References

The following references were used in the process of creating this dissertation. The work of the different authors is acknowledged in this section.

[1] M Honary, *Trellis Decoding of Block Codes*, Kluwer Academic Publishers 1998.

[2] LR Bahl, JCocke, F Jelinek and J Raviv,. "Optimal Decoding of Linear Codes for Minimizing Symbol Error Rate", *IEEE Transactions on Information Theory*, Vol. 20 pp 284-287.

[3] RE Blahut, *Theory and Practice of Error Control Codes*, Addison Wesley 1984.

[4] S Bate, B Honary and PG Farell, "Error Control Techniques Applicable to HF channels", *IEEE Proceedings*, Vol. 136, No. 1, February 1989, pp. 57-63.

[5] GD Forney, "Convolutional Codes I: Algebraic Structure", *IEEE Transactions on Information Theory*, Vol. 16, 1970, pp. 720-738.

[6] R Johannesson and Z Wan, "A Linear Algebra Approach to Minimal Convolutional Encoders", *IEEE Transactions of Information Theory*, Vol. 39, 1993, pp. 1219-1233.

[7] T Kasami and T Fujiwara, "On Complexity of Trellis Structure of Linear Block Codes", *IEEE Transactions on Information Theory*, Vol. 39, 1993, pp 1057-1064.

[8] FJ MacWilliams and NJ Sloane, "The Theory of Error Correcting Codes", Amsterdam: North-Holland, 1983.

[9] DJ Muder, "Minimal Trellises for Block Codes", *IEEE Transactions on Information Theory*, Vol. 34, 1988, pp. 1049-1053.

[10] JG Proakis, *Digital Communications*, New York, McGraw and Hill, 1989.

[11] VR Sidorenko and VV Zyablov, "Bounds on Complexity of Trellis Decoding of Linear Codes", *Proceedings of the Sixth Joint Swedish-Russian International Workshop on Information Theory*, 1993, pp. 164-168.

[12] VR Sidorenkoand VV Zyablov, "Decoding of Convolutional Codes using Syndrome Trellsis", *Proceedings of the Sixth Joint Swedish-Russian International Workshop on Information Theory*, 1993, pp. 46-50.

[13] B Sklar, *Digital Communications*, Englewood Cliffs, New York, Prentice Hall, 1988.

[14] AJ Viterbi, "Error Bounds for Convolutional Codes and an Asymtotically Optimum Decoding Algorithm", *IEEE Transactions on Information Theory*, Vol. 13, 1967, pp. 260-269.

[15] AJ Viterbi and JK Omura, "Principles of Digital Communication and Coding", *New York, McGraw and Hill*, 1979.

[16] JK Wolf, "Efficient Maximum Likelihood Decoding of Linear Block Codes using a Trellis", *IEEE Transactions on Information Theory*, Vol. 24, 1978, pp. 76-80.

[17] R Rotman, *Galois Theory*, Berlin, Springer Verlag, 1990.

[18] S Lin, T Kasami and T Fujiwara "Trellises and Trellis based Decoding Algorithms for Linear Block Codes", *Kluwer Academic Publishers*, 1998.

[19] S Lin and DJ Costello, *Error Control Coding: Fundamentals and their Applications*, Prentice Hall, 1983.

[20] IS Reed and G Solomon, "Polynomial Codes over Certain Finite Fields", *SIAM Journal of Applied Mathematics*, Vol. 8, 1960, pp. 300-304

[21] SB Wicker and VK Bhargava, *Reed-Solomon Codes and their Applications*, IEEE Press, New York, 1994.

[22] JK Wolf, "On Codes derivable from the Tensor Product of Check Matrices", *IEEE*

Transactions on Information Theory, Vol. 11, April 1965, pp. 281-284, April 1965.

[23] CE Shannon, "The Zero Capacity of Noisy Channels", *IRE Transactions on Information Theory,* vol. 2, September 1956, pp. 8-19.

[24] E Biglieri, D Divsalar, PJ McLane and MK Simon, *Introduction to Trellis-Coded Modulation with Applications*, MacMillan Pull. Co, New York, 1991.

[25] RJ McEliece, *Finite Fields for Computer Scientists and Engineers,* Boston: Kluwer Academic, 1987.

[26] GD Forney and AR Calderbank "Coset Codes for Partial Response Channels; or, Coset Codes with Spectral Nulls", *IEEE Transactions on Information Theory,* vol. 35, 1989, pp. 925-943.

[27] GD Forney, "Coset Codes- Part1: Introduction and Geometrical Classification", *IEEE Transactions on Information Theory,* vol. 34, 1988, No5, pp. 1123-1151.

[28] GD Forney, "Coset Codes- Part 2: Binary Lattices and Related Codes", *IEEE Transactions on Information Theory,* vol. 34, No. 5, 1988, pp. 1152-1187.

[29] GD Forney and MD Trott, "The Dynamics of Group Codes: State Spaces, Trellis Diagram and Canonical Encoders", *IEEE Transactions on Information Theory,* vol. 39, 1993, pp. 1491-1513.

[30] GD Forney, "Dimension/Length Profiles and Trellis Complexity of Linear Block Codes", *IEEE Transactions on Information Theory,* vol. 40, 1994, pp. 1741-1752.

[31] GD Forney, "Density/Length Profiles and Trellis Complexity of Lattices", *IEEE Transactions on Information Theory,* vol. 40, 1994, pp. 1753-1772.

[32] E Berlekamp, *Algebraic Coding Theory,* Agean Park Press, Laguna Hills, Revised edition, 1984.

[33] RJ McEliece, "The Viterbi Decoding Complexity of Linear Block Codes" *Proceedings of IEEE International Symposium on Information Theory,*

Trondheim, Norway, 1994, p. 341

[34] S Sonander, *Generalized Array Code Construction Algorithm for Reed-Solomon Codes,* Msc Thesis, Lancaster University, 1994.

[35] O Ytrehus, "On the Trellis Complexity of Certain Binary Block Codes", *IEEE Transactions on Information Theory,* vol. 41, 1995, No. 2, pp. 559-560.

Appendix A

Another Syndrome Trellis Construction Technique

A.1 Introduction

In **Section 3.3** of **Chapter 3** a method of constructing a syndrome trellis diagram as proposed by Wolf was described. In this Appendix, another method of constructing the syndrome trellis diagram will be investigated. The method is similar to the one described by Wolf, but in this method, the syndromes of the parity check matrix are directly used to construct the trellis diagram.

In order to ensure complete understanding of this method, an example to illustrate the various steps is provided.

A.2 Generation of the Code Book

Consider the following binary linear block code (5,3,2) with the following parity check matrix:

$$\boldsymbol{H} = \left(\begin{array}{ccc|cc} 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \end{array}\right) = \left(\begin{array}{c|c} -\boldsymbol{P}' & \boldsymbol{I}_{n-k} \end{array}\right) \tag{A.1}$$

with $n = 5$, $k = 3$ and $(n - k) = 2$.

By using the above mentioned values for n, k and $(n - k)$ together with definitions from **Chapter 2**, the following parameters can be calculated. Firstly, the number of code words is,

$$2^k = 2^3 = 8 = M \qquad \textbf{(A.2)}$$

and the number of states are:

$$2^{n-k} = 2^{5-3} = 2^2 = 4 \qquad \textbf{(A.3)}$$

As shown in **Appendix B**, it is possible to obtain the generator matrix **G** from the parity check matrix **H** of a given block code *C*. The reverse is of course also possible, provided that the parity check matrix is in the systematic form, or transformed into the systematic form first.

For further information on this transformation, **Appendix 3** should be consulted.

As mentioned above, the reverse process (generator matrix **G** from parity check matrix **H**) would look as follows. Note that the parity check matrix was already in systematic form which means that there is no need to first transform the parity check matrix into systematic form before the generator matrix is calculated.

$$\boldsymbol{G} = \left(\boldsymbol{I}_k \quad \middle| \quad \boldsymbol{P} \right) = \left(\begin{array}{ccc|cc} 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \end{array} \right) \qquad \textbf{(A.4)}$$

The code book can now be constructed by calculating the result of taking all the possible linear combinations of the generator matrix.

In other words, each row is Modula-2 added to every other row. The resulting output vector is one of the codewords in the code book. Noting that it is a linear code, it is also necessary to consider the all zero state. The following is a list of all the codewords and their associated code bits.

$$
\begin{array}{llllllllllll}
C_1 = 1 & 0 & 0 & 1 & 1 & = c_{11} & c_{12} & c_{13} & c_{14} & c_{15} & \\
C_2 = 0 & 1 & 0 & 1 & 0 & = c_{21} & c_{22} & c_{23} & c_{24} & c_{25} & \\
C_3 = 0 & 0 & 1 & 0 & 1 & = c_{31} & c_{32} & c_{33} & c_{34} & c_{35} & \\
C_4 = 1 & 1 & 0 & 0 & 1 & = c_{41} & c_{42} & c_{43} & c_{44} & c_{45} & [1\oplus 2] \\
C_5 = 1 & 0 & 1 & 1 & 0 & = c_{51} & c_{52} & c_{53} & c_{54} & c_{55} & [1\oplus 3] \\
C_6 = 0 & 1 & 1 & 1 & 1 & = c_{61} & c_{62} & c_{63} & c_{64} & c_{65} & [2\oplus 3] \\
C_7 = 1 & 1 & 1 & 0 & 0 & = c_{71} & c_{72} & c_{73} & c_{74} & c_{75} & [1\oplus 2\oplus 3] \\
C_8 = 0 & 0 & 0 & 0 & 0 & = c_{81} & c_{82} & c_{83} & c_{84} & c_{85} & [All\, 0]
\end{array}
\qquad \textbf{(A.5)}
$$

The above list contains all the possible codewords. We are now ready to proceed to the next step.

A.3 Calculation of the Syndromes

The next step in the process is to calculate the syndromes for each of the code words.

In order to evaluate the syndromes, some definitions would be in order here.

Syndrome S_{1n} is defined as follows:

$$S_{1n} = \breve{C}_{1n} \cdot \boldsymbol{H}^T \qquad \textbf{(A.6)}$$

The transformed codeword is defined below:

$$
\begin{aligned}
\breve{C}_{11} &= \begin{pmatrix} c_{11} & 0 & 0 & 0 & 0 \end{pmatrix} \\
\breve{C}_{12} &= \begin{pmatrix} c_{11} & c_{12} & 0 & 0 & 0 \end{pmatrix} \\
&\ \ \vdots \\
\breve{C}_{1n} &= \begin{pmatrix} c_{11} & c_{12} & \dots & \dots & c_{1n} \end{pmatrix}
\end{aligned}
\qquad \textbf{(A.7)}
$$

- Consider codeword $C_1 = \begin{pmatrix} c_{11} & c_{12} & c_{13} & c_{14} & c_{15} \end{pmatrix} = \begin{bmatrix} 1 & 0 & 0 & 1 & 1 \end{bmatrix}$

  Start with the first transformed codeword. This is used to calculate the first syndrome.

$$S_{11} = \breve{C}_{11} \cdot \boldsymbol{H}^T = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} 1 & 1 \\ 1 & 0 \\ 0 & 1 \\ 1 & 0 \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 1 \end{pmatrix} = \xi_3 \qquad \textbf{(A.8)}$$

  Using the other transformed codewords, the other syndromes are calculated.

$$S_{12} = \breve{C}_{12} \cdot \boldsymbol{H}^T = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} 1 & 1 \\ 1 & 0 \\ 0 & 1 \\ 1 & 0 \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 1 \end{pmatrix} = \xi_3 \qquad \textbf{(A.9)}$$

$$S_{13} = \breve{C}_{13} \cdot \boldsymbol{H}^T = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} 1 & 1 \\ 1 & 0 \\ 0 & 1 \\ 1 & 0 \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 1 \end{pmatrix} = \xi_3 \qquad \textbf{(A.10)}$$

$$S_{14} = \breve{C}_{14} \cdot \boldsymbol{H}^T = \begin{pmatrix} 1 & 0 & 0 & 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} 1 & 1 \\ 1 & 0 \\ 0 & 1 \\ 1 & 0 \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} 0 & 1 \end{pmatrix} = \xi_1 \qquad \textbf{(A.11)}$$

$$S_{15} = \breve{C}_{15} \cdot \boldsymbol{H}^T = \begin{pmatrix} 1 & 0 & 0 & 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 1 \\ 1 & 0 \\ 0 & 1 \\ 1 & 0 \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} 0 & 0 \end{pmatrix} = \xi_0 \qquad \textbf{(A.12)}$$

