Chapter 4

Performance Issues of Various Block Codes

4.1 Introduction

In the previous chapter, the syndrome method of constructing the trellis diagram of a block code was presented, as well as techniques to decode block codes by means of these trellis diagrams. This chapter will concentrate on performance issues relevant to this decoding method. Several different block codes are investigated, and their simulation results given. All the simulation software was developed from scratch, and was used to achieve the given simulation results.

4.2 Simulation Results

The following simulations results were obtained for both soft-decision and hard-decision block decoding. The generator matrixes that were used for the simulations are given in **Appendix C**. A simulation was done with a (7,4)-Hamming code for both soft decision and hard decision decoding in an AWGN channel. The simulation setup is the same as described in **Figure 4** of **Chapter 2** and consists of a transmitter transmitting over an AWGN channel. The achieved results for the (7,4)-Hamming code are compared to both the theoretical curves and the error curves obtained for the traditional analytical decoding methods. The results prove that the same (maximum likelihood) performance is obtained with the Viterbi algorithm than with conventional algebraic methods. This holds for both soft decisions as well as hard decisions. As all the curves, including the bounds, cluster closely together, this representation (i.e. all curves on one graph) is not used in subsequent simulation results in this dissertation. It is clear form the results in the example (**Figure 16**) that the results fall well within the limits defined by the BER bounds for the decoding methods under discussion. The following equation was used in determining the upper bound for the decoding.

$$P_E \leq \sum_{m=\left\lfloor \frac{d_{\min}-1}{2} \right\rfloor}^{n} \binom{n}{m} p^m (1-p)^{n-m} \qquad (4.1)$$

The following two graphs provide the results achieved for the (7,4)-Hamming code.

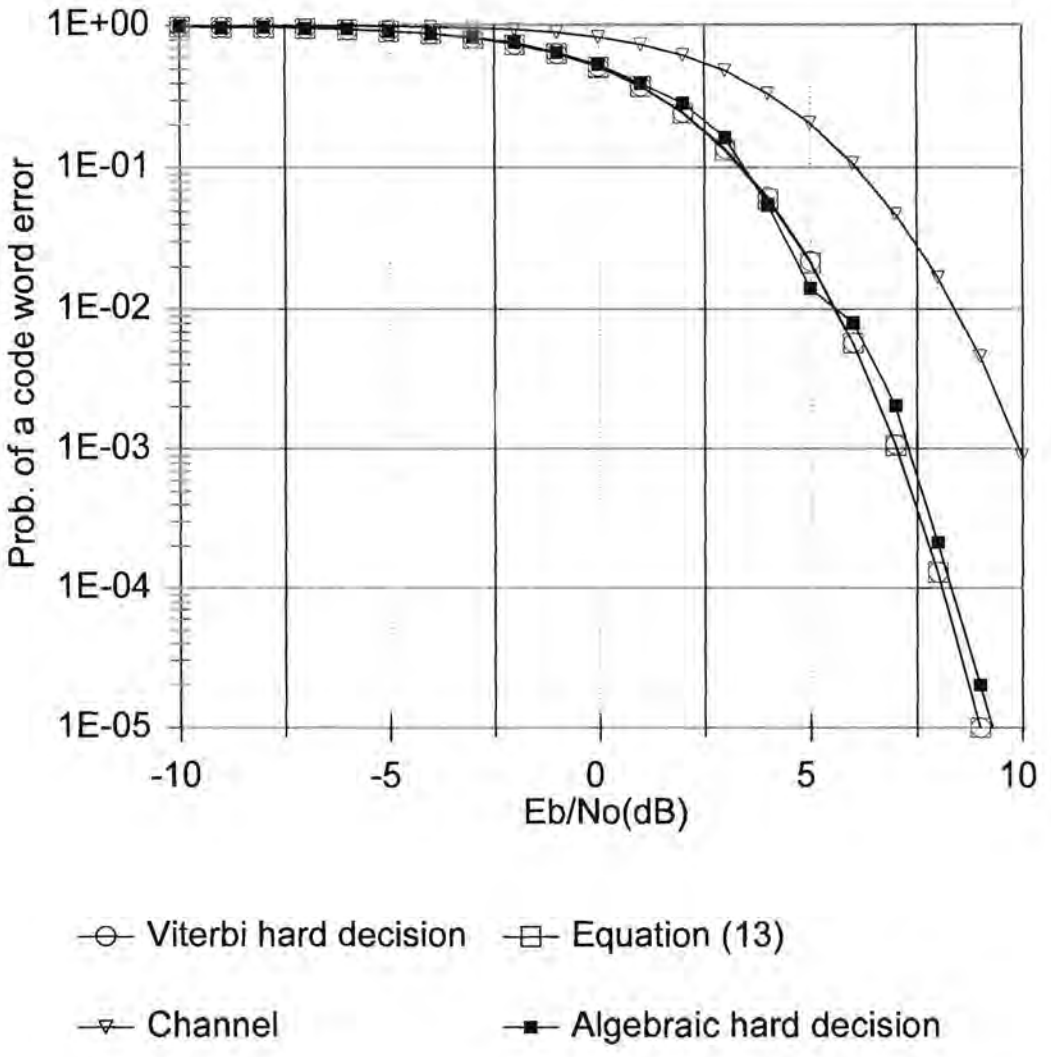

Figure 16 BER Comparison of Various Hard Decision Viterbi Decoding Methods of the (7,4) Hamming Block Code

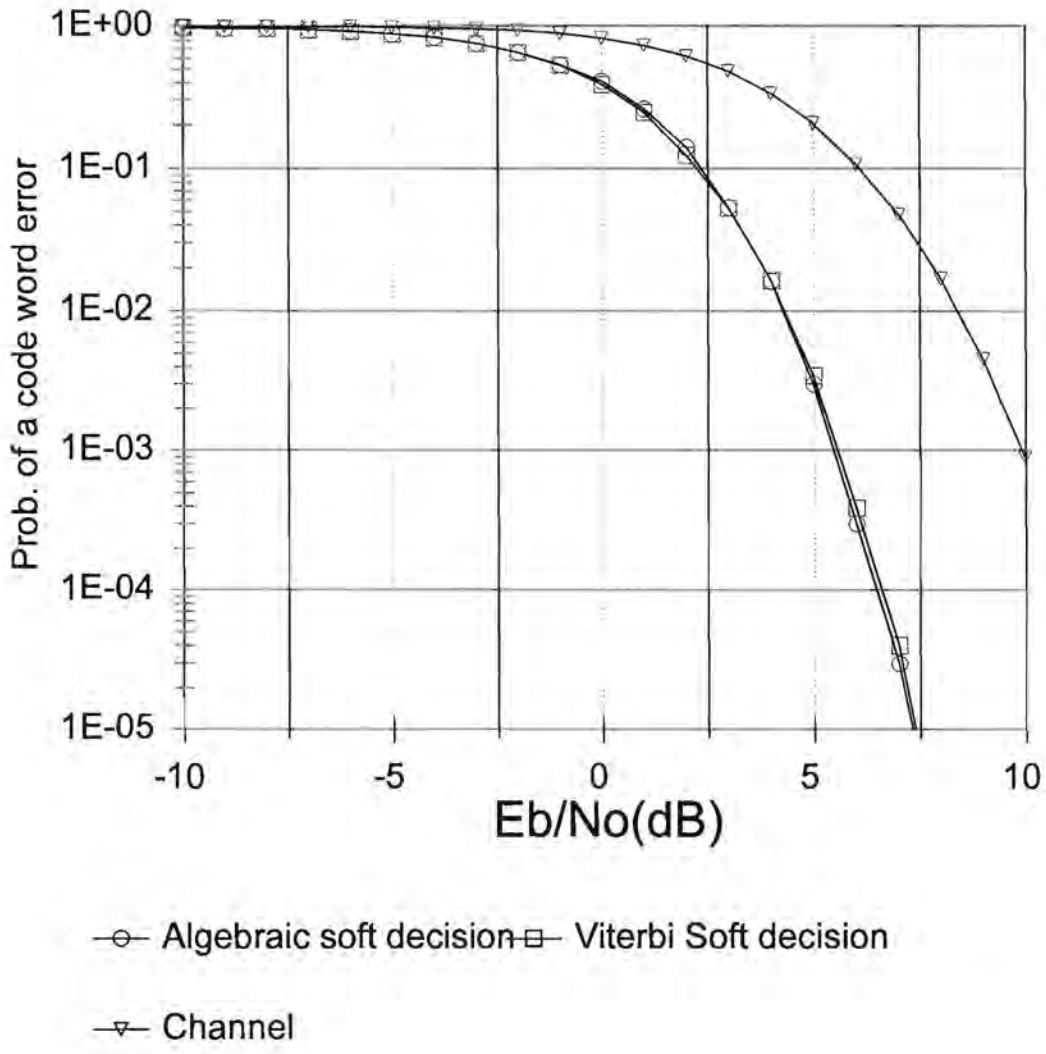

Figure 17 BER Comparison of Various Soft Decision Viterbi Decoding Methods of the (7,4) Hamming Block Code

On the following BER graphs curves for various families of codes are depicted. For the reasons explained above, the error rate curves will not be compared to the algebraic methods, since these curves lie on top of each other. In the figures, a graph termed *Bipolar Reference* is however included. This curve represents the BER performance of the uncoded case under identical channel conditions.

4.2.1 Bit Error Rate Performance of the(7,4)-Hamming-Code with Viterbi Trellis Decoding

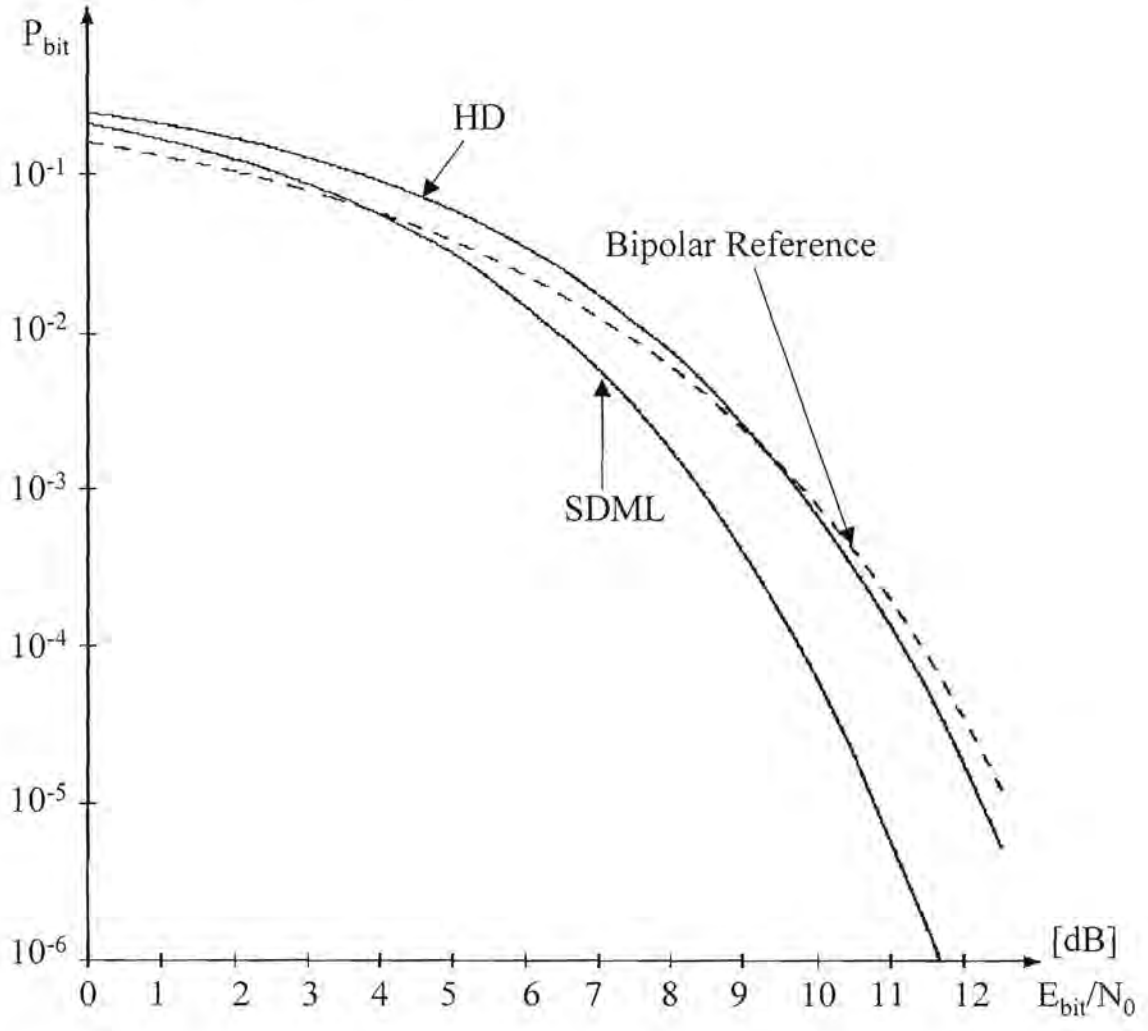

Figure 18 Bit Error Rates of the (7,4)-Hamming-Code in an AWGN Channel

Figure 18 presents the BER performance of a (7,4)-Hamming-Code. As can be seen from the figure, SDML-decoding becomes viable for channels having and E_{bit}/N_0 ratio of 4 dB and higher. Below this cross over value (3.5 dB in this case), SDML-decoding fairs no better than the normal uncoded transmission. The HD-decoding only crosses the uncoded graph at 9.2 dB. It is also only marginally better than the uncoded curve for E_{bit}/N_0 larger than 9.2 dB. At $P_{bit} = 10^{-5}$ the difference between SDML and HD-decoding is 1.4 dB. This corresponds with the calculated [10] value for the (7,4)-Code.

4.2.2 Block Error Rate Performance of the (7,4)-Hamming-Code with Viterbi Trellis Decoding

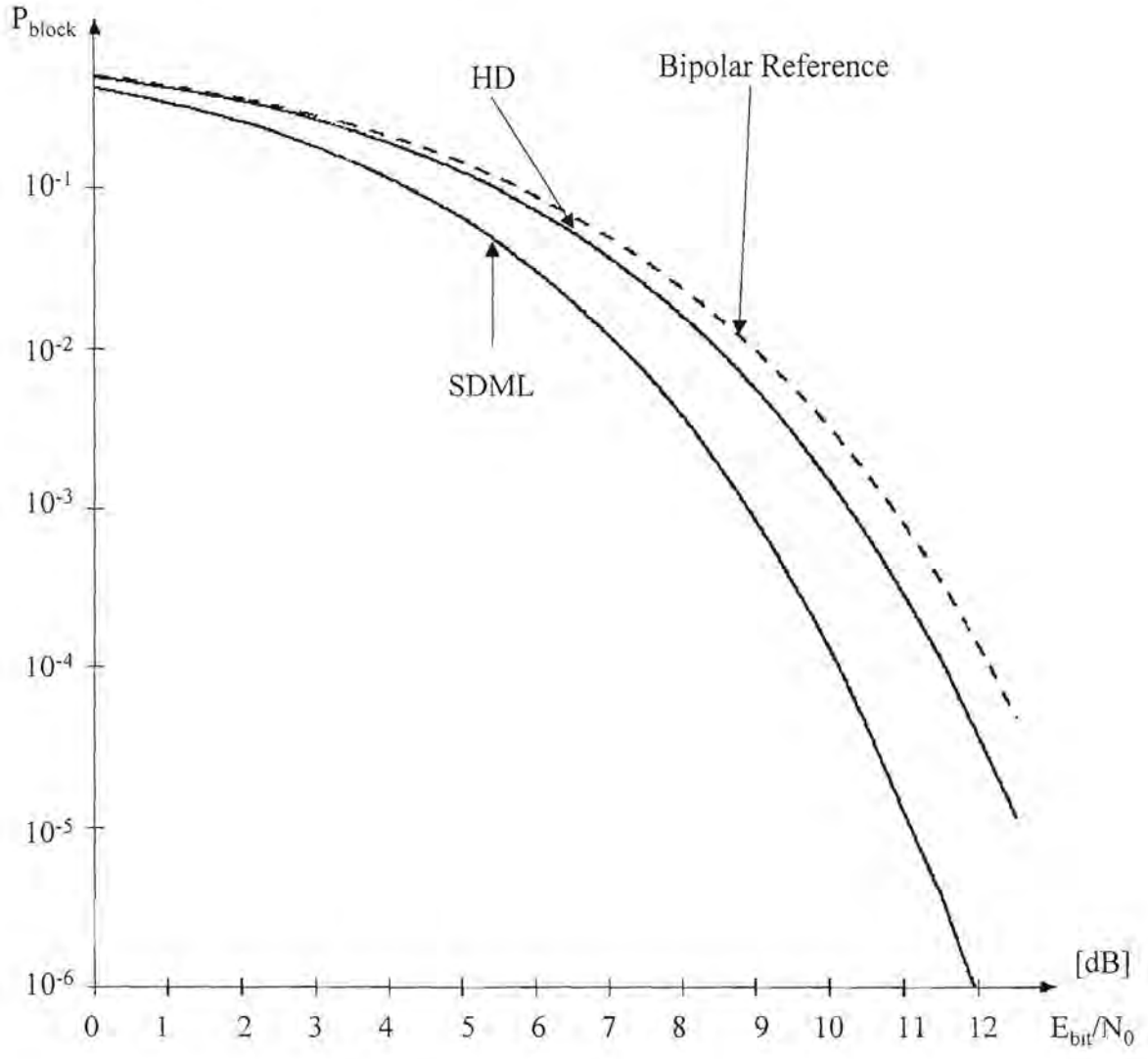

Figure 19 Block Error Rate of the (7,4)-Hamming-Code in an AWGN Channel

In **Figure 19** above, the block error probability of the (7,4)-Hamming-Code is given. The difference between the BER and the block error rate is as follows: Whereas several bit errors may only result in one block error, the BER counts all bit errors. The latter are statistically dependant on a few parameters such as the burst error statistics of the given channel. For instance, a codeword of length 7 bits is decoded incorrectly. This constitutes only one block error, but any number of bits (1 to 7) may be incorrect.

