vol. 151, no. 5, pp. 409–414, Oct. 2004.

[7] M. Rouanne and D. J. Costello, Jr, "An algorithm for computing the distance spectrum of trellis codes," *IEEE J. Sel. Areas Commun.*, vol. 7, no. 6, pp. 929–940, Aug. 1989.

[8] R. D. Wesel, "Reduced-state representations for trellis codes using constellation symmetry," *IEEE Trans. Commun.*, vol. 52, no. 8, pp. 1302–1310, Aug. 2004.

[9] G. F. Franklin, J. D. Powell, A. Emami-Naeini, *Feedback Control of Dynamic Systems*. Addison-Wesley, Third Edition, 1994.

[10] J. B. Buchner, "Ternary Line Codes," *Philips Telecommunications Review*, vol. 34, no. 2, pp. 72–86, June 1976

[11] F. H. Raven, *Automatic Control Engineering*. McGraw-Hill, Fourth Edition, Singapore, 1987.

[12] F. G. Stremler, *Introduction to Communication Systems*. Addison-Wesley, Third Edition, 1990

[13] J. G. Proakis and M. Salehi, *Communication Systems Engineering*. Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1994.

[14] G. L. Stuber, *Principles of Mobile Communication*. Kluwer Academic Publishers, Boston USA, 2002.

[15] A. Dholakia, *Introduction to convolutional codes with applications*. Kluwer Academic Publishers, Boston USA, 1994.

[16] A. Croft, R. Davison and M. Hargreaves, *Engineering mathematics: a modern foundation for electronic, electrical and system engineers* . Harlow : Addison-Wesley, 2nd ed. 1996.

[17] N. Q. Duc, "Line Coding Techniques for Baseband Digital Transmission," *A. T. R.* vol. 9 no. 1, pp. 3–17, 1975.

[18] J. M. Wozencraft, Irwin Mark Jacobs, *Principles of Communication Engineering*. John Wiley Sons, United States of America, First Edition, 1965.

[19] K. Ouahada and H. C. Ferreira, "Soft-decision decoding of some ternary line codes," *Electronics Letters*, vol. 39, no. 14, pp. 1068–1069, 2003.

[20] D. B. Keogh, "Finite-State Machine Descriptions of Filled Bipolar Codes," *A. T. R.* vol. 18, no. 2, pp. 3–12 , June 1984.