- Consider codeword $C_2 = \begin{pmatrix} c_{21} & c_{22} & c_{23} & c_{24} & c_{25} \end{pmatrix} = \begin{bmatrix} 0 & 1 & 0 & 1 & 0 \end{bmatrix}$

  Due to space limitations, only the calculated syndromes for the rest of the codes will be displayed.

$$S_{21} = \breve{C}_{21} \cdot \boldsymbol{H}^T = \begin{pmatrix} 0 & 0 \end{pmatrix} = \xi_0 \qquad \textbf{(A.13)}$$

$$S_{22} = \breve{C}_{22} \cdot \boldsymbol{H}^T = \begin{pmatrix} 1 & 0 \end{pmatrix} = \xi_2 \qquad \textbf{(A.14)}$$

$$S_{23} = \breve{C}_{23} \cdot \boldsymbol{H}^T = \begin{pmatrix} 1 & 0 \end{pmatrix} = \xi_2 \qquad \textbf{(A.15)}$$

$$S_{24} = \breve{C}_{24} \cdot \boldsymbol{H}^T = \begin{pmatrix} 0 & 0 \end{pmatrix} = \xi_0 \qquad \textbf{(A.16)}$$

$$S_{25} = \breve{C}_{25} \cdot \boldsymbol{H}^T = \begin{pmatrix} 0 & 0 \end{pmatrix} = \xi_0 \qquad \textbf{(A.17)}$$

- Consider codeword $C_3 = \begin{pmatrix} c_{31} & c_{32} & c_{33} & c_{34} & c_{35} \end{pmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 & 1 \end{bmatrix}$

  The syndromes for this codeword are given below.

$$S_{31} = \breve{C}_{31} \cdot \boldsymbol{H}^T = \begin{pmatrix} 0 & 0 \end{pmatrix} = \xi_0 \qquad \textbf{(A.18)}$$

$$S_{32} = \breve{C}_{32} \cdot \boldsymbol{H}^T = \begin{pmatrix} 0 & 0 \end{pmatrix} = \xi_0 \qquad \textbf{(A.19)}$$

$$S_{33} = \breve{C}_{33} \cdot \boldsymbol{H}^T = \begin{pmatrix} 0 & 1 \end{pmatrix} = \xi_1 \qquad \textbf{(A.20)}$$

$$S_{34} = \breve{C}_{34} \cdot \boldsymbol{H}^T = \begin{pmatrix} 0 & 1 \end{pmatrix} = \xi_1 \qquad \textbf{(A.21)}$$

$$S_{35} = \breve{C}_{35} \cdot \boldsymbol{H}^T = \begin{pmatrix} 0 & 0 \end{pmatrix} = \xi_0 \qquad \textbf{(A.22)}$$

- Consider codeword $C_4 = \begin{pmatrix} c_{41} & c_{42} & c_{43} & c_{44} & c_{45} \end{pmatrix} = \begin{bmatrix} 1 & 1 & 0 & 0 & 1 \end{bmatrix}$

  The syndromes are given below.

$$S_{41} = \breve{C}_{41} \cdot \boldsymbol{H}^T = \begin{pmatrix} 1 & 1 \end{pmatrix} = \xi_3 \qquad \textbf{(A.23)}$$

$$S_{42} = \breve{C}_{42} \cdot \boldsymbol{H}^T = \begin{pmatrix} 0 & 1 \end{pmatrix} = \xi_1 \qquad \textbf{(A.24)}$$

$$S_{43} = \breve{C}_{43} \cdot \boldsymbol{H}^T = \begin{pmatrix} 0 & 1 \end{pmatrix} = \xi_1 \qquad \textbf{(A.25)}$$

$$S_{44} = \breve{C}_{44} \cdot \boldsymbol{H}^T = \begin{pmatrix} 0 & 1 \end{pmatrix} = \xi_1 \qquad \textbf{(A.26)}$$

$$S_{45} = \breve{C}_{45} \cdot \boldsymbol{H}^T = \begin{pmatrix} 0 & 0 \end{pmatrix} = \xi_0 \qquad \textbf{(A.27)}$$

- Consider codeword $C_5 = \begin{pmatrix} c_{51} & c_{52} & c_{53} & c_{54} & c_{55} \end{pmatrix} = \begin{bmatrix} 1 & 0 & 1 & 1 & 0 \end{bmatrix}$

  The appropriate syndromes for this code are listed below.

$$S_{51} = \breve{C}_{51} \cdot \boldsymbol{H}^T = \begin{pmatrix} 1 & 1 \end{pmatrix} = \xi_3 \qquad \textbf{(A.28)}$$

$$S_{52} = \breve{C}_{52} \cdot \boldsymbol{H}^T = \begin{pmatrix} 1 & 1 \end{pmatrix} = \xi_3 \qquad \textbf{(A.29)}$$

$$S_{53} = \breve{C}_{53} \cdot \boldsymbol{H}^T = \begin{pmatrix} 1 & 0 \end{pmatrix} = \xi_2 \qquad \textbf{(A.30)}$$

$$S_{54} = \breve{C}_{54} \cdot \boldsymbol{H}^T = \begin{pmatrix} 0 & 0 \end{pmatrix} = \xi_0 \qquad \textbf{(A.31)}$$

$$S_{55} = \breve{C}_{55} \cdot \boldsymbol{H}^T = \begin{pmatrix} 0 & 0 \end{pmatrix} = \xi_0 \qquad \textbf{(A.32)}$$

- Consider codeword $C_6 = \begin{pmatrix} c_{61} & c_{62} & c_{63} & c_{64} & c_{65} \end{pmatrix} = \begin{bmatrix} 0 & 1 & 1 & 1 & 1 \end{bmatrix}$

  The syndromes for this codeword are given below.

$$S_{61} = \breve{C}_{61} \cdot \boldsymbol{H}^T = \begin{pmatrix} 0 & 0 \end{pmatrix} = \xi_0 \qquad \textbf{(A.33)}$$

$$S_{62} = \breve{C}_{62} \cdot \boldsymbol{H}^T = \begin{pmatrix} 1 & 0 \end{pmatrix} = \xi_2 \qquad \textbf{(A.34)}$$

$$S_{63} = \breve{C}_{63} \cdot \boldsymbol{H}^T = \begin{pmatrix}1 & 1\end{pmatrix} = \xi_3 \quad \textbf{(A.35)}$$

$$S_{64} = \breve{C}_{64} \cdot \boldsymbol{H}^T = \begin{pmatrix}0 & 1\end{pmatrix} = \xi_1 \quad \textbf{(A.36)}$$

$$S_{65} = \breve{C}_{65} \cdot \boldsymbol{H}^T = \begin{pmatrix}0 & 0\end{pmatrix} = \xi_0 \quad \textbf{(A.37)}$$

- Consider codeword $C_7 = \begin{pmatrix}c_{71} & c_{72} & c_{73} & c_{74} & c_{75}\end{pmatrix} = \begin{bmatrix}1 & 1 & 1 & 0 & 0\end{bmatrix}$

  The syndromes are given below.

$$S_{71} = \breve{C}_{71} \cdot \boldsymbol{H}^T = \begin{pmatrix}1 & 1\end{pmatrix} = \xi_3 \quad \textbf{(A.38)}$$

$$S_{72} = \breve{C}_{72} \cdot \boldsymbol{H}^T = \begin{pmatrix}0 & 1\end{pmatrix} = \xi_1 \quad \textbf{(A.39)}$$

$$S_{73} = \breve{C}_{73} \cdot \boldsymbol{H}^T = \begin{pmatrix}0 & 0\end{pmatrix} = \xi_0 \quad \textbf{(A.40)}$$

$$S_{74} = \breve{C}_{74} \cdot \boldsymbol{H}^T = \begin{pmatrix}0 & 0\end{pmatrix} = \xi_0 \quad \textbf{(A.41)}$$

$$S_{75} = \breve{C}_{75} \cdot \boldsymbol{H}^T = \begin{pmatrix}0 & 0\end{pmatrix} = \xi_0 \quad \textbf{(A.42)}$$

- Consider codeword $C_8 = \begin{pmatrix}c_{81} & c_{82} & c_{83} & c_{84} & c_{85}\end{pmatrix} = \begin{bmatrix}0 & 0 & 0 & 0 & 0\end{bmatrix}$

  The syndromes for this codeword are as below.

$$S_{81} = \breve{C}_{81} \cdot \boldsymbol{H}^T = \begin{pmatrix}0 & 0\end{pmatrix} = \xi_0 \quad \textbf{(A.43)}$$

$$S_{82} = \breve{C}_{82} \cdot \boldsymbol{H}^T = \begin{pmatrix}0 & 0\end{pmatrix} = \xi_0 \quad \textbf{(A.44)}$$

$$S_{83} = \breve{C}_{83} \cdot \boldsymbol{H}^{T} = \begin{pmatrix} 0 & 0 \end{pmatrix} = \xi_0 \qquad \textbf{(A.45)}$$

$$S_{84} = \breve{C}_{84} \cdot \boldsymbol{H}^{T} = \begin{pmatrix} 0 & 0 \end{pmatrix} = \xi_0 \qquad \textbf{(A.46)}$$

$$S_{85} = \breve{C}_{85} \cdot \boldsymbol{H}^{T} = \begin{pmatrix} 0 & 0 \end{pmatrix} = \xi_0 \qquad \textbf{(A.47)}$$

After having calculated the relevant syndromes for the codewords, one can now proceed to draw the syndrome trellis diagram.

A.4 Constructing the Syndrome Trellis Diagram

Each one of the previously calculated syndromes is associated with a node in the syndrome trellis diagram. The time element of the trellis diagram is enclosed in the number of the specific syndrome. The first syndrome of a code would be the first in time, and the last syndrome last in time (further down the trellis).

- Construct single code trellis for $C_1 = \begin{pmatrix}1 & 0 & 0 & 1 & 1\end{pmatrix}$

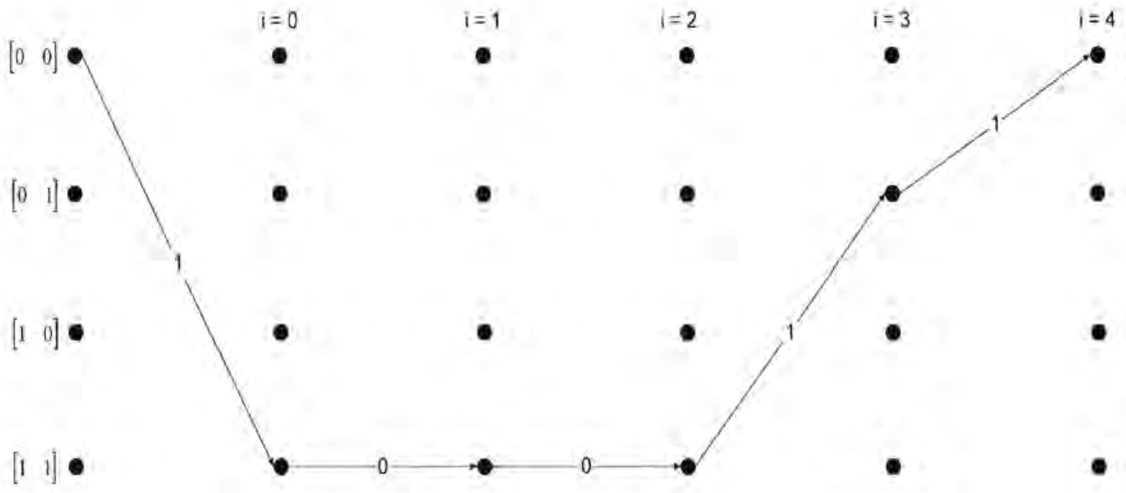

Figure 46 Single Code Trellis for Codeword 1

- Construct single code trellis for $C_2 = \begin{pmatrix}0 & 1 & 0 & 1 & 0\end{pmatrix}$

