Contents

Chapter 10

System Design with Codes

As a system designer, an engineer often uses codes to improve performance. Chapters 8 and 9 have suggested that transmission of sequences with consequent sequence detection can improve performance significantly. An upper bound on data rate for such improvement is the channel capacity of Chapter 8. More than a half-century after the introduction of capacity, the cumulative effort of many fine engineers and mathematicians has produced a repertoire of good codes that with increasing complexity can be used to approach capacity on both the BSC and the AWGN.

This chapter will focus on the tabulation and use of these codes, leaving the inner details and comparisons of the codes as well as the design of yet other codes to the continuing efforts of coding theorists. The objective is to have a reference to which the designer can refer and estimate the various coding gains, complexities, and types of codes applicable in any given application. The codes listed here will be good ones without aberrant undesirable properties or hidden problems, mainly because the efforts of others to understand coding has enabled such undesirable codes to be eliminated when searching for good codes.

Convolutional codes are studied for $\bar{b} < 1$ in Sections 10.1 and 10.2 before progressing to coset/trellis codes (which in fact are based on internal convolutional codes) for $\bar{b} \geq 1$ in Section 10.3. Section 10.2 also discusses code complexity, which is largely in the decoder, and tabulates many of the best-known convolutional codes and their essential parameters, as well as illustrating implementation. Sections 10.4 and 10.5 tabulate a number of multidimensional lattices and coset codes. The appendices delve into some more theoretical aspects of coding and lattices for the interested reader.

Code intricacies and structure is heavily studied and published. We refer the reader to the many fine textbooks in this area, in particular S. Wicker's *Error Control Systems for Digital Communication and Storage* (Prentice Hall, 1995)on block codes, R. Johhannesson and K. Ziogangirov's *Fundamentals of Convolutional Coding* (IEEE Press, 1998), and C. Schlegal and L. Perez' *Trellis Coding* (IEEE Press, 1997).

10.1 Convolutional Codes and Implementation

The theory and realization of convolutional codes makes extensive use of concepts in algebra, most specifically the theory of groups and finite fields. The very interested reader may want to read Appendix A. This section does not attempt rigor, but rather use of the basic concepts of convolutional codes. One needs only know binary modulo-2 arithmetic is closed under addition and multiplication, and basically convolutional codes deal with vectors of binary sequences generated by a sequential encoder. Appendix B describes methods for translation of convolutional encoders.

10.1.1 Notation and Terminology

Let F be the finite (binary or modulo-2) field with elements $\{0, 1\}$.[1] A sequence of bits, $\{a_m\}$, in F can be represented by its D transform

$$a(D) = \sum_{m=-\infty}^{\infty} a_m D^m \quad . \tag{10.1}$$

The sums in this section that deal with sequences in finite fields are presumed to be modulo- 2 sums. In the case of the finite field, the variable D must necessarily be construed as a placeholder, and has no relation to the Fourier transform of the sequence (i.e., $D \neq e^{-\jmath\omega T}$). For this reason, $a(D)$ uses a lower case a in the transform notation $a(D)$, rather than the upper case A that was conventionally used when dealing with sequences in the field of real numbers. Note such lower-case nomenclature in D-transforms has been tacitly presumed in Chapters 8 and 9 also.

$F[D]$ denotes the set of all polynomials (finite-length causal sequences) with coefficients in F,

$$F[D] \triangleq \left\{ a(D) \;\mid\; a(D) = \sum_{m=0}^{\nu} a_m D^m \;,\; a_m \in F,\; \nu \geq 0,\; \nu < \infty,\; \nu \in \mathcal{Z} \right\} \quad ; \tag{10.2}$$

where $\mathcal{Z}$ is the set of integers.[2] The set $F[D]$ can also be described as the set of all polynomials in D with finite Hamming weight. Let $F_r(D)$ be the field formed by taking ratios $a(D)/b(D)$ of any two polynomials in $F[D]$, with $b(D) \neq 0$ and $b_0 = 1$:

$$F_r(D) \triangleq \left\{ c(D) \;\mid\; c(D) = \frac{a(D)}{b(D)} \;,\; a(D) \in F[D],\; b(D) \neq 0,\; b(D) \in F[D] \text{with } b_0 = 1 \right\} \quad . \tag{10.3}$$

This set $F_r(D)$ is sometimes also called the **rational fractions of polynomials** in D.

> **Definition 10.1.1 (Delay of a Sequence)** *The* **delay**, *$del(a)$, of a nonzero sequence, $a(D)$, is the smallest time index, m, for which a_m is nonzero. The delay of the zero sequence $a(D) = 0$ is defined to be ∞. This is equivalent to the lowest power of D in $a(D)$.*

This chapter assumes the sequential encoder starts at some time, call it $k = 0$, and so all sequences of interest will have delay greater than or equal to 0. For example, the delay of the sequence $1 + D + D^2$ is 0, while the delay of the sequence $D^5 + D^{10}$ is 5.

> **Definition 10.1.2 (Degree of a Sequence)** *The* **degree**, *$deg(a)$, of a nonzero sequence, $a(D)$ is the largest time index, m, for which a_m is nonzero. The degree of the zero sequence $a(D) = 0$ is defined to be $-\infty$. This is equivalent to the highest power of D in $a(D)$.*

For example, the degree of the sequence $1 + D + D^2$ is 2, while the degree of the sequence $D^5 + D^{10}$ is 10.

[1] Finite fields are discussed briefly in Appendix A, along with the related concept of a ring, which is almost a field except that division is not well defined or implemented.

[2] The set $F[D]$ can be shown to be a **ring** (see Appendix A).

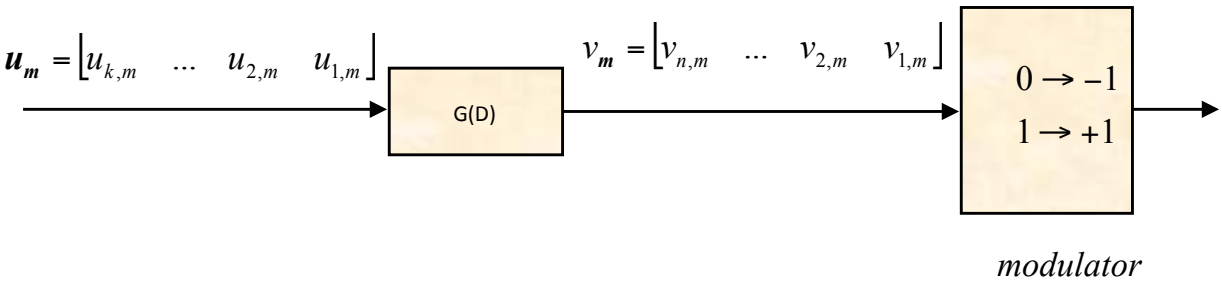


Figure 10.1: General convolutional encoder block diagram.

Definition 10.1.3 (Length of a Sequence) *The* **length**, $len(a)$, *of a nonzero sequence,* $a(D)$ *is defined by*

$$len(a) \triangleq deg(a) - del(a) + 1 \quad . \tag{10.4}$$

The length of the zero sequence $a(D) = 0$ *is defined to be* $len(0) \triangleq 0$.

For example, the length of the sequence $1 + D + D^2$ is 3, while the length of the sequence $D^5 + D^{10}$ is 6.

10.1.2 The Convolutional Code

The general convolutional encoder is illustrated in Figure 10.1. Coding theorists often use a time index of m instead of k, which is instead the number of input bits to a sequential encoder (b earlier in this text). This chapter then tries to maintain consistency with that significant literature on convolutional codes in particular with k input bits at each time m producing n output bits. As will become evident, there is no need to call the input message m because it will be represented by a vector of bits (so ambiguity of the use of m for a message index with the newly introduced interpretation of m as a time index will then not occur).

In convolutional codes, the k-dimensional binary vector sequence of input bits, $\boldsymbol{u}_m = [u_{k,m}\ u_{k-1,m}\ ...\ u_{1,m}]$ produces an n-dimensional vector sequence of output bits $\boldsymbol{v}_m = [v_{n,m}\ v_{n-1,m}\ ...\ v_{1,m}]$. The output vector may then be applied to a modulator that produces the N-dimensional channel input vector $\boldsymbol{x}_m$. The study of convolutional codes focuses on the relationship between $\boldsymbol{u}_m$ and $\boldsymbol{v}_m$. The relationship between $\boldsymbol{v}_m$ and $\boldsymbol{x}_m$ for the AWGN is simple binary translation to the bipolar (2 PAM) modulated waveform with amplitude $\pm\sqrt{\bar{\mathcal{E}}_{\boldsymbol{x}}}$. Note that the vector $\boldsymbol{u}(D)$ becomes a generalization of the message sequence $m(D)$, while $\boldsymbol{v}(D)$ is a vector essentially suited to transmission on a BSC while $\boldsymbol{x}(D)$ is suited to transmission on the AWGN.

The convolutional code is usually described by its **generator** $G(D)$:

Definition 10.1.4 (Convolutional Code) *The* **generator matrix**, $G(D)$ *of a convolutional code can be any* $k \times n$ *matrix with entries in* $F_r(D)$ *and rank* k. *The* **convolutional code**, $C(G)$, *is the set of all* n*-dimensional vector sequences* $\{\boldsymbol{v}(D)\}$ *that can be formed by premultiplying* $G(D)$ *by any* k*-dimensional vector sequence,* $\boldsymbol{u}(D)$, *whose component polynomials are causal and binary. That is, the code* $C(G)$ *is the set of* n*-dimensional sequences, whose components are in* $F_r(D)$, *that fall in the subspace spanned by the rows of* $G(D)$. *Mathematically,*

$$C(G) \triangleq \{\boldsymbol{v}(D) \mid \boldsymbol{v}(D) = \boldsymbol{u}(D) \cdot G(D)\ ,\ \boldsymbol{u}(D) \in F_r(D)\} \quad . \tag{10.5}$$

The **code rate** *for a convolutional coder is defined as* $r = k/n = \bar{b}$.

Two convolutional encoders that correspond to generators $G(D)$ and $G'(D)$ are said to be **equivalent** if they generate the same code, that is $C(G) = C(G')$.

Lemma 10.1.1 (Equivalence of Convolutional Codes) *Two convolutional codes are equivalent if and only if there exists an invertible $k \times k$ matrix, A, of polynomials in $F_r(D)$ such that $G'(D) = AG(D)$.*

The rows of $G(D)$ span a k-dimensional subspace of $F_r(D)$. Any $(n-k) \times n$ matrix that spans the orthogonal complement of the rows of $G(D)$ is known as a **parity matrix** for the convolutional code, that is:

Definition 10.1.5 (Parity Matrix) *An $(n-k) \times n$ matrix of rank $(n-k)$, $H(D)$, is known as the parity matrix for a code if for any codeword $\boldsymbol{v}(D)$, $\boldsymbol{v}(D)H^*(D) = 0$ (where $*$ denotes transpose in this case).*

An alternative definition of the convolutional code is then the set of n-dimensional sequences in $\{F_r(D)\}^n$ described by

$$C(G) = \{\boldsymbol{v}(D) \mid \boldsymbol{v}(D)H^*(D) = 0\} \quad . \tag{10.6}$$

Note $G(D)H^*(D) = 0$. When $(n-k) < k$, the parity matrix is a more compact description of the code. $H(D)$ also describes a convolutional code if it is used as a generator – this code is formally called the **dual code** to that formed by $G(D)$. All codewords in the dual code are obviously orthogonal to those in the original code, and the two codes together span n-space.

The separation between codewords in a convolutional code is described by their **Hamming distance**:

Definition 10.1.6 (Hamming Distance) *The* **Hamming Distance**, *$d_H(\boldsymbol{v}(D), \boldsymbol{v}'(D))$, between two sequences, $v(D)$ and $v'(D)$, is the number of bit positions in which they differ.*

Similarly the **Hamming weight** *$w_H(\boldsymbol{v}(D))$ is defined as the Hamming distance between the codeword and the zero sequence, $w_H(\boldsymbol{v}(D)) \stackrel{\Delta}{=} d_H(\boldsymbol{v}(D), 0)$. Equivalently, the Hamming weight is the number of "ones" in the codeword.*

Definition 10.1.7 (Systematic Encoder) *A* **systematic** *convolutional encoder has $v_{n-i}(D) = u_{k-i}(D)$ for $i = 0, ..., k-1$.*

Equivalently, the systematic encoder has the property that all the inputs are directly passed to the output, with the remaining $n-k$ output bits being reserved as "parity" bits. One can always ensure that a coder with this property satisfies our above definition by properly labeling the output bits.

A common measure of the complexity of implementation of the convolutional code is the **constraint length**. In order to define the constraint length precisely, one can transform any generator $G(D)$ whose entries are not in $F[D]$ into an equivalent generator with entries in $F[D]$ by multiplying every row by the least common multiple of the denominator polynomials in the original generator $G(D)$, call it $\phi(D)$ ($A = \phi(D)I_k$ in Lemma 10.1.1).

Definition 10.1.8 (Constraint Length) *The* **constraint length**, *ν, of a convolutional encoder $G(D)$ with entries in $F[D]$ is $\log_2$ of the number of states in the encoder; equivalently it is the number of D flip-flops (or delay elements) in the obvious realization (by means of a FIR filter).*

If ν_i $(i = 1, ..., k)$ is the degree or constraint length of the i^{th} row of $G(D)$ (that is the maximum degree of the n polynomials in the i^{th} row of $G(D)$), then in the obvious realization of $G(D)$

$$\nu = \sum_{i=1}^{k} \nu_i \quad . \tag{10.7}$$

The **complexity** of a convolutional code measures the complexity of implementation over all possible equivalent encoders:

Definition 10.1.9 (Minimal Encoder) *The* **complexity** *μ of a convolutional code, C, is the minimum constraint length over all equivalent encoders $\{G(D)\}$ such that $C = C(G)$. An encoder is said to be* **minimal** *if the complexity equals the constraint length, $\nu = \mu$.*

A minimal encoder with feedback has a realization with the number of delay elements equal to the complexity.

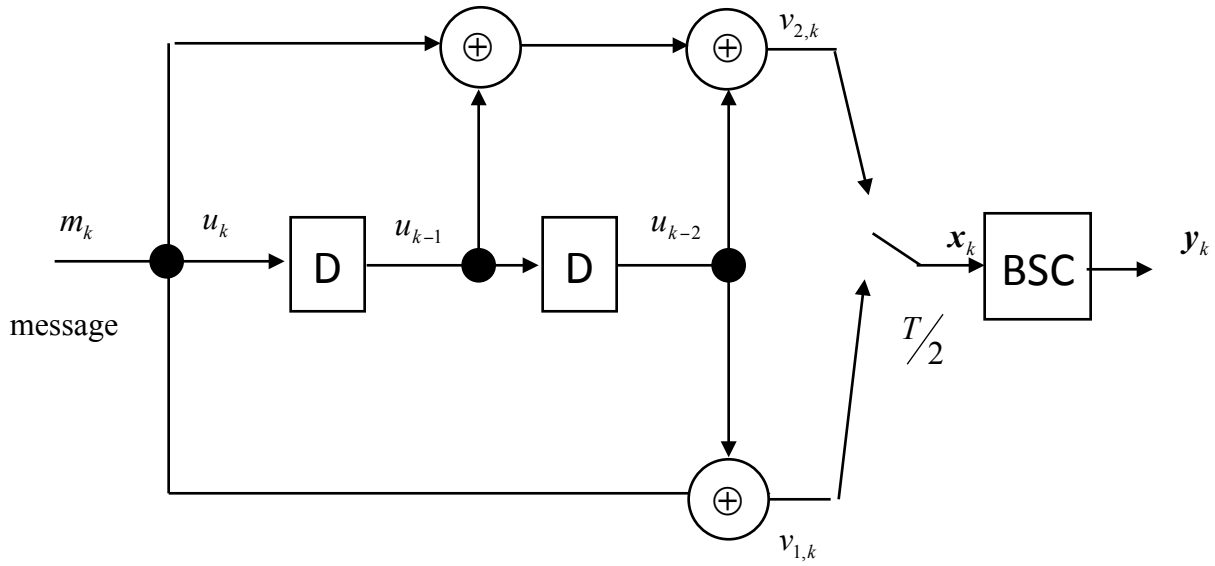


Figure 10.2: 4-state encoder example.

The following is a (brute-force) algorithm to construct a minimal encoder $G_{min}(D)$ for a given generator $G(D)$. This algorithm was given by Piret in his recent text on Convolutional Codes:

1. Construct a list of all possible n-vectors whose entries are polynomials of degree ν or less, where ν is the constraint length of $G(D)$. Sort this list in order of nondecreasing degree, so that the zeros vector is first on the list.

2. Delete from this list all n-vectors that are not codeword sequences in $C(G)$. (To test whether a given row of n polynomials is a codeword sequence, it suffices to test whether all $(k+1) \times (k+1)$ determinants vanish when the $k \times n$ matrix $G(D)$ is augmented by that row.)

3. Delete from this remaining list all codewords that are linear combinations of previous codewords.

The final list should have k codewords in it that can be taken as the rows of $G_{min}(D)$ and the constraint length must be μ because the above procedure selected those codewords with the smallest degrees. The initial list of codewords should have $2^{n(\nu+1)}$ codewords. A well-written program would combine steps 2 and 3 above and would stop on step 2 after generating k independent codeword sequences. This combined step would have to search somewhere between $2^{n\mu}$ and $2^{n(\mu+1)}$ codewords, because the minimum number of codewords searched would be $2^{n\mu}$ if at least one codeword of degree μ exists, and the maximum would be $2^{n(\mu+1)} - 1$ if no codewords of degree $\mu + 1$ exist. A more efficient method for generating the minimal encoder involves the concept of basic encoders (see Appendix A).

10.1.3 Examples

EXAMPLE 10.1.1 (4-state Optimal Rate 1/2 Code) The previous example of a convolutional encoder of Section 8.1 is repeated in Figure 10.2. There are $k = 1$ input bit and $n = 2$ output bits, so that the code rate is $\bar{b} = r = k/n = 1/2$. The input/output relations are

$$v_2(D) = (1 + D + D^2)u_1(D) \tag{10.8}$$
$$v_1(D) = (1 + D^2)u_1(D) \tag{10.9}$$

Thus,

$$G(D) = \left[1 + D + D^2 \quad 1 + D^2 \right] \quad . \tag{10.10}$$

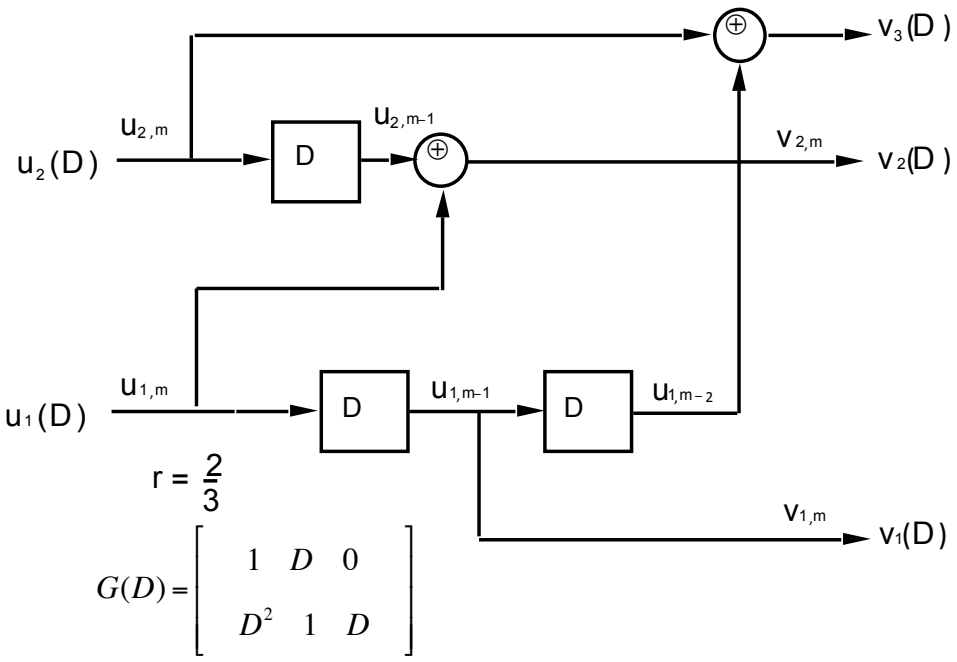


Figure 10.3: 8-state encoder example.