4.2.3 Bit Error Rate Performance of the (16,11)-Reed-Muller-Code with Viterbi Trellis Decoding

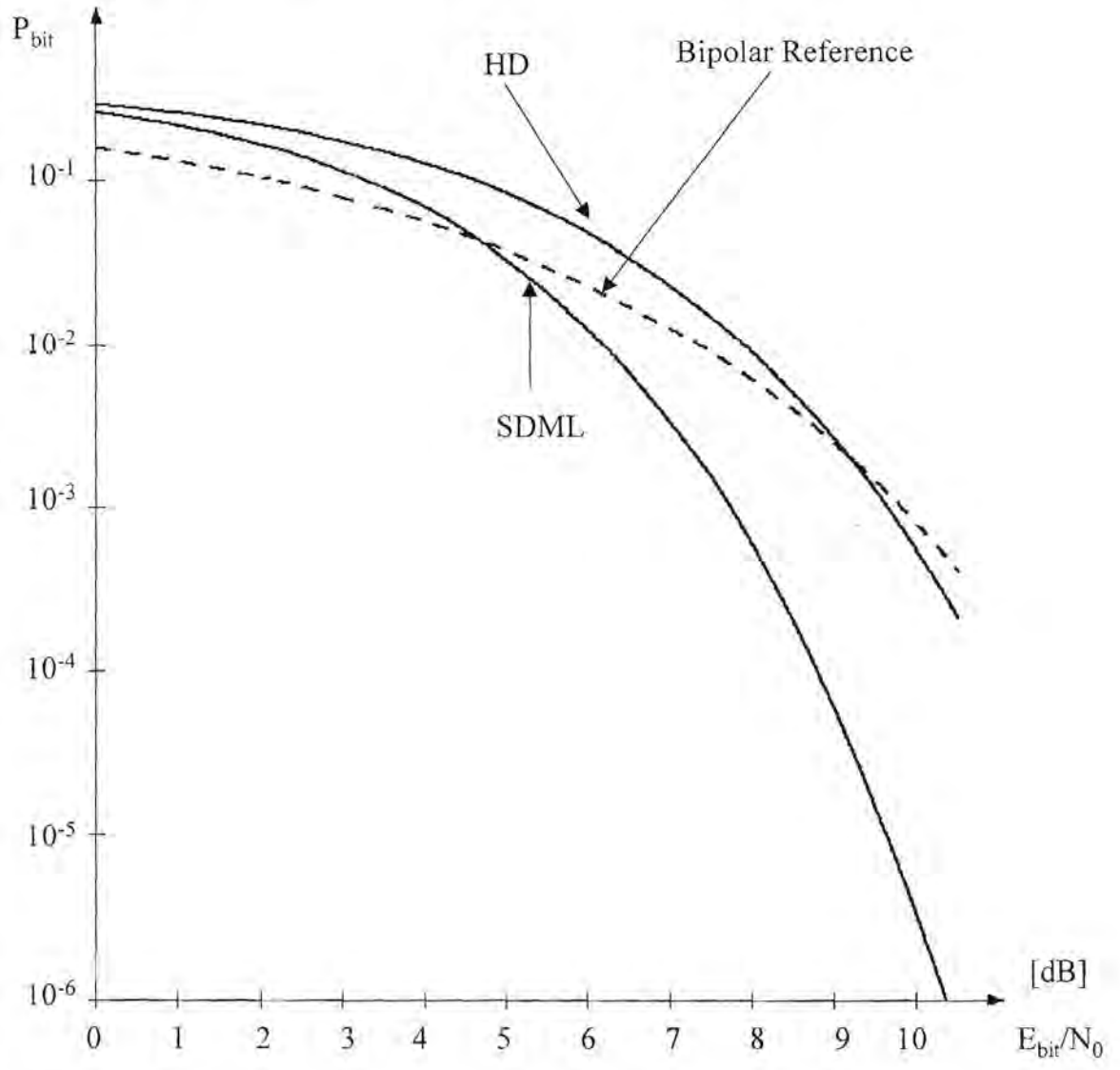

Figure 20 Bit Error Rate of the (16,11)-Reed-Muller-Code in an AWGN Channel

The above curve is very similar to the curve for the (7,4)-Hamming-Code depicted in **Figure 18**. The difference between the two codes is as follows: To achieve a bit error rate of 10^{-6} with the Reed-Muller-Code, an E_{bit}/N_0 ratio of only 10.3 dB is required, whereas the Hamming-Code requires an E_{bit}/N_0 ratio of 11.7 dB. The Reed-Muller code has a 1.4 dB coding advantage on the (7,4)-Hamming code in this particular example.

4.2.4 Bit Error Rate Performance of the (31,21)-BCH-Code with Viterbi Trellis Decoding

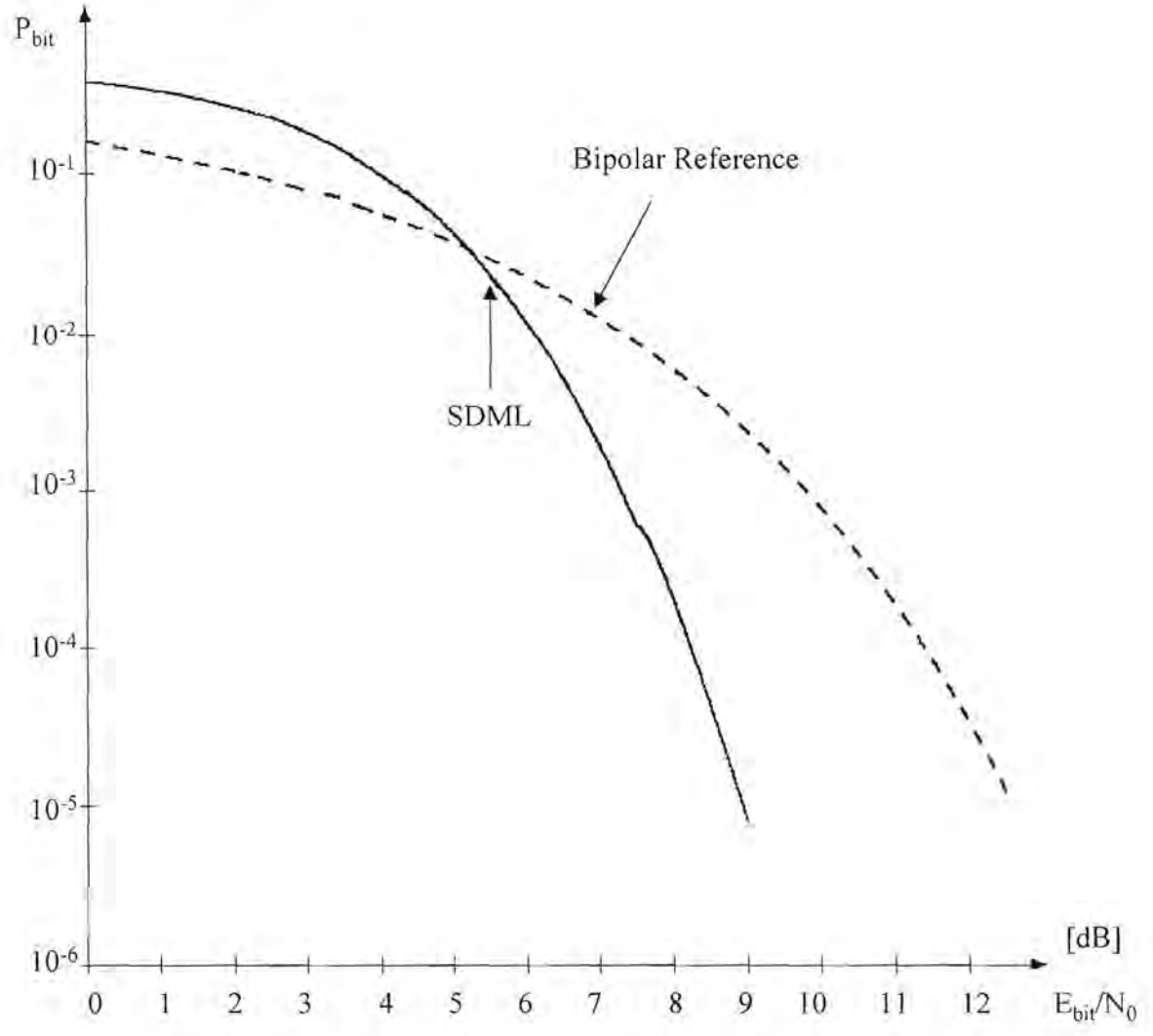

Figure 21 Bit Error Rate of the (31,21)-BCH-Code in an AWGN Channel

The above figure displays the Bit Error Rate of a (31,21)-BCH-Code. It can be seen that this code renders good performance at high values of E_{bit}/N_0. A Bit Error Rate of 10^{-5} is achieved at only 9 dB. In **Figure 22** another BCH-code is displayed, and in **Figure 23**, a comparison between both the BCH-codes and the (7,4)-Hamming-Code is made.

4.2.5 Bit Error Rate Performance of the (15,7)-BCH-Code with Viterbi Trellis Decoding

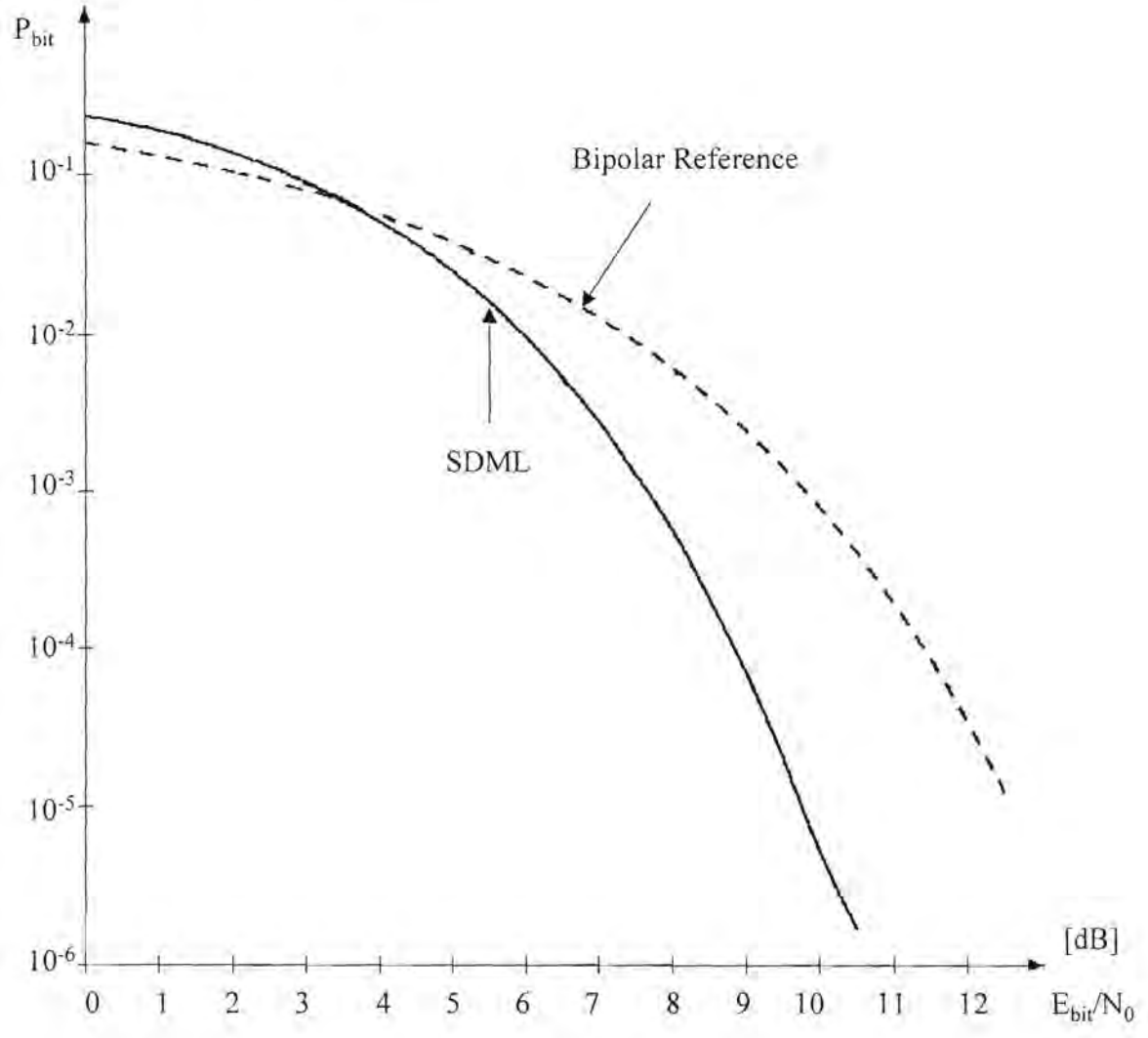

Figure 22 Bit Error Rate of the (15,7)-BCH-Code in an AWGN Channel

In **Figure 22**, the Bit Error Rate of a (15,7)-BCH-Code is displayed. Again, it can be seen that the code provides a relatively good performance at high values of E_{bit}/N_0. To gain some insight into the relative performance of the block codes considered, the previous two figures are superimposed on the (7,4)-Hamming-Code. The result is shown in **Figure 23**.

4.2.6 Performance Comparison of BCH-Codes with (7,4)-Hamming-Code with Viterbi Trellis Decoding

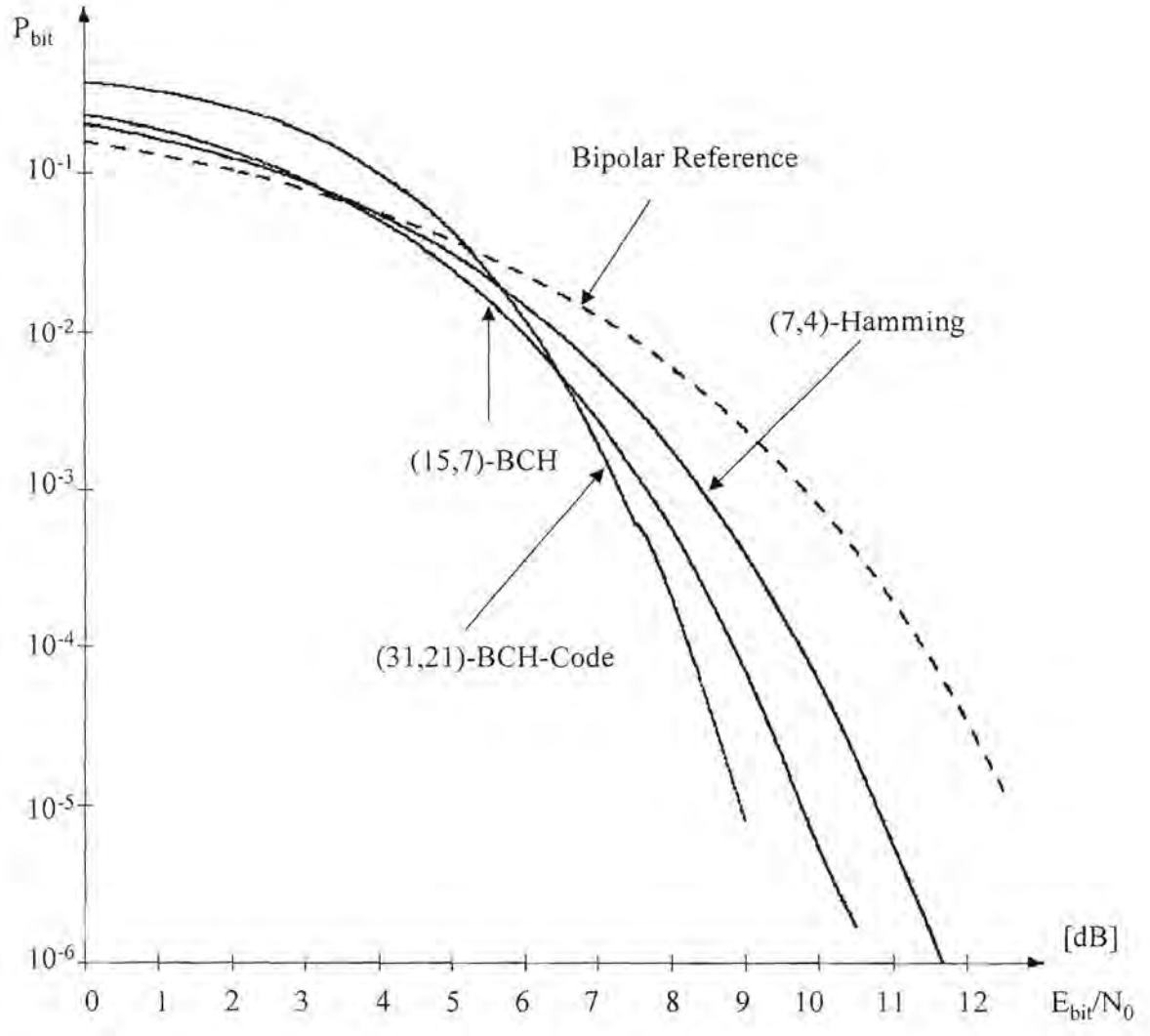

Figure 23 Comparison of BCH-Codes with (7,4)-Hamming-Code in an AWGN Channel

From **Figure 23**, it can be seen that all of the codes perform worse than the uncoded bipolar reference system at low values of E_{bit}/N_0. The cross over point, where the codes begin to perform according to their theoretical specification, is at about 5.5 dB. From this point onward, the larger codes progressively yield better performance as the block length increases. It should be noted that a number of the factors have to be considered when deciding on a code for a particular channel.