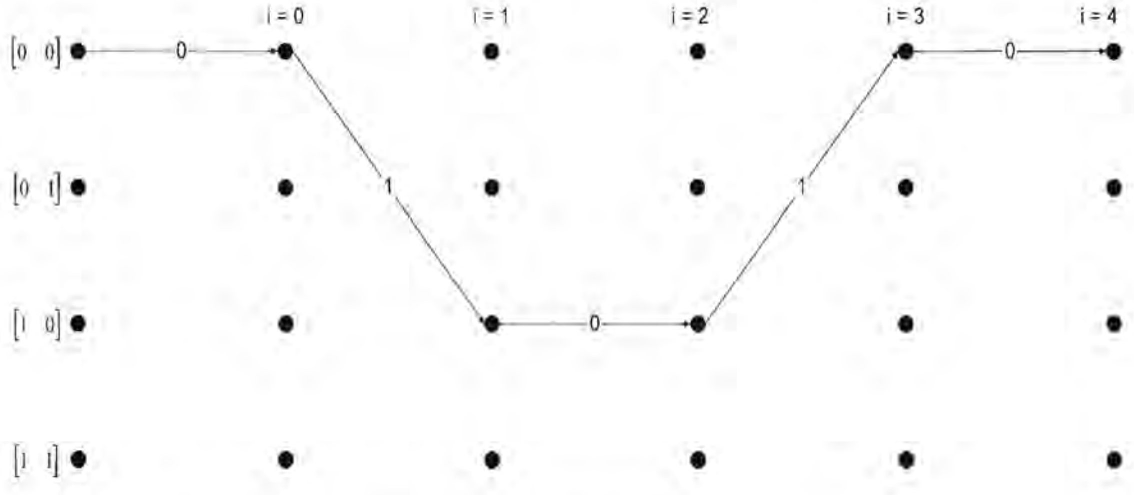

Figure 47 Single Code Trellis for Codeword 2

- Construct single code trellis for $C_3 = \begin{pmatrix} 0 & 0 & 1 & 0 & 1 \end{pmatrix}$

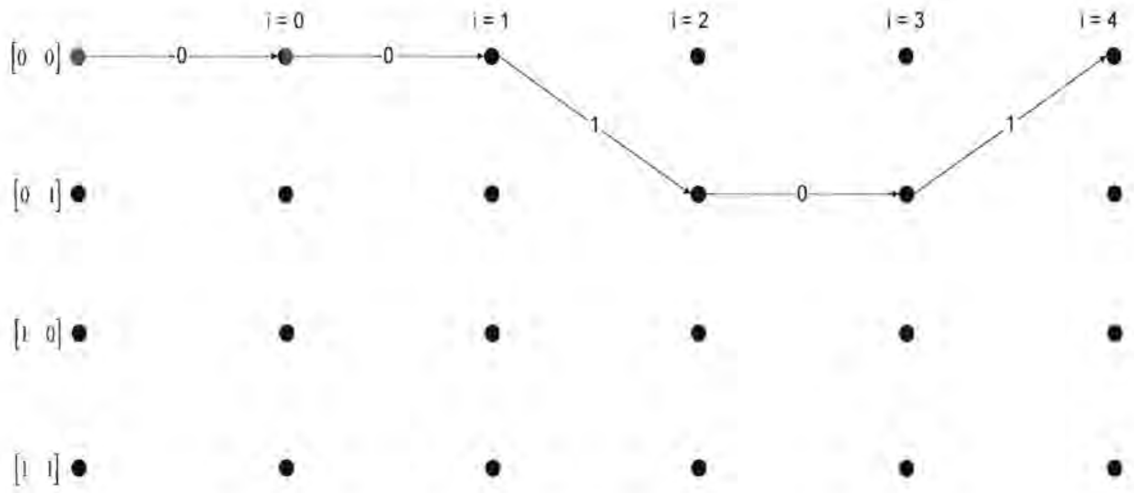

Figure 48 Single Code Trellis for Codeword 3

- Construct single code trellis for $C_4 = \begin{pmatrix} 1 & 1 & 0 & 0 & 1 \end{pmatrix}$

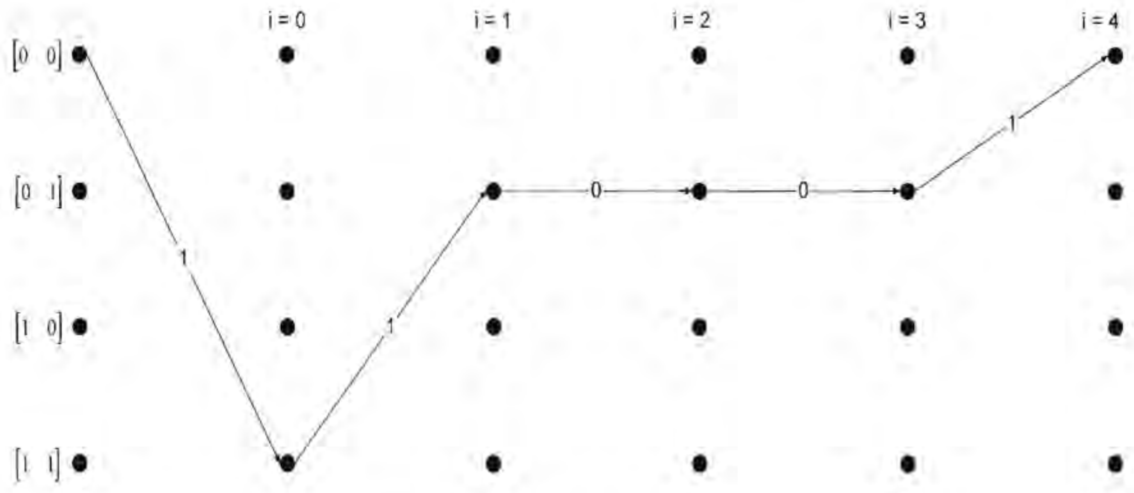

Figure 49 Single Code Trellis for Codeword 4

- Construct single code trellis for $C_5 = \begin{pmatrix}1 & 0 & 1 & 1 & 0\end{pmatrix}$

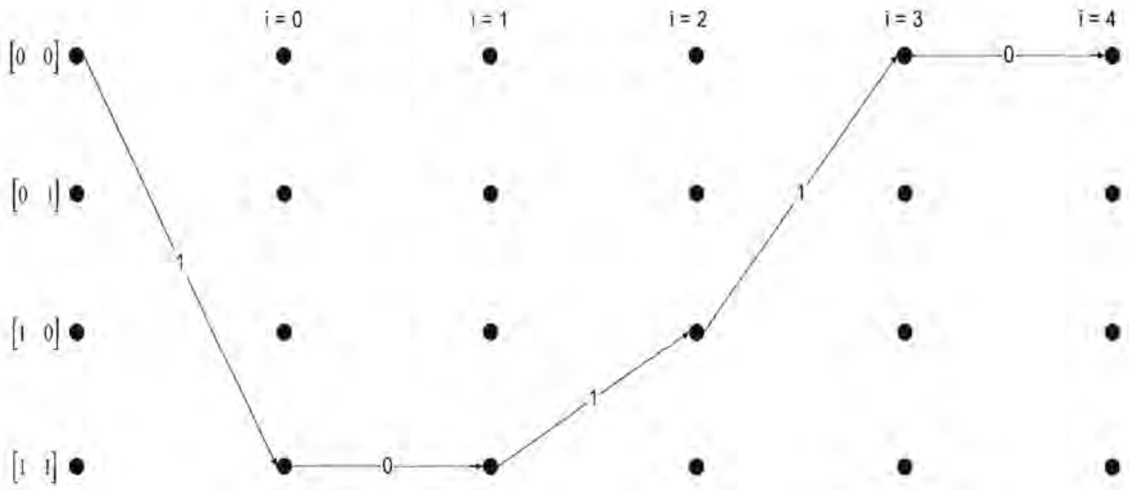

Figure 50 Single Code Trellis for Codeword 5

- Construct single code trellis for $C_6 = \begin{pmatrix}0 & 1 & 1 & 1 & 1\end{pmatrix}$

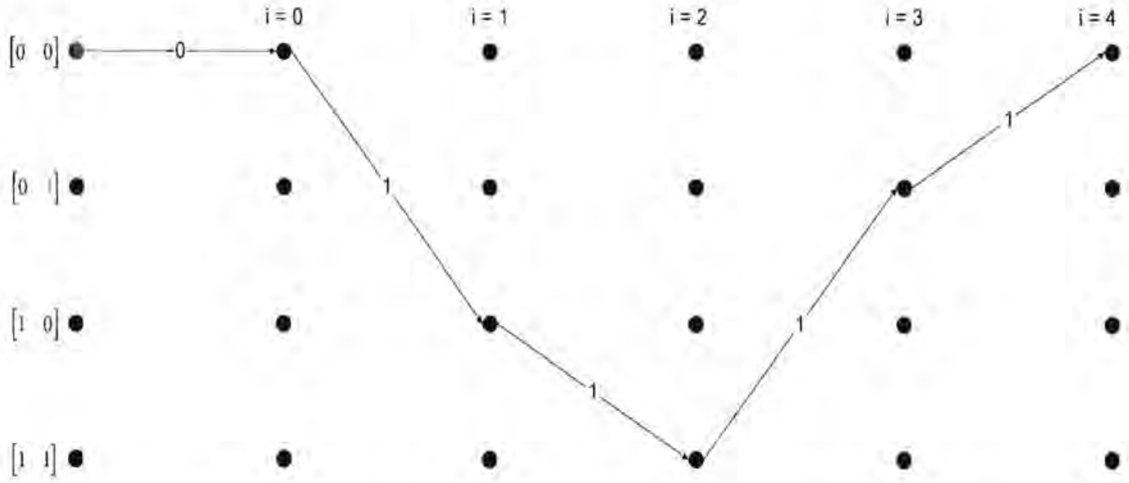

Figure 51 Single Code Trellis for Codeword 6

- Construct single code trellis for $C_7 = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 \end{pmatrix}$

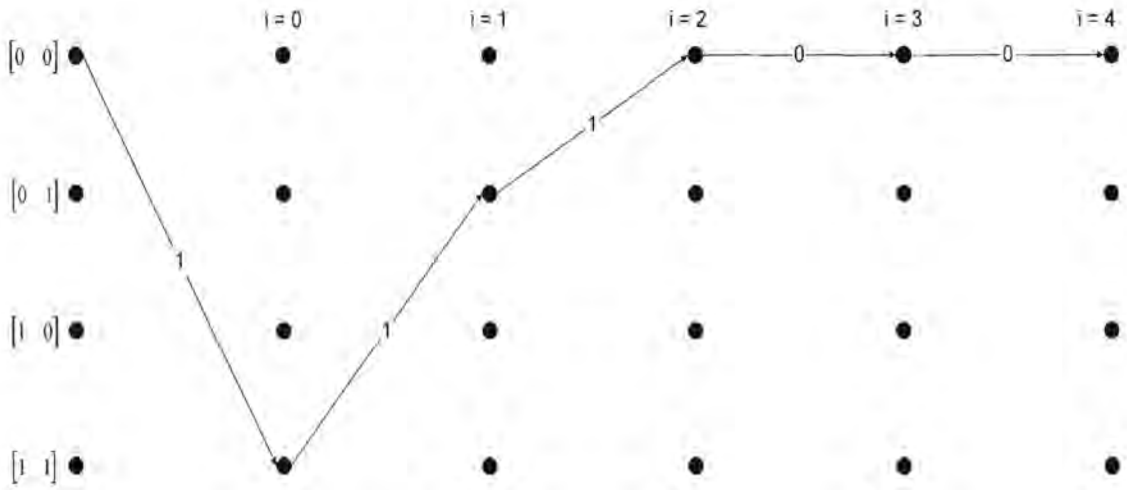

Figure 52 Single Code Trellis for Codeword 7

- Construct single code trellis for $C_8 = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \end{pmatrix}$