The constraint length is $\nu = 2$. The parity matrix is

$$H(D) = \begin{bmatrix} 1+D^2 & 1+D+D^2 \end{bmatrix} \quad . \tag{10.11}$$

The code generated by

$$G(D) = \begin{bmatrix} 1 & \frac{1+D^2}{1+D+D^2} \end{bmatrix} \quad , \tag{10.12}$$

has a systematic encoder; this code also has the same parity matrix, meaning the two codes are the same, even though the mappings from input bits to the same set of codewords may be different.

EXAMPLE 10.1.2 (8-state Ungerboeck Code) Another example appears in Figure 10.3. This encoder has $r = 2/3$, and

$$G(D) = \begin{bmatrix} 1 & D & 0 \\ D^2 & 1 & D \end{bmatrix} \quad . \tag{10.13}$$

$H(D)$ must satisfy $G(D)H^*(D) = 0$, or

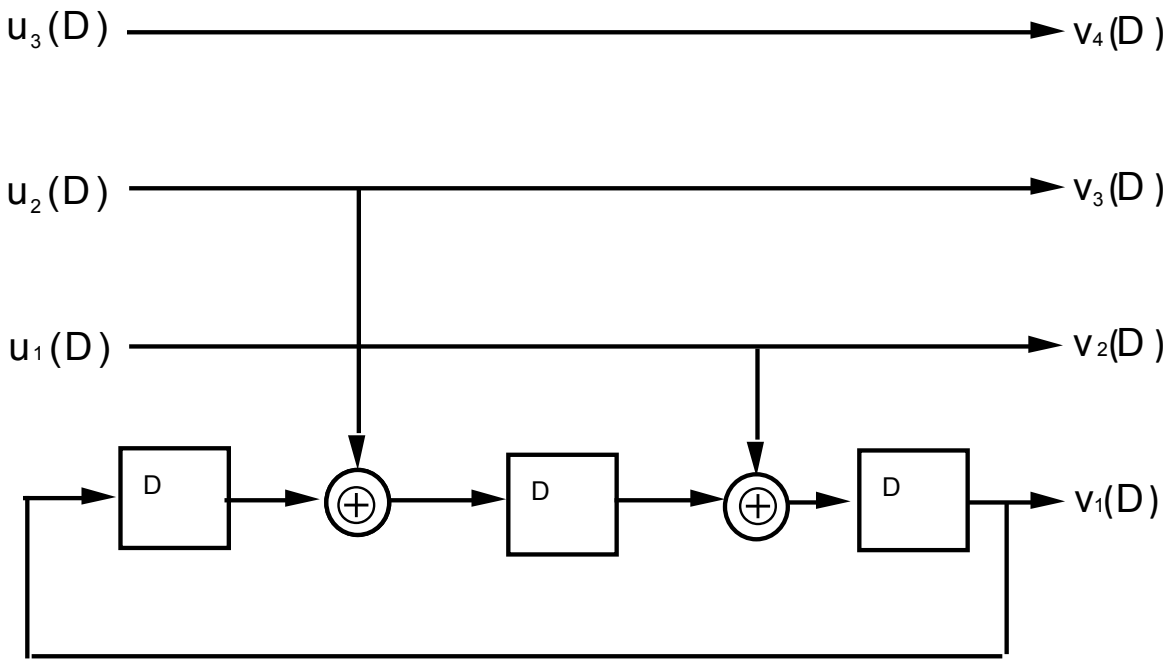
$$0 = h_3 + h_2 \cdot D \tag{10.14}$$

$$0 = h_3 \cdot D^2 + h_2 + D \cdot h_1 \tag{10.15}$$

and that $h_3 = D^2$, $h_2 = D$, and $h_1 = 1 + D^3$ is a solution. Thus,

$$H(D) = \begin{bmatrix} D^2 & D & 1+D^3 \end{bmatrix} \quad . \tag{10.16}$$

EXAMPLE 10.1.3 (8-state Ungerboeck Code with Feedback) A final example illustrating the use of feedback in the encoder appears in Figure 10.4. This encoder is rate $r = 3/4$



$$r = \frac{3}{4} \qquad G(D) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & D^2/(1+D^3) \\ 0 & 0 & 1 & D/(1+D^3) \end{bmatrix}$$

Figure 10.4: 8-State Encoder Example, with Feedback

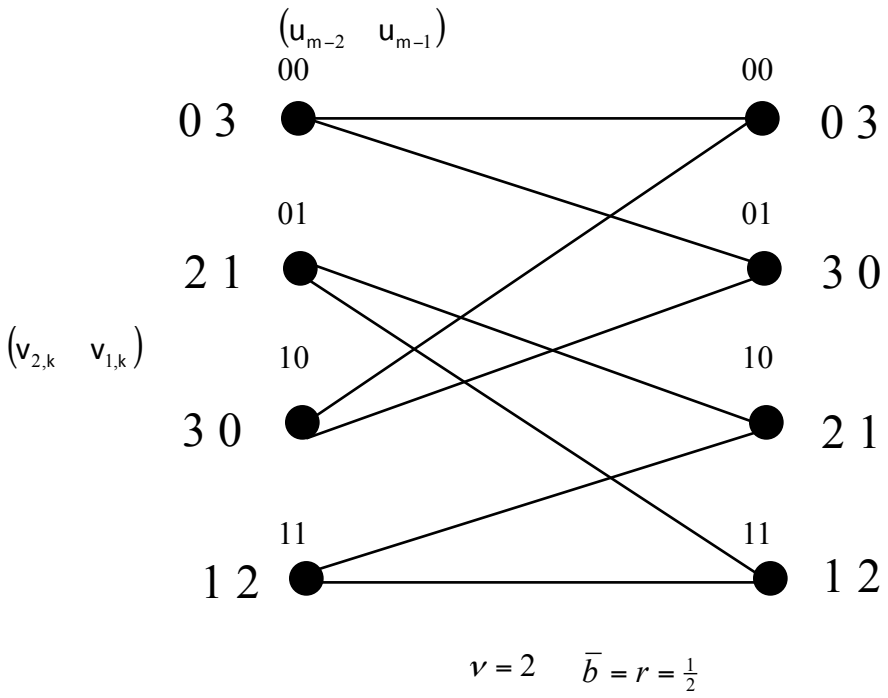


Figure 10.5: Trellis for 4-state encoder example.

and has

$$G(D) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & \frac{D^2}{1+D^3} \\ 0 & 0 & 1 & \frac{D}{1+D^3} \end{bmatrix} \quad . \tag{10.17}$$

For this 8-state feedback example, let us ignore the trivial pass-through bit $u_3 = v_4$. Then, G(D) becomes

$$G(D) = \begin{bmatrix} 1 & 0 & \frac{D^2}{1+D^3} \\ 0 & 1 & \frac{D}{1+D^3} \end{bmatrix} \quad , \tag{10.18}$$

for which

$$0 = h_3 + h_1 \cdot \frac{D^2}{1+D^3} \tag{10.19}$$

$$0 = h_2 + h_1 \cdot \frac{D}{1+D^3} \tag{10.20}$$

and that $h_3 = D^2$, $h_2 = D$, and $h_1 = 1 + D^3$ is a solution. Thus,

$$H(D) = \begin{bmatrix} D^2 & D & 1+D^3 \end{bmatrix} \quad , \tag{10.21}$$

and (ignoring $u_3 = v_4$) Example 10.1.2 and Fig.10.4 have the same code!

10.1.4 Trellis Diagrams for Convolutional Codes

Trellis diagrams can always be used to describe convolutional codes. The trellis diagram in Figure 10.5 is repeated from Section 8.1. The state in the diagram is represented by the ordered pair $(u_{1,m-2}\ ,\ u_{1,m-1})$. The outputs are denoted in mod-4 arithmetic as the mod-4 value for the vector of bits $\boldsymbol{v}$, with the most

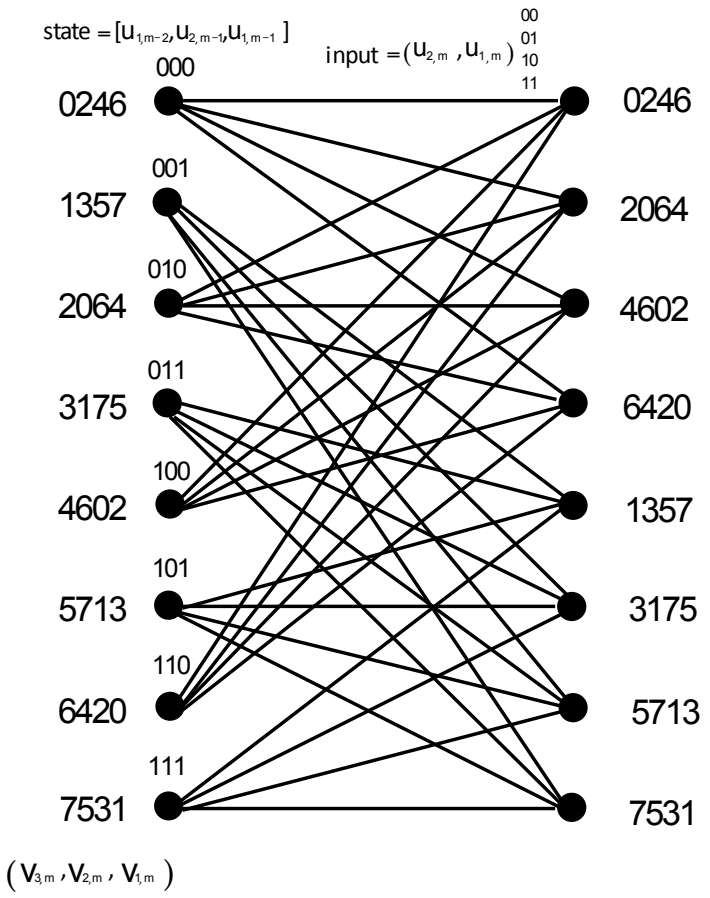


Figure 10.6: Trellis for 8-state encoder example of Figure 10.3.

significant bit corresponding to the left-most bit, $v_{n,m}$. On the left, the mod-4 value written closer to the trellis corresponds to the lowest branch emanating from the state it labels, while the mod-4 value written furthest from the trellis corresponds to the highest branch emanating from the state it labels. The labeling is vice-versa on the right.

The trellis diagram in Figure 10.6 is for the circuit in Figure 10.3. The state in the diagram is represented by the ordered triple $(u_{1,m-2}\ ,\ u_{2,m-1}\ ,\ u_{2,m-1})$. The outputs are denoted in octal arithmetic as the mod-8 value for the vector of bits $\boldsymbol{v}$, with the most significant bit again corresponding to the left-most bit, $v_{n,m}$.

10.1.5 Error probabilities and Performance

Section 9.2 generally investigated performance of codes designed with sequential encoders (or equivalently partial response systems where channel ISI was viewed as absorbed into a sequential encoder). For convolutional codes, they are typically used on either an AWGN or a BSC. For the AWGN, as always, the performance is given by

$$\bar{P}_e \approx \bar{N}_e Q\left(\frac{d_{min}}{2\sigma}\right) = \bar{N}_e Q\left(\sqrt{d_{free}\cdot \mathrm{SNR}}\right) \tag{10.22}$$

$$\bar{P}_b \approx \frac{N_b}{b} Q\left(\sqrt{d_{free}\cdot \mathrm{SNR}}\right) \quad . \tag{10.23}$$

For the BSC, the equivalent relations are

$$\bar{P}_e \approx \bar{N}_e \left[4p(1-p)\right]^{\lceil \frac{d}{2} \rceil} \tag{10.24}$$

$$\bar{P}_b \approx \frac{N_b}{b} \left[4p(1-p)\right]^{\lceil \frac{d}{2} \rceil} \quad . \tag{10.25}$$

10.2 Convolutional Coding Tables and Decoder Complexity

This section describes implementation, complexity, and enumerates tables of convolutional codes.

10.2.1 Implementations

There are two basic implementations of convolutional codes that find most use in practice: feedback free implementations and systematic implementations (possibly with feedback). This subsection illustrates how to convert a generator or a parity description of the convolutional code into these implementations. There are actually a large number of implementations of any particular code with only the exact mapping of input bit sequences to the same set of output codewords that differ between the implementations. When feedback is used it is always possible to determine a systematic implementation with $b = k$ of the bits directly passed to the output. Systematic implementations are not always possible without feedback.

Generally speaking the direct realization without feedback is often enumerated in tables of codes, as appear in Subsection 10.2.4. For instance the rate 1/2 code $G(D) = [1+D+D^2 \;\; 1+D^2]$ is an example of the code with maximum minimum-distance for rate 1/2 with 4 states. The direct realization is obvious and was repeated in Section 10.1, Figure 10.2. To convert this encoder to a systematic realization, a new description of the input may be written as

$$u'(D) = \frac{1}{1+D+D^2} \cdot u(D) \quad . \tag{10.26}$$

Clearly $u'(D)$ can take on all possible causal sequences, just as can $u(D)$ and the two inputs are simply a relabeling of the relationship of input sequences to output code sequences. Thus, the codewords are generated by

$$\boldsymbol{v}'(D) = u'(D)G(D) = u(D)\frac{1}{1+D+D^2}G(D) = u(D)\left[1 \;\; \frac{1+D^2}{1+D+D^2}\right] \quad . \tag{10.27}$$

The new matrix on the right above is a new generator for the same code (the same set of codewords). This new generator or encoder is systematic, but also has feedback. Another way to generate this same encoder is to realize that for any systematic encoder, one may write

$$G_{sys} = [I_k \;\; h(D)] \tag{10.28}$$

where $h(D)$ defines the parity matrix through

$$H_{sys} = [h^T(D) \;\; I_{n-k}] \tag{10.29}$$

and the superscript of T denotes transpose to prevent confusion with prime here. Thus for rate $\frac{n-1}{n}$ codes, the parity matrix can often be easily converted to this form by simply dividing all elements by the last. Then (10.28) defines the systematic realization. Thus for the 4-state example $h(D) = \frac{1+D^2}{1+D+D^2}$ and the circuit realization with two D memory elements and feedback appears in Figure 10.7.

More generally for a convolutional code, the systematic realization is found by finding the inverse of the first k columns of the matrix

$$G(D) = [G_{1:k}(D) \; G_{k+1:n}(D)] \tag{10.30}$$

and premultiplying $G(D)$ by $G_{1:k}^{-1}$ to get

$$G_{sys} = G_{1:k}^{-1}G = \left[I \; G_{1:k}^{-1}G_{k+1:n}\right] \quad , \tag{10.31}$$

where the argument of (D) has been dropped for notational simplification. The inverse is implemented with modulo two arithmetic and of course the first k columns must form an invertible matrix for this conversion to work. For the codes that this chapter lists in tables, such invertibility is guaranteed. Such encoders can be replaced by other encoders for the same code that are better according to procedures in

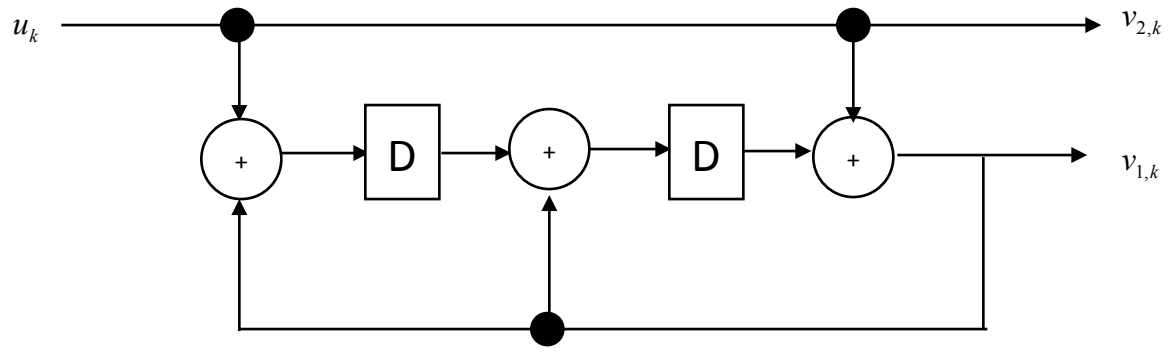


Figure 10.7: Systematic realization of 4-state convolutional code.

Appendix B. However, this text does not emphasize poor codes because most designers are well served to just use the codes in the tables. Premultiplication by the inverse does not change the codewords because this amounts to replacing rows of the generator by linear combinations of the rows of the matrix in an invertible manner. Since the set of codewords is all possible combinations of the rows of $G(D)$, this set is simply reindexed in terms of mappings to all possible input sequences by the matrix premultiplication.

As an example, the previous 8-state code in Example 10.3 has the generator

$$G(D) = \begin{bmatrix} 1 & D & 0 \\ D^2 & 1 & D \end{bmatrix} \tag{10.32}$$

and premultiplication by

$$G_{1:2}^{-1}(D) = \frac{1}{1+D^3} \cdot \begin{bmatrix} 1 & D \\ D^2 & 1 \end{bmatrix} \tag{10.33}$$

leaves

$$G_{1:2}^{-1}(D) \cdot G(D) = \begin{bmatrix} 1 & 0 & \frac{D^2}{1+D^3} \\ 0 & 1 & \frac{D}{1+D^3} \end{bmatrix} \tag{10.34}$$

and creates the systematic encoder.

Definition 10.2.1 (Catastrophic encoder) *A* **catastrophic encoder** *is one for which at least one codeword with finite Hamming weight corresponds to an input of infinite Hamming weight.*

Since the set of all possible codewords is also the set of all possible error events, this is the same as saying a finite number of decoding errors in a sequence could lead to an infinite number of input bit errors, clearly a catastrophic event. Catastrophic implementations are thus avoided. There are many tests for catastrophic codes, but one will suffice here:

Lemma 10.2.1 (Catastrophic Encoder Test) *An encoder is non-catastrophic if and only if the greatest common divisor of the determinants of all the $k \times k$ submatrices of $G(D)$ is a nonnegative power of D – that is D^δ for $\delta \geq 0$.*

The proof is in Appendix B.

A non-catastrophic encoder always exists for any code, and all the encoders listed in the tables of this section are non catastrophic. Trivially, a systematic encoder can never be catastrophic.

It is also possible to find a minimal encoder for any code, which will be non catastrophic, and then to convert that into a systematic encoder through the method above. That systematic encoder will also be minimal and thus non catastrophic. For this reason, most convolutional codes are implemented with systematic encoders and minimal complexity. However, the tables of this section allow both the feedback free and systematic realizations to be readily implemented.