4.2.7 Bit Error Rate Performance of the (16,5)-Reed-Muller-Code *R*(1,4) with Viterbi Trellis Decoding

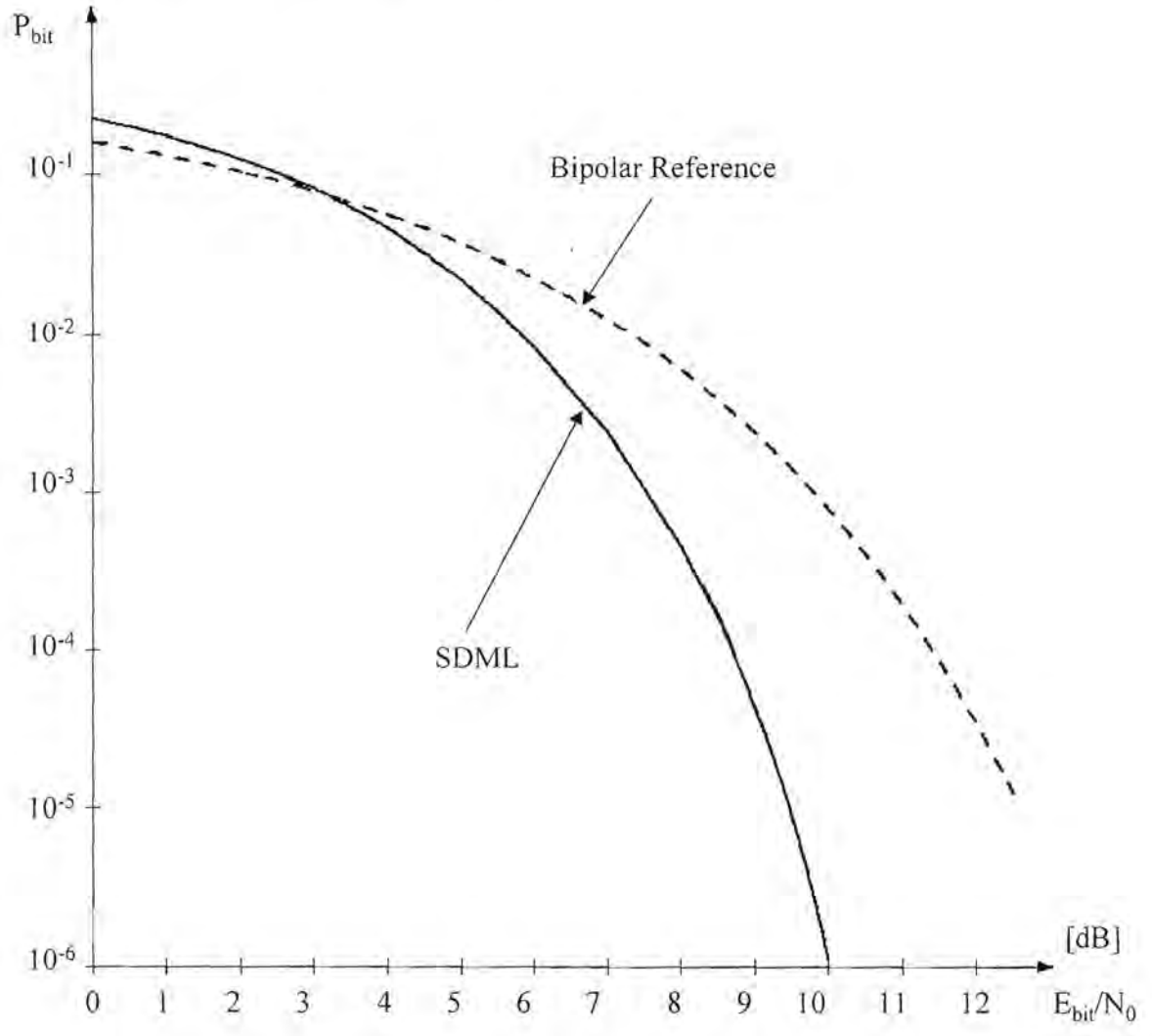

Figure 24 Bit Error Rate of the (16,5)-Reed-Muller-Code $R(1,4)$ in an AWGN Channel

4.2.8 Bit Error Rate Performance of the (16,11)-Reed-Muller-Code *R*(2,4) with Viterbi Trellis Decoding

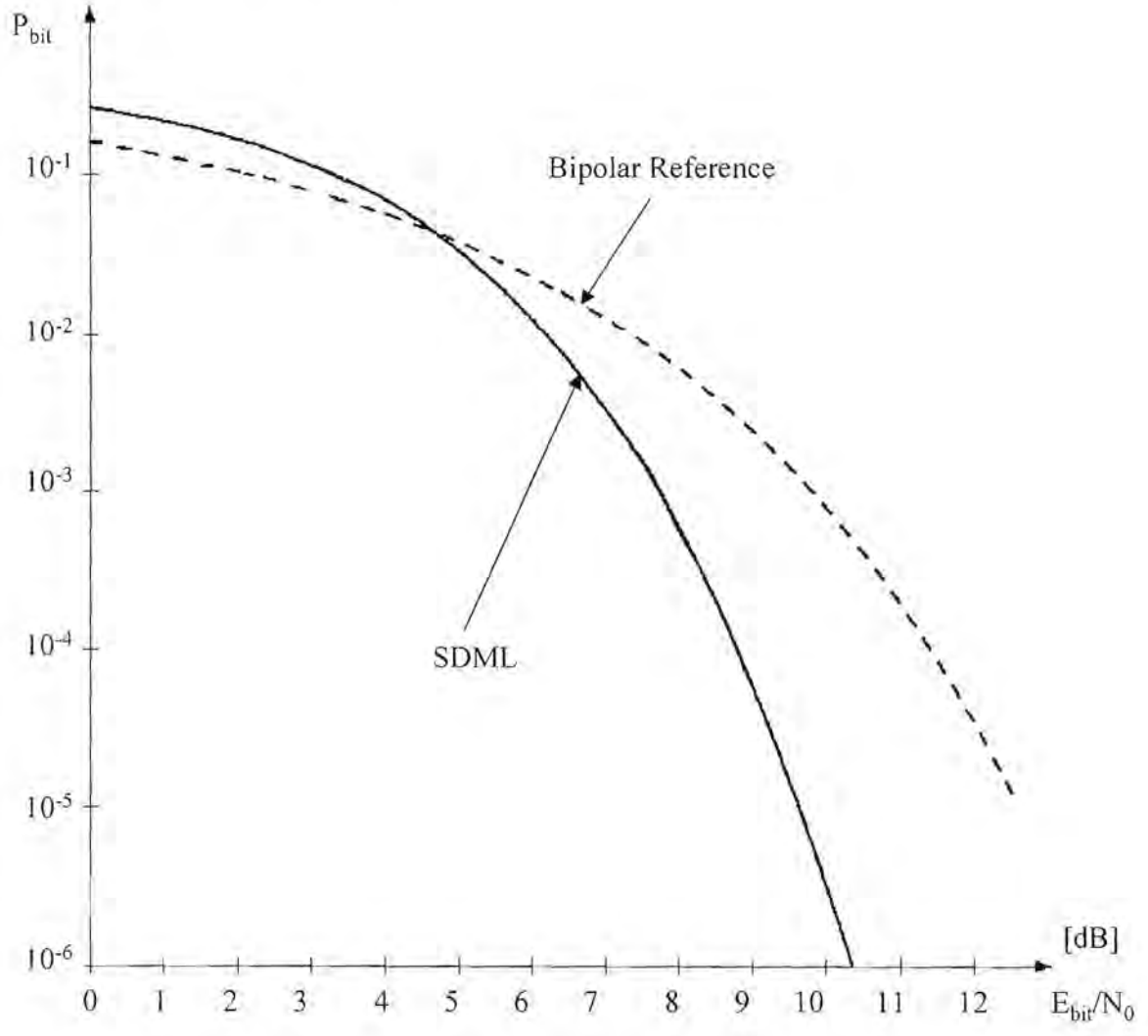

Figure 25 Bit Error Rate of the (16,11)-Reed-Muller-Code $R(2,4)$ in an AWGN Channel

4.2.9 Bit Error Rate Performance of the (32,16)-Reed-Muller-Code *R*(2,5) with Viterbi Trellis Decoding

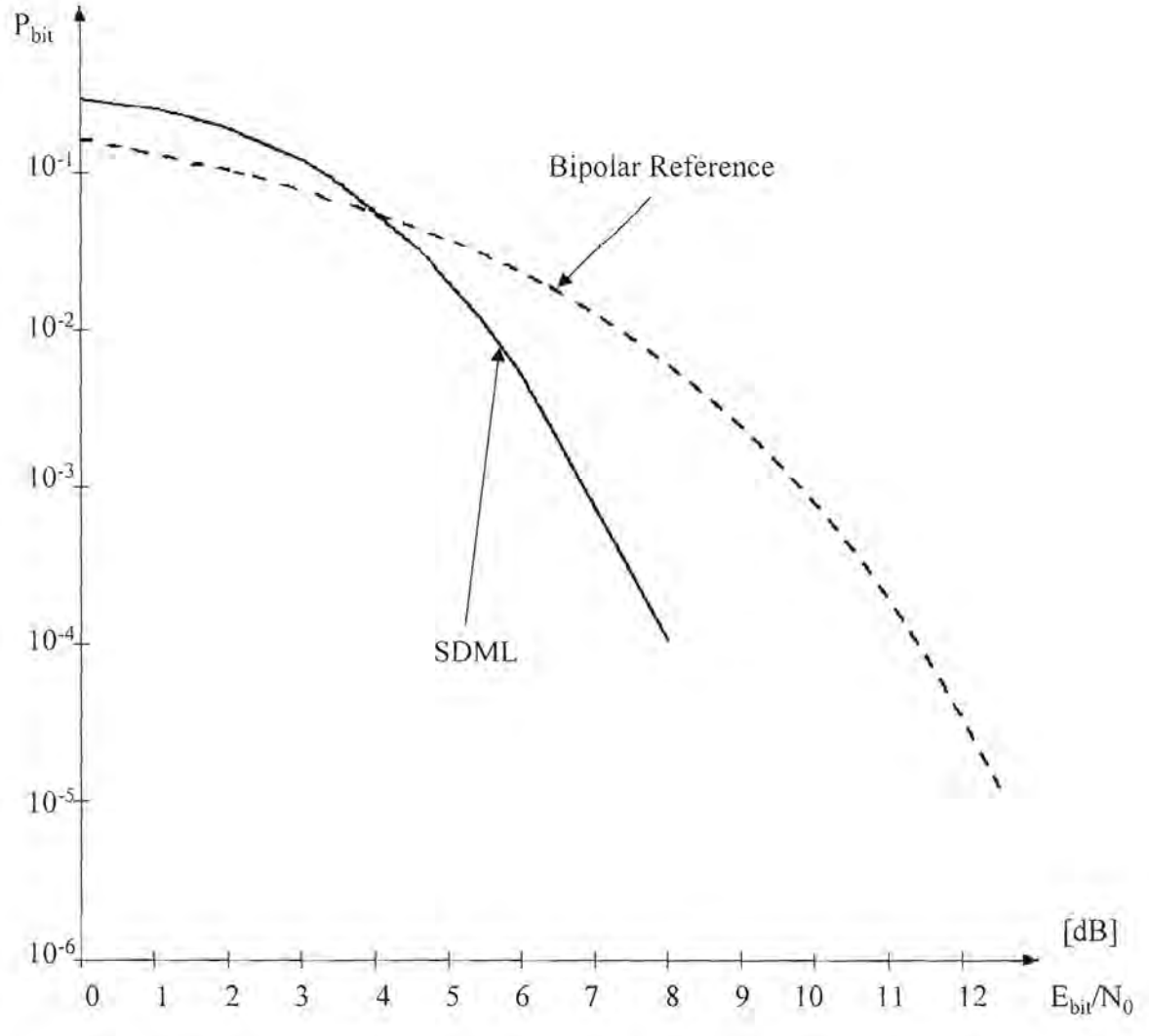

Figure 26 Bit Error Rate of the (32,16)-Reed-Muller-Code *R*(2,5) in an AWGN Channel

4.2.10 Bit Error Rate Performance of the (32,26)-Reed-Muller-Code *R*(3,5) with Viterbi Trellis Decoding

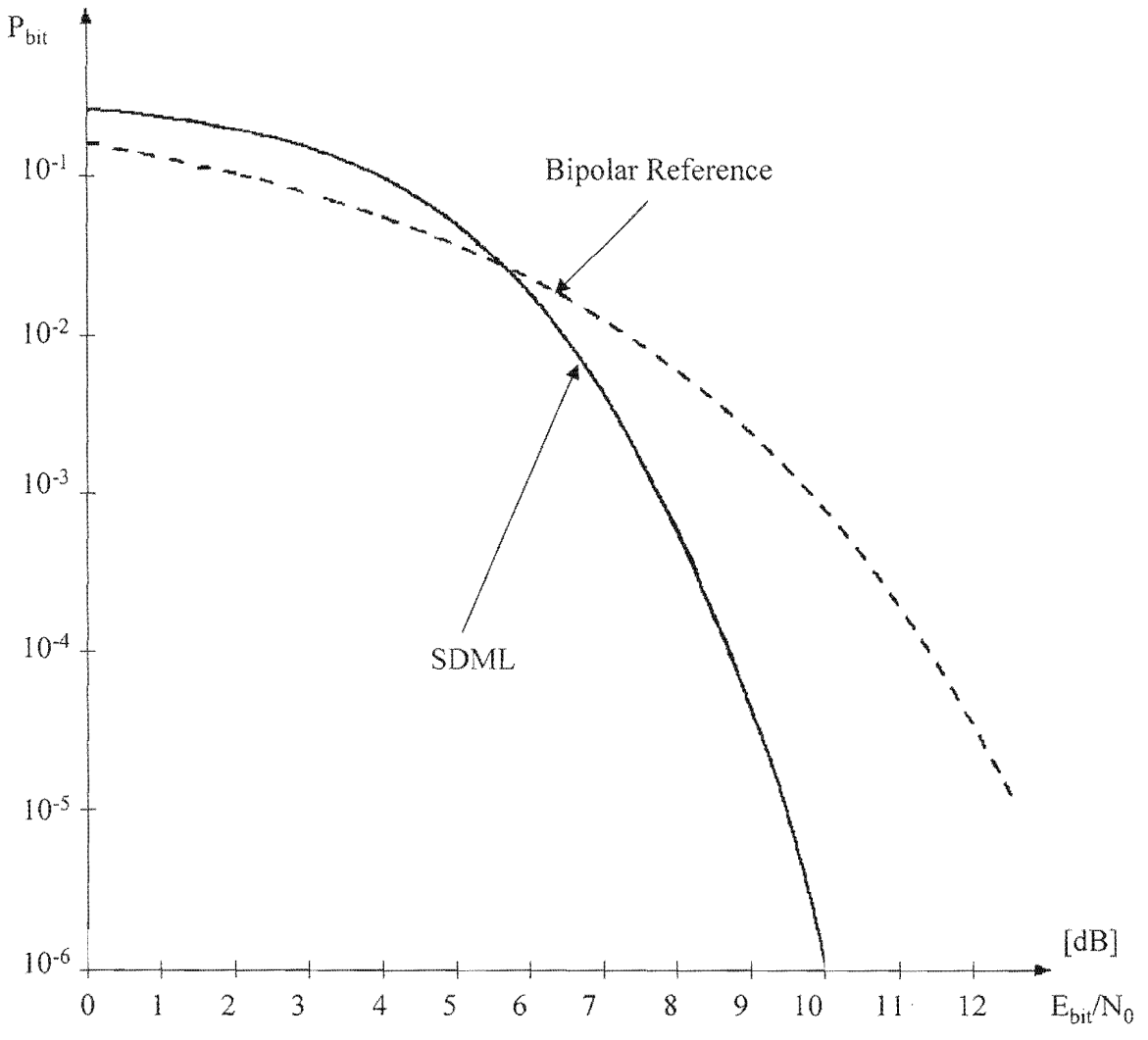

Figure 27 Bit Error Rate of the (32,26)-Reed-Muller-Code *R*(3,5) in an AWGN Channle

4.2.11 Bit Error Rate Performance of the (32,31)-Reed-Muller-Code *R*(4,5) with Viterbi Trellis Decoding

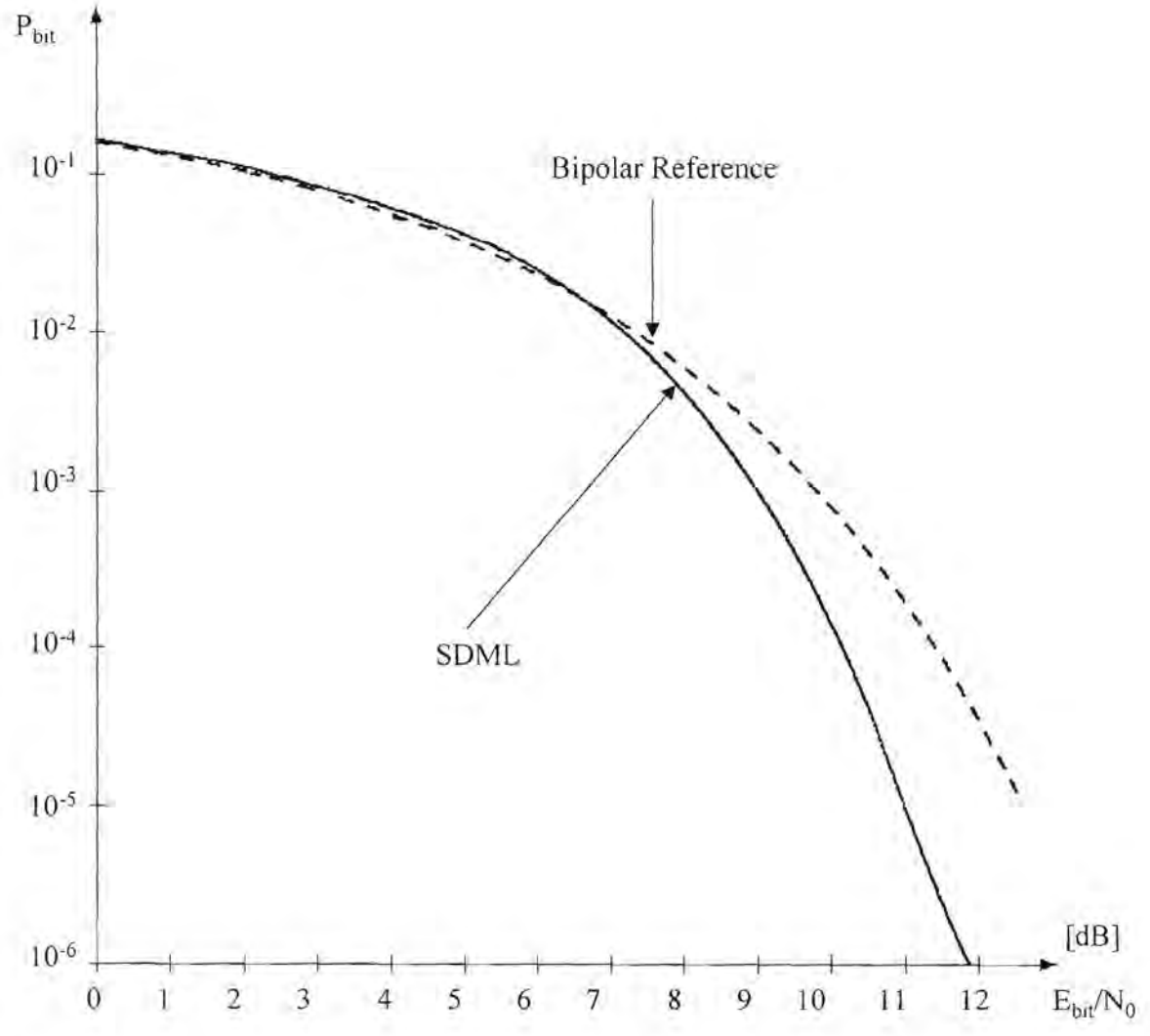

Figure 28 Bit Error Rate of the (32,31)-Reed-Muller-Code *R*(4,5) in an AWGN Channel