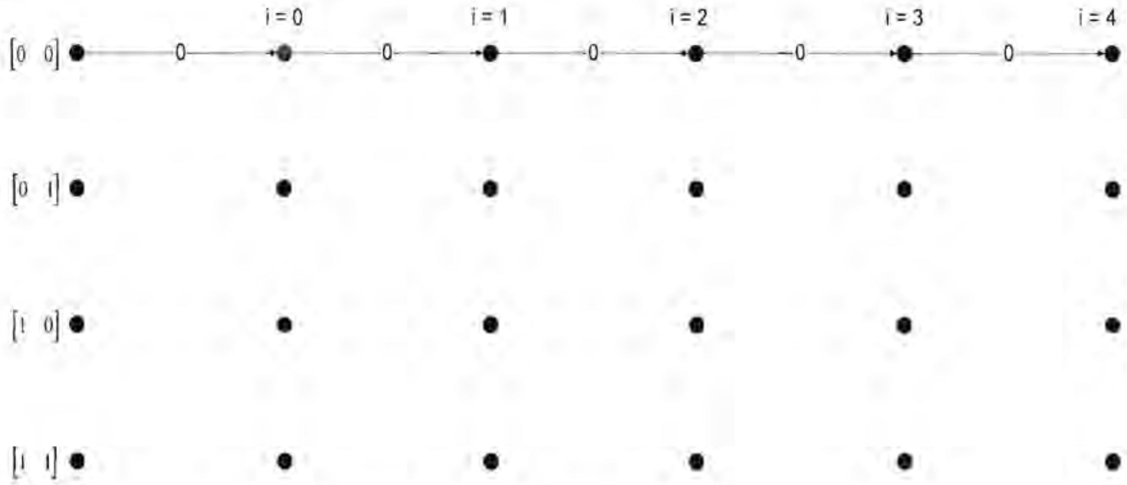

Figure 53 Single Code Trellis for Codeword 8

- Combination of all single code trellis diagrams

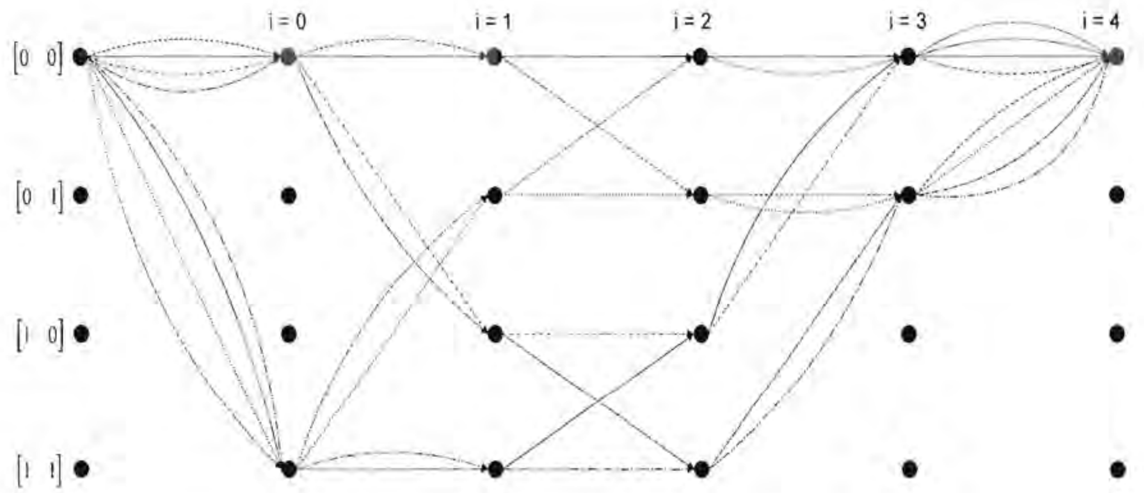

Figure 54 Combined Single Code Trellis Diagram

- Create the final syndrome trellis diagram

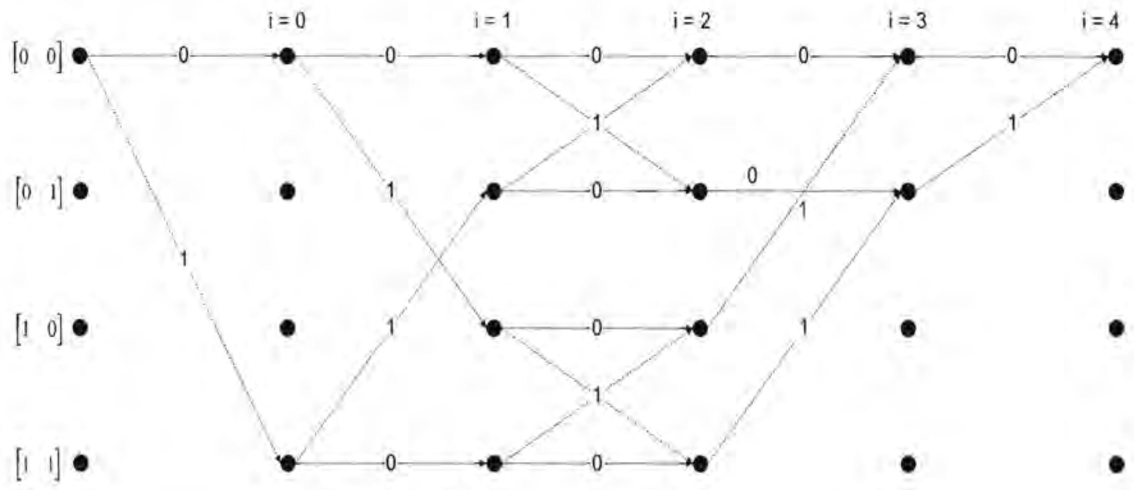

Figure 55 Final Syndrome Trellis Diagram

Appendix B

Computation of the Parity Check Matrix from the Generator Matrix

B.1 Introduction

In **Section 3.4** it was described analytically how the parity check matrix **H** of a linear block code was generated from the generator matrix **G**. An example to illustrate the proposed method is given below:

B.2 Explanation by Example

Given the following generator matrix

$$\boldsymbol{G} = \begin{pmatrix} 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 & 1 & 0 \end{pmatrix} \quad \textbf{(B.1)}$$

of a (7,4)-Hamming-Code.

Firstly, we will try and transform a column of this generator matrix G in such a manner that it represents the first unit vector after transformation

$$\boldsymbol{e}_1 = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} \quad \textbf{(B.2)}$$

The first step in achieving this is the swapping of the first and second row. Note that according to **Chapter 2** such elementary matrix operations are allowed.

$$\begin{pmatrix} 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 & 1 & 0 \end{pmatrix} \quad \textbf{(B.3)}$$

In order to complete the task, the third and fourth rows are replaced by their respective sums with the first row. When this is done, the first column is the first unit vector.

$$\begin{pmatrix} 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 \end{pmatrix} \quad \textbf{(B.4)}$$

For the further transformation, the element of the second row in the second column has to be a 1. This is done by swapping the second and the third row.

$$\begin{pmatrix} 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 \end{pmatrix} \quad \textbf{(B.5)}$$

The elements of the first and the fourth row in the second column are now made 0, in that the second row is added to these two rows.

$$\begin{pmatrix} 1 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 \end{pmatrix} \quad \textbf{(B.6)}$$

In the third row of the third column there is already a 1 present. This makes the next operation easier since all that is require is to make the rest of the elements in this column 0. This is done by adding the third row to the particular rows.

$$\begin{pmatrix} 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 \end{pmatrix} \tag{B.7}$$

Preferably, only a 1 in the fourth row of the 4 column should be present. There is however a zero. There is also no lower row with a 1 in the fourth column, which can be exchanged with the fourth row in order to meet the requirement. Rows further up in the matrix can also not be used to perform the row swapping operation, since this would also disturb the unit vectors already present in several of the columns. The only possible solution is not to use the fourth column, but to move along to another column further to the right of the matrix. The column that we choose for this is the fifth one. This one is chosen due to the fact that it already has a one in the fourth row. This makes it a lot easier. The other ones present in this column will also have to be made 0, and this is accomplished by adding the fourth row to the particular rows. By doing this, we obtain the following generator matrix which still produces exactly the same code. It does however have a different mapping between the information vectors $\boldsymbol{i}$ and the code vectors $\boldsymbol{c}$ (for clarification on this, the reader is referred back to **Section 2.1** in **Chapter 2**).

$$\begin{pmatrix} 1 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 \end{pmatrix} \tag{B.8}$$

In the first, second, third and fifth columns the following unit vectors are present. This completes the first step in finding the parity check matrix.

$$\boldsymbol{e}_1 = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \qquad \boldsymbol{e}_2 = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix} \tag{B.9}$$

$$e_3 = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix} \qquad e_4 = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} \tag{B.10}$$

To bring this matrix into the systematic form **G'** it is necessary to exchange the fourth and the fifth columns. In the systematic form matrix **G'**, the following matrix

$$A = \begin{pmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \\ 0 & 1 & 1 \end{pmatrix} \tag{B.11}$$

is contained in the fourth, sixth and seventh columns. In these columns of the corresponding parity check matrix **H**, the unit matrix $\mathbf{I}_{n-k}$ is present. The remaining columns (first, second, third and fifth columns) make up the transposed matrix

$$-A^T = \begin{pmatrix} 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{pmatrix} \tag{B.12}$$

This results in the parity check matrix that was used to construct the syndrome trellis diagram of **Chapter 3**. It can easily be shown that all the relevant equations in **Section 3.4** are satisfied.

$$\boldsymbol{H} = \begin{pmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 1 \end{pmatrix} \tag{B.13}$$

Appendix C

Generator Matrixes of Simulated Codes

C.1 Introduction

For reasons of completeness and also so that simulation results can be repeated by other people basing their work on this piece, this **Appendix C** will contain most of the generator matrixes G of the codes used in this work. In order to obtain the parity check matrix H for a particular code, the generator matrix G as given in this appendix can be transformed using the process described in **Chapter 3** and illustrated in **Appendix B**.

C.2 Generator Matrixes of given BCH-Codes

The (15,7)-BCH-Code has the following generator polynomial given in the exponential form below.

$$g(x) = 1 + x^4 + x^6 + x^7 + x^8 \tag{C.1}$$

The generator matrix can be obtained from the polynomial and is as follows.

$$\boldsymbol{G} = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{pmatrix} \tag{C.2}$$

For the (31,21)-BCH-Code, the generator polynomial and generator matrix is as given below.

$$g(x) = 1 + x^3 + x^5 + x^6 + x^8 + x^9 + x^{10} \tag{C.3}$$

$$G = \begin{pmatrix}
1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1
\end{pmatrix} \tag{C.4}$$

C.3 Generator Matrixes of given Reed-Muller-Codes

Here follow several Reed-Muller-Codes and their generator matrixes.

- (16,5)-Reed-Muller-Code $R(1,4)$

$$\boldsymbol{G}=\begin{pmatrix}1&0&1&0&1&0&1&0&1&0&1&0&1&0&1&0\\1&1&0&0&1&1&0&0&1&1&0&0&1&1&0&0\\1&1&1&1&0&0&0&0&1&1&1&1&0&0&0&0\\0&0&0&0&1&1&1&1&0&0&0&0&1&1&1&1\\0&0&0&0&0&0&0&0&1&1&1&1&1&1&1&1\end{pmatrix} \tag{C.5}$$

- (16,11)-Reed-Muller-Code $R(2,4)$

$$\boldsymbol{G}=\begin{pmatrix}1&0&0&0&1&0&0&0&1&0&0&0&1&0&0&0\\1&0&1&0&0&0&0&0&0&0&0&0&1&0&1&0\\0&0&0&0&1&0&1&0&0&0&0&0&1&0&1&0\\0&0&0&0&0&0&0&0&1&0&1&0&1&0&1&0\\1&1&0&0&0&0&0&0&0&0&0&0&0&0&1&1\\0&0&1&1&0&0&0&0&0&0&0&0&0&0&1&1\\0&0&0&0&1&1&0&0&0&0&0&0&0&0&1&1\\0&0&0&0&0&0&1&1&0&0&0&0&0&0&1&1\\0&0&0&0&0&0&0&0&1&1&0&0&0&0&1&1\\0&0&0&0&0&0&0&0&0&0&1&1&0&0&1&1\\0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1\end{pmatrix} \tag{C.6}$$

- (32,31)-Reed-Muller-Code $R(4,5)$ (Parity-Check-Code)

$$G=\begin{pmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1
\end{pmatrix} \quad \textbf{(C.7)}$$