10.2.2 Coding Gain for Convolutional Codes

In Volume I and also again here in Volume II, coding gain has been defined as a comparison of distance-to-energy for two systems, a coded system and an uncoded system. This measure of performance is actually the most fair - from Chapter 1, one recalls that fair comparisons of two systems should evaluate the 3 parameters $\bar{b}$, $\bar{\mathcal{E}}_{\boldsymbol{x}}$, and $\bar{P}_e$. Any two of the 3 can be fixed and the other compared. Convolutional coding gain essentially fixes $\bar{b}$ and $\bar{\mathcal{E}}_{\boldsymbol{x}}$ and looks at performance in terms of d_{min}, which is presumed directly related to $\bar{P}_e$.

Unfortunately in convolutional codes, $\bar{b}$ is not easily fixed and so the previous measure of coding gain is not then applicable. The designer thus then must look more carefully and understand his criteria and comparison. There are two methods for comparison: bandwidth expansion and data-rate reduction.

Method One - Bandwidth Expansion

In this criteria, data rate $R = b/T$, power $P = \mathcal{E}_{\boldsymbol{x}}/T$, and symbol period T are held fixed while bandwidth is allowed to increase (with n) and performance compared. This comparison is thus somewhat unfair in that it presumes more dimensions can be allocated (i.e., more bandwidth is available when more than likely it is not available) via bandwidth expansion. When so doing, the designer then simply compares d_{min} values for convolutionally coded systems with 2-level PAM at the same data rate. The convolutional code system thus has $1/\bar{b}$ more bandwidth than the uncoded system, and at fixed power and T, this means that $\bar{\mathcal{E}}_{\boldsymbol{x}}$ is reduced to $\bar{b}\bar{\mathcal{E}}_{\boldsymbol{x}}$. Thus the coded minimum distance is then $d_{free}\bar{b}\bar{\mathcal{E}}_{\boldsymbol{x}}$. The ratio of coded to uncoded distance is then just

$$\gamma = 10\log_{10}\left(\bar{b}d_{free}\right) \quad . \tag{10.35}$$

This quantity is then listed in tables as the coding gain for the convolutional code, even though it implies a bandwidth increase. The probability of codeword/sequence error is

$$\begin{aligned} P_e &\approx N_e \cdot Q\left[\sqrt{d_{free}\cdot\bar{b}\cdot \mathrm{SNR}_{uncoded}}\right] = N_e \cdot Q\left[\left(2\cdot\gamma\cdot\frac{\mathcal{E}_b}{\mathcal{N}_0}\right)^{1/2}\right] & (10.36)\\ &= N_e \cdot Q\left[\sqrt{d_{free}\cdot \mathrm{SNR}_{coded}}\right] \quad . & (10.37)\end{aligned}$$

Method Two - Data-Rate Reduction

This criteria is perhaps more applicable in practice, but the argument that leads to the same measure of coding gain is a little harder to follow. The designer fixes power P and bandwidth to W (positive frequencies only). For the coded system, the fixed bandwidth then leads to a fixed value for $1/(\bar{b}T)$, or equivalently a data rate reduction by a factor of $\bar{b}$ to $R_{code} = \bar{b}2W$ with respect to uncoded 2-level PAM, which would have $R_{code} = 2W$. The squared distance does increase to $d_{free}\bar{\mathcal{E}}_{\boldsymbol{x}}$ for the coded system, but could have used a lower-speed uncoded system with $1/\bar{b}$ more energy per dimension for 2-level PAM transmission. Thus the ratio of squared distance improvement is still $\bar{b}d_{free}$, the coding gain.

10.2.3 Binary Symmetric Channel Error Probability

For the binary symmetric channel, the simplest approximation to error probability is repeated here as

$$P_e \approx N_3\left[4p(1-p)\right]^{\left\lceil\frac{d_{free}}{2}\right\rceil} \quad . \tag{10.38}$$

A slightly more complicated expression from Section 9.2 is

$$P_e \le \sum_{d=d_{free}}^{\infty} a(d)\left[\sqrt{4p(1-p)}\right]^d \quad , \tag{10.39}$$

where $a(d)$ is the number of error events of weight d. Expressions for bit error probability also occur in Seciton 9.2.

2^ν	$g_{11}(D)$	$g_{12}(D)$	d_{free}	γ_f	(dB)	N_e	N_1	N_2	N_b	L_D
4	7	5	5	2.5	3.98	1	2	4	1	3
8	17	13	6	3	4.77	1	3	5	2	5
16	23	35	7	3.5	5.44	2	3	4	4	8
(GSM) 16	31	33	7	3.5	5.44	2	4	6	4	7
32	77	51	8	4	6.02	2	3	8	4	8
64	163	135	10	5	6.99	12	0	53	46	16
(802.11a) 64	155	117	10	5	6.99	11	0	38	36	16
(802.11b) 64	133	175	9	4.5	6.53	1	6	11	3	9
128	323	275	10	5	6.99	1	6	13	6	14
256	457	755	12	6	7.78	10	9	30	40	18
(IS-95) 256	657	435	12	6	7.78	11	0	50	33	16
512	1337	1475	12	6	7.78	1	8	8	2	11
1024	2457	2355	14	7	8.45	19	0	80	82	22
2048	6133	5745	14	7	8.45	1	10	25	4	19
4096	17663	11271	15	7.5	8.75	2	10	29	6	18
8192	26651	36477	16	8	9.0	5	15	21	26	28
16384	46253	77361	17	8.5	9.29	3	16	44	17	27
32768	114727	176121	18	9	9.54	5	15	45	26	37
65536	330747	207225	19	9.5	9.78	9	16	48	55	33
131072	507517	654315	20	10	10	6	31	58	30	27

Table 10.1: Rate 1/2 Maximum Free Distance Codes

Probabilty of error with coding gain and energy per bit expression

From section 10.1, the probability of symbol error is given by

$$\bar{P}_e \approx \bar{N}_e \cdot Q\left(\frac{d_{min}}{2\sigma}\right) = \bar{N}_e Q\left(\sqrt{d_{free} \cdot \mathrm{SNR}}\right) \tag{10.40}$$

Explicit inclusion of coding gain is tacit because the SNR reduces as $\bar{b}$. An alternate expression is to use energy per bit $\mathcal{E}_b \triangleq \frac{\mathcal{E}_{\boldsymbol{x}}}{b}$ so that

$$\bar{P}_e \approx \bar{N}_e \cdot Q\left(\sqrt{d_{free} \cdot \frac{b}{n} \cdot \frac{\mathcal{E}_b}{\sigma^2}}\right) = \bar{N}_e \cdot Q(2 \cdot \gamma \cdot \frac{\mathcal{E}_b}{\mathcal{N}_0}) \tag{10.41}$$

which includes the coding gain $\gamma = \bar{b} \cdot d_{free}$ directly in the formulae and views the quantity $\frac{\mathcal{E}_b}{\mathcal{N}_0}$ as fixed. Such a quantity is reasonably interpretted as fixed (ignoring bandwidth expansion) ONLY in the study of convolutional codes when $\bar{b} < 1$.

10.2.4 Tables of Convolutional Codes

This subsection lists several of the most popular convolutional codes in tabular format. These tables are based on a very exhaustive set found by R. Johannesson K. Sigangirov in their text *Fundamentals of Convolutional Coding* (IEEE Press, 1999), but in which we found many errors. Using Ginis dmin calculation program, they have been corrected. The generators are specified in octal with an entry of 155 for $g_{11}(D)$ in Table 10.1 for the 64-state code corresponding to $g_{11}(D) = D^6 + D^5 + D^3 + D^2 + 1$.

EXAMPLE 10.2.1 (Rate 1/2 example) The 8-state rate 1/2 code in the tables has a generator given by [17 13]. The generator polynomial is thus $G(D) = [D^3 + D^2 + D + 1 \;\; D^3 + D + 1]$. Both feedback-free and systematic-with-feedback encoders appear in Figure 10.8.

2^ν	$g_{11}(D)$	$g_{12}(D)$	$g_{13}(D)$	d_{free}	γ_f	(dB)	N_e	N_1	N_2	N_b	L_D
4	5	7	7	8	2.67	4.26	2	0	5	3	4
8	15	13	17	10	3.33	5.23	3	0	2	6	6
16	25	33	37	12	4	6.02	5	0	3	12	8
32	71	65	57	13	4.33	6.37	1	3	6	1	6
64	171	165	133	15	5	6.99	3	3	6	7	11
128	365	353	227	16	5.33	7.27	1	5	2	2	10
256	561	325	747	17	5.67	7.53	1	2	6	1	9
512	1735	1063	1257	20	6.67	8.24	7	0	19	19	15
1024	3645	2133	3347	21	7	8.45	4	1	4	12	16
2048	6531	5615	7523	22	7.33	8.65	3	0	9	8	15
4096	13471	15275	10637	24	8.00	9.03	2	8	10	6	17
8192	32671	27643	22617	26	8.67	9.38	7	0	23	19	20
16384	47371	51255	74263	27	9	9.54	6	4	6	22	24
32768	151711	167263	134337	28	9.33	9.70	1	6	5	2	17
65536	166255	321143	227277	28	10	10	1	9	20	4	22

Table 10.2: Rate 1/3 Maximum Free Distance Codes

2^ν	$g_{11}(D)$	$g_{12}(D)$	$g_{13}(D)$	$g_{14}(D)$	d_{free}	γ_f	(dB)	N_e	N_1	N_2	N_b	L_D
4	7	7	7	5	10	2.5	3.98	1	1	1	2	4
8	17	15	13	13	13	3.25	5.12	2	1	0	4	6
16	37	35	33	25	16	4	6.02	4	0	2	8	7
32	73	65	57	47	18	4.5	6.53	3	0	5	6	8
64	163	147	135	135	20	5	6.99	10	0	0	37	16
128	367	323	275	271	22	5.5	7.40	1	4	3	2	9
256	751	575	633	627	24	6.0	7.78	1	3	4	2	10
512	0671	1755	1353	1047	26	6.5	8.13	3	0	4	6	12
1024	3321	2365	3643	2277	28	7.0	8.45	4	0	5	9	16
2048	7221	7745	5223	6277	30	7.5	8.75	4	0	4	9	15
4096	15531	17435	05133	17627	32	8	9.03	4	3	6	13	17
8192	23551	25075	26713	37467	34	8.5	9.29	1	0	11	3	18
16384	66371	50575	56533	51447	37	9.25	9.66	3	5	6	7	19
32768	176151	123175	135233	156627	39	9.75	9.89	5	7	10	17	21
65536	247631	264335	235433	311727	41	10.25	10.1	3	7	7	7	20

Table 10.3: Rate 1/4 Maximum Free Distance Codes

2^ν	$h_3(D)$	$h_2(D)$	$h_1(D)$	d_{free}	γ_f	(dB)	N_e	N_1	N_2
8	17	15	13	4	2.667	4.26	1	5	24
16	05	23	27	5	3.33	5.22	7	23	59
32	53	51	65	6	4.00	6.02	9	19	80
64	121	113	137	6	4.0	6.02	1	17	47
128	271	257	265	7	4.67	6.69	6	26	105
256	563	601	475	8	5.33	7.27	8	40	157
512	1405	1631	1333	8	5.33	7.27	1	20	75
1024	1347	3641	3415	9	6	7.78	9	45	166
2048	5575	7377	4033	10	6.67	8.24	29	0	473
4096	14107	13125	11203	10	6.67	8.24	4	34	127
8192	29535	31637	27773	11	7.33	8.65	9	72	222
16384	66147	41265	57443	12	8	9.02	58	0	847
32768	100033	167575	155377	12	8	9.02	25	0	462
65536	353431	300007	267063	12	8	9.02	7	26	120

Table 10.4: Rate 2/3 Maximum Free Distance Codes

2^ν	$h_4(D)$	$h_3(D)$	$h_2(D)$	$h_1(D)$	d_{free}	γ_f	(dB)	N_e	N_1	N_2
32	45	71	63	51	5	3.75	5.74	9	47	218
64	151	121	177	111	6	4.5	6.53	30	112	640

Table 10.5: Rate 3/4 Maximum Free Distance Codes

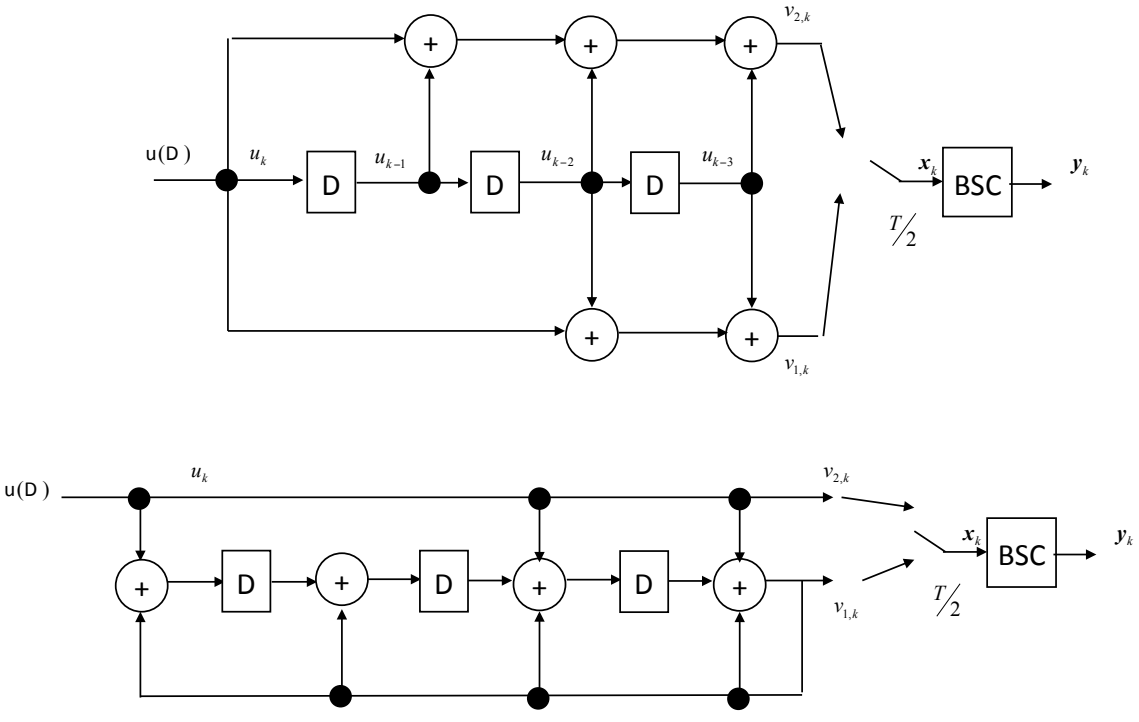


Figure 10.8: 8-State rate-1/2 equivalent encoders from Tables

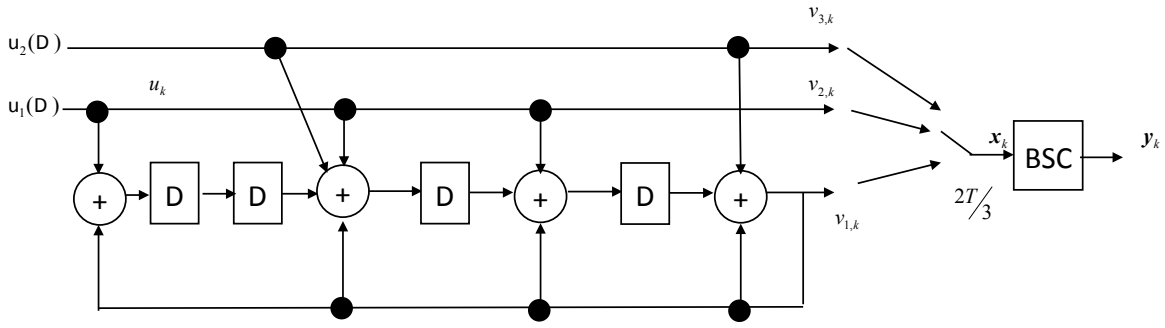


Figure 10.9: 16-State rate-2/3 equivalent encoders from Tables

EXAMPLE 10.2.2 (Rate 2/3 example) The 16-state rate 2/3 code in the tables has a parity matrix given by [05 23 27]. The parity polynomial is thus $H(D) = [D^2+1 \;\; D^4+D^2+D \;\; D^4+D^2+D+1]$. Equivalently, $H_{sys}(D) = [\frac{D^2+1}{D^4+D^2+D+1} \;\; \frac{D^4+D^2+D}{D^4+D^2+D+1} \;\; 1]$ A systematic generator is then

$$G(D) = \begin{bmatrix} 1 & 0 & \frac{D^2+1}{D^4+D^2+D+1} \\ 0 & 1 & \frac{D^4+D^2+D}{D^4+D^2+D+1} \end{bmatrix} \quad . \tag{10.42}$$

The systematic-with-feedback encoder appears in Figure 10.9.

10.2.5 Complexity

Code complexity is measured by the number of adds and compares that are executed per symbol period by the Viterbi ML decoder. This number is always

$$N_D = 2^\nu \left(2^k + 2^k - 1\right) \tag{10.43}$$

for convolutional codes. This is a good relative measure of complexity for comparisons, but not necessarily an accurate count of instructions or gates in an actual implementation.

10.2.6 Forcing a Known Ending State

For encoders without feedback, the input $\boldsymbol{u}(D)$ can be set to zero for ν symbol periods. Essentially, this forces a known return to the all zeros state. The encoder outputs for these ν symbol periods are sent through the channel and the decoder knows they correspond to zeros. Thus, a sequence decoder would then know to find the best path into only the all zeros state at the end of some packet of transmission. If the packet length is much larger than ν, then the extra symbols corresponding to no inputs constitute a small fraction of the channel bandwidth. In some sense, symbols at the end of the packet before the intentional "zero-stuffing" are then treated at least equally to symbols earlier within the packet.

Tail Biting

The state of a recursive (i.e., having feedback) encoder can also be forced to zero by what is called "tail byting." Tail byting is the finite-field equivalent of providing an input to a circuit that "zeros the poles." Figure 10.10 illustrates the basic operation. A switch is used in the feedback section. After a packet of L input symbols has been input to the encoder with the switch in the upper position, then the switch is lowered and the feedback elements are successively zeroed. The output of the feedback section is input to the filter (which because of the exclusive or operation zeros the input to the first delay element). This delay-line output value is also transmitted as ν additional input symbols. The state is again forced to zero so that a decoder can exploit this additional knowledge of the ending state for the packet of $L + \nu$ transmitted symbols.

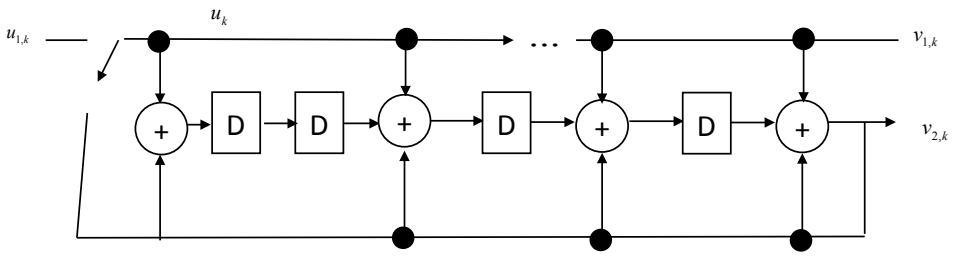


Figure 10.10: Illustration of Tail Biting.