4.2.12 Performance Comparison of Reed-Muller-Codes *R(x,y)* for different (*n,k*) with Viterbi Trellis Decoding

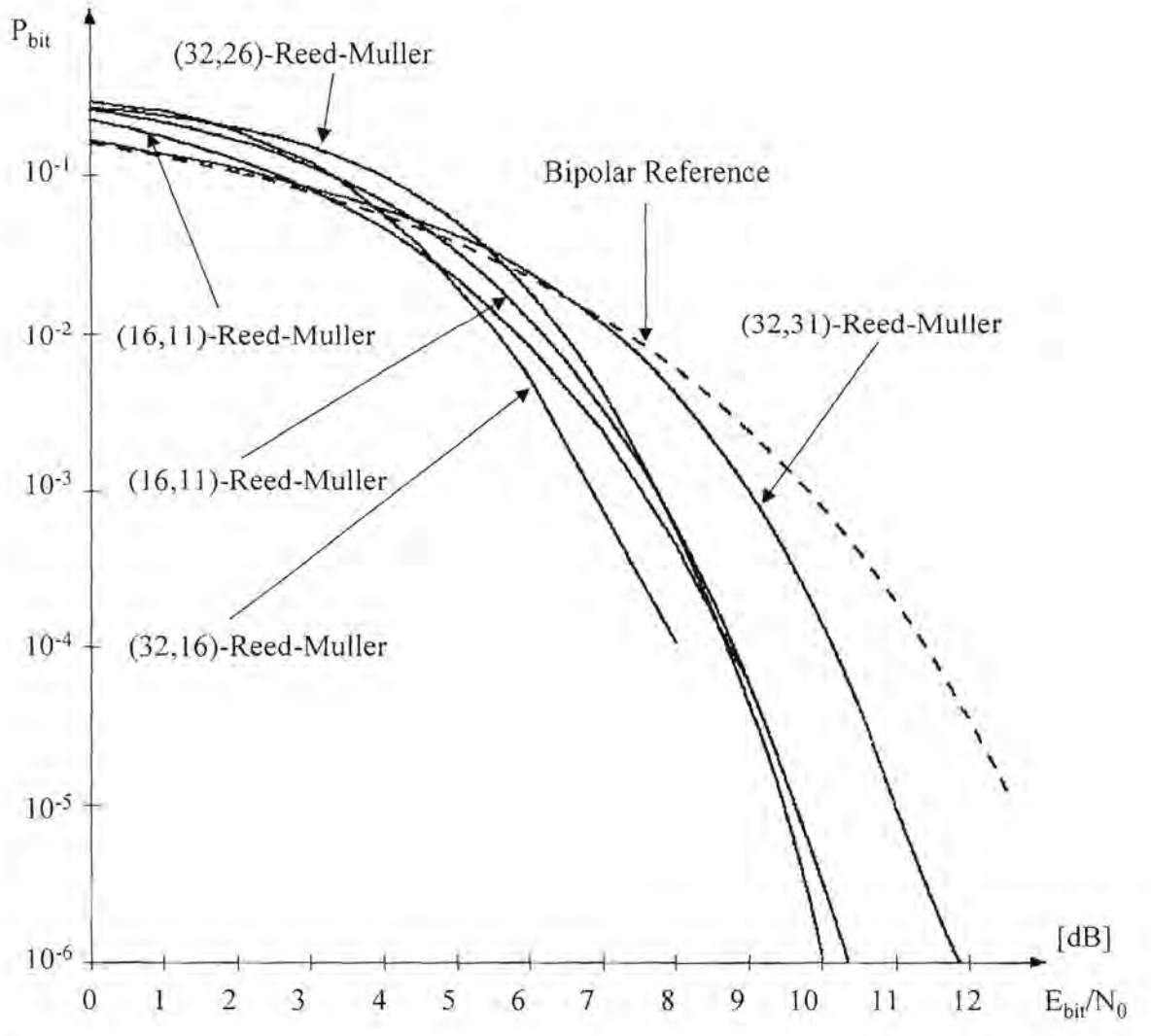

Figure 29 Comparison of different Reed-Muller-Codes

The above figure compares all the different simulation results which were obtained for Reed-Muller codes. What is apparent from the above results, is that the performance achieved is not only dependent on the size of the code, but also on the parameters (*n,k*). This is illustrated by the fact that the (32,31)-Reed-Muller-Code performs a lot worse than the (16,11)-Reed-Muller-Code. Again, the choice of the block code is a very critical step in the design of a error correcting code.

4.2.13 Tabulation of Results

The results achieved by simulation are presented in the table below.

Code	Cross Over HD	Cross Over SDML	Asymptotic Gain HD	Asymtotic Gain SDML
(7,4)-Hamming	9.2 dB	4.0 dB	0.75 dB	2.0 dB
(16,11)-Reed-Muller	9 dB	4.5 dB	1.0 dB	2.5 dB
(16,5)-Reed-Muller	-	3.5 dB	-	4.0 dB
(32,16)-Reed-Muller	-	4.0 dB	-	3.5 dB
(32,26)-Reed-Muller	-	5.9 dB	-	4.0 dB
(32,31)-Reed-Muller	-	7.0 dB	-	1.5 dB
(31,21)-BCH	-	5.1 dB	-	3.5 dB
(15,7)-BCH	-	3.5 dB	-	3.0 dB

Table 1 Tabulation of Results achieved with Viterbi Trellis Decoding

4.2.14 General Discussion of Results

In the above simulations it was shown that it is indeed possible to employ the Viterbi algorithm to decode block codes. Furthermore, it is proven that all block codes which are decoded via their trellis diagram representations conform to the term maximum likelihood decoding. This means that the bit error rate curves that are obtained for the Viterbi decoding of block codes matches the bit error rate curves obtained by analytical decoding - which is a maximum likelihood technique. No coding gain is thus lost when block codes are decoded employing their trellis diagrams. Put differently, it can be said that no sacrifice or penalty is paid in transforming a block code into its respective trellis representation. It is also shown that the simulation results are supported by the theoretical analyses which were presented for reference purposes.

Chapter 5

Trellis Construction and Decoding of Non-Binary Reed-Solomon Codes

5.1 Introduction to Reed Solomon Codes

On the 21st of January 1959 Irvine Reed and Gus Solomon [21] submitted a dull sounding title for a paper to the "Journal of the Society for Industrial and Applied Mathematics"[20]. The paper was accepted and published in June of 1960 under the title "Polynomial Codes over Certain Finite Fields". Little did the unsuspecting world know at that stage what dramatic impact this paper would have on the coding ideology known to man. The enormous contribution that this class of error correcting codes, appropriately called Reed-Solomon Codes[22], has made to the communications sphere, cannot be described in this short paragraph. Reed-Solomon codes have proved their value in channels with a predominantly bursty error characteristic.

5.2 Algebraic Reed-Solomon Code Generation

The class of Reed-Solomon [1], [10],[18],[19],[20,[21],[22] codes are constructed and decoded through the use of finite field arithmetic. These fields are sometimes also referred to as Galois Fields [24][25], after their discoverer. They too have a very wide application in modern communication systems. In order to provide the reader with a more complete set of information, **Appendix F** gives a brief overview of finite fields. The reader may find it necessary to consult this appendix first before proceeding with the following chapter, since none of the information given in **Appendix F** is repeated here.

The original approach to the construction of the Reed-Solomon codes over finite fields was very simple indeed. Assume that k information symbols

$$\left[m_0, m_1, \ldots, m_{k-2}, m_{k-1}\right] \tag{5.1}$$

exist over the finite field GF(q). From these information symbols, the following polynomial can be constructed

$$P(x) = m_0 + m_1 \cdot x + \ldots + m_{k-2} \cdot x^{k-2} + m_{k-1} \cdot x^{k-1} \tag{5.2}$$

As with any code, Reed-Solomon codes have a limited set of code words for every specific Reed-Solomon code. These code words are found by evaluating the polynomial of **Equation 5.2** at each of th q elements of the finite Galois field GF(q). The code words will thus be in the following form

$$c = \left(c_0, c_1, c_2, \ldots, c_{q-1}\right) = \left(P(0), P(\alpha), P\left(\alpha^2\right), \ldots, P\left(\alpha^{q-1}\right)\right) \tag{5.3}$$

The full set of code words can be formed by allowing the k information symbols to assume all possible combinations of values over GF(q). The information symbols are taken from GF(q), resulting in each being able to assume q different values. There is no need to explain why a Reed-Solomon code has a huge set of possible code words. A Reed-Solomon code will have q^k unique code words. From the discussion and definitions given in Chapter 3, it follows that a Reed-Solomon code is linear in nature.

It is quite common to call the parameter k of the Reed-Solomon code the dimension of the code, since the value k forms a vector space of dimension k over the Galois field GF(q). The length of a Reed-Solomon code is commonly termed n in accordance with block code conventions.

Due to the polynomial nature of Reed-Solomon codes, it is possible to revert each Reed-Solomon code to a system of q linear equations with k parameters each. A generic system is given in the equation below:

$$\begin{aligned}
&P(0) = m_0 \\
&P(\alpha) = m_0 + m_1 \cdot \alpha + m_2 \cdot \alpha^2 + \ldots + m_{k-1} \cdot \alpha^{k-1} \\
&P\left(\alpha^2\right) = m_0 + m_1 \cdot \alpha^2 + m_2 \cdot \alpha^4 + \ldots + m_{k-1} \cdot \alpha^{2(k-1)} \\
&\vdots \\
&P\left(\alpha^{q-1}\right) = m_0 + m_1 \cdot \alpha^{q-1} + m_2 \cdot \alpha^{2(q-1)} + \ldots + m_{k-1} \cdot \alpha^{(k-1)(q-1)}
\end{aligned} \tag{5.4}$$

From normal algebra principles, it follows that any k of these expressions can be used in order to construct a system of k equations with k variables. As an example, the first

k expressions of **Equation 5.4** form the following system:

$$\begin{pmatrix} 1 & 0 & 0 & \dots & 0 \\ 1 & \alpha & \alpha^2 & \dots & \alpha^{(k-1)} \\ 1 & \alpha^2 & \alpha^{42} & \dots & \alpha^{2\cdot(k-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \alpha^{k-1} & \alpha^{2\cdot(k-1)} & \dots & \alpha^{(k-1)\cdot(k-1)} \end{pmatrix} \cdot \begin{pmatrix} m_0 \\ m_1 \\ m_2 \\ \vdots \\ m_{k-1} \end{pmatrix} = \begin{pmatrix} P(0) \\ P(\alpha) \\ P(\alpha^2) \\ \vdots \\ P(\alpha^{k-1}) \end{pmatrix} \tag{5.5}$$

It can be shown that this system has a unique solution for the k information symbols by computing the determinant of the following coefficient matrix:

$$\begin{pmatrix} 1 & 0 & 0 & \dots & 0 \\ 1 & \alpha & \alpha^2 & \dots & \alpha^{(k-1)} \\ 1 & \alpha^2 & \alpha^{42} & \dots & \alpha^{2\cdot(k-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \alpha^{k-1} & \alpha^{2\cdot(k-1)} & \dots & \alpha^{(k-1)\cdot(k-1)} \end{pmatrix} \tag{5.6}$$

It can be shown that the above matrix can be reduced to a Vandermonde matrix, and that all Vandermonde matrixes are non-singular. From this, it follows that any k of the expressions in **Equation 5.4** can be used to solve the system.

In the modern construction of Reed-Solomon codes, a construction technique is employed in which a generator matrix approach is used. It is however not necessary at this point to go into the detail of this construction method, as only an overview of Reed-Solomon codes is provided here.

5.3 Conventional Decoding Methods for Reed-Solomon Codes

After the discovery of Reed-Solomon codes, a search for an efficient decoding algorithm commenced. None of the standard decoding techniques at the time were useful for this purpose, for example, simple codes can be decoded through the use of a syndrome look-

up table. This approach is however out of the question for the standard Reed-Solomon codes. A (65, 53, 10) Reed-Solomon code, capable of correcting 5 errors, will have to have a look-up table of 10^{20} symbols. No hardware would be able to handle such immense data sizes.

In their 1960 paper, Gus Reed and Irvine Solomon proposed a decoding algorithm based on the solution of sets of simultaneous equations as described above in **Section 5.2** [60]. Although more useful than the look-up table approach, this techniques only allows for the decoding of very small Reed-Solomon codes. During the 1960's a large amount of work was done towards finding effective decoding algorithms for Reed-Solomon codes. Some of the contributors were people like Peterson, Chien and Forney [26],[27],[28],[29],[30]. Although several techniques were devised, no major improvement on the earlier decoding techniques were found.

The eventual breakthrough came late in 1967 when Berlekamp [32] devised an effective decoding algorithm for nonbinary BCH and Reed-Solomon codes. Following the good work of Berlekamp, Massey [21] showed in 1968 that the decoding problem is equivalent to the generation of linear feedback shift registers which generate the Reed-Solomon codes. The original Berlekamp decoding technique was adapted by Massey to conform to his theory of decoding by linear feedback shift registers. This decoding algorithm is suitably termed the Berlekamp-Massey [21] decoding algorithm.

A lot of different decoding procedure exist, but these will not be described or mentioned here, as this would serve no purpose at all.

The "Holy Grail" of Reed-Solomon decoding research is the maximum likelihood soft decision decoder. A soft decision decoder accepts analog values directly from the channel. The demodulator is not forced to decide which of the q possible symbols a given signal is supposed to represent. The decoder is thus able to make decisions based on the quality of a received signal. For example, the decoder is more willing to assume that a noisy value represents an incorrect symbol than a clean, noise-free signal. All of the information on the "noisiness" of a particular received signal is lost when the demodulator assigns a symbol to the signal prior to decoding.

A lot of work has been done on soft decision maximum likelihood decoders for Reed-Solomon codes, but as to yet, no practical solution has been found. This dissertation

attempts to provide a possible solution for more effective SDML decoding for Reed-Solomon codes.

5.4 Error Correcting Characteristics of Reed-Solomon Codes

The Reed-Solomon code's error correcting capability and performance is now presented. For the purpose of this discussion, it is assumed that t of the code word coordinates are corrupted during transmission, and are received incorrectly. This state of affairs would lead to an incorrect representation in the system of **Equation 5.4**, and would lead to an incorrect solution if **Equation 5.5** is solved. Assuming that there is no knowledge of where the errors are, all possible constructions of systems from **Equation 5.4** are made. There are in fact $\binom{q}{k}$ such systems, $\binom{t+k-1}{k}$ of which produce, when solved, incorrect information symbols. Correct information bits are received as long as the following holds true:

$$\binom{t+k-1}{k} < \binom{q-t}{k} \tag{5.7}$$

This condition only applies when $t + k - 1 < q - t$, which in itself only holds true if the following condition is met $2t < q - k + 1$. A Reed-Solomon code of length q and dimension k can correct up to t errors, where t is given as:

$$t = \left(\frac{q-k+1}{2} \right) \tag{5.8}$$

It was proved in 1964 by Singleton [31] that the family of Reed-Solomon codes provides the best error correction capability for any code with the same length and dimension.

5.5 Various Trellis Construction Techniques

Trellis construction techniques for Reed-Solomon codes can basically be divided into two groups [21],[22],[33],[34]. The first group is termed "Syndrome Trellises of Reed-Solomon Codes" and the second group "Coset Trellises of Reed Solomon Codes". Both these methods aim at obtaining a trellis diagram which would be considered as being minimal. In the following subsections, a brief overview on both these methods is presented. Another method also exists. This method is called the "Shannon Product of Trellises". Due to the way in which the dissertation is organized, this method is presented in Appendix D, since it is referred to by several chapters. It is advisable that the reader familiarize himself with above mentioned procedure, as it is essential to the understanding of the discussion below.

5.5.1 Syndrome Trellis Design for Reed-Solomon Codes

Wolf's method of trellis construction was already presented in **Chapter 3**. This method of trellis construction is under investigation in more detail in **Section 5.6**, and is not pursued further.

A method which is very similar to Wolf's [16] method is presented here as another form of syndrome trellis construction.

A few years back, McElice [19] has proven that trellises found via syndrome techniques are minimal. According to a bound calculated by Wolf [16], the maximum number of states in a syndrome trellis can be estimated as:

$$N_{synd}^{\max} \leq \min\left\{q^{k}, q^{(n-k)}\right\} \tag{5.9}$$

Similar to this, Forney has shown that the minimum number of states at the i-th level of the trellis is equal to:

$$N_i = \frac{q^k}{q_{past}^k \cdot q_{future}^k} \quad i = 1,2,\ldots,n \tag{5.10}$$

where

$$\begin{aligned} k_{past} &= \dim\left(C_{past}\right) \\ k_{future} &= \dim\left(C_{future}\right) \\ C_{past} &= (i, k_{past}, d) \\ C_{future} &= (i, k_{future}, d) \end{aligned} \tag{5.11}$$

As shown before, it is necessary to find the syndrome trellis in a certain Galois field. To be as general as possible, the determination of the trellis diagram is done in GF(q). Firstly, a few definitions are in order at this stage.

The vector of information symbols is defined as

$$X = \left(x_1, x_2, \ldots, x_k\right) \qquad x_i \in GF(q) \tag{5.12}$$

and the encoded vector as

$$Y = \left(y_1, y_2, \ldots, y_n\right) \qquad y_i \in GF(q) \tag{5.13}$$

with $i = 1,2,\ldots,k$ and $j = 1,2,\ldots,n$.