- (32,26)-Reed-Muller-Code $R(3,5)$

$$G = \begin{pmatrix}
1&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0\\
1&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&1&0&0&0\\
0&0&0&0&0&0&0&0&1&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&1&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&1&0&0&0&1&0&0&0&1&0&0&0\\
1&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&1&0\\
0&0&0&0&1&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&1&0\\
0&0&0&0&0&0&0&0&1&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&1&0\\
0&0&0&0&0&0&0&0&0&0&0&0&1&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&1&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&1&0&0&0&0&0&0&0&0&0&1&0&1&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&1&0&0&0&0&0&1&0&1&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&1&0&1&0&1&0\\
1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1\\
0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1\\
0&0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1\\
0&0&0&0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1\\
0&0&0&0&0&0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1\\
0&0&0&0&0&0&0&0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1\\
0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&1&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&1&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&0&0&0&0&0&0&0&1&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&0&0&0&0&0&1&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&0&0&0&1&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&0&1&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1
\end{pmatrix} \qquad (C.8)$$

- (32,16)-Reed-Muller-Code $R(2,5)$

$$G = \begin{pmatrix}
1&0&0&0&1&0&0&0&1&0&0&0&1&0&0&0&1&0&0&0&1&0&0&0&1&0&0&0&1&0&0&0\\
1&0&1&0&0&0&0&0&1&0&1&0&0&0&0&0&1&0&1&0&0&0&0&0&1&0&1&0&0&0&0&0\\
1&0&1&0&1&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&1&0&1&0&1&0\\
0&0&0&0&0&0&0&0&1&0&1&0&1&0&1&0&0&0&0&0&0&0&0&0&1&0&1&0&1&0&1&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&1&0&1&0&1&0&1&0&1&0&1&0&1&0\\
1&1&0&0&0&0&0&0&1&1&0&0&0&0&0&0&1&1&0&0&0&0&0&0&1&1&0&0&0&0&0&0\\
1&1&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&0&1&1&0&0\\
0&0&0&0&0&0&0&0&1&1&0&0&1&1&0&0&0&0&0&0&0&0&0&0&1&1&0&0&1&1&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&0&1&1&0&0&1&1&0&0&1&1&0&0\\
1&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1\\
0&0&0&0&1&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1\\
0&0&0&0&0&0&0&0&1&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1\\
0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1&0&0&0&0&0&0&0&0&1&1&1&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1&0&0&0&0&1&1&1&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1&1&1&1&1
\end{pmatrix} \quad \textbf{(C.9)}$$

Appendix D

Shannon Product of Trellises

D.1 Introduction

In **Chapter 5**, extensive use of the Shannon Product of trellises theory was made. This Appendix focuses on this unique trellis "combination" method developed by Shannon.

it should however be noted that the Shannon product of trellises is in itself also a fully functional method of designing a trellis diagram from first principles. It does however not always deliver a minimal trellis representation.

D.2 Analytical Approach to the Shannon Trellis Product

The method will first be described on an analytical basis, and will be illustrated with an example afterwards.

Let C_1 and C_2 be binary linear block codes with parameters (N,K_1,D_1) and (N,K_2,D_2) respectively. The two generator matrixes for the codes will be $\mathbf{G}_1$ and $\mathbf{G}_2$. It is further assumed that C1 and C2 only have the all-zero code word in common. From this would follow that:

$$C_1 \cap C_2 = \{0\} \tag{D.1}$$

Their direct sum is defined as follows:

$$C \overset{\Delta}{=} C_1 \oplus C_2 \overset{\Delta}{=} \left\{ u + v : u \in C_1, v \in C_2 \right\} \tag{D.2}$$

From this it would follow that C is an (N,K_1+K_2,d) linear block code with minimum distance equal to

$$d_{\min} = \min\left\{ d_1, d_2 \right\} \tag{D.3}$$

and generator matrix

$$\boldsymbol{G} = \begin{bmatrix} \boldsymbol{G}_1 \\ \boldsymbol{G}_2 \end{bmatrix}. \qquad \textbf{(D.4)}$$

Let T_1 and T_2 be N-section trellis diagrams for C_1 and C_2 respectively. Then a N-section trellis T for the direct-sum code C can be constructed by taking the Shannon product of T_1 and T_2.

Let

$$T = T_1 \times T_2 \qquad \textbf{(D.5)}$$

denote the Shannon product of T_1 and T_2.

The state spaces of $T1$ and $T2$ at time i are given as

$$\Sigma_i(C_1) \quad and \quad \Sigma_i(C_2) \quad for\ 0 \le i \le N \qquad \textbf{(D.6)}$$

respectively.

The construction of the Shannon product T is carried out according to the following procedure:

- For all i, the state space product from of $C1$ and C2 is obtained according to **Equation D.6**. This product can then be given as

  $$\Sigma_i(C_1) \times \Sigma_i(C_2) = \left\{\left(\sigma_i^{(1)}, \sigma_i^{(2)}\right) : \sigma_i^{(1)} \in \Sigma_i(C_1), \sigma_i^{(2)} \in \Sigma_i(C_2)\right\} \qquad \textbf{(D.7)}$$

  Then $\Sigma_1(C_1) \times \Sigma_2(C_2)$ forms the state space of $T_1 \times T_2$ at time i. In other words, the two-tuples from $\Sigma_1(C_1) \times \Sigma_2(C_2)$ form the nodes of T at level i.

- A state $\left(\sigma_i^{(1)}, \sigma_i^{(2)}\right) \in \Sigma_i(C_1) \times \Sigma_i(C_2)$ is adjacent to a state $\left(\sigma_{i+1}^{(1)}, \sigma_{i+1}^{(2)}\right) \in \Sigma_{i=1}(C_1) \times \Sigma_{i+1}(C_2)$ if and only if $\sigma_i^{(1)}$ is adjacent to $\sigma_{i+1}^{(1)}$ and $\sigma_i^{(2)}$ is adjacent to $\sigma_{i+1}^{(2)}$. Let $l\left(\sigma_i^{(1)}, \sigma_{i+1}^{(1)}\right)$ denote the label of the branch that connects $\sigma_i^{(1)}$ to $\sigma_{i+1}^{(1)}$ in trellis T_1 and $l\left(\sigma_i^{(2)},\sigma_{i+1}^{(2)}\right)$ denote the label of the branch that connects $\sigma_i^{(2)}$ to $\sigma_{i+1}^{(2)}$ in trellis T_2. Then two adjacent states $\left(\sigma_i^{(1)},\sigma_i^{(2)}\right)$ and $\left(\sigma_{i+1}^{(1)},\sigma_{i+1}^{(1)}\right)$ in trellis T are connected by a branch with label

$$l\left(\sigma_i^{(1)},\sigma_{i+1}^{(1)}\right)+l\left(\sigma_i^{(2)},\sigma_{i+1}^{(2)}\right) \qquad \textbf{(D.8)}$$

For $0 \leq i \leq N$, let $\rho_i\left(C_1\right)$ and $\rho_i\left(C_2\right)$ be the dimensions of the state spaces given in Equation D.6 respectively. Then the state space dimension profile of $T_1 \times T_2$ is given by

$$\left(\rho_0\left(C_1\right)+\rho_0\left(C_2\right),\rho_1\left(C_1\right)+\rho_1\left(C_2\right),\ldots,\rho_N\left(C_1\right)+\rho_N\left(C_2\right)\right) \qquad \textbf{(D.9)}$$

D.3 Shannon Product of Trellises by Example

The following example will illustrate the technique as described in **Section D.2**. The two block codes C_1 and C_2 are assumed to be linear, and have generator matrixes in the following form:

$$\boldsymbol{G}_1=\begin{pmatrix}1&1&1&1&0&0&0&0\\0&1&0&1&1&0&1&0\end{pmatrix} \qquad \textbf{(D.10)}$$

and

$$\boldsymbol{G}_2=\begin{pmatrix}0&0&1&1&1&1&0&0\\0&0&0&0&1&1&1&1\end{pmatrix} \qquad \textbf{(D.11)}$$

From the discussion in **Section D.2** it follows that

$$C_1 \cap C_2=\{0\} \qquad \textbf{(D.12)}$$

and that the direct-sum is given as

$$G = \begin{pmatrix} G_1 \\ G_2 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{pmatrix}. \qquad \textbf{(D.13)}$$

The following two figures show the trellis diagrams for the two codes C_1 and C_2 respectively.

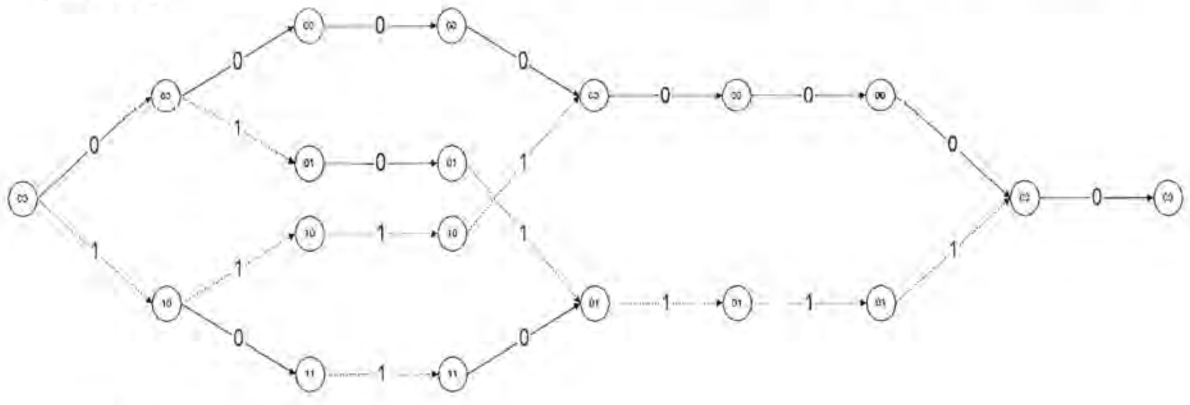

Figure 56 Trellis Diagram for Code C_1

These trellises are then combined with the Shannon procedure as described in **Section D.2**. The result is given in the figure below.