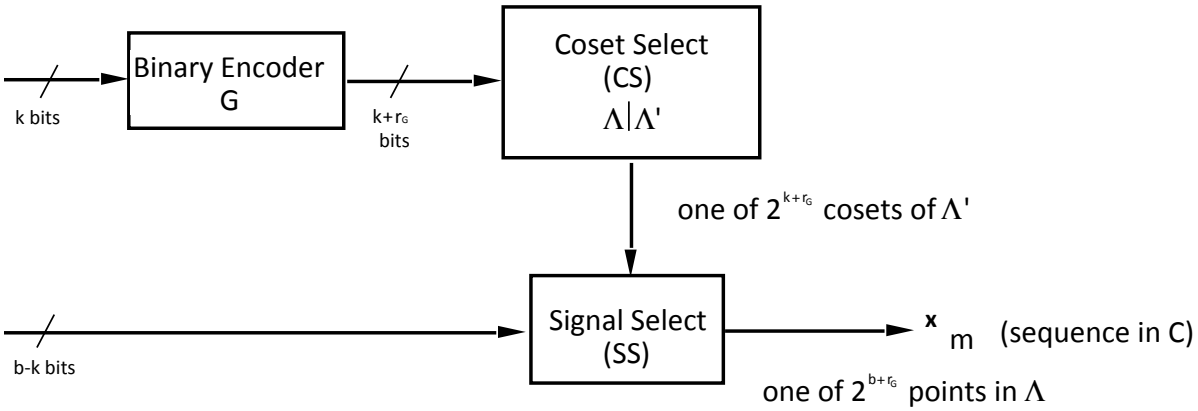


Figure 10.11: The coset-code encoder

10.3 Coset Codes, Lattices, and Partitions

The general coset-code encoder is shown in Figure 10.11. This encoder consists of 3 major components: the **binary encoder** (G), the **coset select** (CS), and the **signal select** (SS). The encoder output, $\boldsymbol{x}_m$, is an N-dimensional vector sequence of points; m is a symbol-time index. Each (N-dimensional) symbol of this sequence is chosen from an N-dimensional constellation. The sequences of $\boldsymbol{x}_m$ are the codewords $\boldsymbol{x}(D) = \sum_m \boldsymbol{x}_m D^m$. This signal constellation consists of 2^{b+r_G} signal points in some coset of an N-dimensional real lattice, Λ (for a definition of a lattice and its cosets, see Appendix C). The basic idea in designing a coset code is to select carefully N-dimensional sequences that are separated by a large minimum distance. The signal constellation contains 2^{k+r_G} subsets (**cosets**) of the constellation that each contain 2^{b-k} points. A good trellis code is designed by carefully selecting sequences of cosets that will lead to the desired increase in separation. The vector sequence $\boldsymbol{x}_m$ is converted by modulation to a continuous-time waveform according to the techniques discussed in Chapter 1.

At any time m, there are b input bits to the encoder, of which k are input to a conventional binary encoder (convolutional or block), which produces $n = k + r_G$ output bits, with r_G specifying the number of redundant bits produced by the encoder ($r_G = n - k$ in Sections 10.1 and 10.2). The quantity $\bar{r}_G \triangleq r_G/N$ is called the **normalized redundancy** of the convolutional encoder in the coset-code encoder. The quantity $\bar{k}_G \triangleq k/N$ is called the **informativity** of the convolutional encoder in the coset-code encoder. The $k + r_G$ output bits of the binary encoder specify one of 2^{k+r_G} disjoint subsets (or cosets) into which the signal constellation has been divided. This subset selection is performed by the coset select. The current output signal point $\boldsymbol{x}_m$ at time m is selected by the signal select from the 2^{b-k} remaining points in that coset that is currently selected by the coset select. The set of all possible sequences that can be generated by the coset encoder is called the **coset code** and is denoted by C.

In Figure 10.11, the sublattice Λ' is presumed to partition the lattice Λ, written $\Lambda|\Lambda'$, into 2^{k+r_G} cosets of Λ', where each coset has all of its points contained within the original lattice Λ. Such a partitioning always accompanies the specification of a coset code.

> **Definition 10.3.1 (Coset Partitioning $\Lambda|\Lambda'$)** *A* **coset partitioning** *is a partition of the Lattice Λ into $|\Lambda|\Lambda'|$ (called the "order" of the partition) cosets of a sublattice Λ' such that each point in the original lattice Λ is contained in one, and only one, coset of the sublattice Λ'.*

The coset select in Figure 10.11 then accepts the $k + r_G$ bits from G as input and outputs a corresponding index specifying one of the cosets of Λ' in the coset partitioning $\Lambda|\Lambda'$. There are 2^{b+r_G} constellation points chosen from the lattice Λ, and each is in one of the 2^{k+r_G} cosets. There are an equal number, 2^{b-k}, of constellation points in each coset. The signal select accepts $b - k$ uncoded input bits and selects which of the 2^{b-k} points in the coset of Λ' specified by the coset select that will be transmitted as the modulated vector $\boldsymbol{x}_m$.

If the encoder G is a convolutional encoder, then the set of all possible transmitted sequences $\{\boldsymbol{x}(D)\}$ is a **Trellis Code**, and if G is a block encoder, the set of N-dimensional vectors is a **Lattice Code**. So, both trellis codes and lattice codes are coset codes.

10.3.1 Gain of Coset Codes

There are several important concepts in evaluating the performance of a coset code, but the gain is initially the most important.

The fundamental gain will be taken in Volume II to always be with respect to a uncoded system, $\tilde{\boldsymbol{x}}$, that uses points on the N-dimensional integer lattice (denoted Z^N). It is easy (and often harmless, but not always) to forget the difference between $\mathcal{V}_{\boldsymbol{x}}$ and $\mathcal{V}^{2/N}(\Lambda)$. $\mathcal{V}_{\boldsymbol{x}}$ is the volume of the constellation and is equal to the number of points in the constellation times $\mathcal{V}^{2/N}(\Lambda_{\boldsymbol{x}})$. For coding gain calculations where the two compared systems have the same number of points, this difference is inconsequential. However, in trellis codes, the two are different because the coded system often has extra redundant points in its constellation. For the uncoded reference Z^N lattice, $\mathcal{V}(Z^N) = 1$ and $d_{min}(Z^N) = 1$, so that the fundamental gain of a coset code reduces to

$$\gamma_f = \frac{\frac{d^2_{min}(\boldsymbol{x})}{\mathcal{V}_{\boldsymbol{x}}^{2/N}}}{\frac{d^2_{min}(\tilde{\boldsymbol{x}})}{\mathcal{V}_{\tilde{\boldsymbol{x}}}{}^{2/N}}} = \frac{\frac{d^2_{min}(\boldsymbol{x})}{2^{2(\bar{b}+\bar{r}_G)}\mathcal{V}^{2/N}(\Lambda)}}{\frac{d^2_{min}(\tilde{\boldsymbol{x}})}{2^{2\bar{b}}\cdot\mathcal{V}^{2/N}(\Lambda)}} = \frac{\frac{d^2_{min}(C)}{2^{2\bar{r}_G}\cdot\mathcal{V}^{2/N}(\Lambda)}}{\frac{1}{1}} = \frac{d^2_{min}(C)}{\mathcal{V}(\Lambda)^{2/N}2^{2\bar{r}_G}} \tag{10.44}$$

The quantity $\bar{r}_G = \frac{r_G}{N}$ is the normalized redundancy of the encoder G. The gain γ_f in dB is

$$\begin{aligned}\gamma_f &= 10\cdot\log_{10}\left(\frac{d^2_{min}(C)}{\mathcal{V}(\Lambda)^{2/N}2^{2\bar{r}_G}}\right) &(10.45)\\ &= 20\log_{10}\left(\frac{d_{min}(C)}{\mathcal{V}(\Lambda)^{1/N}2^{\bar{r}_G}}\right) &(10.46)\end{aligned}$$

The redundancy of the overall coset code requires the concept of the redundancy of the original lattice Λ.

Definition 10.3.2 (Redundancy of a Lattice) *The* **redundancy of a lattice** *is defined by r_Λ such that*

$$\mathcal{V}(\Lambda) = 2^{r_\Lambda} = 2^{N\bar{r}_\Lambda} \quad , \tag{10.47}$$

or $r_\Lambda = \log_2(\mathcal{V}(\Lambda))$ bits per symbol. The quantity $\bar{r}_\Lambda = r_\Lambda/N$ is called the **normalized redundancy** *of the lattice, which is measured in bits/dimension.*

Then, the fundamental coding gain of a coset code is

$$\gamma_f = \frac{d^2_{min}(C)}{2^{2(\bar{r}_G+\bar{r}_\Lambda)}} = \frac{d^2_{min}(C)}{2^{2\bar{r}_C}} \tag{10.48}$$

where

$$\bar{r}_C = \bar{r}_G + \bar{r}_\Lambda \quad . \tag{10.49}$$

Equation (10.48) is often used as a measure of performance in evaluating the performance of a given trellis code. Good coset codes typically have 3 dB $\leq \gamma_f \leq$ 6 dB.

Shaping gain is also important in evaluating trellis codes, but is really a function of the shape of the constellation used rather than the spacing of sequences of lattice points. The shaping gain is defined as (again comparing against a zero-mean translate of Z^N lattice)

$$\gamma_s = \frac{\mathcal{V}^{2/N}(\Lambda)\cdot 2^{2\bar{r}_G}}{\bar{\mathcal{E}}(\Lambda)} / \frac{1}{(2^{2\bar{b}}-1)/12} = \frac{2^{2\bar{r}_C}}{12\bar{\mathcal{E}}}(2^{2\bar{b}}-1) \quad , \tag{10.50}$$

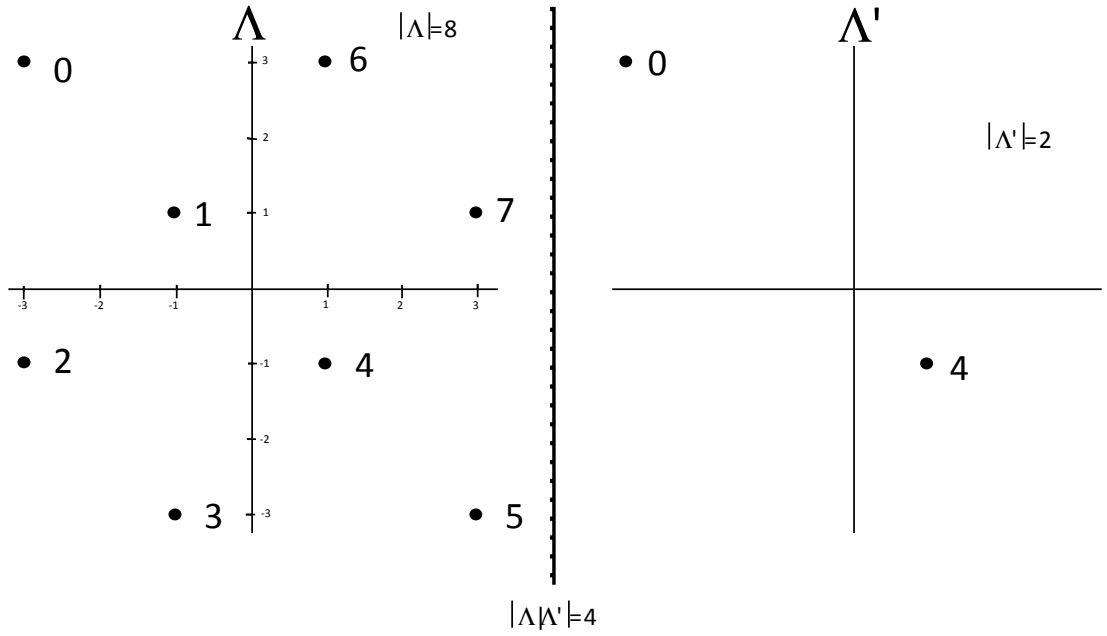


Figure 10.12: Partitioning (a coset of) the D_2 Lattice

where $\bar{\mathcal{E}}(C)$ is the energy per dimension required for the coset code. Using the so-called "continuous approximation" (which holds accurately for $\bar{b} \geq 3$), the shaping gain is often approximated by

$$\gamma_s \approx \frac{2^{2\bar{r}_C} \cdot 2^{2\bar{b}}}{12\bar{\mathcal{E}}(\Lambda)} \quad . \tag{10.51}$$

This can also be written in dB as

$$\begin{aligned} \gamma_s &\approx 10\log_{10}\left(\frac{\mathcal{V}^{2/N}(\Lambda)2^{2(\bar{b}+\bar{r}_G)}}{12\bar{\mathcal{E}}(\Lambda)}\right) &(10.52)\\ &= 10\log_{10}\left(\frac{2^{2(\bar{b}+\bar{r}_C)}}{12\bar{\mathcal{E}}(\Lambda)}\right) &(10.53)\end{aligned}$$

This shaping gain has a theoretical maximum of 1.53dB - the best known shaping methods achieve about 1.1dB (see Section 10.8).

EXAMPLE 10.3.1 (Ungerboeck's Rate 1/2 3 dB Trellis Code) In Figure 10.12, the 8AMPM (or 8CR) constellation is subset of a (possibly scaled and translated) version of what is known as the $\Lambda = D_2$ Lattice that contains $|\Lambda| = 8$ points. The average energy per symbol for this system is $\mathcal{E} = 10$ or $\bar{\mathcal{E}} = 5$. The (translated) sublattice Λ' has a coset, Λ_0 that contains $|\Lambda_0| = 2$ points, so that there are $|\Lambda|\Lambda'| = 4$ cosets of Λ' in Λ; they are $\Lambda_0 = \{0, 4\}$, $\Lambda_1 = \{1, 5\}$, $\Lambda_2 = \{2, 6\}$, and $\Lambda_3 = \{3, 7\}$. These 4 cosets are selected by the two bit output $\boldsymbol{v}(D)$ of a rate 1/2 convolutional encoder with generator:

$$G(D) = \begin{bmatrix} 1 + D^2 & D \end{bmatrix} \quad . \tag{10.54}$$

The corresponding trellis and trellis encoder are shown in Figures 10.13 and 10.14, respectively. The convolutional encoder, G, has rate $r = 1/2$, but the overall coset-code has $\bar{b} = 2$

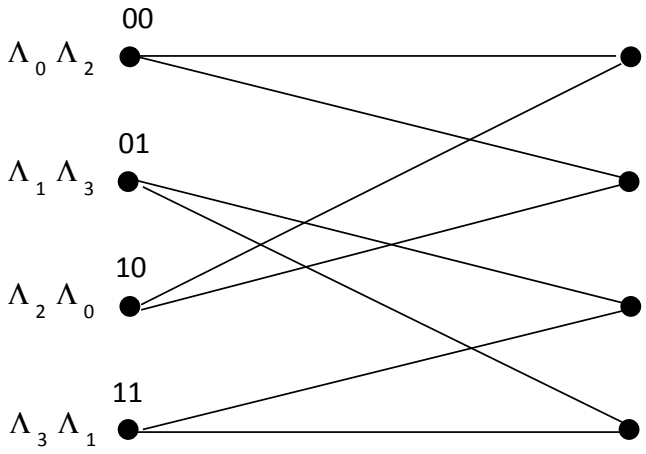


Figure 10.13: Trellis for 4-state rate 1/2 Ungerboeck Code

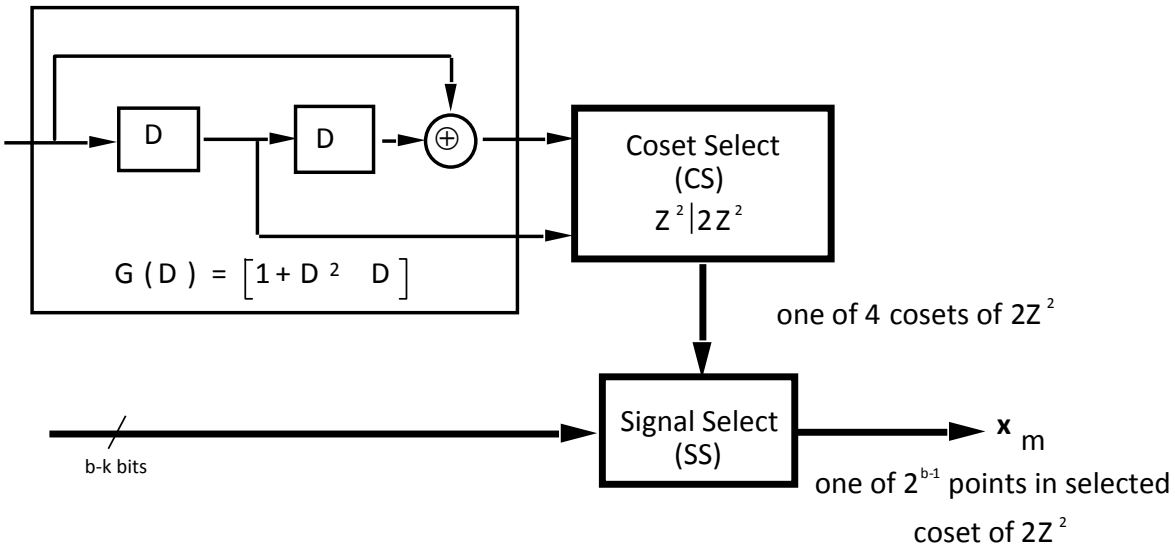


Figure 10.14: Encoder for 4-state rate 1/2 Ungerboeck Code

bits/2dimensions = 1 bit/dimension. The input bit $u_{1,k}$ passes through G, where the two output bits select the index of the desired coset Λ_0, Λ_1, Λ_2, or Λ_3. The minimum distance between the points in any of these cosets is $d_{min}(\Lambda') = 2d_{min}(\Lambda) = 4\sqrt{2}$. This distance is also the distance between the parallel transitions that are tacit in Figure 10.13. Figure 10.13 inherently illustrates that any two (non-parallel) paths that start and terminate in the same pair of states, must have a distance that is $d' = \sqrt{16+8+16}$, which is always greater than $4\sqrt{2}$, so that the parallel transition distance is the minimum distance for this code. This parallel-transition distance (normalized to square-root energy and ignoring constellation shape effects) is $\sqrt{2}$ better than the distance corresponding to no extra bit or just transmitting (uncoded) 4 QAM. More precisely, the coding gain is

$$\gamma = \frac{\left(d^2_{min}/\mathcal{E}_{\boldsymbol{x}_p}\right)_{\text{coded}}}{\left(d^2_{min}/\mathcal{E}_{\boldsymbol{x}_p}\right)_{\text{uncoded}}} \tag{10.55}$$

This expression evaluates to (for comparison against 4 QAM, which is also Z^N and exact for γ_f and γ_s calculations)

$$\gamma = \frac{\frac{16\cdot 2}{10}}{\frac{1}{1/2}} = 1.6 = 2 \text{ dB} \quad . \tag{10.56}$$

The fundamental coding gain is (realizing that $\bar{r}_C = \bar{r}_\Lambda + \bar{r}_G = 1.5 + .5 = 2$)

$$\gamma_f = \left(\frac{d^2_{min}}{2^{2\bar{r}_C}}\right) = \frac{32}{2^{2\cdot 2}} = 2 \text{ (3 dB)} \quad . \tag{10.57}$$

Thus, the shaping gain is then $\gamma_s = \gamma - \gamma_f = 2 - 3 = -1$ dB, which verifies according to

$$\gamma_s = \frac{2^{2\cdot 2}}{12 \cdot 5}\left(2^2 - 1\right) = \frac{4}{5} = \text{ -1 dB} . \tag{10.58}$$

This coset code of Figure 10.14 can be extended easily to the case where $b = 3$, by using the constellation in Figure 10.15. In this case, $\bar{r}_C = .5 = \bar{r}_\Lambda + \bar{r}_G = 0 + .5$, $\bar{b} = 1.5$, and $\bar{\mathcal{E}} = 1.25$. This constellation contains 16 points from the scaled and translated Z^2 Lattice. The sublattice and its 4 cosets are illustrated by the labels 0,1,2, and 3 in Figure 10.15. This coset code uses a circuit almost identical to that in Figure 10.14, except that two bits enter the Signal Select to specify which of the 4 points in the selected coset will be the modulated signal.