Let **G** be the generator matrix of the Reed-Solomon code in the cyclic form. (see **Chapter2**):

$$\boldsymbol{G} = \begin{pmatrix} g_1 \\ \vdots \\ g_k \end{pmatrix} = \begin{pmatrix} g_1 \\ cs_1(g_1) \\ cs_2(g_2) \\ \vdots \\ cs_{k-1}(g_k) \end{pmatrix} \tag{5.14}$$

In the above equation, $g_i \quad i = 1,2,\ldots,k$ is the i-th row of the generator matrix **G**, and $cs_j(g_1)$ denotes j cyclic shifts of g_1.

The desired Reed-Solomon code can be constructed as a sum of k codes:

$$C = \sum_{j=1}^{k} C_j \tag{5.15}$$

where the j-th code C_j is an (n,1,d) code (k = 1) over GF(q) generated by $\mathbf{G}_j=[g_j]$.

It is relatively easy to show that q code words in C_j can be obtained as:

$$C_j = x_j\left[g_j\right] \tag{5.16}$$

The corresponding sub-trellises T_j, where $j=1,2,...,k$, start at the generic node called the root node, and terminate in the finishing node called the destination node. These sub-trellises have $(n+1)$ vertices and the number of states in the i^{th} vertice is defined as being:

$$\begin{aligned} N_0 &= N_n = 1 \\ N_t &= \begin{cases} q & \text{if } g_j^i \neq 0 \text{ and } g_j^{i+1} \neq 0 \\ 1 & \text{for all other cases} \end{cases} \end{aligned} \tag{5.17}$$

where $t = 0,1,2,3,...,n$ and represents the t^{th} element of g_j.

If T_j is a syndrome trellis of the elementary code C_j generated by g_i then the combined trellis can be obtained from $T = T_1 \cdot T_2 \cdot \ldots \cdot T_k$. In other words, the syndrome trellis of the code C generated by G is given as T.

The state profile of the syndrome trellis can be obtained from:

$$\begin{aligned} N_0 &= N_n = 1 \\ N_t &= q^m \quad t = 1,2,\ldots,n-1 \end{aligned} \tag{5.18}$$

where m is the number of non-zero elements in the t^{th} column of **G** which are followed by any other non-zero element.

The above process is illustrated by means of an example in **Appendix E**.

5.5.2 Coset Trellis Design for Reed-Solomon Codes

A coset trellis represents a set of parallel sub-trellises [5], each corresponding to one of the cosets of the basic code. Such a trellis allows a reduction in the decoder complexity, since all the sub-trellises have identical structure and differ only in the labeling of their respective trellis branches. Again, the reader is referred to **Appendix D**, since this method also involves Shannon's "Product of Trellises" method. Below, it is now shown how the Shannon product of trellises can be successfully employed to find a minimal coset trellis representation of a Reed-Solomon code.

The first step in obtaining the coset trellis representation, is the calculation of the state profile from the minimal syndrome trellis representation of the Reed-Solomon code. This profile can be obtained by calculating the minimal number of states at every node of the trellis.

From the above calculated profile, N_{synd} splitting points are chosen which have a similar number of states. The next step is defining the state and label size profiles of the desirable trellis:

$$N_{\text{coset}} = \left[1, N_1, N_2, \ldots, N_c - 1, 1\right] \tag{5.19}$$

$$l_{\text{coset}} = \left[l_1, l_2, \ldots, l_{N_c-1}\right] \tag{5.20}$$

where N_c is a number of columns (vertices) in the desired coset trellis, and all vertices have a similar number of states:

$$N_i = N_j \qquad i, j = 1,2,\ldots,N_c - 1 \tag{5.21}$$

It is apparent, that in the general case the following applies:

$$l_i \neq l_j \qquad i, j = 1,2,\ldots,N_c - 1 \tag{5.22}$$

Finally, the overall generator matrix of the complete code can be presented in the following form:

$$\boldsymbol{G} = \begin{pmatrix} g_1 \\ g_2 \\ \vdots \\ g_k \end{pmatrix} = \begin{pmatrix} \boldsymbol{G}_1 & \boldsymbol{G}_2 & \cdots & \boldsymbol{G}_{N_c-1} \end{pmatrix} \tag{5.23}$$

where $\mathbf{G}_i$, with $i = 1,2,...,N_c - 1$, has l columns and k rows. Each row of **G** is used to design the trellis diagram of the (n,l,d) code over GF(q) with the label size profile given by **Equation 5.20**. The overall trellis diagram can be obtained as the Shannon product of k constituent trellises.

Again, as mentioned before, examples to illustrate the process are given in the second part of **Appendix E**.

5.5.3 Modified Trellis Design Procedure for Reed-Solomon Codes

This next method is basically a variation on Wolf's method. The basic procedure outlined in **Chapter 3** still applies to the trellis design procedure for Reed-Solomon codes. It is however essential to understand all the implications of using a higher order Galois field than GF(2).

In order to provide a very clear example of the procedure and also due to the large complexity, a very simple example is given, namely a (7,5) RS-code in GF(4). Since it was decided to keep the Galois field set as small as possible (not counting the binary case),a Galois field of GF(4) was chosen. In order to have a full understanding on this issue, the reader is referred back to the previous sections of this chapter, in which the basics of Reed-Solomon codes and Galois fields are presented.

The following definitions are in order here.

The Galois field that was chosen is:

$$GF(2^2) = GF(4) \tag{5.24}$$

The number of states in the trellis are:

$$N = (2^{GF(2^2)})^{n-k} = (2^2)^2 = (4)^2 = 16 \tag{5.25}$$

The following generator polynomial was chosen for the example:

$$\begin{aligned} g(x) &= (x+\alpha)\cdot(x+\alpha^2) \\ &= (x^2 + \alpha\cdot x + \alpha^2\cdot x + \alpha^3) \\ &= (1\cdot x^2 + (\alpha^2+\alpha)\cdot x^1 + 1\cdot x^0) \\ &= (1\cdot x^2 + 1\cdot x^1 + 1\cdot x^0) \end{aligned} \tag{5.26}$$

The following generator matrix can be constructed from the above given generator polynomial. Note that the generator matrix is in cyclic form, as each row is just a shifted replica of the generator polynomial coefficients.

$$\boldsymbol{G} = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 \end{pmatrix} \tag{5.27}$$

From this generator matrix, the following information is obtained:

The trellis depth equals:

$$T_d = Num(\boldsymbol{G}_{columns}) = n = 7 \tag{5.28}$$

The number of possible code words are:

$$N_{CW} = (GF(2^2))^n = (2^2)^7 = 4^7 = 16384 \tag{5.29}$$

The number of branches emanating from each node is determined by the number of symbols in the code. Thus, for the binary case, the number of symbols would be 2 and the number of associated branches also 2. However, for a code in GF(4), there are 4 branches associated with the following 4 symbols:

$$Symbols = \{0,\ \ 1,\ \ \alpha,\ \ \alpha^2\} \tag{5.30}$$

At this point it can already be said that each node will have the following branch structure depicted in **Figure 30** below:

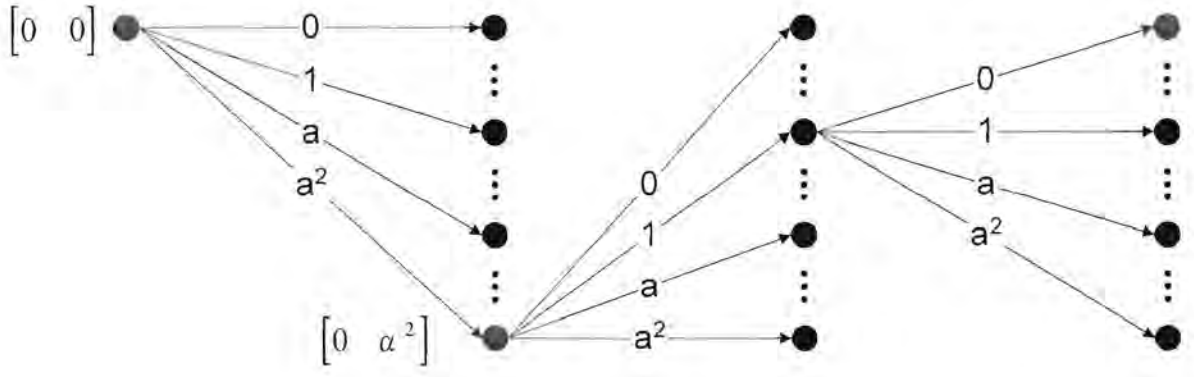

Figure 30 Branch Structure of Reed-Solomon Trellis is Galois Field GF(4)

Next, from the generator matrix **G** given by **Equation 5.27**, the parity check matrix can be calculated. This is done by performing elementary matrix operations on the matrix as outlined in **Chapter 2**. For a complete discussion on this topic, the reader is referred to **Appendix B**. Only a few steps are shown below.

$$\boldsymbol{G} = \left(\begin{array}{ccccc|cc} 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 \end{array}\right) = \left(\begin{array}{ccccc|cc} 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 \end{array}\right) \tag{5.31}$$

This is further transformed into:

$$\boldsymbol{G} = \left(\begin{array}{ccccc|cc} 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 \end{array}\right) \tag{5.32}$$

From the above systematic generator matrix **G** it is possible to read off the parity check matrix in the following form:

$$\boldsymbol{H} = \begin{pmatrix} 1 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 & 0 \end{pmatrix} \tag{5.33}$$

Next a lookup table can be constructed to simplify the derivation of the trellis diagram. The lookup table is based on elementary Galois field mathematics presented above in **Section 5.2**.

The lookup table for $GF(2^2)$ is divided into 4 separate lookup tables, one for each symbol in the Galois field. These tables are constructed using the primary polynomial in GF(4). This polynomial can be expressed as:

$$\alpha^2 = \alpha + 1 \tag{5.34}$$

The table for code symbol "0":

$$\begin{vmatrix} 0+0=0 \\ 0+1=1 \\ 0+\alpha=\alpha \\ 0+\alpha^2=\alpha^2 \end{vmatrix} \tag{5.35}$$

The table for code symbol "1":

$$\begin{vmatrix} 1+0=1 \\ 1+1=0 \\ 1+\alpha=\alpha^2 \\ 1+\alpha^2=\alpha \end{vmatrix} \tag{5.36}$$

The table for code symbol "α ":

$$\begin{vmatrix} \alpha+0=\alpha \\ \alpha+1=\alpha^2 \\ \alpha+\alpha=0 \\ \alpha+\alpha^2=1 \end{vmatrix} \tag{5.37}$$

The table for code symbol "α^2":

$$\begin{vmatrix} \alpha^2+0=\alpha^2 \\ \alpha^2+1=\alpha \\ \alpha^2+\alpha=1 \\ \alpha^2+\alpha^2=0 \end{vmatrix} \tag{5.38}$$

Since the number of states in the trellis is 16, they can be divided into 4 sub-parts containing 4 states each. The significance of this is explained later. The 4 sub-parts along with their respective 4 states each are numbered as follows:

Sub-part 1 along with first set of four states:

$$SP_1 = \left\{\begin{bmatrix}0 & 0\end{bmatrix},\begin{bmatrix}0 & 1\end{bmatrix},\begin{bmatrix}0 & \alpha\end{bmatrix},\begin{bmatrix}0 & \alpha^2\end{bmatrix}\right\} \tag{5.39}$$

Sub-part 2 along with second set of four states:

$$SP_2 = \left\{\begin{bmatrix}1 & 0\end{bmatrix},\begin{bmatrix}1 & 1\end{bmatrix},\begin{bmatrix}1 & \alpha\end{bmatrix},\begin{bmatrix}1 & \alpha^2\end{bmatrix}\right\} \tag{5.40}$$

Sub-part 3 along with third set of four states:

$$SP_3 = \left\{\begin{bmatrix}\alpha & 0\end{bmatrix},\begin{bmatrix}\alpha & 1\end{bmatrix},\begin{bmatrix}\alpha & \alpha\end{bmatrix},\begin{bmatrix}\alpha & \alpha^2\end{bmatrix}\right\} \tag{5.41}$$

Sub-part 4 along with fourth set of four states:

$$SP_4 = \left\{\left[\alpha^2 \quad 0\right],\left[\alpha^2 \quad 1\right],\left[\alpha^2 \quad \alpha\right],\left[\alpha^2 \quad \alpha^2\right]\right\} \tag{5.42}$$

The next step in the design process is to identify trellis branches associated with the individual code words. This is done by finding all the connecting branches between nodes of the trellis. The starting node is taken as state [0,0], and paths are calculated from here onwards. The reader is again reminded that this process is very similar to the one provided in **Chapter 3**. It should also be noted that paths emanating from nodes are only calculated for states that have at least one incoming branch. An exception to the rule is the above mentioned starting state.

As shown in **Figure 20**, each symbol in the Galois field causes a branch to emanate from each node, implying, at first glance that 4 calculations will have to be done in order to find the 4 branches emanating from every node or state.

For the first section of the trellis, the first column of the parity check matrix is used for all calculations.

For reference, the parity check matrix is repeated. For completeness, the unit matrix has been appended to the end:

$$\boldsymbol{H} = \left(\begin{array}{ccccc|cc} 1 & 1 & 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 \end{array}\right) \tag{5.43}$$

The following calculations yield the first 4 branches emanating from node [0 0] where each branch is labeled with a specific input symbol, 0, 1, α or α^2:

$$h_1 = \begin{pmatrix} 1 \\ 1 \end{pmatrix} \tag{5.44}$$

$$\begin{aligned}
\begin{bmatrix}0 & 0\end{bmatrix} \rightarrow 0: &\quad \begin{bmatrix}0 & 0\end{bmatrix} + 0\cdot\begin{bmatrix}1 & 1\end{bmatrix} = \begin{bmatrix}0 & 0\end{bmatrix} \\
1: &\quad \begin{bmatrix}0 & 0\end{bmatrix} + 1\cdot\begin{bmatrix}1 & 1\end{bmatrix} = \begin{bmatrix}1 & 1\end{bmatrix} \\
\alpha: &\quad \begin{bmatrix}0 & 0\end{bmatrix} + \alpha\cdot\begin{bmatrix}1 & 1\end{bmatrix} = \begin{bmatrix}\alpha & \alpha\end{bmatrix} \\
\alpha^2: &\quad \begin{bmatrix}0 & 0\end{bmatrix} + \alpha^2\cdot\begin{bmatrix}1 & 1\end{bmatrix} = \begin{bmatrix}\alpha^2 & \alpha^2\end{bmatrix}
\end{aligned} \tag{5.45}$$

It can be seen that quite a number of calculations are involved in order to find the nodes that are connected by a specific connection branch associated with a specific symbol. In order to still give a good overview of the process, a shorthand notation is employed to summarise all the relevant results of the calculations.

It can be seen that the next batch of calculations will deliver a set of 16 active nodes, due to the fact that 4 branches exit any active node. An active node is any node which has at least one trellis branch entering it. When the trellis is "fanned-out", in other words such that each node or state has at least one branch entering it, the calculations will yield 64 branches interconnecting the various nodes.

The method used to obtain the trellis branches, is based on the same structure used in **Equation 5.45**, but by only displaying the starting and ending nodes of each branch. The details of the calculations are left out. These may be readily filled in by the reader without a lot of effort.

For each depth of the trellis, the next column of the parity check matrix is used. This means, that at the last section of the trellis, the last column of the parity check matrix is employed to base the calculations on and so on.

On the next few pages, the complete results of all the calculations are presented.

The results for the first section of the trellis are presented on this page.

The relevant column of the parity check matrix is the following:

$$h_1 = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

The following equation was used to calculate the table below:

$$NewState = OldState + Symbols \cdot \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

Old State	New States	Old State	New States	Old State	New States	Old State	New States
[0 0]	[0 0]	[1 0]	-	[α 0]	-	[α² 0]	-
	[1 1]		-		-		-
	[α α²]		-		-		-
	[α² α²]		-		-		-
[0 1]	-	[1 1]	-	[α 1]	-	[α² 1]	-
	-		-		-		-
	-		-		-		-
	-		-		-		-
[0 α]	-	[1 α]	-	[α α]	-	[α² α]	-
	-		-		-		-
	-		-		-		-
	-		-		-		-
[0 α²]	-	[1 α²]	-	[α α²]	-	[α² α²]	-
	-		-		-		-
	-		-		-		-
	-		-		-		-

Table 2 Nodes for State 1 of (7,5) RS-Code in GF(4)

The results for the second section of the trellis are presented on this page.

The relevant column of the parity check matrix is the following:

$$\boldsymbol{h}_2 = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

The following equation was used to calculate the table below:

$$NewState = OldState + Symbols \cdot \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

Old State	New States	Old State	New States	Old State	New States	Old State	New States
[0 0]	[0 0]	[1 0]	-	[α 0]	-	[α^2 0]	-
	[1 0]		-		-		-
	[α 0]		-		-		-
	[α^2 0]		-		-		-
[0 1]	-	[1 1]	[1 1]	[α 1]	-	[α^2 1]	-
	-		[0 1]		-		-
	-		[α^2 1]		-		-
	-		[α 1]		-		-
[0 α]	-	[1 α]	-	[α α]	[α α]	[α^2 α]	-
	-		-		[α^2 α]		-
	-		-		[0 α]		-
	-		-		[1 α]		-
[0 α^2]	-	[1 α^2]	-	[α α^2]	-	[α^2 α^2]	[α^2 α^2]
	-		-		-		[α α^2]
	-		-		-		[1 α^2]
	-		-		-		[0 α^2]

Table 3 Nodes for State 2 of (7,5) RS-Code in GF(4)

The results for the third section of the trellis are presented on this page.

The relevant column of the parity check matrix is the following:

$$\boldsymbol{h}_3 = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

The following equation was used to calculate the table below:

$$NewState = OldState + Symbols \cdot \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

Old State	New States	Old State	New States	Old State	New States	Old State	New States
[0 0]	[0 0]	[1 0]	[1 0]	[α 0]	[α 0]	[α^2 0]	[α^2 0]
	[0 1]		[1 1]		[α 1]		[α^2 1]
	[0 α]		[1 α]		[α α]		[α^2 α]
	[0 α^2]		[1 α^2]		[α α^2]		[α^2 α^2]
[0 1]	[0 1]	[1 1]	[1 1]	[α 1]	[α 1]	[α^2 1]	[α^2 1]
	[0 0]		[1 0]		[α 0]		[α^2 0]
	[0 α^2]		[1 α^2]		[α α^2]		[α^2 α^2]
	[0 α]		[1 α]		[α α]		[α^2 α]
[0 α]	[0 α]	[1 α]	[1 α]	[α α]	[α α]	[α^2 α]	[α^2 α]
	[0 α^2]		[1 α^2]		[α α^2]		[α^2 α^2]
	[0 0]		[1 0]		[α 0]		[α^2 0]
	[0 1]		[1 1]		[α 1]		[α^2 1]
[0 α^2]	[0 α^2]	[1 α^2]	[1 α^2]	[α α^2]	[α α^2]	[α^2 α^2]	[α^2 α^2]
	[0 α]		[1 α]		[α α]		[α^2 α]
	[0 1]		[1 1]		[α 1]		[α^2 1]
	[0 0]		[1 0]		[α 0]		[α^2 0]

Table 4 Nodes for State 3 of (7,5) RS-Code in GF(4)

The results for the fourth section of the trellis are presented on this page.