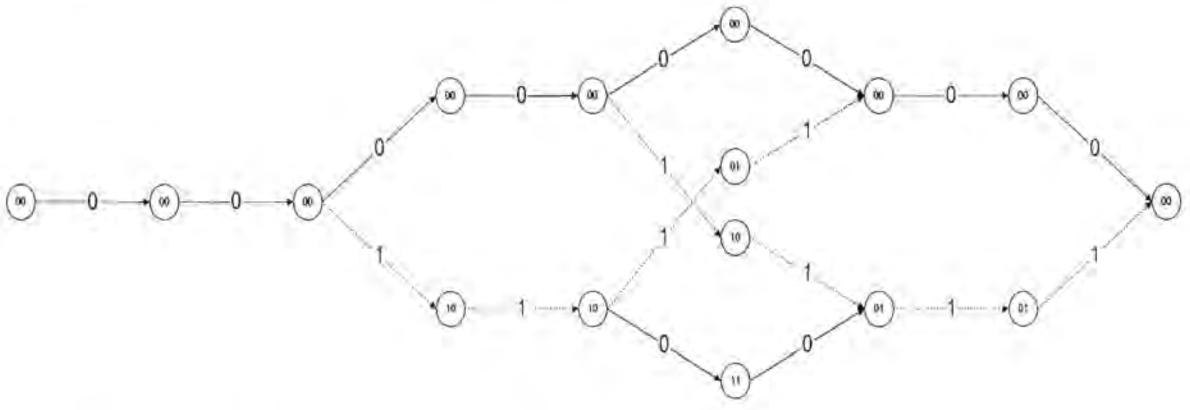

Figure 57 Trellis Diagram for Code C_2

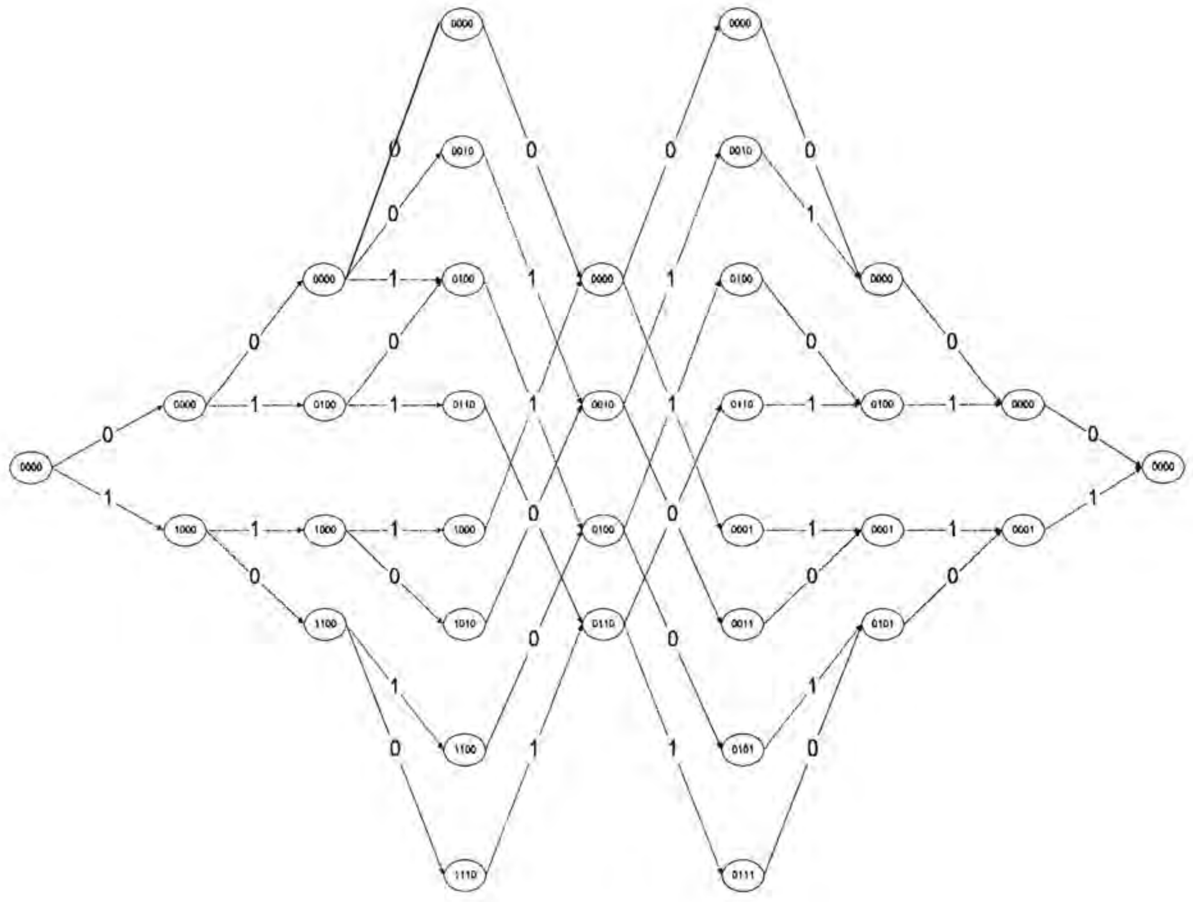

Figure 58 Combined Shannon Product Trellis Diagram

Appendix E

Coset and Syndrome Trellises by Example

E.1 Introduction

In this Appendix, the two techniques for Reed-Solomon trellis construction as presented in **Section 5.5** of **Chapter 5**, are illustrated by example. The first part is concerned with the syndrome construction technique, and the second part is concerned with the coset trellis design technique.

E.2 Syndrome Construction

This example will illustrate the use of the syndrome technique in obtaining a trellis diagram for a (7, 5, 3) Reed-Solomon code, with symbols taken from GF(8). The generator polynomial for this specific Reed-Solomon code is given as

$$g(X) = \alpha^3 + \alpha^4 \cdot X + X^2 \tag{E.1}$$

from which the generator matrix **G** can be constructed.

$$\boldsymbol{G} = \begin{pmatrix} g_1 \\ g_2 \\ g_3 \\ g_4 \\ g_5 \end{pmatrix} = \begin{pmatrix} \alpha^3 & \alpha^4 & 1 & 0 & 0 & 0 & 0 \\ 0 & \alpha^3 & \alpha^4 & 1 & 0 & 0 & 0 \\ 0 & 0 & \alpha^3 & \alpha^4 & 1 & 0 & 0 \\ 0 & 0 & 0 & \alpha^3 & \alpha^4 & 1 & 0 \\ 0 & 0 & 0 & 0 & \alpha^3 & \alpha^4 & 1 \end{pmatrix} \tag{E.2}$$

The overall Reed-Solomon code can be presented in the following form:

$$C = \sum_{j=1}^{5} C_j \tag{E.3}$$

In **Equation E.3** C_j with j=1,2,...,5 is a (7, 1, 3) sub-code generated by g_j. These subcodes can be expressed in the following format:

$$\begin{aligned}
C_1 &= x_1 \cdot g_1 = \left(\alpha^3 x_1,\ \alpha^4 x_1,\ x_1,\ 0,\ 0,\ 0,\ 0\right)\\
C_2 &= x_2 \cdot g_2 = \left(0,\ \alpha^3 x_2,\ \alpha^4 x_2,\ x_2,\ 0,\ 0,\ 0\right)\\
C_3 &= x_3 \cdot g_3 = \left(0,\ 0,\ \alpha^3 x_3,\ \alpha^4 x_3,\ x_3,\ 0,\ 0\right)\\
C_4 &= x_4 \cdot g_4 = \left(0,\ 0,\ 0,\ \alpha^3 x_4,\ \alpha^4 x_4,\ x_4,\ 0\right)\\
C_5 &= x_5 \cdot g_5 = \left(0,\ 0,\ 0,\ 0,\ \alpha^3 x_5,\ \alpha^4 x_5,\ x_5\right)
\end{aligned} \tag{E.4}$$

The component trellises have a very simple structure with the following state profiles:

$$\begin{aligned}
N_1(t) &= \left(1,\ 8,\ 8,\ 1,\ 1,\ 1,\ 1,\ 1\right)\\
N_2(t) &= \left(1,\ 1,\ 8,\ 8,\ 1,\ 1,\ 1,\ 1\right)\\
N_3(t) &= \left(1,\ 1,\ 1,\ 8,\ 8,\ 1,\ 1,\ 1\right)\\
N_4(t) &= \left(1,\ 1,\ 1,\ 1,\ 8,\ 8,\ 1,\ 1\right)\\
N_5(t) &= \left(1,\ 1,\ 1,\ 1,\ 1,\ 8,\ 8,\ 1\right)
\end{aligned} \tag{E.5}$$

The state profiles from **Equation E.5** are represented in graphical format below.

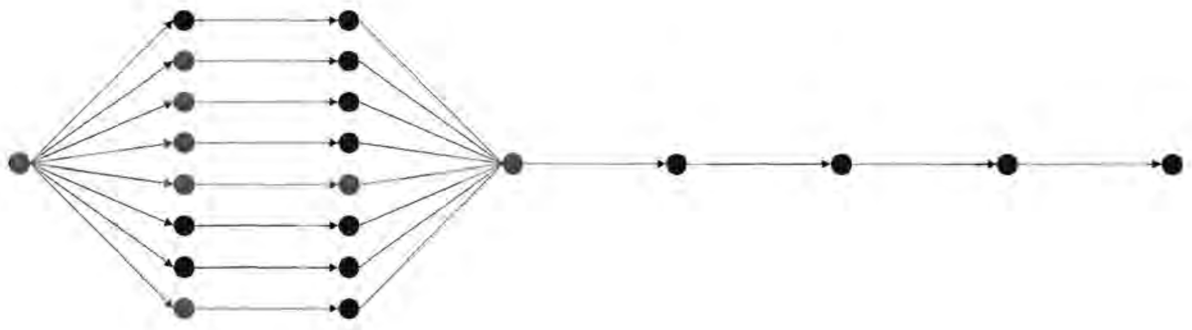

Figure 59 Sate Profile for C_1

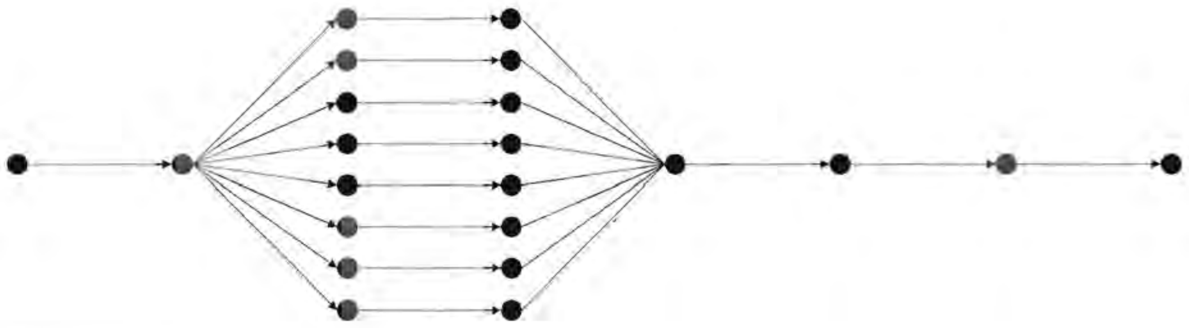

Figure 60 State Profile for C_2

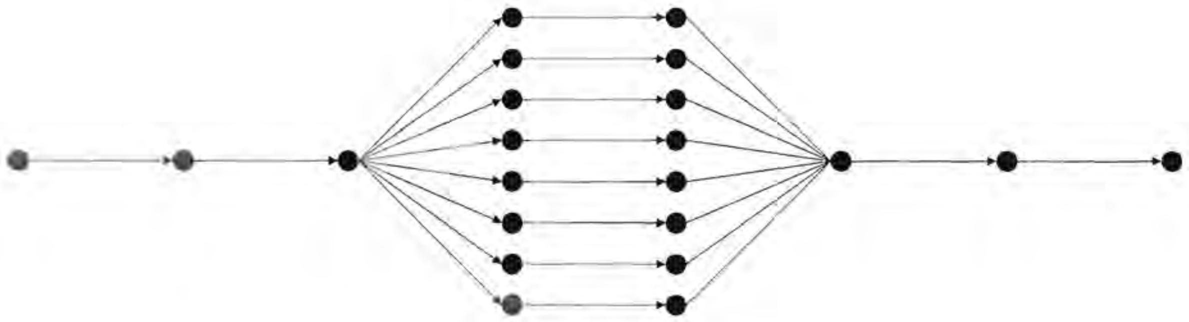

Figure 61 State Profile for C_3

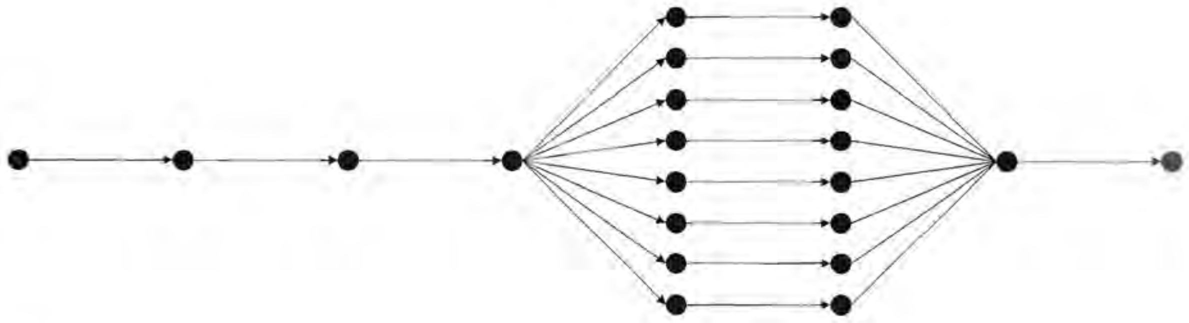

Figure 62 State Profile for C_4

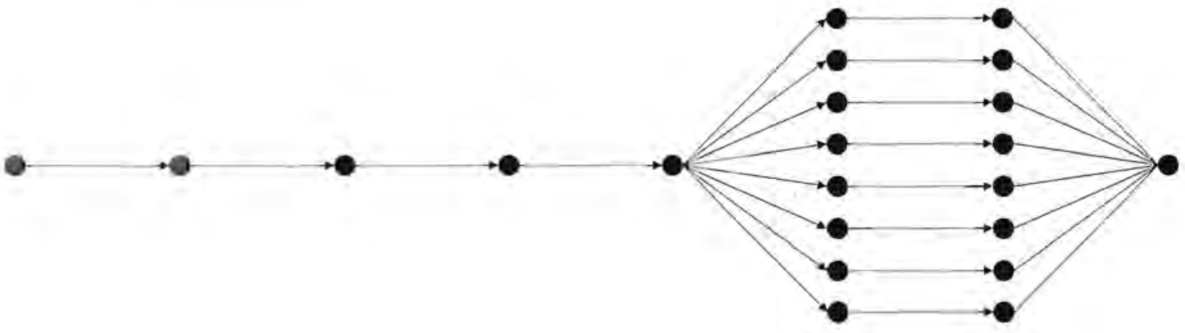

Figure 63 State Profile for C_5

When the procedure as described in **Section 6.2** is applied to the above state profiles, then a new state profile for the entire code can be written as:

$$N(t) = \{1,\ 8,\ 64,\ 64,\ 64,\ 64,\ 64,\ 8,\ 1\} \quad \textbf{(E.6)}$$

The result of this procedure is given in the figure on the next page. Due to the huge amount of states and branches, no labeling is done on the graph itself. It should be noted that several branches have been omitted in state 3, state 4 and state 5.