The minimum distance of the coded signal is still twice the distance between points in Λ. The fundamental coding gain is

$$\gamma_f = \frac{4}{2^{2\cdot .5}} = 2 \text{ (3 dB)}. \tag{10.59}$$

The fundamental gain γ_f will remain constant at 3 dB if b further increases in the same manner; however the shaping gain γ_s will vary slightly. The shaping gain in this case is

$$\gamma_s = \frac{2^{2\cdot .5}}{12 \cdot (5/4)}\left(2^3 - 1\right) = \frac{14}{15} = \text{ (-.3 dB)}. \tag{10.60}$$

The coding gain of this code with respect to 8 CR is (16/5)/(8/5)= 3dB. This code class is known as Ungerboeck's 4-state Rate 1/2 Trellis Code, and is discussed further in Section 10.4.

10.3.2 Mapping By Set Partitioning

The example of the last section suggests that the basic partitioning of the signal constellation $\Lambda|\Lambda'$ can be extended to larger values of b. The general determination of Λ', given Λ, is called **mapping-by-set-partitioning.** Mapping-by-set-partitioning is instrumental to the development and understanding of coset codes.

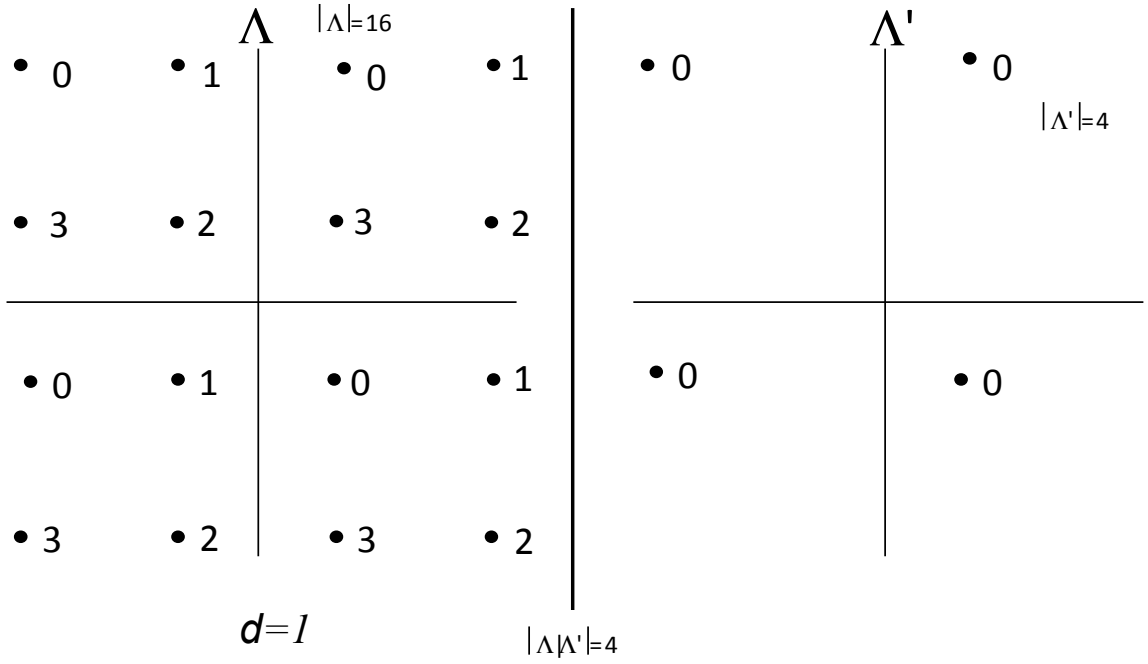


Figure 10.15: Partitioning 16 QAM

Partitioning of the Integer Lattice

Example 10.3.1 partitioned the 16QAM constellation used to transmit 3 bits of information per symbol. The basic idea is to divide or to partition the original constellation into two equal size parts, which both have 3 dB more distance between points than the original lattice. Figure 10.16 repeats the partitioning of 16 QAM each of the partitions to generate 4 sublattices with distance 6dB better than the original Lattice.

The cosets are labeled according to an **Ungerboeck Labeling**:

> **Definition 10.3.3 (Ungerboeck Labeling in two dimensions)** *An Ungerboeck labeling for a Coset Code C uses the LSB, v_0, of the encoder G output to specify which of the first 2 partitions (B0, $v_0 = 0$ or B1 $v_0 = 1$) contains the selected coset of the sublattice Λ', and then uses v_1 to specify which of the next level partitions (C0,C2,C1,C3) contains the selected coset of the sublattice, and finally when necessary, v_2 is used to select which of the 3rd level partitions is the selected coset of the sublattice. This last level contains 8 cosets, D0,D4,D2,D6,D1,D5,D3, and D7.*

The remaining bits, v_{k+r}, ..., v_{b+r-1} are used to select the point within the selected coset. The partitioning process can be continued for larger constellations, but is not of practical use for two-dimensional codes.

In practice, mapping by set partitioning is often used for $N = 1$, $N = 2$, $N = 4$, and $N = 8$. One-dimensional partitioning halves a PAM constellation into sets of "every other point," realizing a 6dB increase in intra-partition distance for each such halving. In 4 and 8 dimensions, which will be considered later, the distance increase is (on the average) 1.5 dB per partition and .75 dB per partition, respectively.

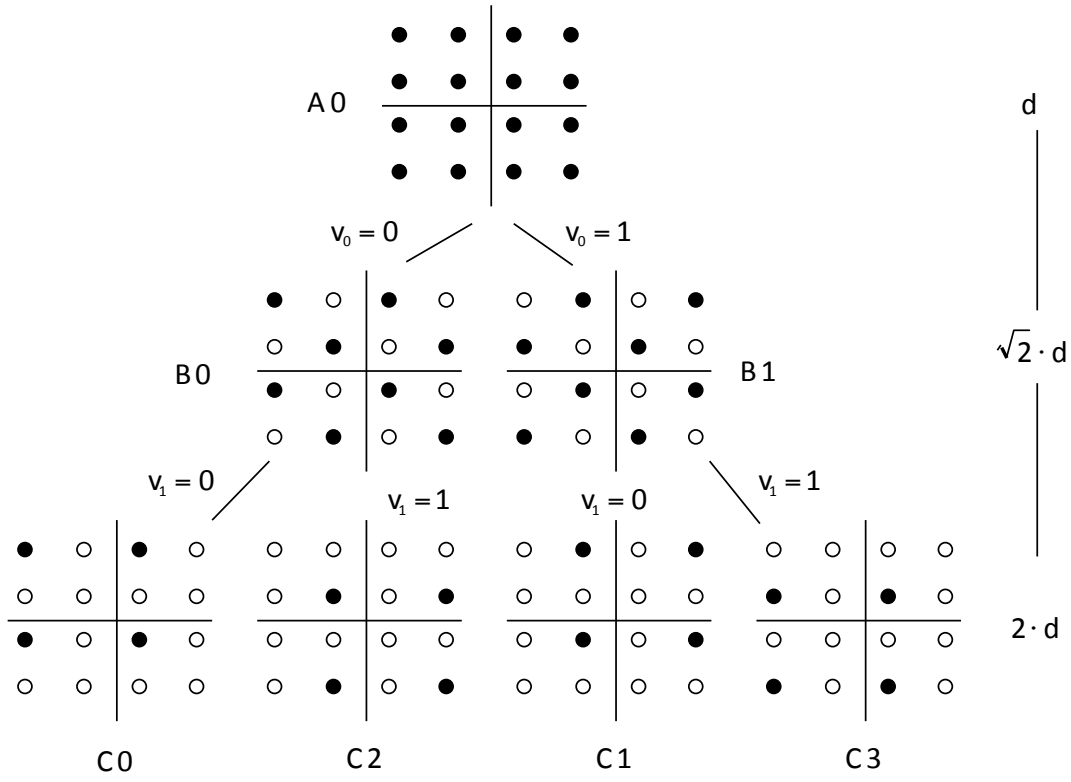


Figure 10.16: Illustration of Mapping by Set Partitioning for 16QAM Constellation

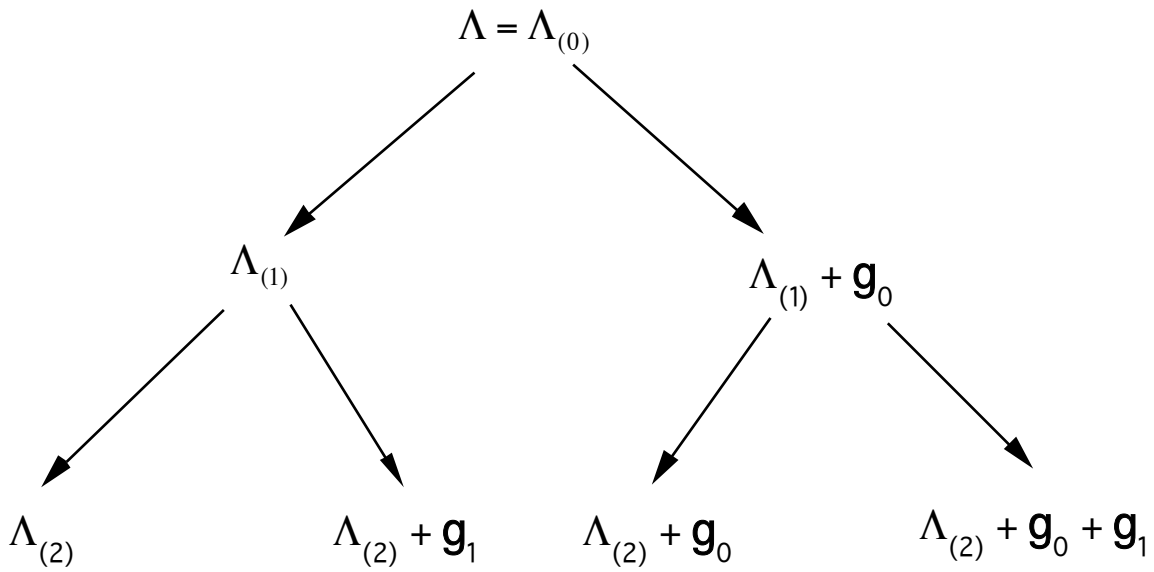


Figure 10.17: Partition tree.

Partition Trees and Towers

Forney's **Partition trees** and **Towers** are alternative more mathematical descriptions of mapping by set partitioning and the Ungerboeck Labeling process.

The partition tree is shown in Figure 10.17. Each bit of the convolutional encoder (G) output is used to delineate one of two cosets of the parent lattice at each stage in the tree. The constant vector that is added to one of the partitions to get the other is called a **coset leader**, and is mathematically denoted as $\boldsymbol{g}_i$. The specification of a vector point that represents any coset in the i^{th} stage of the tree is

$$\boldsymbol{x} = \Lambda' + \sum_{i=0}^{k+r_G-1} v_i \boldsymbol{g}_i = \Lambda' + \boldsymbol{v} \begin{bmatrix} \boldsymbol{g}_{k+r_G-1} \\ \vdots \\ \boldsymbol{g}_1 \\ \boldsymbol{g}_0 \end{bmatrix} = \Lambda' + \boldsymbol{v}\boldsymbol{G} \quad , \tag{10.61}$$

where $\boldsymbol{G}$ is a "generator" for the coset code. Usually in practice, a constant offset $\boldsymbol{a}$ is added to all $\boldsymbol{x}$ so that 0 (a member of all lattices) is not in the constellation, and the constellation has zero mean. The final $b-k$ bits will specify an offset from $\boldsymbol{x}$ that will generate our final modulation symbol vector. The set of all possible binary combinations of the vectors $\boldsymbol{g}_i$ is often denoted

$$[\Lambda|\Lambda'] \triangleq \{\boldsymbol{v}\boldsymbol{G}\} \tag{10.62}$$

Thus,

$$\Lambda = \Lambda' + [\Lambda|\Lambda'] \quad , \tag{10.63}$$

symbolically, to abbreviate that every point in Λ can be (uniquely) decomposed as the sum of a point in Λ' and one of the "coset leaders" in the set $[\Lambda|\Lambda']$, or recursively

$$\Lambda_{(i)} = \Lambda_{(i+1)} + \left[\Lambda_{(i)}|\Lambda_{(i+1)}\right] \quad . \tag{10.64}$$

There is thus a **chain rule**

$$\Lambda_{(0)} = \Lambda_{(2)} + \left[\Lambda_{(1)}|\Lambda_{(2)}\right] + \left[\Lambda_{(0)}|\Lambda_{(1)}\right] \tag{10.65}$$

or

$$\left[\Lambda_{(0)}|\Lambda_{(2)}\right] = \left[\Lambda_{(0)}|\Lambda_{(1)}\right] + \left[\Lambda_{(1)}|\Lambda_{(2)}\right] \tag{10.66}$$

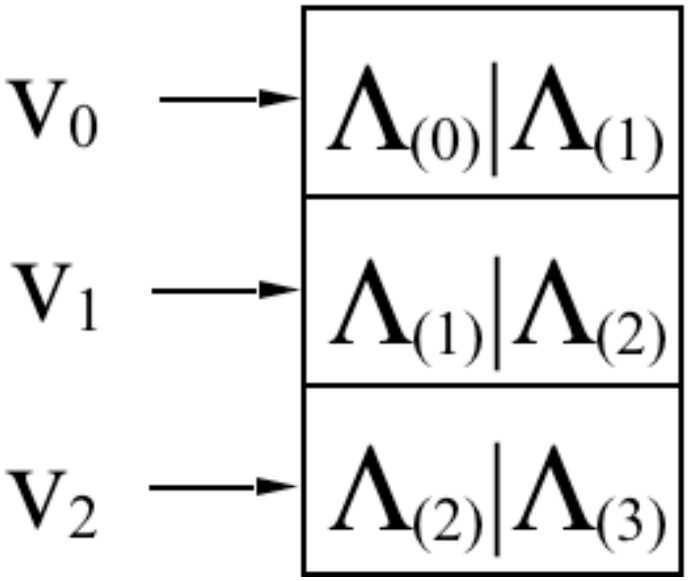


Figure 10.18: Partition tower.

where $\left[\Lambda_{(i)}|\Lambda_{(i+1)}\right] = \{0,\ \boldsymbol{g}_i\}$ and $\left[\Lambda_{(0)}|\Lambda_{(2)}\right] = \{0,\ \boldsymbol{g}_0,\ \boldsymbol{g}_1,\ \boldsymbol{g}_0 + \boldsymbol{g}_1\}$. This concept is probably most compactly described by the partition tower of Forney, which is illustrated in Figure 10.18.

The partition chain is also compactly denoted by

$$\Lambda_{(0)}\left|\Lambda_{(1)}\right|\Lambda_{(2)}\left|\Lambda_{(3)}\right|.... \tag{10.67}$$

EXAMPLE 10.3.2 (Two-dimensional integer lattice partitioning) The initial lattice $\Lambda_{(0)}$ is Z^2. Partitioning selects "every other" point from this lattice to form the lattice D_2, which has only those points in Z^2 that have even squared norms (see Figures 10.16 and 10.12). Mathematically, this partitioning can be written by using the rotation matrix

$$R_2 \triangleq \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \tag{10.68}$$

as

$$D_2 = R_2 Z^2 \quad , \tag{10.69}$$

where the multiplication in (10.69) symbolically denotes taking each point in Z^2 and multiplying by the rotation matrix R_2 to create a new set of points that will also be a lattice. D_2 is also a sublattice of Z^2 and therefore partitions Z^2 into two subsets. Thus in partitioning notation:

$$Z^2\,|D_2 \quad . \tag{10.70}$$

The row vector that can be added to D_2 to get the other coset is $\boldsymbol{g}_0 = [0\ 1]$. So, $Z^2\,|D_2$ with coset leaders $\left[Z^2\,|D_2\right] = \{[0\ 0],\ [0\ 1]\}$. D_2 decomposes into two sublattices by again multiplying by the rotation operator R_2

$$2Z^2 = R_2 D_2 \quad . \tag{10.71}$$

Points in $2Z^2$ have squared norms that are multiples of 4. Then, $D_2\left|2Z^2\right.$ with $\left[D_2\left|2Z^2\right.\right] = \{[0\ 0],\ [1\ 1]\}$. Thus, $\left[Z^2\left|2Z^2\right.\right] = \{[0\ 0],\ [0\ 1],\ [1\ 0]\ [1\ 1]\}$. The generator for a coset code with convolutional codewords $[v_1(D)\ v_0(D)]$ as input is then

$$\boldsymbol{G} = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \quad . \tag{10.72}$$

Partitioning continues by successive multiplication by R_2: $Z^2\,|D_2|\,2Z^2\,|2D_2|\,4Z^2....$

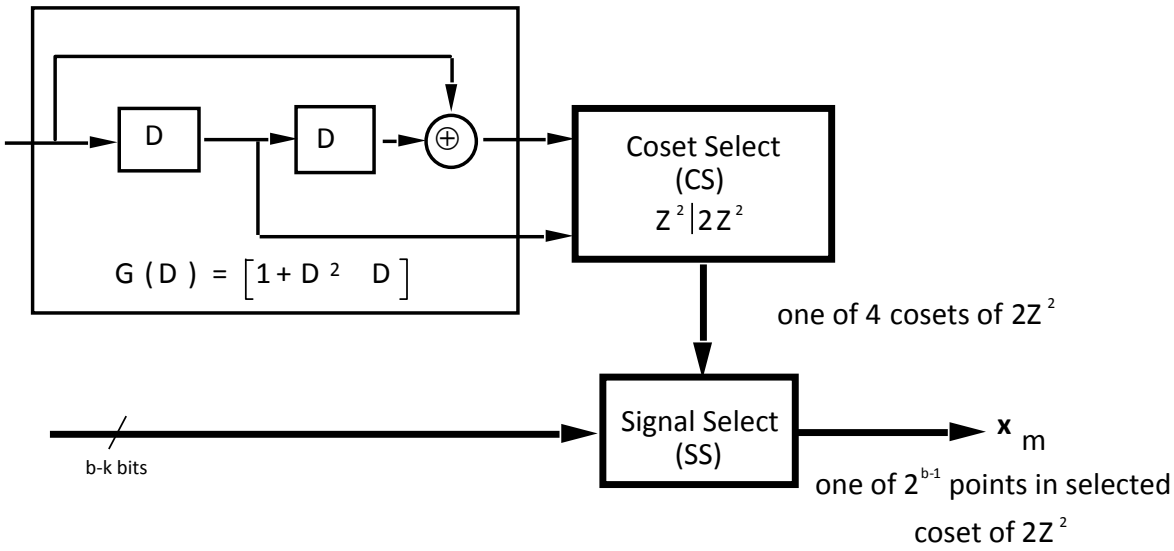


Figure 10.19: Ungerboeck's 4-state rate 1/2 Trellis Code

10.4 One- and Two-dimensional Trellis Codes

Ungerboeck's two most famous codes are in wide use and used for illustrative purposes in this section.