The relevant column of the parity check matrix is the following:

$$\boldsymbol{h}_4 = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

The following equation was used to calculate the table below:

$$NewState = OldState + Symbols \cdot \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

Old State	New States	Old State	New States	Old State	New States	Old State	New States
[0 0]	[0 0]	[1 0]	[1 0]	[α 0]	[α 0]	[α^2 0]	[α^2 0]
	[1 1]		[0 1]		[α^2 1]		[α 1]
	[α α^2]		[α^2 α]		[0 α]		[1 α]
	[α^2 α^2]		[α α^2]		[1 α^2]		[0 α^2]
[0 1]	[0 1]	[1 1]	[1 1]	[α 1]	[α 1]	[α^2 1]	[α^2 1]
	[1 0]		[0 0]		[α^2 0]		[α 0]
	[α α^2]		[α^2 α^2]		[0 α^2]		[1 α^2]
	[α^2 α]		[α α]		[1 α]		[0 α]
[0 α]	[0 α]	[1 α]	[1 α]	[α α]	[α α]	[α^2 α]	[α^2 α]
	[1 α^2]		[0 α^2]		[α^2 α^2]		[α α^2]
	[α 0]		[α^2 0]		[0 0]		[1 0]
	[α^2 1]		[α 1]		[1 1]		[0 1]
[0 α^2]	[0 α^2]	[1 α^2]	[1 α^2]	[α α^2]	[α α^2]	[α^2 α^2]	[α^2 α^2]
	[1 α]		[0 α]		[α^2 α]		[α α]
	[α 1]		[α^2 1]		[0 1]		[1 1]
	[α^2 0]		[α 0]		[1 0]		[0 0]

Table 5 Nodes for State 4 of (7,5) RS-Code in GF(4)

The results for the fifth section of the trellis are presented on this page.

The relevant column of the parity check matrix is the following:

$$\boldsymbol{h}_5 = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

The following equation was used to calculate the table below:

$$NewState = OldState + Symbols \cdot \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

Old State	New States	Old State	New States	Old State	New States	Old State	New States
[0 0]	[0 0]	[1 0]	[1 0]	[α 0]	[α 0]	[α^2 0]	[α^2 0]
	[1 0]		[0 0]		[α^2 0]		[α 0]
	[α 0]		[α^2 0]		[0 0]		[1 0]
	[α^2 0]		[α 0]		[1 0]		[0 0]
[0 1]	[0 1]	[1 1]	[1 1]	[α 1]	[α 1]	[α^2 1]	[α^2 1]
	[1 1]		[0 1]		[α^2 1]		[α 1]
	[α 1]		[α^2 1]		[0 1]		[1 1]
	[α^2 1]		[α 1]		[1 1]		[0 1]
[0 α]	[0 α]	[1 α]	[1 α]	[α α]	[α α]	[α^2 α]	[α^2 α]
	[1 α]		[0 α]		[α^2 α]		[α α]
	[α α]		[α^2 α]		[0 α]		[1 α]
	[α^2 α]		[α α]		[1 α]		[0 α]
[0 α^2]	[0 α^2]	[1 α^2]	[1 α^2]	[α α^2]	[α α^2]	[α^2 α^2]	[α^2 α^2]
	[1 α^2]		[0 α^2]		[α^2 α^2]		[α α^2]
	[α α^2]		[α^2 α^2]		[0 α^2]		[1 α^2]
	[α^2 α^2]		[α α^2]		[1 α^2]		[0 α^2]

Table 6 Nodes for State 5 of (7,5) RS-Code in GF(4)

The results for the sixth section of the trellis are presented on this page.

The relevant column of the parity check matrix is the following:

$$\boldsymbol{h}_6 = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

The following equation was used to calculate the table below:

$$NewState = OldState + Symbols \cdot \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

Old State	New States	Old State	New States	Old State	New States	Old State	New States
[0 0]	[0 0]	[1 0]	[1 0]	[α 0]	[α 0]	[α^2 0]	[α^2 0]
	[1 0]		[0 0]		[α^2 0]		[α 0]
	[α 0]		[α^2 0]		[0 0]		[1 0]
	[α^2 0]		[α 0]		[1 0]		[0 0]
[0 1]	[0 1]	[1 1]	[1 1]	[α 1]	[α 1]	[α^2 1]	[α^2 1]
	[1 1]		[0 1]		[α^2 1]		[α 1]
	[α 1]		[α^2 1]		[0 1]		[1 1]
	[α^2 1]		[α 1]		[1 1]		[0 1]
[0 α]	[0 α]	[1 α]	[1 α]	[α α]	[α α]	[α^2 α]	[α^2 α]
	[1 α]		[0 α]		[α^2 α]		[α α]
	[α α]		[α^2 α]		[0 α]		[1 α]
	[α^2 α]		[α α]		[1 α]		[0 α]
[0 α^2]	[0 α^2]	[1 α^2]	[1 α^2]	[α α^2]	[α α^2]	[α^2 α^2]	[α^2 α^2]
	[1 α^2]		[0 α^2]		[α^2 α^2]		[α α^2]
	[α α^2]		[α^2 α^2]		[0 α^2]		[1 α^2]
	[α^2 α^2]		[α α^2]		[1 α^2]		[0 α^2]

Table7 Nodes for State 6 of (7,5) RS-Code in GF(4)

The results for the seventh section of the trellis are given on this page.

The relevant column of the parity check matrix is the following:

$$\boldsymbol{h}_7 = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

The following equation was used to calculate the table below:

$$NewState = OldState + Symbols \cdot \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

Old State	New States	Old State	New States	Old State	New States	Old State	New States
[0 0]	[0 0]	[1 0]	[1 0]	[α 0]	[α 0]	[α^2 0]	[α^2 0]
	[0 1]		[1 1]		[α 1]		[α^2 1]
	[0 α]		[1 α]		[α α]		[α^2 α]
	[0 α^2]		[1 α^2]		[α α^2]		[α^2 α^2]
[0 1]	[0 1]	[1 1]	[1 1]	[α 1]	[α 1]	[α^2 1]	[α^2 1]
	[0 0]		[1 0]		[α 0]		[α^2 0]
	[0 α^2]		[1 α^2]		[α α^2]		[α^2 α^2]
	[0 α]		[1 α]		[α α]		[α^2 α]
[0 α]	[0 α]	[1 α]	[1 α]	[α α]	[α α]	[α^2 α]	[α^2 α]
	[0 α^2]		[1 α^2]		[α α^2]		[α^2 α^2]
	[0 0]		[1 0]		[α 0]		[α^2 0]
	[0 1]		[1 1]		[α 1]		[α^2 1]
[0 α^2]	[0 α^2]	[1 α^2]	[1 α^2]	[α α^2]	[α α^2]	[α^2 α^2]	[α^2 α^2]
	[0 α]		[1 α]		[α α]		[α^2 α]
	[0 1]		[1 1]		[α 1]		[α^2 1]
	[0 0]		[1 0]		[α 0]		[α^2 0]

Table 8 Nodes for State 7 of (7,5) RS-Code in GF(4)

From the above tables, it is possible to construct a complete trellis by connecting all relevant branches on an empty trellis grid.

Before this is done however, a few important comments are in order. It is obvious that quite a lot of calculations are needed in order to be able to draw the final trellis diagram. It is easy to see that the complexity will increase as n and k of the code increases. However, as the order of the Galois field increases an even greater increase in complexity is affected.

One possible solution to overcome the complexity is to employ a lookup table, which alleviates the need for extensive branch calculations.

Observing the tables that have been constructed a definite pattern emerges. The practical implication of this is, that once a point in the lookup table is found, the next outcomes may be uniquely determined from the entries of those tables which yielded the last output result. The other elements can just be read of in sequence. In this way, the derivation of the complete trellis is vastly simplified.

The next step in the design process is to produce a graphical display of the trellis diagram from the calculated data. It is a fairly simple process, in which a starting and ending node or state are connected with an interconnecting branch. The branches in the above tables of results are defined by listing the starting and ending states of a particular section in the trellis. The associated symbol, which is the output for that particular branch, may also be found from above the tables, since for each given node, the first result was obtained from the first symbol, the second result was obtained from the second symbol and so on. This is illustrated on the next few pages, where trellis patterns for subsequent sections of the trellis diagrams are displayed consecutively.

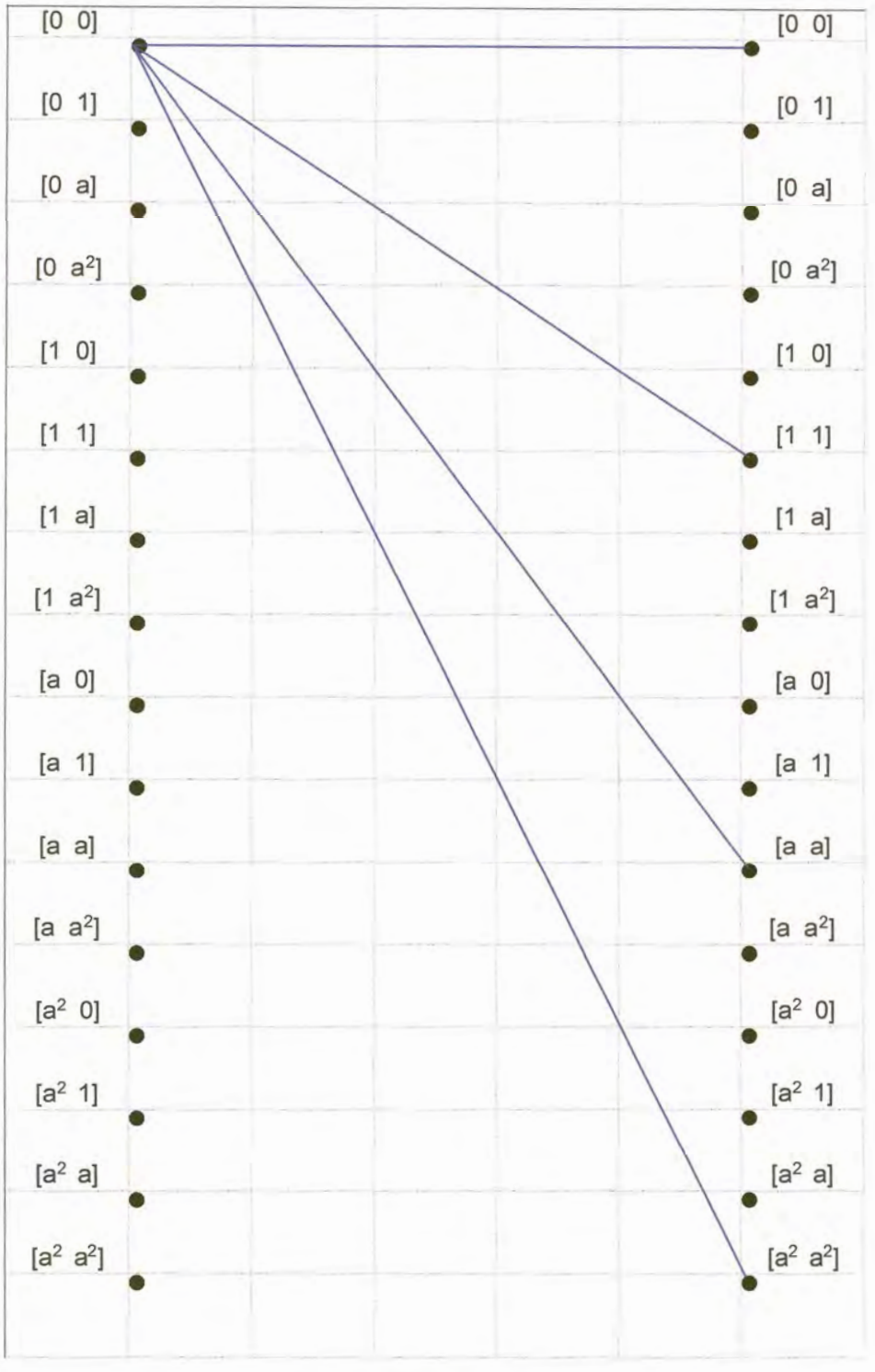

Figure 31 Section 1 of (7,5) Reed-Solomon Trellis in GF(4).

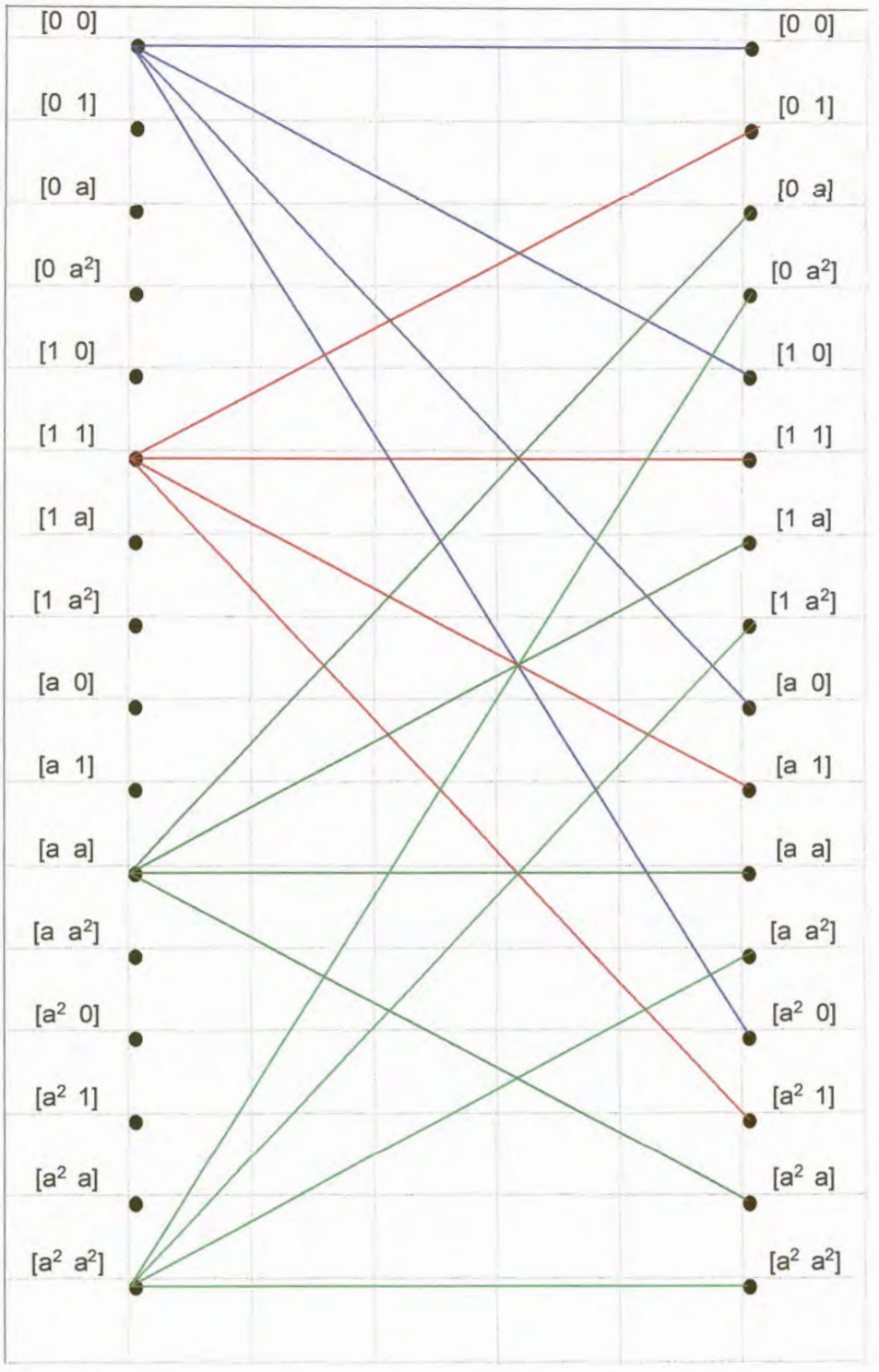

Figure 32 Section 2 of (7,5) Reed-Solomon Trellis in GF(4).