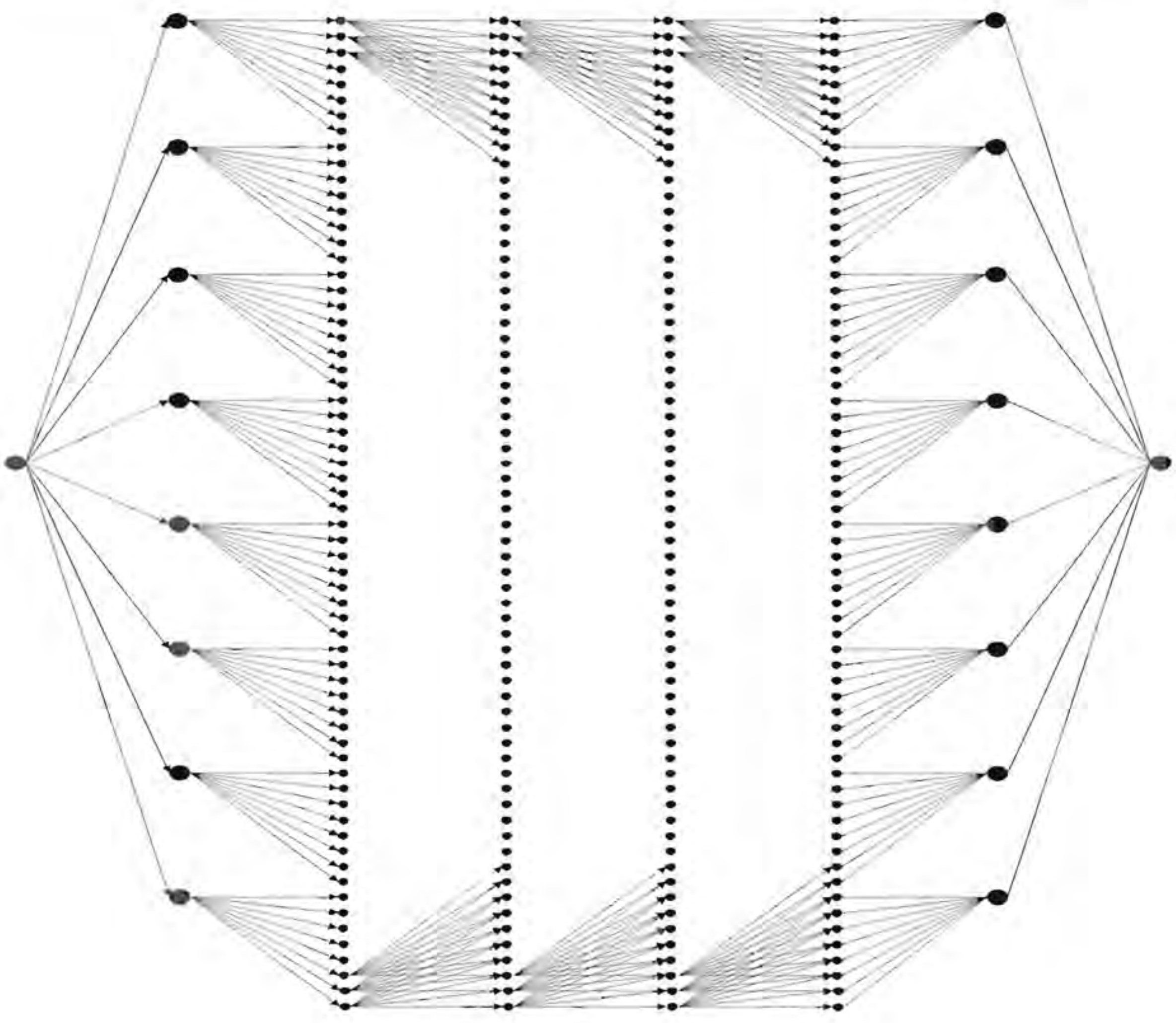

Figure 64 Syndrome Trellis Diagram for the (7, 5, 3) Reed-Solomon Code

E.3 Coset Construction

For this example, a (7, 5, 3) Reed-Solomon code is chosen with symbols taken from GF(8). The chosen code has the following generator polynomial

$$g(X) = \alpha^3 + \alpha \cdot X + X^2 + \alpha^3 \cdot X^3 + X^4 \tag{E.7}$$

From the above generator polynomial, the generator matrix can be constructed as follows:

$$\boldsymbol{G} = \begin{pmatrix} \alpha^3 & \alpha & 1 & \alpha^3 & 1 & 0 & 0 \\ 0 & \alpha^3 & \alpha & 1 & \alpha^3 & 1 & 0 \\ 0 & 0 & \alpha^3 & \alpha & 1 & \alpha^3 & 1 \end{pmatrix} \tag{E.8}$$

The state profile can be obtained as given below:

$$\begin{aligned} N_0 &= \frac{q^3}{q^0 \cdot q^3} = 1 \\ N_1 &= \frac{q^3}{q^0 \cdot q^2} = q = 8 \\ N_2 &= \frac{q^3}{q^0 \cdot q} = q^2 = 64 \\ N_3 &= \frac{q^3}{q^0 \cdot q^0} = q^3 = 512 \\ N_4 &= \frac{q^3}{q^0 \cdot q^0} = q^3 = 512 \\ N_5 &= \frac{q^3}{q^1 \cdot q^0} = q^2 = 64 \\ N_6 &= \frac{q^3}{q^2 \cdot q^0} = q = 8 \\ N_7 &= \frac{q^3}{q^3 \cdot q^0} = 1 \end{aligned} \tag{E.9}$$

From this point onwards, a number of different but isomorphic trellis diagrams exist. The one chosen depends solely on the designer, and the procedure as given in **Section 5.5** is applied. The trellis is then again the Shannon product of a number of sub-trellises. Due to the large complexity of even this small code, the resulting trellis diagrams will not be presented here.

Appendix F

Galois Field Arithmetic

F.1 Introduction

This appendix will give a brief introduction into the arithmetic of fields. A field is a set of elements in which one is able to do addition, subtraction, multiplication and division without ever having to leave the set. Addition and multiplication must satisfy the commutative, associative and distributive laws.

F.2 Fields

Let F be a set of elements on which two binary operations called addition and multiplication are defined. The set F together with the two binary operations mentioned above is a field if the following conditions are satisfied:

- F is a commutative group under addition . The identity element with respect to addition is called the zero element or the additive identity of F and is denoted by 0.
- The set of nonzero elements in F is a commutative group under multiplication. The identity element with respect to multiplication is called the unit element or the multiplicative identity of F and is denoted by 1.
- Multiplication is distributive over addition, that is, for any three elements a, b and c in F the following holds:
$$a \cdot (b + c) = a \cdot b + a \cdot c. \qquad \textbf{(F.1)}$$

It follows from the above definition, that a field consists of at least two elements, the additive identity and the multiplicative identity.

The number of elements in a field is called the order of the field. A field with a finite number of elements is called a finite field. In a field, the additive inverse of an element a is denoted by $-a$ and the multiplicative inverse of a is denoted by a^{-1}, provided that a is not equal to zero. Subtracting a field element *b* from another field element *a* is defined as adding the additive inverse. If *b* is a nonzero element dividing by *b* is defined as multiplying by the multiplicative inverse.

A number of basic properties of fields exist and are given below.

- For every element *a* in a field $a \cdot 0 = 0 \cdot a = 0$
- For any two nonzero elements *a* and *b* in a field $a \cdot b \neq 0$
- $a \cdot b = 0$ and $a \neq 0$ implies that $b = 0$
- For any two elements *a* and *b* in a field $-(a \cdot b) = (-a) \cdot (b) = (a) \cdot (-b)$
- For $a \neq 0$, $a \cdot b = a \cdot c$ implies that $b = c$

Fields are normally indicated by GF(q). This notation is used in honor of Galois who discovered these fields. The binary field in which most everyday calculations are done is called GF(2). Every Galois field contains a given number of symbols. In GF(4) the set is limited to 4 symbols. The symbols can be expressed in exponential or polynomial form, which makes addition and multiplication easier depending on which representation is used. A table is given below to illustrate the above process with 8 symbols.

No.	Exponential Representation	Polynomial Representation
1	0	0
2	1	1
3	α	α
4	α^2	α^2
5	α^3	$\alpha^3 = \alpha + 1$
6	α^4	$\alpha^3 \cdot \alpha = (\alpha + 1) \cdot \alpha = \alpha^2 + \alpha$
7	α^5	$\alpha^4 \cdot \alpha = \ldots = \alpha^2 + \alpha + 1$
8	α^6	$\alpha^5 \cdot \alpha = \ldots = \alpha^2 + 1$

Table 23 Galois Symbol Representation in Exponential and Polynomial Form

This discussion was just a very brief introduction to the study of finite fields and Galois fields. For further information, the reader is referred to standard works[21],[24],[25] on this topic.

Appendix G

Simulation Code

G.1 Introduction

This is just a partial reproduction of the simulation code used in the dissertation. The full code is available on request from the authors.