10.4.1 Rate 1/2 Code

The earlier 4-state rate 1/2 code discussed in Section 10.3 can be generalized as shown in Figure 10.19. The encoder matrix $G = [1 + D^2 \; D]$ can be equivalently written as a systematic encoder

$$G_{sys} = \left[1 \; \frac{D}{1+D^2}\right] \quad , \tag{10.73}$$

which is shown in Figure 10.19. The parity matrix H appears in Figure 10.19 because most trellis codes are more compactly expressed in terms of H. In this code $r_G = 1$ and 2^{b+1} points are taken from the Lattice coset $Z^2 + \left[\frac{1}{2} \; \frac{1}{2}\right]$ to form the constellation Λ, while the sublattice that is used to partition Λ is $\Lambda' = 2Z^2$, upon which there are 2^{b-1} constellation points, and there are 4 cosets of Λ' in Λ, $|\Lambda/\Lambda'| = 4$. The fundamental gain remains at

$$\gamma_f = \left(\frac{d^2_{min}(2Z^2)}{\mathcal{V}(Z^2)\cdot 2}\right) / \left(\frac{d^2_{min}(Z^2)}{\mathcal{V}(Z^2)}\right) = \frac{4}{2}/\frac{1}{1} = 2 = 3 \text{ dB} \quad . \tag{10.74}$$

Shaping gain is again a function of the constellation shape, and is not considered to be part of the fundamental gain of any code. For any number of input bits, b, the structure of the code remains mainly the same, with only the number of points within the cosets of $\Lambda' = 2Z^2$ increasing with 2^b.

The partition chain for this particular code is often written $Z^2|D_2|2Z^2$.

10.4.2 A simple rate 2/3 Trellis Code

In the rate 2/3 code of interest, the encoder G will be a rate 2/3 encoder. The objective is to increase fundamental gain beyond 3dB, which was the parallel transition distance in the rate 1/2 code of Section 10.3. Higher gain necessitates constellation partitioning by one additional level/step to ensure that the parallel transition distance will now be 6dB, as in Figure 10.20. Then, the minimum distance will usually

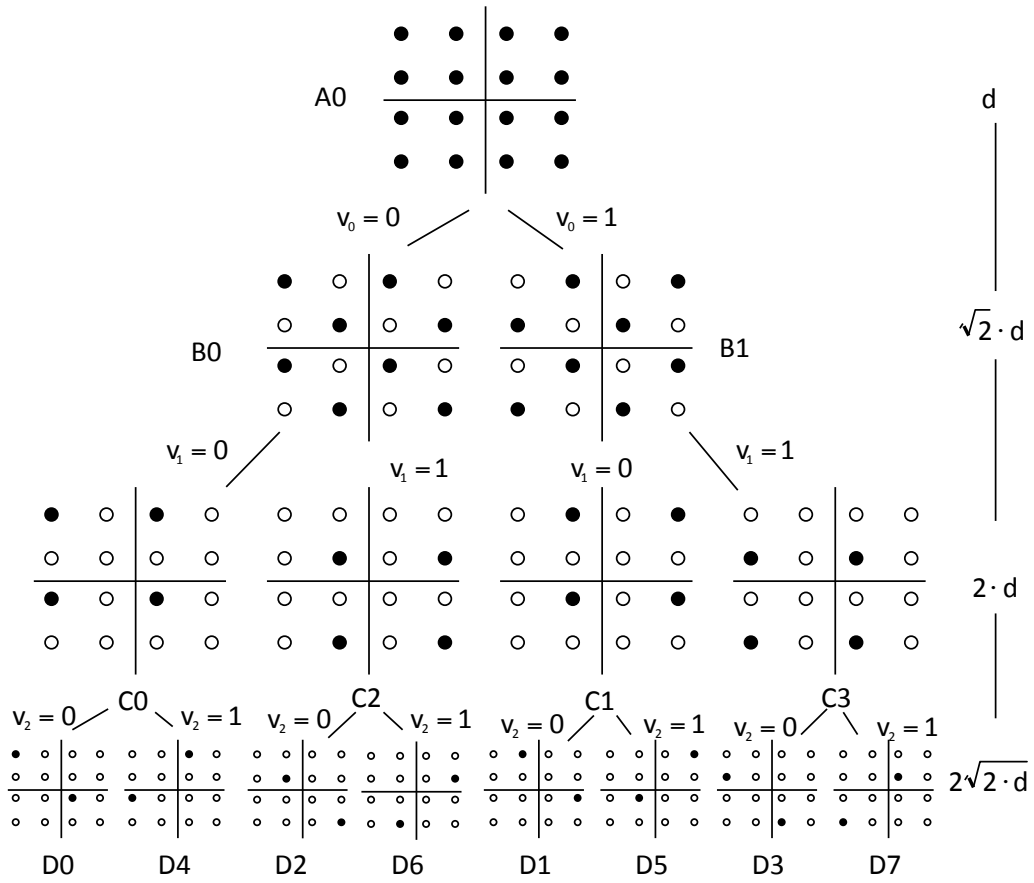


Figure 10.20: D-level partitioning of 16 QAM.

occur between two longer length sequences through the trellis, instead of between parallel transitions. The mapping-by-set-partitioning principle of the Section 10.3.2 extends one more level to the chain $Z^2|D_2|2Z^2|2D_2$, which is illustrated in detail in Figure 10.20 for a 16 SQ QAM constellation. Figure 10.21 shows a trellis for the successive coset selection from D-level sets in Figure **??** and also illustrates an example of the worst-case path for computation of d_{min}.

The worst case path has distance

$$d_{min} = \sqrt{2+1+2} = \sqrt{5} < \sqrt{8} \quad . \tag{10.75}$$

The fundamental gain is thus

$$\gamma_f = \frac{d_{min}^2}{2^{2\bar{r}_C}} = \frac{5}{2^{2(1/2)}} = 2.5 = 4 \text{ dB} \quad . \tag{10.76}$$

Essentially, this rate 2/3 code with 8 states, has an extra 1 dB of coding gain. It is possible to yet further increase the gain to 6dB by using a trellis with more states, as Section 10.4.3 will show.

Recognizing that this trellis is the same that was analyzed in Sections 10.1 and 10.2 or equivalently reading the generator from the circuit diagram in Figure 10.22,

$$G(D) = \begin{bmatrix} 1 & 0 & \frac{D^2}{1+D^3} \\ 0 & 1 & \frac{D}{1+D^3} \end{bmatrix} \tag{10.77}$$

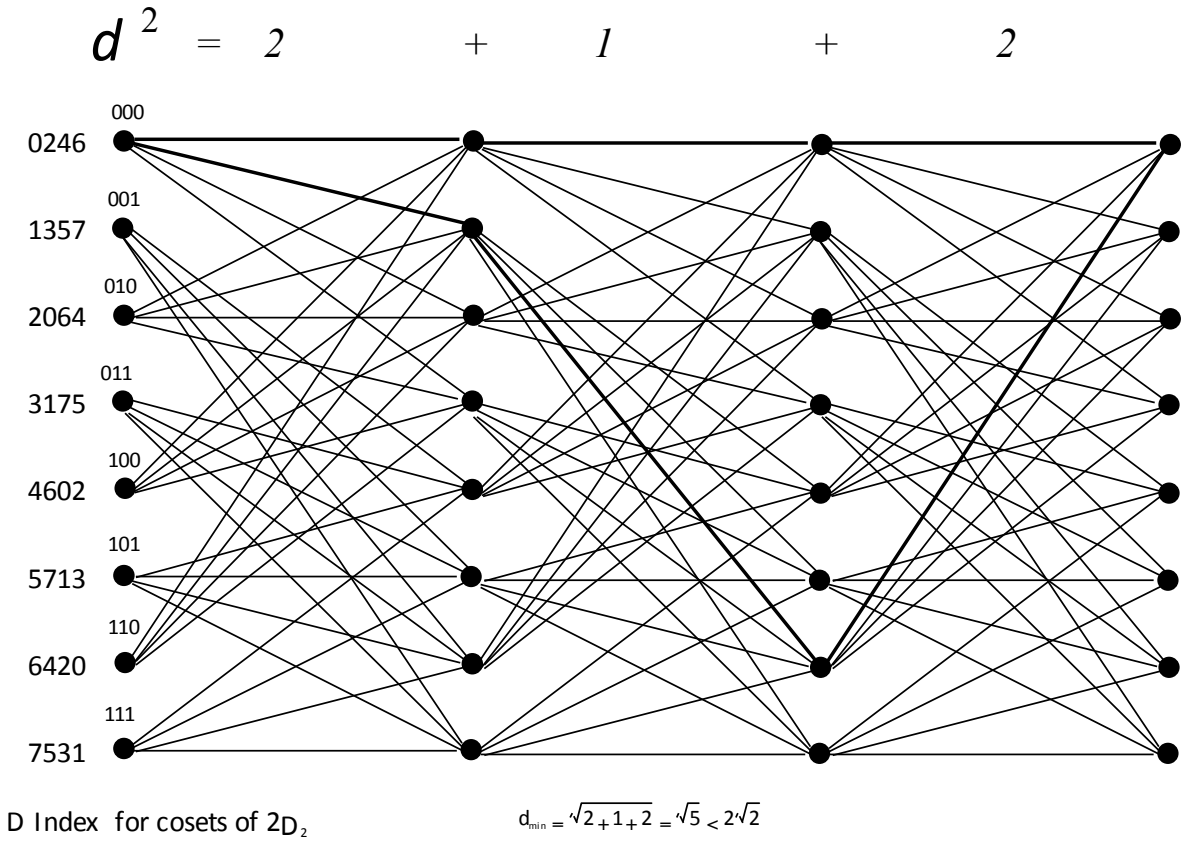


Figure 10.21: Trellis for 8-state rate 2/3 Trellis Code ($\gamma_f = 3$ dB)

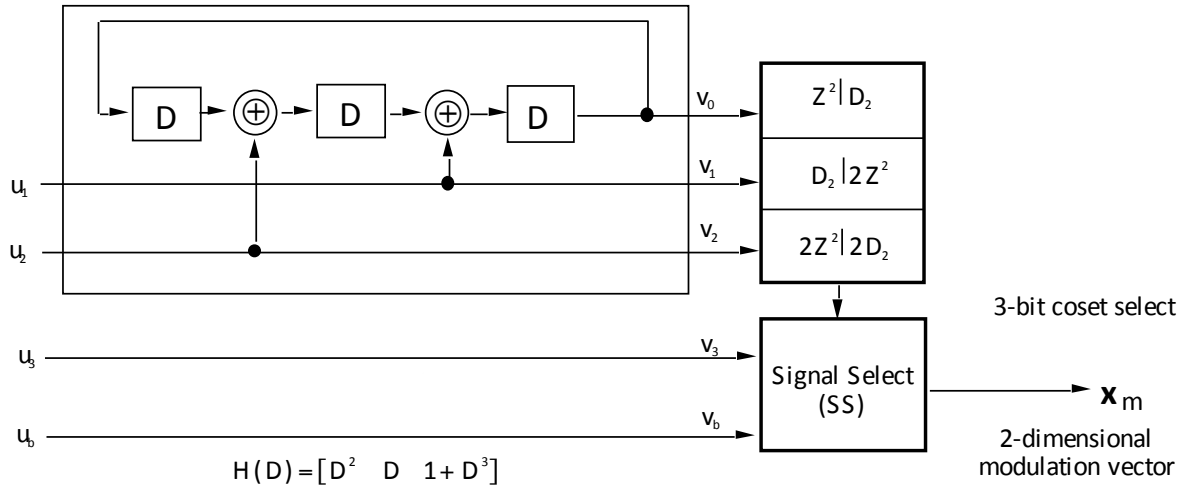


Figure 10.22: Encoder for 8-state rate 2/3 trellis code ($\gamma_f = 4$ dB).

Since this is a systematic realization,

$$G_{sys}H' = 0 = \begin{bmatrix} I & h^T \end{bmatrix} \begin{bmatrix} h^T \\ I \end{bmatrix} \quad , \tag{10.78}$$

so that the last $n-k=1$ column(s) are the rows of the parity matrix, or

$$H(D) = [h \;\; 1] \quad . \tag{10.79}$$

In the case of the code in question (after clearing the denominator),

$$H(D) = \begin{bmatrix} D^2 & D & 1+D^3 \end{bmatrix} \quad . \tag{10.80}$$

With trellis codes of rate $k/k+1$, the extra bit is named v_0 instead of v_{k+1}. A variant of the code in Figure 10.22 is used in the voiceband modem standards v.32 and v.32bis.

10.4.3 Code Design in One and Two Dimensions

Previous sections specifically studied two 2-dimensional trellis codes in detail: the first being a rate 1/2 4-state code with fundamental gain $\gamma_f = 3$ dB, and the second being a rate 2/3 8-state code with $\gamma_f = 4$ dB. There are many yet more powerful (higher γ_f) codes that have been mainly tabulated by Ungerboeck and Forney[3] that this subsection lists again shortly. There are also codes designed for the one-dimensional channel, which appear next.

One-Dimensional Trellis Codes

For one dimensional codes to have up to 6 dB of fundamental gain, the designer need only partition twice, so that $\Lambda' = \Lambda_{(2)}$ to realize a minimum separation between any two parallel transitions that is 6 dB higher than uncoded one-dimensional PAM. Stated more precisely, the partition chain for the one-dimensional trellis codes is, with $\Lambda = Z$, $Z|2Z|4Z$. The corresponding minimum distances between points are $d_{min}(Z) = 1$, $d_{min}(2Z) = 2$, and $d_{min}(4Z) = 4$, respectively, and $r_G = 1$ for the one-dimensional codes in this chapter. Since $r_G = 1$ implies a doubling of the constellation size $|\Lambda|$, with respect to uncoded transmission, the maximum fundamental gain will be limited to

$$\gamma_f \leq 16/(2^{2\cdot 1}) = 6 \text{ dB} \quad , \tag{10.81}$$

because the parallel transition separation is never more than $d^2 = 16$. Since $G(D)$ must then be a rate 1/2 code, then $G(D)$ and $H(D)$ are both 1×2 matrices. In one-dimensional codes, the actual signal set Λ is offset to eliminate any nonzero mean, and has equal numbers of positive and negative points, so that $\Lambda \rightarrow Z + \frac{1}{2}$. d^2_{min} for any such code must be an integer because it is a sum of squared integers.

Table 10.6 lists most of the best-known one-dimensional codes (and was essentially copied from Forney's 1988 Coset Codes I paper). Recall that $\bar{N}_e = N_e$ is the normalized number of nearest neighbors. The quantities $\bar{N}_1$ and $\bar{N}_2$ are the numbers of next-to-nearest neighbors (with squared distance $d^2 = d^2_{min} + 1$), and next-to-next-to-nearest neighbors (with squared distance $d^2_{min} + 2$). The effective gain $\tilde{\gamma}_f$ is the fundamental gain of the code, reduced by .2dB for each factor of 2 increase in nearest neighbor over the minimum of 2 nearest neighbors per dimension, which occurs for uncoded PAM transmission. $\bar{N}_D$ is a measure of decoder complexity that will be described in Section 10.4.4. An example of the use of these tables appears in Subsection 10.4.3.

For more complete understanding of the entries in the one-dimensional coding tables, Table 10.7 summarizes some partitioning results in one-dimension in Table 10.7. The subscript on a partition refers to the dimension of the underlying coset code, while the superscript refers to the coset index with an Ungerboeck labeling. The distances in the table correspond to

$$A^0_1 \quad = \quad Z \tag{10.82}$$

[3]The author would like to acknowledge help from Nyles Heise of IBM Almaden Research who updated and corrected some of these tables - Heise's corrections on the tables of Unberboeck and Forney are included here.

2^ν	h_1	h_0	d^2_{min}	γ_f	(dB)	$\bar{N}_e$	$\bar{N}_1$	$\bar{N}_2$	$\bar{N}_3$	$\bar{N}_4$	$\tilde{\gamma}_f$	$\bar{N}_D$
4	2	5	9	2.25	3.52	4	8	16	32	64	3.32	12
8	04	13	10	2.50	3.98	4	8	16	40	72	3.78	24
16	04	23	11	2.75	4.39	8	8	16	48	80	3.99	48
16	10	23	11	2.75	4.39	4	8	24	48	80	4.19	48
32	10	45	13	3.25	5.12	12	28	56	126	236	4.60	96
64	024	103	14	3.50	5.44	36	0	90	0	420	4.61	192
64	054	161	14	3.50	5.44	8	$\underline{32}$	66	84	236	4.94	192
128	126	235	16	4.00	6.02	66	0	256	0	1060	5.01	384
128	160	267	15	3.75	5.74	8	34	$\underline{100}$	164	344	5.16	384
128	124	207	14	3.50	5.44	4	8	14	56	136	5.24	384
256	362	515	16	4.00	6.02	2	32	$\underline{80}$	132	268	5.47	768
256	370	515	15	3.75	5.74	4	6	$\underline{40}$	68	140	5.42	768
512	0342	1017	16	4.00	6.02	2	0	56	0	$\underline{332}$	5.51	1536

Table 10.6: One-Dimensional Trellis Codes and Parameters
(Underlined quantities correspond to cases where worst-case performance is caused by large next-to-nearest neighbor counts.)

	A_1^0			
d_{min} w.r.t. A_1^0	1			
N_e w.r.t. A_1^0	2			
	B_1^0		B_1^1	
d_{min} w.r.t. B_1^0	2		1	
N_e w.r.t. B_1^0	2		2	
	C_1^0	C_1^2	C_1^1	C_1^3
d_{min} w.r.t. C_1^0	4	2	1	1
N_e w.r.t. C_1^0	2	2	1	1

Table 10.7: One-Dimensional Partitioning

2^ν	h_2	h_1	h_0	d^2_{min}	γ_f	(dB)	$\bar{N}_e$	$\bar{N}_1$	$\bar{N}_2$	$\bar{N}_3$	$\bar{N}_4$	$\tilde{\gamma}_f$	$\bar{N}_D$
4	-	2	5	4	2	3.01	2	16	64	256	1024	3.01	8
8	04	02	11	5	2.5	3.98	8	36	160	714	3144	3.58	32
16	16	04	23	6	3	4.77	28	80	410	1952	8616	4.01	60
32	10	06	41	6	3	4.77	8	52	202	984	4712	4.37	116
32	34	16	45	6	3	4.77	4	64	202	800	4848	4.44	116
64	064	016	101	7	3.5	5.44	28	130	504	2484	12236	4.68	228
64	060	004	143	7	3.5	5.44	24	146	592	2480	12264	4.72	228
64	036	052	115	7	3.5	5.44	20	126	496	2204	10756	4.78	228
128	042	014	203	8	4	6.02	172	0	2950	0	73492	4.74	451
128	056	150	223	8	4	6.02	86	312	1284	6028	29320	4.94	451
128	024	100	245	7	3.5	5.44	4	94	484	1684	8200	4.91	451
128	164	142	263	7	3.5	5.44	4	66	376	1292	6624	5.01	451
256	304	056	401	8	4	6.02	22	152	658	2816	13926	5.23	900
256	370	272	417	8	4	6.02	18	154	612	2736	13182	5.24	900
256	274	162	401	7	3.5	5.44	2	32	124	522	2732	5.22	900
512	0510	0346	1001	8	4	6.02	2	64	350	1530	6768	5.33	1796

Table 10.8: Two-Dimensional Trellis Codes and Parameters
(Underlined quantities correspond to cases where worst-case performance is caused by large next-to-nearest neighbor counts).

$$B_1^0 = 2Z \tag{10.83}$$
$$B_1^1 = 2Z+1 \tag{10.84}$$
$$C_1^0 = 4Z \tag{10.85}$$
$$C_1^1 = 4Z+1 \tag{10.86}$$
$$C_1^2 = 4Z+2 \tag{10.87}$$
$$C_1^3 = 4Z+3 \quad . \tag{10.88}$$

Two-Dimensional Codes

Two-dimensional codes use 3-level partitioning[4], so that $\Lambda' = \Lambda_{(3)}$ to realize a minimum separation between any two parallel transitions that is 6dB higher than uncoded two-dimensional QAM. Stated more precisely, the partition chain for the two-dimensional trellis codes of interest is, with $\Lambda = Z^2$, $Z^2|D_2|2Z^2|2D_2$. The corresponding minimum distances between points are $d_{min}(Z^2) = 1$, $d_{min}(D_2) = \sqrt{2}$, $d_{min}(2Z^2) = 2$, and $d_{min}(2D_2) = 2\sqrt{2}$ respectively, and $r_G = 1$ for the two-dimensional codes presented here. Since $r_G = 1$ implies a doubling of the two-dimensional constellation size $|\Lambda|$, with respect to uncoded transmission, the maximum fundamental gain will be limited to

$$\gamma_f \leq 8/2 = 6\text{dB} \quad . \tag{10.89}$$

Since $G(D)$ must then be a rate 2/3 code, then $G(D)$ is a 2×3 matrix and $H(D)$ is a 1×3 matrix, making $H(D)$ the more compact description. In the two-dimensional codes, the actual signal set Λ is offset to eliminate any nonzero mean, and has equal numbers of points in each quadrant, so that $\Lambda \rightarrow Z^2 + [\frac{1}{2}, \frac{1}{2}]$. d^2_{min} for any code must be an integer because it is a sum of integers.