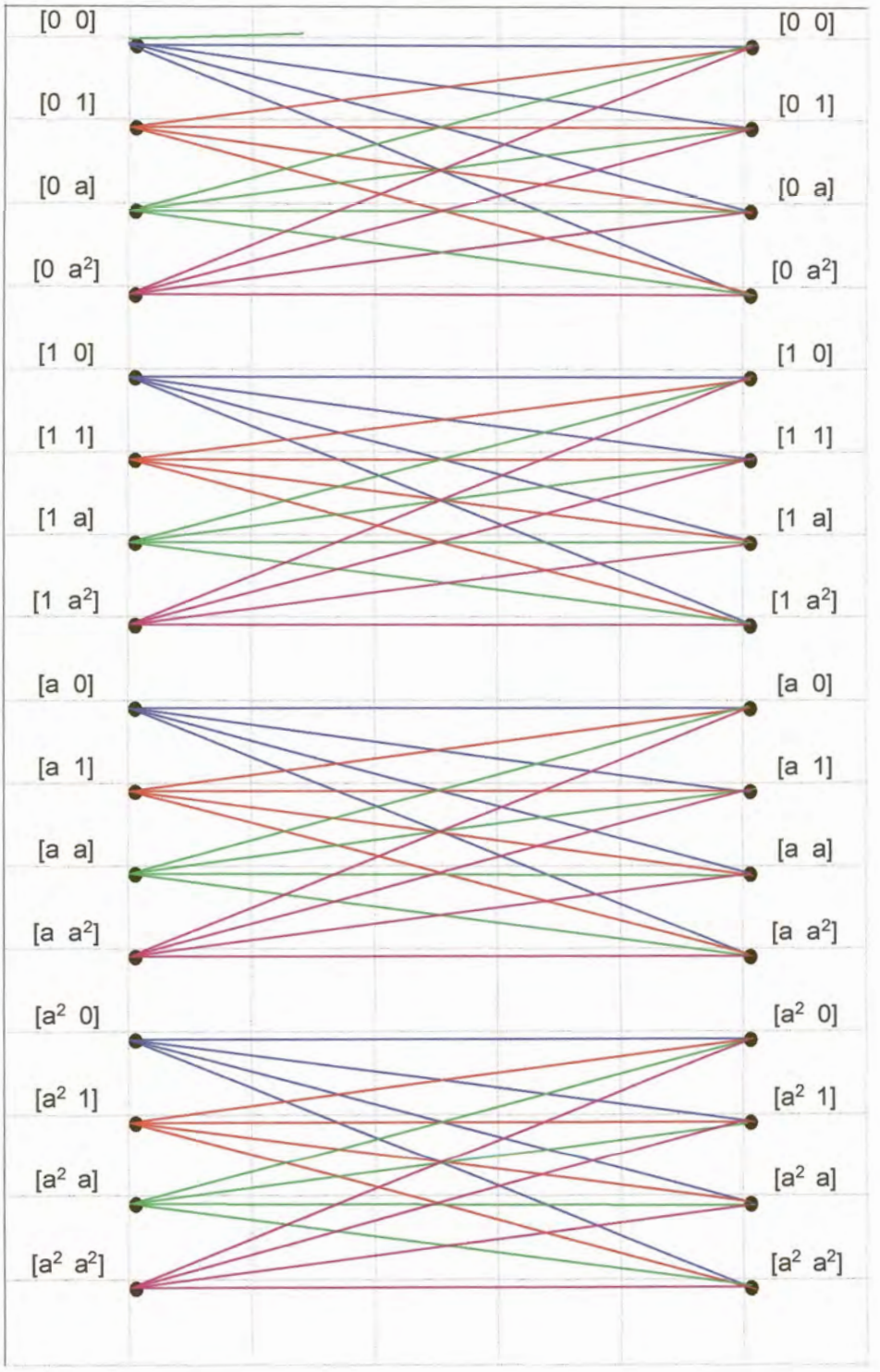

Figure 33 Section 3 of (7,5) Reed-Solomon Trellis in GF(4)

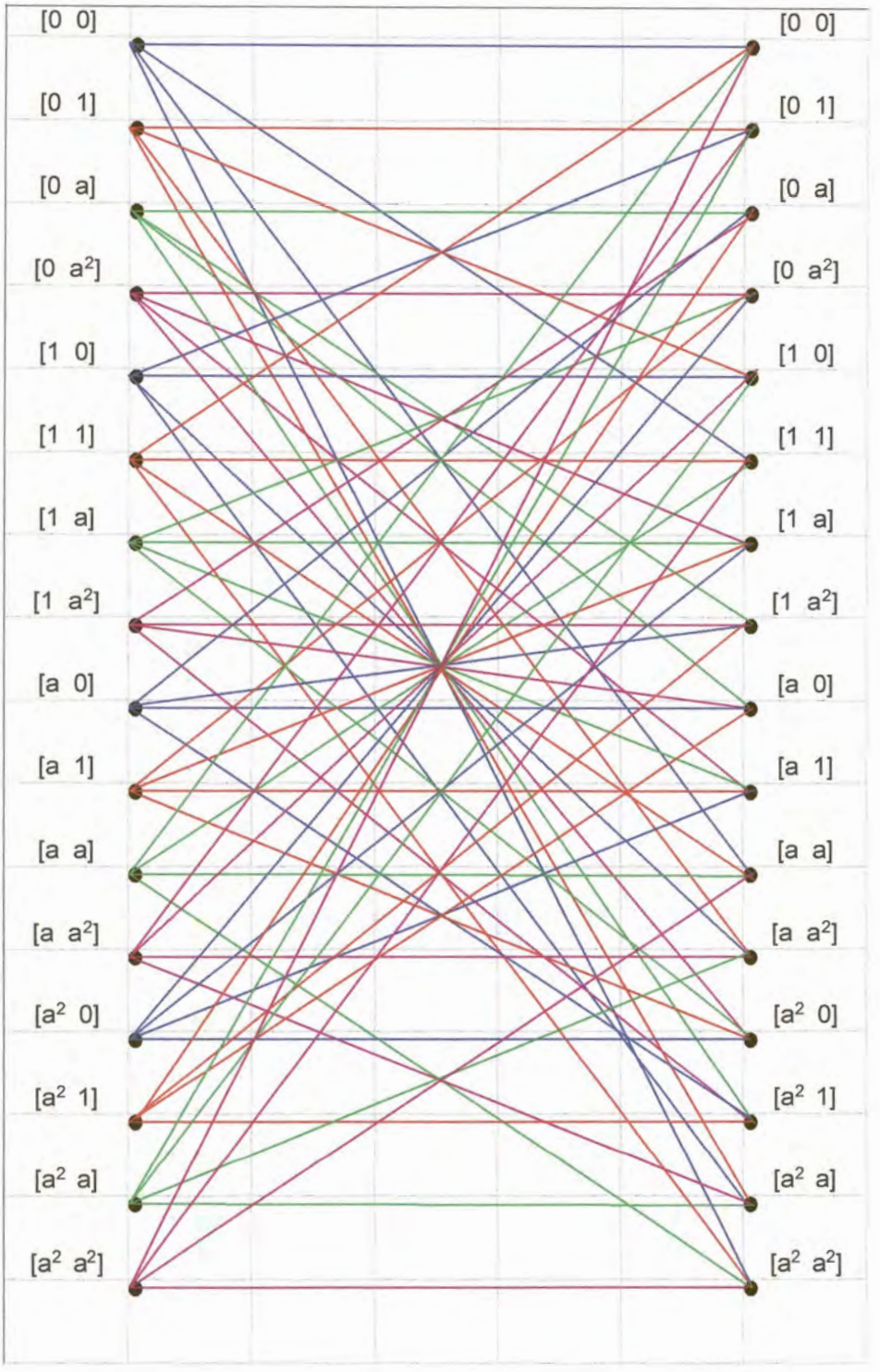

Figure 34 Section 4 of (7,5) Reed-Solomon Trellis in GF(4).

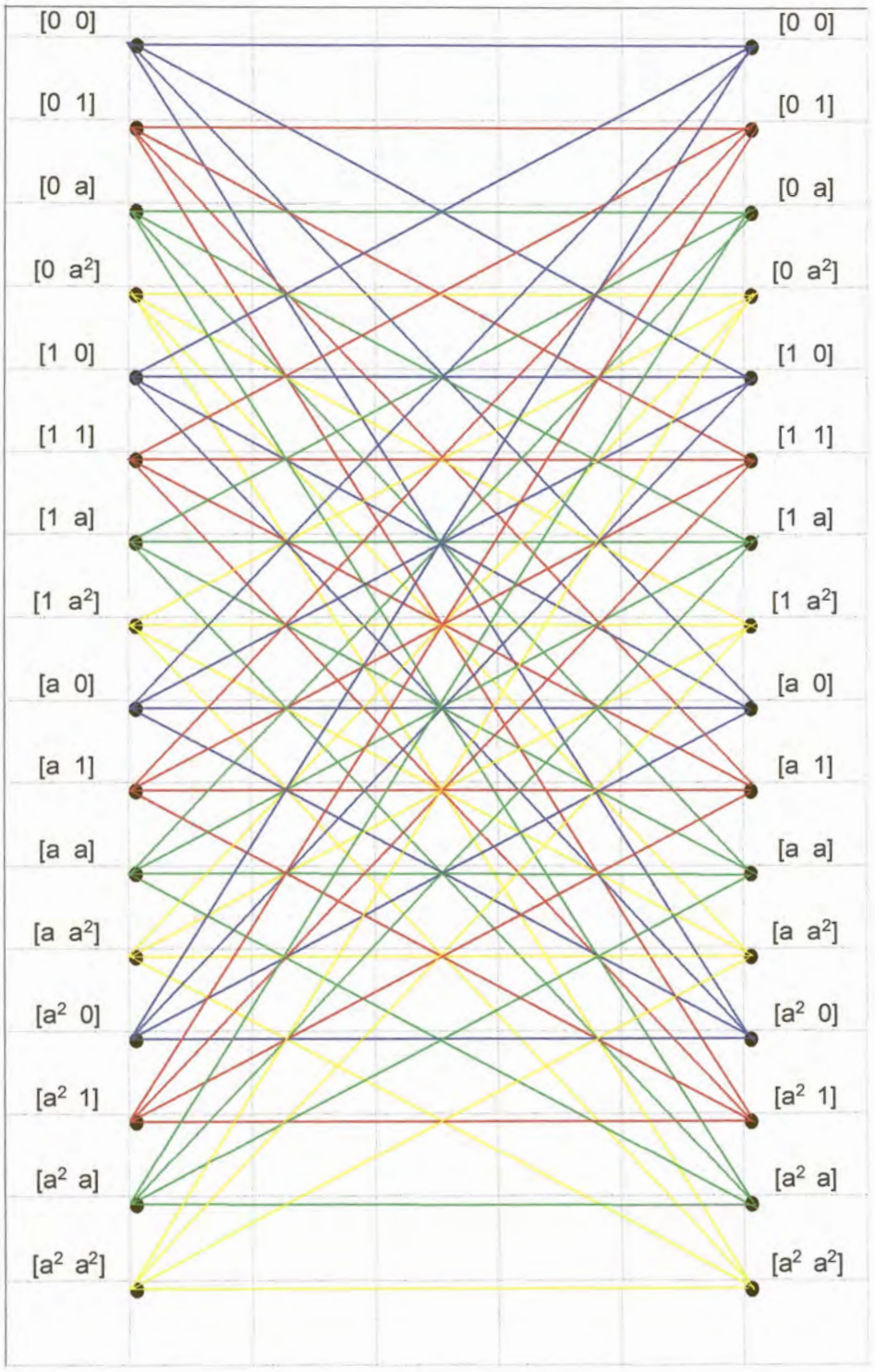

Figure 35 Section 5 of (7,5) Reed-Solomon Trellis in GF(4).

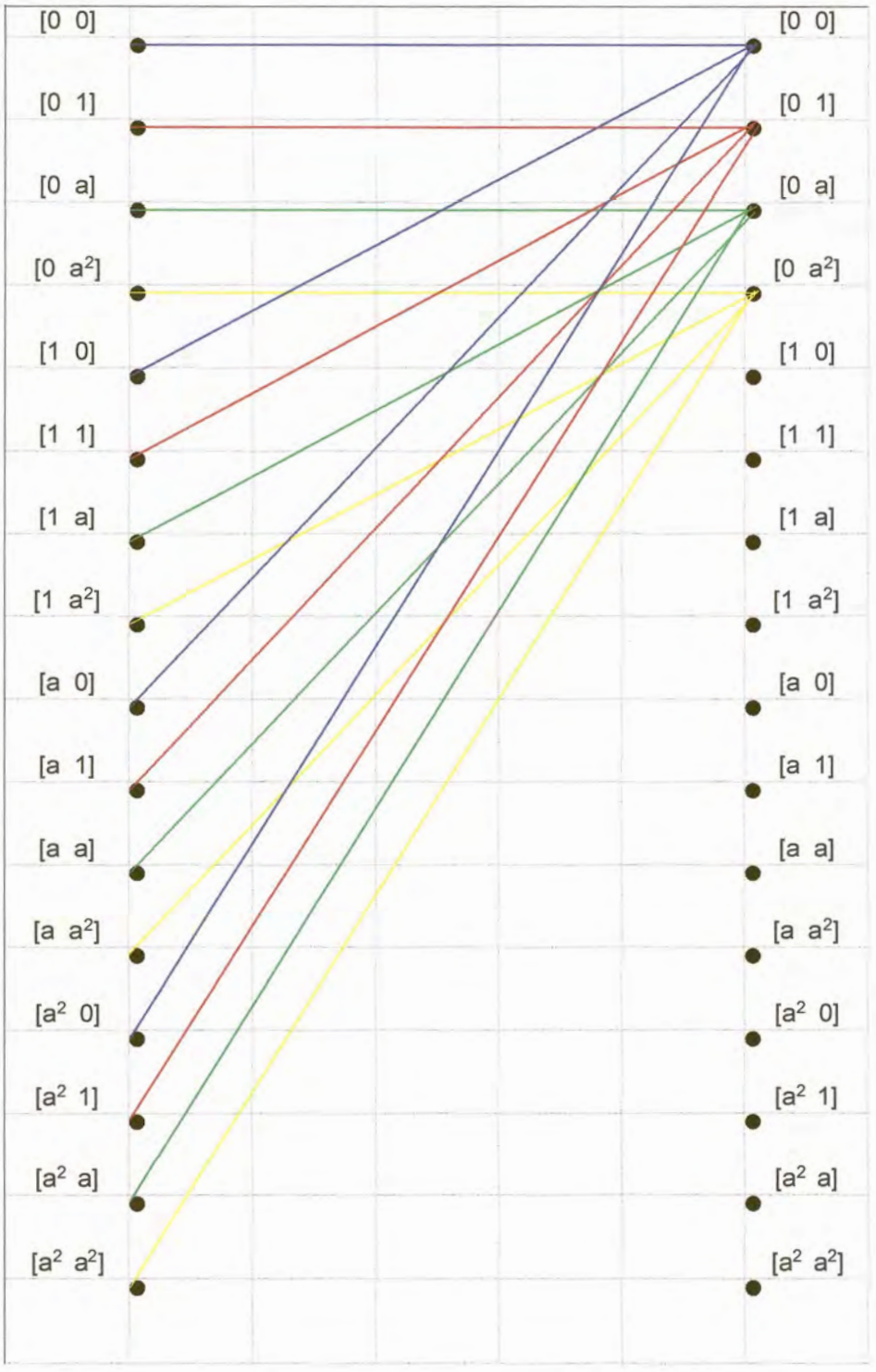

Figure 36 Section 6 of (7,5) Reed-Solomon Trellis in GF(4).

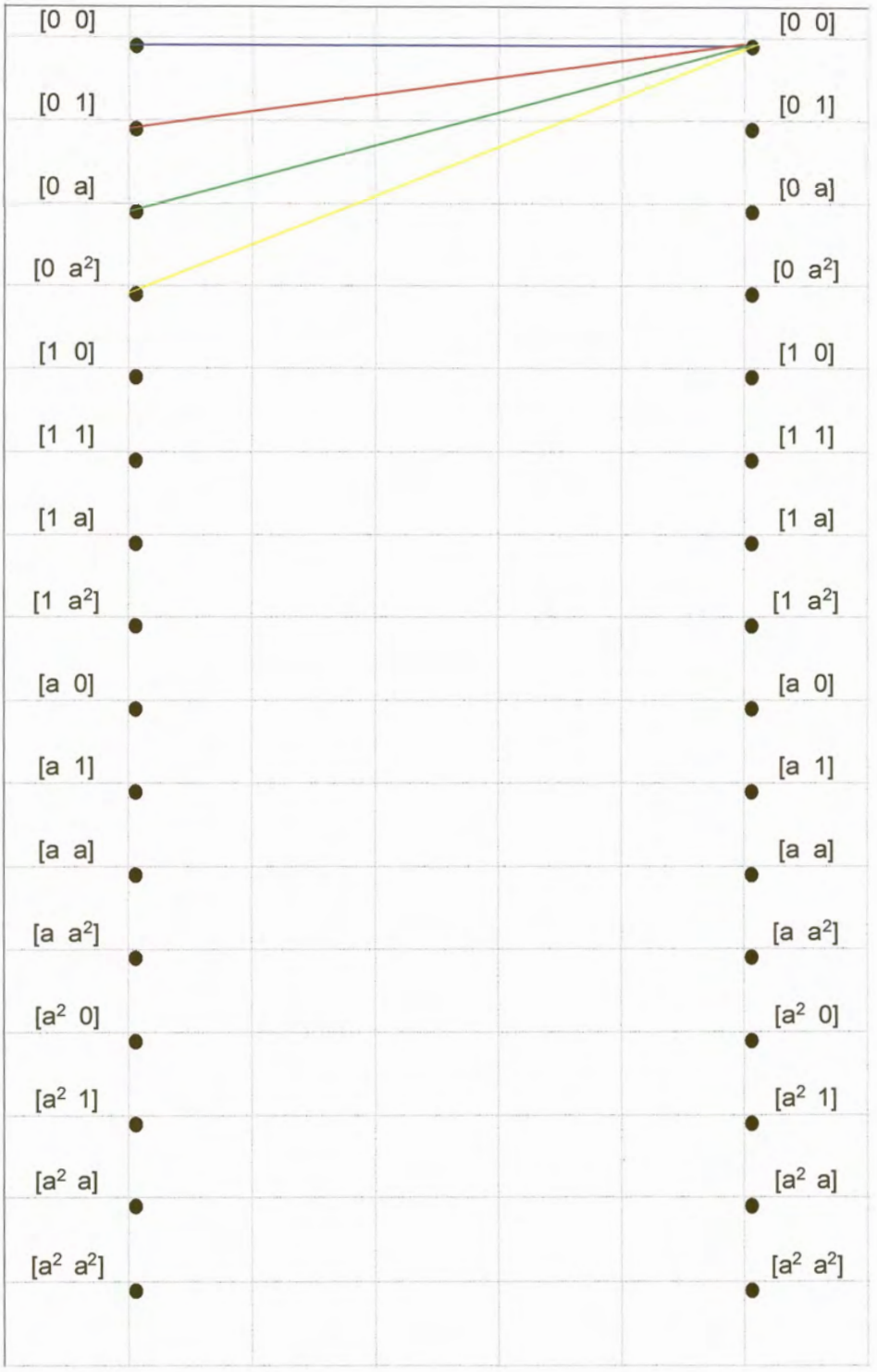

Figure 37 Section 7 of (7,5) Reed-Solomon Trellis in GF(4).

5.6 A Topological Trellis Construction Scheme for Reed-Solomon Codes

As mentioned before, one of the largest restrictions on the use of a Reed-Solomon trellis, is the large complexity involved. The small code described above, viz the (7,5) RS-code, already had a fairly huge trellis structure. This of course makes the decoding process complex and slow to perform.