G.2 Simulation Code

```
//---------------------------------------------------------------------------
#include <vcl\vcl.h>
#pragma hdrstop
#include <stdio.h>
#include <math.h>
#include <vcl/clipbrd.hpp>
#include "main.h"
//---------------------------------------------------------------------------
#pragma resource "*.dfm"
TForm1 *Form1;

//---------------------------------------------------------------------------
// Constructor
__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
{
        // Setup the system
  ReadSpecs();
  states = pow(2,(float)(n-k));

  // 20-stage PN Generator
  int data_source_feedback[2] = {2,19};
  DataSource = new PN_Gen(20,2,data_source_feedback);

        // Create the coder
  HammingCoder = new Cyclic(n,k,gen_pol);

  // Create channel
  Pe = 0.1;
  Channel = new BSC(1,-1,Pe);
  No = Channel->Compute_No_needed_for_Pe();
  AWGN = new Noise(0,No);

  // Create the block code's trellis
  HammingTrellis = new Trellis(states,(n+1));
  HammingTrellis->BlockTrellis(n,k,parity);

  // Create the Viterbi decoder
  HammingDecoder = new Viterbi(states,(n+1));

  // Create a few buffers
  message = new Matrix(1,k);
  received = new Matrix(1,n);
  code = new Matrix(1,n);

  // Display startup Pe and associated Eb/No
  Edit1->Text=(10*log10((float)(n)/(float)(k*No)));
  Edit5->Text=Pe;

  // Initial node spacing factors
  zi = (Image1->Width)/(4*(n+2));
  zo = (Image1->Height)/(4*(states+1));

  // Display the startup trellis
  DrawTrellis();
}
//---------------------------------------------------------------------------


//---------------------------------------------------------------------------
void TForm1::ReadSpecs()
{
        FILE *specs;
  char temp[30];

  OpenDialog1->Filter = "Block code specs (*.bcs)|*.BCS";
        if (OpenDialog1->Execute())
  {
        specs = fopen((OpenDialog1->FileName).c_str(),"rt");

            // Read n from file
            fscanf(specs,"%s",temp);
            fscanf(specs,"%i",&n);

            // Read k from file
            fscanf(specs,"%s",temp);
            fscanf(specs,"%i",&k);

            // Read generator polynomial from file
```

```
          fscanf(specs,"%s",temp);
          gen_pol = new int[n-k+1];
          for (int i=0; i<(n-k+1); i++)
          {
                fscanf(specs,"%s",temp);
                gen_pol[i] = atoi(temp);
          }

          // Read the H matrix from file
          fscanf(specs,"%s",temp);
          VECALLOC(parity,(n-k),n);
          for (int i=0; i<(n-k); i++)
          {
                      for (int j=0; j<n; j++)
            {
              fscanf(specs,"%s",temp);
                                  parity[i][j] = atoi(temp);
                 }
               }

                fclose(specs);
   }
   else exit(0);
}
//------------------------------------------------------------------------


//------------------------------------------------------------------------
// Display the trellis information
void TForm1::DrawTrellis()
{
   // Determine spacing between nodes
   int x_step = zi;
   int y_step = zo;

   // Refresh the canvas
         Image1->Canvas->Pen->Color=clBlack;
         Image1->Canvas->Brush->Color=clWhite;
   Image1->Canvas->Rectangle(0,0,Image1->Width,Image1->Height);

   // Draw nodes
   Image1->Canvas->Brush->Color=clBlack;
   for (int m=1; m<=(n+1); m++)
   {
         for (int l=1; l<=states; l++)
         Image1->Canvas->Ellipse((m*x_step-3),(l*y_step-3),(m*x_step+3),(l*y_step+3));
   }
   Image1->Canvas->Brush->Color=clWhite;

   // Draw the branches of the trellis
```

```
  for (int j=0; j<n; j++)
  {
          for (int m=0; m<states; m++)
          {
                for (int l=0; l<states; l++)
                {
                  Image1->Canvas->MoveTo((j+1)*x_step, y_step*(m+1));
                  if (HammingTrellis->OutgoingBranches(j,m,l)==1)
                  { if (HammingTrellis->ReturnBranchCause(j,m,l)==0)
                    Image1->Canvas->Pen->Color=clBlue;
                    else Image1->Canvas->Pen->Color=clRed;
                    Image1->Canvas->LineTo((j+2)*x_step, y_step*(l+1));
                        }
                }
          }
  }

  Image1->Canvas->Pen->Color=clBlack;
}
//---------------------------------------------------------------------------


//---------------------------------------------------------------------------
// Display the computed metrics
void TForm1::ShowMetrics()
{
        char metric[4];
        Image1->Canvas->Pen->Color=clBlack;

   // Determine the spacing between metrics
        int x_step = zi;
  int y_step = zo;

   // Write metris to the canvas
  for (int m=1; m<=(n+1); m++)
  {
        for (int l=1; l<=states; l++)
     {
        if (HammingTrellis->ReturnStateActive(m-1,l-1))
        {
        sprintf(metric,"%2.2f",(HammingDecoder->ReturnMetric(l-1,m-1)));
          Image1->Canvas->TextOut((m*x_step),(l*y_step-20),metric);
        }
     }
  }
}
//---------------------------------------------------------------------------


//---------------------------------------------------------------------------
```

```
// Display the most probable path
void TForm1::DrawPath()
{
        // Redraw the trellis
        DrawTrellis();
        Image1->Canvas->Pen->Color=clGreen;
  Image1->Canvas->Pen->Width=3;

  // Draw best path
  int x_step = zi;
  int y_step = zo;

  for (int j=0; j<n; j++)
  {
          for (int m=0; m<states; m++)
          {
          for (int l=0; l<states; l++)
              {
                 Image1->Canvas->MoveTo((j+1)*x_step, y_step*(m+1));
                 i                                                                 f
(HammingDecoder->ReturnPath(m,j)*HammingDecoder->ReturnPath(l,j+1)*HammingTrellis-
>OutgoingBranches(j,m,l))
             Image1->Canvas->LineTo((j+2)*x_step, y_step*(l+1));
        i f   ( H a m m i n g D e c o d e r - > R e t u r n P a t h ( m , j ) = = 1 )
Image1->Canvas->Ellipse(((j+1)*x_step-5),((m+1)*y_step-5),((j+1)*x_step+3),((m+1)*y_step+5));
              }
          }
  }

  Image1->Canvas->Pen->Color=clBlack;
  Image1->Canvas->Pen->Width=1;
}
//---------------------------------------------------------------------------


//---------------------------------------------------------------------------
// Transmit and receive a single code word
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  AnsiString c;
  AnsiString r;
  AnsiString v;
  float receive[15];
  float noise;

  // Flush shift register based coder
        HammingCoder->Flush();

  // Construct message vector
  for (int x=0; x<k; x++)
  {
      message->ChangeElement(0,x,DataSource->Output(0));
      DataSource->Shift();
   }

   // Construct code word
   HammingCoder->Compute(message);

   // Buffer the code word that is going to be transmitted
   for (int y=0; y<n; y++)          code->ChangeElement(0,y,HammingCoder->Return_code_bit(y));

   // Transverse channel
   for (int y=0; y<n; y++)
   {
         noise = AWGN->Output(1);

received->ChangeElement(0,y,Channel->Transverse_Channel((float)(2*code->ReturnElement(0
,y)-1)+noise));

      // Differentiate between hard decision and soft decision decoding approaches
      if (RadioGroup1->ItemIndex==0) receive[y] = (float)received->ReturnElement(0,y);
            if (RadioGroup1->ItemIndex==1) receive[y] = 2*code->ReturnElement(0,y)-1+noise;
   }

   // Decode the received code word
   HammingDecoder->Decode(receive,HammingTrellis);

   // Draw the best path as determined by the Viterbi decoder
   DrawPath();

   // Display computed metrics if necessary
   if (CheckBox1->State==1) ShowMetrics();

   // Display all info
         for (int y=0; y<n; y++)
   {
         c += code->ReturnElement(0,y);
               if (RadioGroup1->ItemIndex==0)  r += ((received->ReturnElement(0,y)+1)/2);
            else  r = "";
      v += HammingDecoder->ReturnCodeBit(y);
   }

         Edit2->Text=c.c_str();
   Edit3->Text=r.c_str();
   Edit4->Text=v.c_str();
}
//---------------------------------------------------------------------------


//---------------------------------------------------------------------------
// Expurgate the trellis
void __fastcall TForm1::Button2Click(TObject *Sender)
```

```
{
        HammingTrellis->Expurgate();
   DrawTrellis();
   DrawPath();
   if (CheckBox1->State==1) ShowMetrics();
}
//---------------------------------------------------------------------------


//---------------------------------------------------------------------------
// Unexpurgate the trellis
void __fastcall TForm1::Button3Click(TObject *Sender)
{
   // Re-create the block code's trellis
   HammingTrellis->BlockTrellis(n,k,parity);

   // Draw the unexpurgated trellis
   DrawTrellis();
   DrawPath();

        // Display metrics if necesaary
   if (CheckBox1->State==1) ShowMetrics();
}
//---------------------------------------------------------------------------


//---------------------------------------------------------------------------
// Compute the Pe associated with a give noise power
float TForm1::Get_Pe(float N)
{
  float pe=0;
  float y=0;

  for (int i=0; i<1000; i++)
  {
     pe += (0.01/sqrt(3.1415*N))*exp(-(y+1)*(y+1)/N);
     y += 0.01;
  }
  return pe;
}
//---------------------------------------------------------------------------


//---------------------------------------------------------------------------
// Transmit x code words at different Eb/No values
void __fastcall TForm1::Button6Click(TObject *Sender)
{
   float receive[30];
   float noise;
   float error;
   int temp;
  FILE *output;

  output = fopen("results.txt","wt");

  // Turn displays off
  Edit2->Enabled = false;
  Edit2->Update();
  Edit3->Enabled = false;
  Edit3->Update();
  Edit4->Enabled = false;
  Edit4->Update();
  Edit5->Enabled = false;
  Edit5->Update();

        // Step through the given range of Eb/No
         for   (float   step=atof(Edit6->Text.c_str());   step<=atof(Edit7->Text.c_str());
step+=atof(Edit8->Text.c_str()))
  {
        No = ((float)(n)/(float)(k))/(pow(10,(step/10)));
        AWGN->Change_variance(No/2);
     Pe=Get_Pe(No);
        error=0;
     Edit1->Text=(10*log10((float)(n)/(float)(k*No)));
     Edit1->Update();

     // Step through the given amount of code words
          for (float i=0; i<atoi(Edit9->Text.c_str()); i++)
          {
                ProgressBar1->Position = (i/atoi(Edit9->Text.c_str()))*100;

        // Flush the shift register based coder
                        HammingCoder->Flush();

                // Construct message vector
                for (int x=0; x<k; x++)
                {
                    message->ChangeElement(0,x,DataSource->Output(0));
                    DataSource->Shift();
                }

                // Construct code word
                HammingCoder->Compute(message);
                                    for    (int    y=0;    y<n;    y++)
code->ChangeElement(0,y,HammingCoder->Return_code_bit(y));

                // Transverse channel
                for (int y=0; y<n; y++)
                {
                       noise = AWGN->Output(1);

received->ChangeElement(0,y,Channel->Transverse_Channel((float)(2*code->ReturnElement(0
,y)-1)+noise));

          if (RadioGroup1->ItemIndex==0) receive[y] = (float)received->ReturnElement(0,y);
                                        if   (RadioGroup1->ItemIndex==1)   receive[y] =
2*code->ReturnElement(0,y)-1 + noise;
                    }

        // Decode the received code word
        HammingDecoder->Decode(receive,HammingTrellis);

        // Count errors
        temp = 0;
                                        for   (int   y=0;   y<n;   y++)   temp   +=
HammingDecoder->ReturnCodeBit(y)^(code->ReturnElement(0,y));
        error += (temp>0);
            }
     fprintf(output,"%f %f\n",10*log10((float)(n)/(float)(k*No)),error/atof(Edit9->Text.c_str()));
 }
  fclose(output);
        Edit2->Enabled = true;
  Edit3->Enabled = true;
  Edit4->Enabled = true;
  Edit5->Enabled = true;
}
//---------------------------------------------------------------------------


//---------------------------------------------------------------------------
// Compute the new Eb/No when Pe is changed
void __fastcall TForm1::Changing_Pe(TObject *Sender, WORD &Key, TShiftState Shift)
{
        if (Key == 13)
  {
        Pe = atof(Edit5->Text.c_str());
          Channel->Change_Pe(Pe);
          No = Channel->Compute_No_needed_for_Pe();
          Edit1->Text=(10*log10((float)(n)/(float)(k*No)));
  }
}
//---------------------------------------------------------------------------



//---------------------------------------------------------------------------
// Refresh shown Pe
void __fastcall TForm1::Exit_Pe_Change(TObject *Sender)
{
        Edit5->Text = Pe;
}
//---------------------------------------------------------------------------
```

```
//----------------------------------------------------------------------
// Toggles between displaying the metrics and not displaying the metrics
void __fastcall TForm1::Metric_State(TObject *Sender)
{
        if (CheckBox1->State == 1) ShowMetrics();
   else {DrawTrellis(); DrawPath();}
}
//----------------------------------------------------------------------


//----------------------------------------------------------------------
// Zoom in
void __fastcall TForm1::Test1Click(TObject *Sender)
{
        // Adjust spacing between nodes
        zi *= 2;
   zo *= 2;

   // Redisplay all
   DrawTrellis();
   DrawPath();
        if (CheckBox1->State == 1) ShowMetrics();
}
//----------------------------------------------------------------------


//----------------------------------------------------------------------
// Zoom out
void __fastcall TForm1::ZoomOut1Click(TObject *Sender)
{
        if ((zo>10)*(zi>10))
   {
        zi /= 2;
     zo /= 2;
   }

   DrawTrellis();
   DrawPath();
        if (CheckBox1->State == 1) ShowMetrics();
}
//----------------------------------------------------------------------


//----------------------------------------------------------------------
// Copy current displayed trellis, path and metrics to the clipboard
void __fastcall TForm1::Dowhat1Click(TObject *Sender)
{
        Clipboard()->Assign(Image1->Picture);
}
//----------------------------------------------------------------------
```