Table 10.8 lists most of the best-known two-dimensional codes (and was also essentially copied from Forney's Coset Codes I paper). Recall that $\bar{N}_e$ is the normalized number of nearest neighbors. The quantities $\bar{N}_1$ and $\bar{N}_2$ mean the same thing they did for the one-dimensional codes, allowing comparisons on a per-dimensional basis between one and two dimensional codes.

For more complete understanding of the entries in the two-dimensional coding tables, Table 10.9 summarizes some partitioning results in two-dimensions. The subscript on a partition refers to the

[4]With the only exception being the 4-state 3dB code that was already studied

	A_2^0			
d_{min} w.r.t. A_2^0	1			
N_e w.r.t. A_2^0	4			
	B_2^0		B_2^1	
d_{min} w.r.t. B_2^0	$\sqrt{2}$		1	
N_e w.r.t. B_2^0	4		4	
	C_2^0	C_2^2	C_2^1	C_2^3
d_{min} w.r.t. C_2^0	2	$\sqrt{2}$	1	1
N_e w.r.t. C_2^0	4	4	2	2
	D_2^0	D_2^2	D_2^1	D_2^3
d_{min} w.r.t. D_2^0	$\sqrt{8}$	$\sqrt{2}$	1	1
N_e w.r.t. D_2^0	4	2	1	1
	D_2^4	D_2^6	D_2^5	D_2^7
d_{min} w.r.t. D_2^0	2	$\sqrt{2}$	1	1
N_e w.r.t. D_2^0	4	2	1	1

Table 10.9: Two-Dimensional Partitioning

dimension of the underlying coset code, while the superscript refers to the coset index with an Ungerboeck labeling. The distances in the table correspond to

$$
\begin{aligned}
A_2^0 &= Z^2 && (10.90)\\
B_2^0 &= RZ^2 && (10.91)\\
B_2^1 &= RZ^2 + [1,0] && (10.92)\\
C_2^0 &= 2Z^2 && (10.93)\\
C_2^1 &= 2Z^2 + [1,0] && (10.94)\\
C_2^2 &= 2Z^2 + [1,1] && (10.95)\\
C_2^3 &= 2Z^2 + [0,1] = 2Z^2 + [2,1] && (10.96)\\
D_2^0 &= 2RZ^2 && (10.97)\\
D_2^1 &= 2RZ^2 + [1,0] && (10.98)\\
D_2^2 &= 2RZ^2 + [1,-1] && (10.99)\\
D_2^3 &= 2RZ^2 + [2,-1] && (10.100)\\
D_2^4 &= 2RZ^2 + [0,-2] && (10.101)\\
D_2^5 &= 2RZ^2 + [1,-2] && (10.102)\\
D_2^6 &= 2RZ^2 + [1,-3] && (10.103)\\
D_2^7 &= 2RZ^2 + [0,1] = 2RZ^2 + [2,-3] \quad . && (10.104)
\end{aligned}
$$

Phase-Shift Keying Codes

Although, PSK codes fall properly outside the domain of coset codes as considered here - gains can be computed and codes found for two cases of most practical interest. Namely 4PSK/8PSK systems and 8PSK/16PSK systems. The corresponding code tables are illustrated in Tables 10.10 and 10.11. Partitioning proceeds as in one-dimension, except that dimension is wrapped around a circle of circumference 2^{b+1}.

All PSK trellis codes have their gain specified with respect to the uncoded circular constellation - that is with respect to QPSK for $b = 2$ or to 8PSK for $b = 3$.

2^ν	h_2	h_1	h_0	γ	(dB)	$\bar{N}_e$	$\bar{N}_1$	$\bar{N}_2$	$\tilde{\gamma}$
4	-	2	5	2	3.01	.5	?	?	3.41
8	04	02	11	2.291	3.60	2	?	?	3.80
16	16	04	23	2.588	4.13	1.15	?	?	4.29
32	34	16	45	2.877	4.59	2	?	?	4.59
64	066	030	103	3.170	5.01	2.5	?	?	4.95
128	122	054	277	3.289	5.17	.25	?	?	5.67?
256	130	072	435	3.758	5.75	.75	?	?	6.03?

Table 10.10: 4PSK/8PSK Trellis Codes and Parameters
(Effective gains are suspect, as next-to-nearest neighbor counts are not presently available.)

2^ν	h_2	h_1	h_0	γ	(dB)	$\bar{N}_e$	$\bar{N}_1$	$\bar{N}_2$	$\tilde{\gamma}$
4	–	2	5	2.259	3.54	2	?	?	3.54
8	–	04	13	2.518	4.01	2	?	?	4.01
16	–	04	23	2.780	4.44	4	?	?	4.24
32	–	10	45	3.258	5.13	4	?	?	4.93
64	–	024	103	3.412	5.33	1	?	?	5.53?
128	–	024	203	3.412	5.33	1	?	?	5.53?
256	374	176	427	3.556	5.51	4	?	?	5.31?

Table 10.11: 8PSK/16PSK Trellis Codes and Parameters
Effective gains are suspect, as next-to-nearest neighbor counts are not presently available.

Design Examples

This subsection presents two design examples to illustrate the use of Tables 10.6 and 10.8.

EXAMPLE 10.4.1 (32CR Improved by 4.5dB) A data transmission system transmits 5 bits/2D-symbol over an AWGN channel with channel SNR=19.5dB. This SNR is only sufficient, using 32CR QAM, to achieve a probability of error

$$P_e = 4\left(1 - \frac{1}{\sqrt{2 \cdot 32}}\right) Q\left[\sqrt{\frac{3\text{SNR}}{(31/32)32 - 1}}\right] \approx 4Q(2.985) \approx .0016 \quad , \tag{10.105}$$

which is (usually) insufficient for reliable data transmission. An error rate of approximately 10^{-6} is desired. To get this improved error rate, the applied code needs to increase the SNR in (10.105) by approximately 4.5dB. Before using the tables, the designer computes the shaping gain (or loss) from doubling the signal set size from 32CR to 64SQ (presuming no more clever signal constellation with 64 points is desirable for this example) as

$$\gamma_s(32CR) = 10\log_{10}\left(\frac{1 \cdot (2^5 - 1)}{12 \cdot (5/2)}\right) = .14dB \tag{10.106}$$

and

$$\gamma_s(64QAM) = 10\log_{10}\left(\frac{2 \cdot (2^5 - 1)}{12 \cdot (10.5/2)}\right) = -0.07dB \quad , \tag{10.107}$$

thus the design loses .21dB in going to the 64SQ QAM constellation for trellis coding with respect to the 32CR QAM. This is because 32CR QAM is closer to a circular boundary than is 64 SQ QAM.

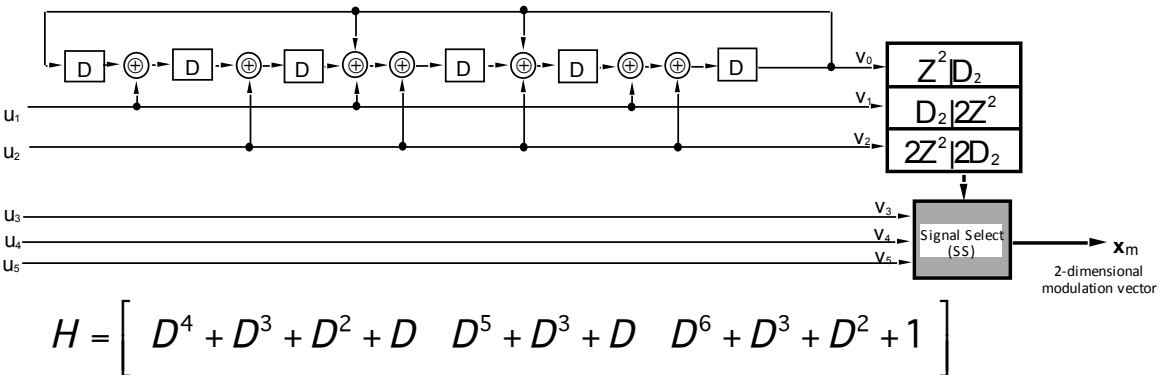


Figure 10.23: Circuit for 64QAM code with 4.57dB gain.

Table 10.8 contains two 64-state codes that achieve an effective coding gain of at least 4.72dB. Since the 2 satisfactory codes listed have the same complexity, a better choice is the last, which has effective gain 4.78dB. Taking the shaping gain penalty gain, the full gain of this code with respect to 32CR is 4.78-.21 = 4.57 dB, and the error probability is then

$$P_e \approx 4Q\left[\sqrt{\frac{3(\text{SNR} + 4.57\text{dB})}{(31/32)32 - 1}}\right] \approx 4Q(5.05) \approx 9 \times 10^{-7} \quad , \tag{10.108}$$

which is below the desired 10^{-6}. From the table, $h_2 = 036$, $h_1 = 052$, and $h_0 = 115$, so that

$$H = \begin{bmatrix} D^4 + D^3 + D^2 + D & D^5 + D^3 + D & D^6 + D^3 + D^2 + 1 \end{bmatrix} \tag{10.109}$$

or equivalently in systematic form

$$H = \begin{bmatrix} \dfrac{D^4 + D^3 + D^2 + D}{D^6 + D^3 + D^2 + 1} & \dfrac{D^5 + D^3 + D}{D^6 + D^3 + D^2 + 1} & 1 \end{bmatrix} \quad . \tag{10.110}$$

(Recall that $GH^* = 0$, so that $G = [I\ h^*]$ when $H = [h\ 1]$.) The circuit for implementation of this code, in systematic form, is illustrated in Figure 10.23 and a labeled (using Ungerboeck labeling) 64QAM constellation for the code is shown in Figure 10.24.

The second example uses a 1-dimensional code in conjunction with the 1+.9D channel that was studied extensively in EE379A. For this system, given developments in this text to this point, the best means of applying trellis coding is to use a decision feedback equalizer (infinite length) to eliminate ISI, and for which our mean-square error sequence will be white (but not necessarily Gaussian, which this text's analysis will assume anyway).

EXAMPLE 10.4.2 ($1 + .9D^{-1}$ Revisited for Code Concatenation) From previous study of this example in earlier chapters, the SNR for the MMSE-DFE was 8.4dB on this $1 + .9D^{-1}$ channel with SNR_{mfb} = 10dB and 1 bit/T transmission. The corresponding error rate on such a system would be completely unacceptable in most applications, so improvement is desirable. A one-dimensional 256-state trellis code from Table 10.6 has effective gain 5.47dB. The shaping gain loss in going from 1bit/T to the 4 levels/T necessary in this code is easily computed as 4/5=-.97dB. The coding gain for this application would then be 5.47-.97=4.50dB. The probability of error for the coded system would then be

$$P_e = 2 \cdot Q(8.4 + 4.5dB) = 2 \cdot Q(4.41) = 10^{-5} \quad . \tag{10.111}$$

Unfortunately, error propagation in the internal DFE is now more likely, not only because an uncoded system would have had a high probability of error of about .01, but also because

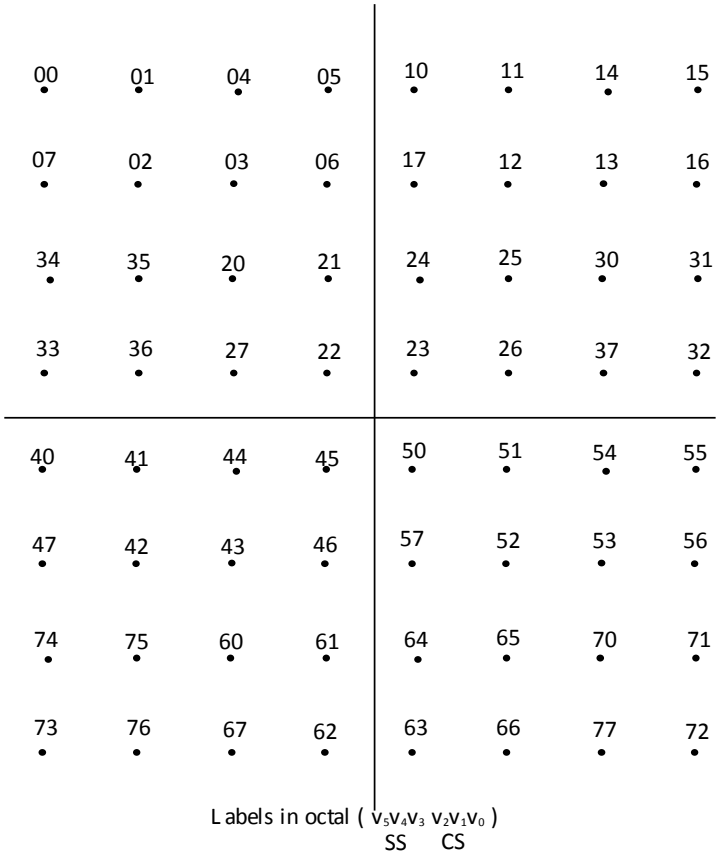


Figure 10.24: Constellation for 64QAM code example with 4.57dB gain.

the enlarged constellation from which a DFE would have to make instantaneous decisions now has more points and smaller symbol-by-symbol-detection distance. A Laroia precoder here with the enlarged constellation would only lose a few tenths of a dB. The parity matrix is 1×2 and can be read from Table 10.6 as

$$H = \begin{bmatrix} D^7 + D^6 + D^5 + D^4 + D & D^8 + D^6 + D^3 + D^2 + 1 \end{bmatrix} \tag{10.112}$$

which corresponds to a systematic G of

$$G_{sys} = \begin{bmatrix} 1 & \dfrac{D^7 + D^6 + D^5 + D^4 + D}{D^8 + D^6 + D^3 + D^2 + 1} \end{bmatrix} \quad . \tag{10.113}$$

Decision-Feedback Sequence Estimation

Decision-Feedback Sequence Estimation (DFSE) essentially eliminates error propagation when DFE's are used with coset codes. In DFSE, the survivor into each state is used to determine 2^ν possible feedback-section outputs, one for each state. The ISI-subtraction associated with that correct survivor is then used in computing the branch metric into each state.

10.4.4 Decoder Complexity Measures

The implementation complexity of a trellis or lattice code depends heavily on the details of the application. Nevertheless, it is possible to associate with each coset code, a meaningful measure of complexity. This measure is mainly relative and used for comparing the use of two different codes in the same application. The measure used in this book is equivalent to one introduced by Forney in 1988.

This measure is computed for a maximum-likelihood (Viterbi for trellis codes) detector and ignores encoder complexity. The complexity measure counts the number of additions and number of comparisons that need to be performed in the straightforward implementation of the decoder and adds these two numbers together. This measure is essentially a count of the number of instruction cycles that are required to implement the decoder on a programmable signal processor.

Definition 10.4.1 (Decoder Complexity) *The quantity N_D is called the* **decoder complexity** *for a coset code and is defined to be the total number of additions and comparisons that are needed to implement the decoder in the straightforward maximum-likelihood implementation. The* **normalized decoder complexity** *is $\bar{N}_D \triangleq N_D/N$.*

In decoding a coset code, the N-dimensional channel output vector $\boldsymbol{y}$ is used to resolve which of the possible points in each of the cosets used by the code is closest to this received sample $\boldsymbol{y}$. This step chooses the parallel transition between states that is more likely than the other such parallel transitions. Once these parallel transitions (one for each coset) have been chosen, sequence detection (via the Viterbi algorithm) chooses the sequence of cosets that was most likely.

For one-dimensional coset codes, the resolution of the closet point in each of the 4 cosets is trivial, and is essentially a truncation of the received vector $\boldsymbol{y}$, so this operation is not included in N_D. Then, for a rate $k/(k + r_{_G})$ code with 2^ν states, the decoder requires 2^k adds and $2^k - 1$ binary comparisons for each state per received one-dimensional output. This is a total of

$$N_D(one - dimensional) = 2^\nu \left(2^k + 2^k - 1\right) = 2^{\nu+k}\left(2 - 2^{-k}\right) \quad . \tag{10.114}$$

This computational count is independent of b because it ignores the truncation associated with choosing among parallel transitions. This also permits the inclusion of N_D in the tables of Section 10.4.3, which do not depend on b.

For two-dimensional codes, the parallel-transition resolution that was trivial in one dimension now requires one addition for each two-dimensional coset. While table look-up or truncation can be used to resolve each of the one-dimensional components of the two-dimensional $\boldsymbol{y}$ for every coset, N_D includes an operation for the addition of the component one-dimensional metrics to get the two-dimensional metric. Since there are 2^{k+r_G} cosets in the coset code, then

$$N_D(two - dimensional) = 2^\nu \left(2^k + 2^k - 1\right) + 2^{k+r_G} = 2^{\nu+k}\left(2 - 2^{-k}\right) + 2^{k+r_G} \quad , \tag{10.115}$$

or

$$\bar{N}_D = 2^{\nu+k}\left(1 - 2^{-k-1}\right) + 2^{k+r_G-1} \quad . \tag{10.116}$$

The computation of N_D for the higher dimensional codes that is discussed later in this chapter is similar, except that the resolution of the parallel transitions becomes increasingly important and complex as the code dimensionality grows.

10.5 Multidimensional Trellis Codes

Multidimensional lattices can be combined with trellis coding to get a larger coding gain for a given complexity, and can also reduce the expansion of the constellation slightly (which may be of importance if channel nonlinearity is important). This section begins in Section 10.5.1 with a discussion of multidimensional partitioning, before then enumerating 4D and 8D coset codes in Section 10.5.2.

10.5.1 Lattice Codes and Multidimensional Partitioning

Lattice Codes are sometimes construed as coset codes with G a $k \times (k + r_G)$ constant matrix (or block code). For the lattices of interest in this chapter, we need not develop the connection with block codes and instead consider $r_G = 0$ and any redundancy as part of r_Λ. For those interested in the strong connection, see the work by Forney and by Conway and Sloane. Appendix A of this chapter introduces lattices.