The first task in the decoding process is the construction of the trellis diagram which is used by the decoder. The realtime construction of the trellis can be omitted if the trellis is hard coded in the decoder. This however imposes a serious limitation on the versatility of the encoder/decoder, as a new decoder/encoder will have to be designed for every new application. It would be very advantageous if an elegant trellis construction technique could be developed.

As the base of the Galois field increases, the number of possible symbols in the code increases. This in turn has a negative impact on the complexity of the system. The number of branches per node in the trellis expands exponentially. A method is required which would limit the required number of calculations involved in obtaining a trellis diagram. Such a method would lead to large savings in computational complexity.

In this discussion, the code to be considered is the one used in **Subsection 6.2.3**, namely the (7,5) RS-code. This code has a parity check matrix in which the number of values in a column is 2. The whole procedure can easily be expanded in order to accommodate larger parity matrixes and codes.

This is where the previously mentioned partitioning of the states into 4 sub-parts is applied. As the state numbering follows a binary sequence, it can be seen that each state is numbered by a unique ordering of the 4 symbols in the field. Due to this, each set of four sub-parts has the same unique symbol in the first position of this state numbering scheme. The second symbol is then just one of the four remaining symbols. In other words, for each unique symbol in the first position of a state which identifies the sub-part of the states, any of the four other symbols are employed to identify a state within the sub-parts. This might seem very insignificant, but will prove invaluable later on.

As can be seen from the trellis diagrams, the first node of each sub-part is coded in a specific colour, which it retains in all other states as well. Due to the fact that 4 nodes or states exist within each sub-part, 4 colours are used. These colours are given in the order blue, red, green and yellow. Again, this seems very unimportant, but together with the first observation, it will provide a very practical method by which the complexity of trellis design can be simplified and confined.

The next important observation is the parity check matrix. Every column of the parity check matrix determines the connecting branches for a specific section of the trellis, viz the first column specifies the first section of the trellis, the second column the second section, and so on.

The values inside the parity check matrix depends on the Galois field used. In other words, if GF(2) is used, the only possible values in the matrix are the binary values 0 and 1. If however GF(4) is used, then the possible values the parity check matrix can assume are the 4 symbols in this field, namely $0,1,\alpha$ and α^2. At the moment all that needs to be remembered, is that there is a different construction associated with the element 0, than with the non zero elements 1, α and α^2.

The examples show that the calculations made to obtain the branches of the trellis seem to follow a cyclic pattern. This can be explained as follows. A lookup table for each symbol can be constructed, as was done before. This lookup table gives the result of adding any possible valid symbol to the symbol of the lookup table. When the first calculation for a node is made, a specific position in the lookup table will provide the answer. If this answer is found, no more calculations need to be made, since the remaining branch connections can be read off from the subsequent positions of the lookup table i.e. the specification of trellis branches follow a very specific topological pattern. The table has to be wrapped when the bottom of the table is reached before all branches have been specified.

The tricky task now is to formalize all these findings, in order to provide a systematic algorithm for trellis construction with low complexity.

The foregoing findings may be summarized point-wise as follows:

- The trellis state numbering is divided into several sub-parts, in which the first

element of the numbering stays the same symbol in the sub-part.

- Colour coding a specific node is important. The colour of a node or state has to remain the same in each of the sub-parts.
- The parity check matrix's columns can be divided into those that have a zero element in the first position and those that do not.
- The symbol in the second position of the parity check matrix column, is important as far as the selection of subsequent points is concerned.
- The results of the calculations are cyclic. The answers can thus be read off from a lookup table as soon as the starting point in the table has been established.

The discussion will commence by describing and explaining the last few steps in the above list based on the (7,5) RS-example. It is said that the results and calculations are cyclic. This can be illustrated by using the first node, and the first column of the parity check matrix to calculate the 4 branches that emanate from this active starting node. The node is labeled [0 0]. In order to find the 4 nodes to which it is connected, the first column of the parity check matrix is multiplied by the symbols and added to the starting node [0 0]. In this way four new nodes in the second section of the trellis are obtained namely [0 0], [0 1], [0 a] and [0 a^2]. Ignoring the first element of the node, the second one changes cyclically. The first calculation can be solved by using the first row of the lookup table of **Equation 5.35**. The answer of the lookup is 0. In order to find the next three nodes, all that has to be done is to continue reading off the rows of the lookup table in **Equation 5.35**. If the bottom of the table is reached prematurely, i.e. before the desired amount of nodes have been numbered, then the table entries are wrapped around. The cyclic procedure therefore follows a mod N cyclic pattern where N denotes the number of branches emanating from a specific node in the code trellis. This wrap-around does not occur since it only occurs when the first element is read off a row other than the first one. The sequence of branch labels obtained when reading the elements off the lookup table in order are 0, 1, a and a^2. Comparing this with the results obtained by calculation, it can be seen that they correspond precisely.

This is the first step in constructing a trellis with low complexity.

The next step is to describe how the values in the columns of the parity check matrix affect the structure of the trellis diagram. Recall, that the columns of the parity check matrix can be divided into two large groups. The first group includes those columns that have the zero element in the first position and the second group those columns that do

not have a zero element in the first position. At this point, the state division into 4 sub-parts becomes important. Two possibilities now exist. If the parity check matrix column has a zero in the first position, then a branch emanating from a node in a specific sub-part will terminate in a node in the same sub-part. If however a column of the parity check matrix is non-zero, then a branch emanating from a node in a specific sub-part, will have a termination in a node from each sub-part of the trellis diagram. As an example of the first case, consider **Figure 26**. It can be seen that the branches emanating from the [0 0] node all end in the same sub-part. This holds true for all instances. The second case is illustrated in **Figure 27**. Here it can be seen that the 4 branches emanating from node [0 0] terminate in each of the 4 sub-parts. This is due to the cyclic phenomenon described in the previous paragraph. Here it is just applicable to the second position of the column of the parity check matrix. It is important to note that the sub-part in which a branch terminates is also determined by the same cyclic principle described in the above paragraph on cycling. If the first branch enters sub-part 3, then the second branch will enter sub-part 4, the third branch sub-part 1 and the last branch sub-part 2. This becomes extremely important when numbering the branches with their corresponding symbols from the Galois field set.

After discussing all of the important issues, a description of the colour coding part is now in order. The colour coding displays the culmination of all the above mentioned steps. **Figure 28** is used as a starting point. This sub-trellis was formed using the parity check column [1 1]. As discussed before, the fact that the first element is a 1, forces the branches leaving a node in a specific sub-part to enter nodes in different sub-parts. This can be seen by just observing the node [0 0]. The branches enter node [0 0] in sub-part 1, node [1 0] in sub-part 2, [α 0] in sub-part 3 and [α^2 0] in sub-part 4. These 4 branches are all marked with blue. Now the interesting part reveals itself. The first node of the second sub-part is also coloured in blue, and it can be seen that they enter the nodes activated by blue branches leaving node [0 0]. This phenomenon occurs throughout the figure. In order to draw the trellis, only the branches of the first sub-part have to be calculated. Hereafter, due to the cyclic phenomenon, all that is required to uniquely identify and label subsequent branches in this section of the trellis.

There is just one other form of topology, which occurs if the first element of the column of the parity check matrix is zero. In this case, as illustrated by **Figure 26**, all the branches emanating from a node in a specific sub-part activate all other nodes in the same sub-part. All that needs to be calculated for this section of the trellis is the first

section. Hereafter all remaining trellis branches may be labeled according to the cyclic pattern described.

In fact, it should be noticed that all that is really necessary to calculate are the branches of the first node in the trellis. This, together with the two types of topologies and the cycling phenomenon determines the trellis uniquely and completely.

For the above trellis, it would normally be necessary to do 404 complex calculations in GF(4). If the cyclic topology is exploited, the whole trellis can be calculated with only 28 complex calculations in GF(4) a saving in the order of a factor of 14.

This method provides an enormous step forward in the whole trellis design procedure for Reed-Solomon codes. Note that the proposed algorithm does not provide a minimal (i.e. minimum number of states and branches) trellis, but only simplifies the trellis construction. A procedure to find a minimal trellis is given in the following chapter.

On the next page, a flow diagram of the topological trellis design procedure is presented.

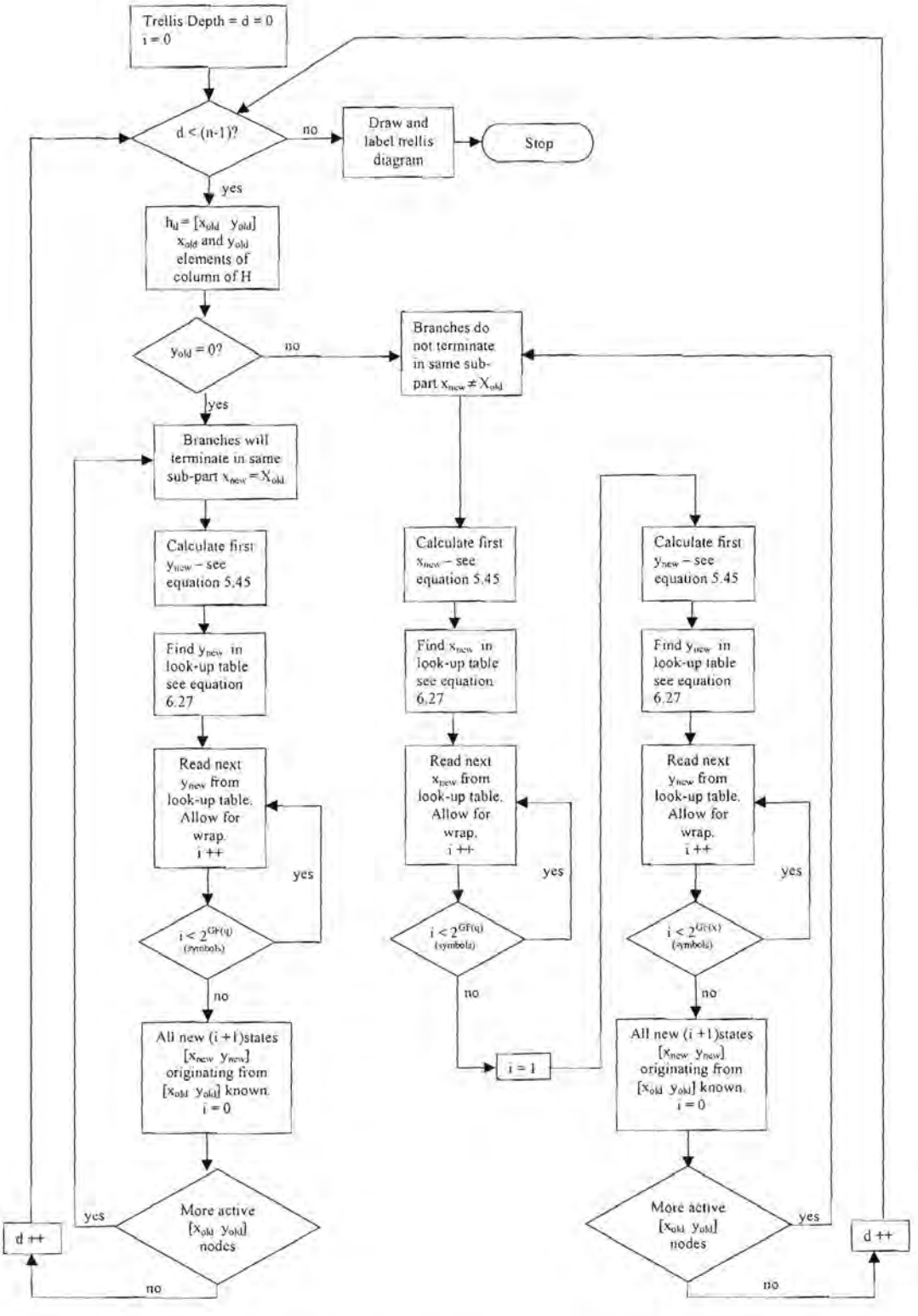

Figure 38 Flow Diagram of the Topological Trellis Design Algorithm

5.7 Symbol Error Rate of Reed-Solomon Code

In this section, the performance of Reed-Solomon codes is investigated. The main task at hand is to show that Reed-Solomon codes can be decoded with the Viterbi Algorithm. The decoding performance is shown to be the same as for algebraic techniques, since the Viterbi algorithm is a maximum likelihood technique.

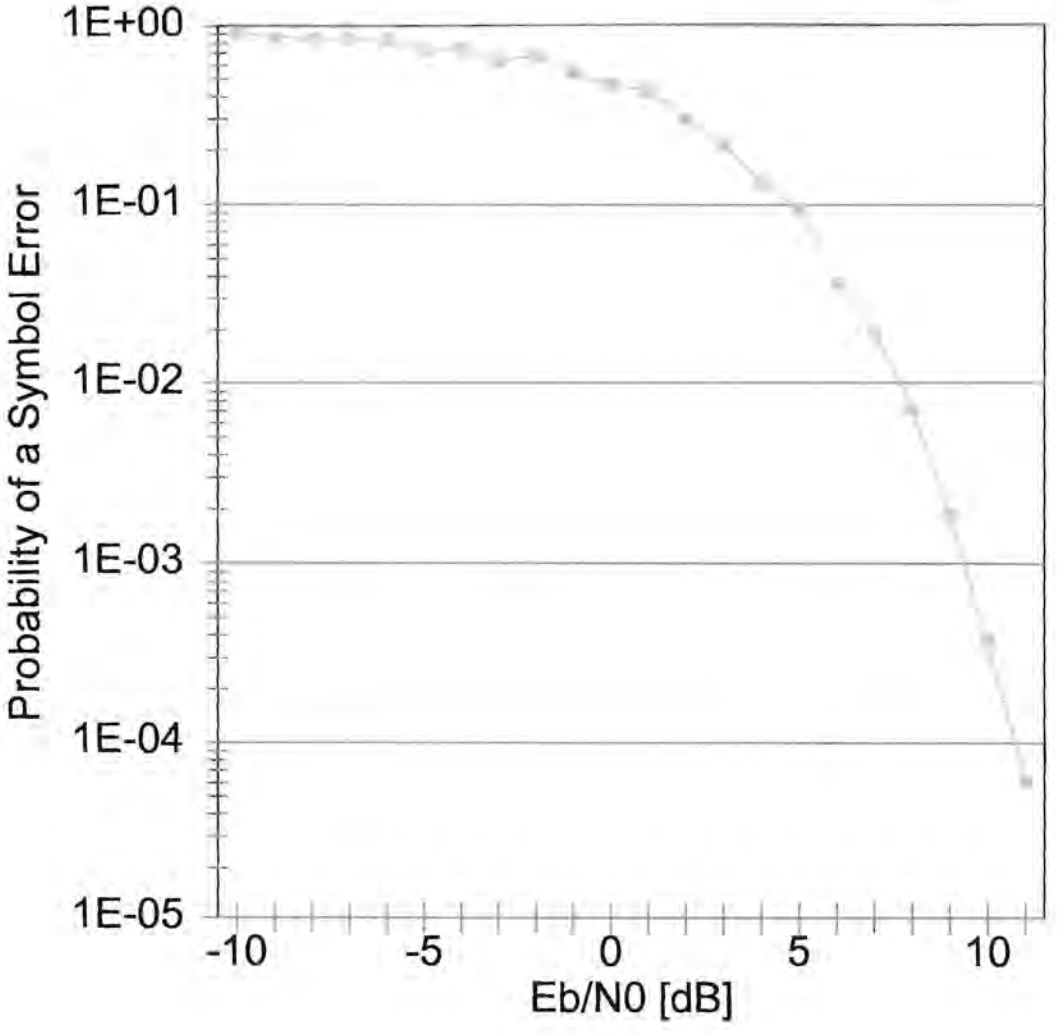

Figure 39 Simulated Symbol Error Rate of (7,5)-Reed-Solomon Code

As mentioned and shown previously, the trellis diagrams for Reed-Solomon codes are very large. This is the main problem with Reed-Solomon codes. As it was possible to construct a trellis for a Reed-Solomon code in the previous chapter, it is now possible to Viterbi decode the Reed-Solomon code. This is illustrated on a relatively small code, as it takes quite a long time to construct a symbol error rate curve. The same principles of course apply to larger codes also.

Again consider the(7,5)-Reed-Solomon code from the previous chapter. The simulation setup employs a Reed-Solomon encoder, the output of which is transmitted over an AWGN channel. In the decoder, a SDML Viterbi decoder is employed. The simulation setup is the same as depicted in **Chapter 2**.

On the previous page, the symbol error rate for the Reed-Solomon code is shown.

5.8 General Discussion on Reed-Solomon Performance

Reed-Solomon codes were traditionally decoded in the frequency domain using the Berlekamp Massey algorithm [20]. This method uses Fourier transforms to do the decoding. Due to the Fourier transform, the decoding is only an approximation, and marginally better decoding results can be obtained employing a maximum likelihood decoding technique such as the Viterbi algorithm. Although the scope of this dissertation was not to show that Viterbi decoding is better than Berlekamp Massey decoding of Reed-Solomon codes, an attempt was made to use existing standard software to find symbol error rate curves for a Berlekamp Massey algorithm. This was possible, but these software packages do not provide for such small codes as the (7,5)-Reed-Solomon code used here. Reed-Solomon codes usually have typical specifications such as $n = 255$ and $k = 251$ or the likes. This meant that a Berlekamp Massey algorithm would have to be re-written in order to do this comparison. The other problem was that the trellis complexity sky-rockets with high values of n and k and that a normal personal computer would not be able to calculate the required symbol error rates for the large block lengths. This does not even take into account the Galois fields which are commonly used in VHDL Reed-Solomon coders. These often run in fields as high as $GF(128)$ or $GF(256)$ compared to the $GF(4)$ field used in the Viterbi simulation presented here.

The comparison which seemed possible at first proved not to be possible with the current state of computer hardware available. The solution to this problem would be the VHDL implementation of such a Viterbi Reed-Solomon decoder. This could then be compared with the normal Berlekamp Massey decoders using a bit error rate analyzer, but this falls beyond the scope of this dissertation.

What has been shown is that it is indeed possible to decode Reed-Solomon codes with decoding techniques usually reserved for convolutional code decoding. This opens many possibilities for integrating coding schemes employing both powerful Reed-Solomon codes and convolutional codes. It is possible to use the same decoder in the decoding process by deriving the resultant trellis of the concatenation of all codes employed.