The fundamental gain of a lattice is defined similar to that of a coset code (where the reference uncoded system is again a Z^N lattice)

$$\gamma_f(\Lambda) \triangleq \frac{d^2_{min}(\Lambda)}{\mathcal{V}^{2/N}(\Lambda)} = \frac{d^2_{min}(\Lambda)}{2^{2\bar{r_\Lambda}}} \quad . \tag{10.117}$$

Multidimensional symbols are often formed by concatenation of lower-dimensional symbols. For instance, a four-dimensional code symbol is often formed from two-dimensional QAM signals as [QAM1 , QAM2], or perhaps from 4 one-dimensional PAM signals. Eight-dimensional symbols are also formed by concatenation of four two-dimensional symbols or eight one-dimensional symbols, or even two 4-dimensional symbols. Certain symbols may be allowed to follow other symbols, while certain other symbols cannot. This section attempts to enumerate and study the most common four and eight dimensional constellation lattices, and in particular their partitioning for trellis code application.

Rectangular Lattice Family

The rectangular lattice in N dimensions is just the lattice Z^N, or any coset (translation) thereof. The volume of such a lattice is

$$\mathcal{V}(Z^N) = 1 \tag{10.118}$$

and the minimum distance is

$$d_{min} = 1 \tag{10.119}$$

leaving a fundamental lattice gain of $\gamma_f(Z^N) = 1$ or 0 dB.

Simple Lattice Constructions:
For Example 10.3.2,

$$D_2 \triangleq R_2 Z^2 \quad , \tag{10.120}$$

$\mathcal{V}(D_2) = 2$ and $d_{min}(D_2) = \sqrt{2}$, so that $\gamma_f(D_2) = 0$ dB. D_2 partitions Z^2, and $|Z^2/D_2| = 2$ so that

$$\mathcal{V}(D_2) = |Z^2/D_2| \cdot \mathcal{V}(Z^2) \quad . \tag{10.121}$$

More generally:

> **Theorem 10.5.1 (Volume and Partitioning)** *if a sublattice Λ' partitions the lattice Λ, then*
>
> $$\mathcal{V}(\Lambda') = |\Lambda/\Lambda'| \cdot \mathcal{V}(\Lambda) \quad . \tag{10.122}$$
>
> **Proof:** Because there are $\frac{1}{|\Lambda/\Lambda'|}$ as many points in Λ' as in Λ, and the union of fundamental volumes for all points in a lattice must cover N-dimensional space, then $\mathcal{V}(\Lambda')$ must be $|\Lambda/\Lambda'|$ times larger. **QED**.

Two successive applications of R_2 produce

$$R_2^2 = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix} \quad , \tag{10.123}$$

a scaling of the original lattice by a factor of 2, so that d_{min} increases by 2, and volume (area) increases by a factor of 4, and γ_f remains at 0 dB. The resultant lattice is

$$R_2^2 Z^2 = R_2 D_2 = 2Z^2 \quad . \tag{10.124}$$

The semi-infinite partition chain is:

$$Z^2/R_2Z^2/R_2^2Z^2/R_2^3Z^2/R_2^4Z^2... \tag{10.125}$$

$$Z^2/D_2/2Z^2/2D_2/4Z^2/... \tag{10.126}$$

The two-dimensional partitioning for two-dimensional trellis codes is generated by successive application of the rotation operator R_2.

The concept of the rotation operator can be extended to 4 dimensions by defining

$$R_4 \triangleq \begin{bmatrix} R_2 & 0 \\ 0 & R_2 \end{bmatrix} \quad , \tag{10.127}$$

and multiplication of a four-dimensional lattice by R_4 amounts to applying R_2 independently to the first two coordinates of a four-dimensional lattice and to the last two coordinates of that same lattice to generate a new four-dimensional set of points. Thus,

$$R_4 Z^4 = R_2 Z^2 \otimes R_2 Z^2 \tag{10.128}$$

and that R_4Z^4 partitions Z^4 four ways, that is $|Z^4/R_4Z^4| = 4$.

$$Z^4 = R_4Z^4 + \{[0000], [0001], [0100], [0101]\} \quad . \tag{10.129}$$

Then,

$$\mathcal{V}(R_4Z^4) = |Z^4/R_4Z^4| \ \cdot \ \mathcal{V}(Z^4) = 4 \cdot 1 = 4 \quad , \tag{10.130}$$

and that $d_{min}^2(R_4Z^4) = 2$, so that

$$\gamma_f(R_4Z^4) = \frac{2}{4^{2/4}} = 1 \ \ (0 \text{ dB}) \quad . \tag{10.131}$$

Similarly, R_8 is

$$R_8 \triangleq \begin{bmatrix} R_4 & 0 \\ 0 & R_4 \end{bmatrix} \quad . \tag{10.132}$$

Then, $d_{min}^2(R_8Z^8) = 2$, $\mathcal{V}(R_8Z^8) = 16$, and

$$\gamma_f(R_8Z^8) = \frac{2}{16^{2/8}} = 1 \ \ (0 \text{ dB}) \quad . \tag{10.133}$$

Lemma 10.5.1 (Invariance of fundamental gain to squaring and rotation) *The fundamental gain of a lattice is invariant under rotation and/or squaring.*

Proof: The rotation operation increases volume by $2^{N/2}$ and squared minimum distance by 2, thus $\gamma_f(R\Lambda) = 2/\{2^{[(N/2)(2/N)]}\}\gamma_f(\Lambda) = \gamma_f(\Lambda)$. The squaring operation squares the volume and doubles the dimensionality, but does not alter the minimum distance, thus $\gamma_f(\Lambda)$ is not altered. **QED.**

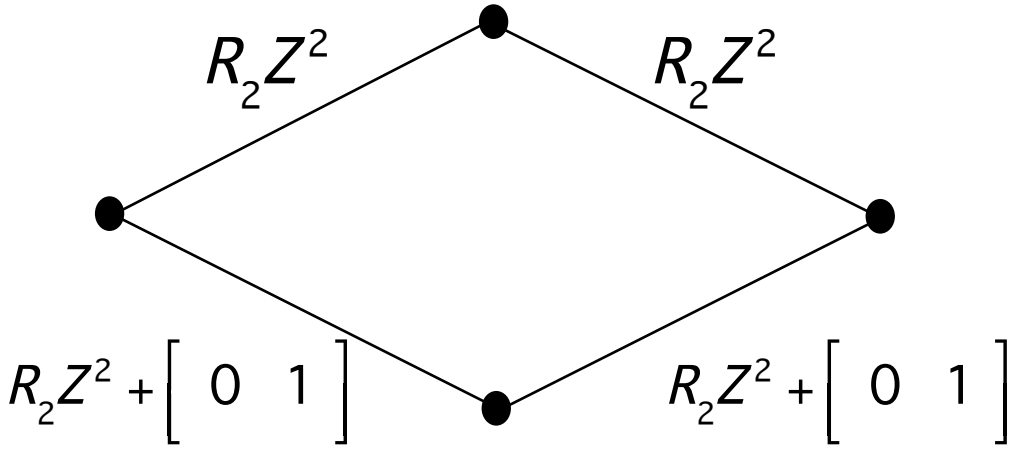


Figure 10.25: Trellis for the D_4 lattice.

D - Lattice Family

The lattice $D_2 = R_2Z^2$ is a rotated (and scaled) version of the Z^2 lattice. It partitions Z^2 into two sets, D_2 and $D_2 + [0,1]$, that is

$$Z^2 = D_2 \bigcup (D_2 + [0,1]) \tag{10.134}$$

In four dimensions, that R_4Z^4 partitions Z^4 into 4 sets – the D_4 lattice partitions Z^4 into two sets. This lattice is sometimes called the "Schlafi Lattice":

$$\begin{aligned} D_4 &\triangleq R_4Z^4 \bigcup \left(R_4Z^4 + [0,1,0,1]\right) && (10.135) \\ &= \left(R_2Z^2 \otimes R_2Z^2\right) \bigcup \left((R_2Z^2 + [0,1]) \otimes (R_2Z^2 + [0,1])\right) \quad , && (10.136) \end{aligned}$$

which can be identified as all those points in Z^4 that have even squared norms. Thus, D_4 partitions Z^4 into two sets of points (evens and odds). For D_4, $d^2_{min}(D_4) = 2$ and $\mathcal{V}(D_4) = 2$, so that

$$\gamma_f(D_4) = \frac{2}{2^{2/4}} = \sqrt{2} \quad = 1.5 \text{ dB} \quad . \tag{10.137}$$

Equation (10.137) relates that the D_4 lattice is better in terms of packing points per unit volume than the rectangular lattice family by a factor of 1.5 dB. In fact, D_4 is the best such **lattice** in four dimensions.

The D_4 lattice is a simple coset code or lattice code that can be described by a trellis that starts and ends with a single state, but which has two states in the middle as shown in Figure 10.25. The Viterbi Algorithm for MLSD can be applied to the trellis in Figure 10.25 to decode the D_4 lattice in the same way that the Viterbi Algorithm is used for trellis codes.

In eight dimensions, R_8Z^8 partitions Z^8 into 16 cosets of R_8Z^8 so that

$$|Z^8/R_8Z^8| = 16 \quad . \tag{10.138}$$

Also since $\left|Z^4/D_4\right| = 2$, then

$$|Z^8/(D_4)^2| = 4 \quad . \tag{10.139}$$

A binary partition (of order 2) is desirable:

$$D_8 \triangleq (D_4 \otimes D_4) \bigcup ((D_4 + [0,0,0,1]) \otimes (D_4 + [0,0,0,1])) \quad , \tag{10.140}$$

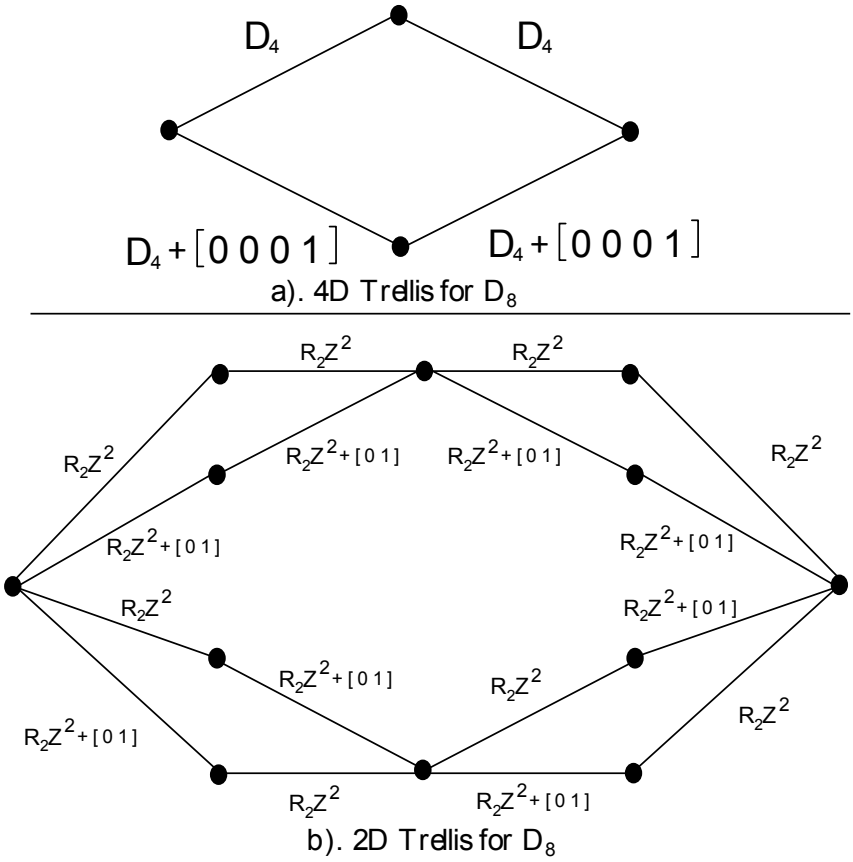


Figure 10.26: Trellis for the D_8 lattice.

which is the subset of Z^8 of all points with even squared norms. Thus,

$$Z^8 = D_8 \bigcup (D_8 + [0,0,0,0,0,0,0,1]) \quad . \tag{10.141}$$

$d^2_{min}(D_8) = 2$ and $\mathcal{V}(D_8) = 2$, so that

$$\gamma_f(D_8) = \frac{2}{2^{2/8}} = 2^{.75} \quad = 2.27 \text{ dB} \quad . \tag{10.142}$$

Thus, D_8 is somewhat better than D_4. Also observe that $(D_4)^2$ partitions D_8 from (10.140), that $|D_8/(D_4)^2| = 2$, and that $\gamma_f(D_4^2) = 1.5$ dB, which also follows from Lemma 10.5.1

A trellis similar to that in Figure 10.25 can be drawn for the D_8 lattice as shown in Figure 10.26. Each of the D_4 trellises in Figure 10.26 can be replaced by D_4 trellises leading to the more informative 2-dimensional trellis for D_8 in Figure 10.26. Again, the Viterbi algorithm can decode the D_8 lattice using these trellises.

The DE_8 Lattice

The DE_8 lattice is defined by

$$DE_8 \quad \stackrel{\Delta}{=} \quad (R_4 Z^4)^2 \bigcup \left((R_4 Z^4)^2 + [0,1,0,1,0,1,0,1]\right) \tag{10.143}$$

$$= \quad R_8 Z^8 \bigcup \left(R_8 Z^8 + [0,1,0,1,0,1,0,1]\right) \tag{10.144}$$

DE_8 partitions $(D_4)^2$ into two groups by

$$\begin{aligned}(D_4)^2 &= \Big[\big(R_4Z^4\big)\bigcup\big(R_4Z^4+[0,1,0,1]\big)\Big]^2 \\ &= \big(R_4Z^4\big)^2\bigcup\big(R_4Z^4+[0,1,0,1]\big)^2\bigcup\big(R_4Z^4\otimes(R_4Z^4+[0,1,0,1])\big)\bigcup\big((R_4Z^4+[0,1,0,1])\otimes R_4Z^4\big) \\ &= DE_8\bigcup(DE_8+[0,1,0,1,0,0,0,0]) \quad . \end{aligned} \tag{10.145}$$

The last step notes that with respect to the R_4Z^4 lattice, adding [0,2,0,2] is equivalent to adding [0,0,0,0]. $d^2_{min}(DE_8)=2$ and $\mathcal{V}(DE_8)=8$, so that

$$\gamma_f(DE_8)=\frac{2}{8^{2/8}}=2^{.25} \quad =.73 \text{ dB} \quad . \tag{10.146}$$

So far, previous results have established the partition chain

$$Z^8/D_8/D_4^2/DE_8 \tag{10.147}$$

with a factor of 2 increase in lattice fundamental volume at each step in the partitioning. One is tempted to complete the chain by partitioning again into R_8Z^8, which would be a valid partition. The next subsection shows another partition into a much better 8 dimensional lattice.

The trellis diagram for the DE_8 lattice is trivial and left to the reader as an exercise.

The Gosset (E_8) Lattice

The Gosset or E_8 Lattice is the most dense **lattice** in 8 dimensions. It is defined by

$$E_8 \triangleq R_8D_8\bigcup(R_8D_8+[0,1,0,1,0,1,0,1]) \quad . \tag{10.148}$$

All norms of points in E_8 are integer multiples of 4. Since rotation by R_N increases distance by a factor of $\sqrt{2}$, inspection of the coset leader in the second term of (10.148) leads to

$$d^2_{min}(E_8)=4 \tag{10.149}$$

Further, $|E_8/R_8D_8|=2$, so that

$$\mathcal{V}(E_8)=\frac{1}{2}\mathcal{V}(R_8D_8)=\frac{1}{2}16\cdot 2=16 \quad . \tag{10.150}$$

Then,

$$\gamma_f(E_8)=\frac{4}{16^{2/8}}=2 \ \ (3 \text{ dB}) \quad . \tag{10.151}$$

A simple trellis in terms of the 4 dimensional constituents appears in Figure 10.27, where the Cartesian product decomposition for D_8 in Equation (10.140) has been used along with the fact that rotation by R_8 is the same as rotating each four-dimensional constituent by R_4. A two-dimensional trellis is then constructed by drawing the 2D trellises for D_4 wherever they appear in Figure 10.27. This two-dimensional trellis appears in Figure 10.28.

The assumption that E_8 partitions DE_8 is justified by taking

$$Z^8=D_8\bigcup(D_8+[0,0,0,0,0,0,0,1]) \tag{10.152}$$

and premultiplying by R_8

$$R_8Z^8=R_8D_8\bigcup(R_8D_8+[0,0,0,0,0,0,1,-1]) \quad , \tag{10.153}$$

making (10.144) the same as

$$\begin{aligned} DE_8 &= R_8D_8\bigcup(R_8D_8+[0,0,0,0,0,0,1,-1]) && (10.154)\\ &\quad \bigcup(R_8D_8+[0,1,0,1,0,1,0,1])\bigcup(R_8D_8+[0,1,0,1,0,1,1,0]) && (10.155)\\ &= E_8\bigcup(E_8+[0,0,0,0,0,0,1,-1]) \quad . && (10.156)\end{aligned}$$

Thus, E_8 partitions DE_8 and $|DE_8/E_8|=2$.

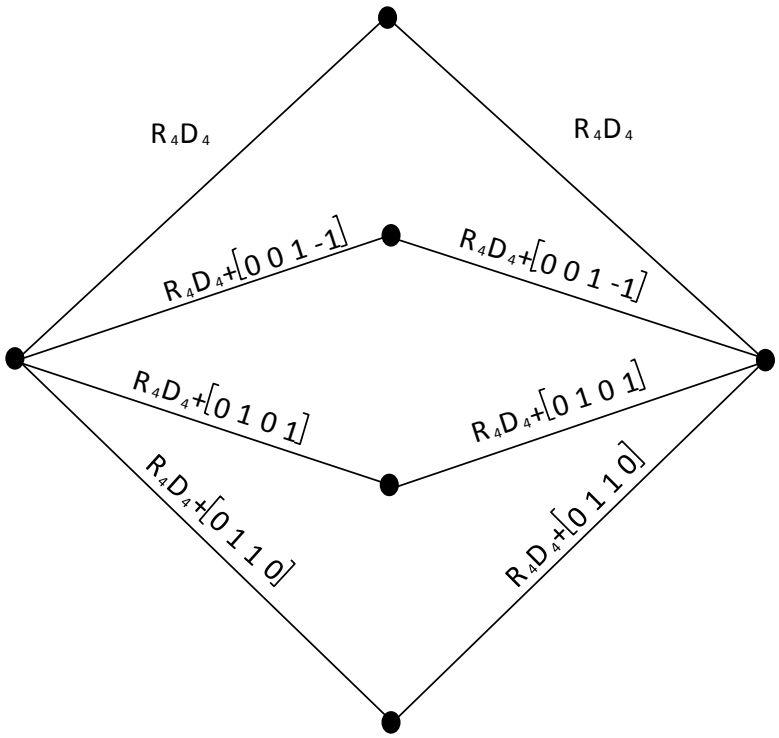


Figure 10.27: 4D Trellis for the E_8 lattice.

v_i		$d^2_{min}(\Lambda)$	$\mathcal{V}(\Lambda)$	$\gamma_f(\Lambda)$ dB
–	Z^4	1	1	0
v_0	D_4	2	2	1.5
v_1	R_4Z^4	2	4	0
v_2	R_4D_4	4	8	1.5
v_3	$2Z^4$	4	16	0
v_4	$2D_4$	8	32	1.5

Table 10.12: Four-dimensional partition tower and lattice parameters.

4 and 8 Dimensional Partition Chains

The previous subsections established the four-dimensional partition chain

$$Z^4/D_4/R_4Z^4/R_4D_4/2Z^4/2D_4/... \tag{10.157}$$

and the eight-dimensional partition chain

$$Z^8/D_8/D_4^2/DE_8/E_8/R_8D_8/(R_4D_4)^2/R_8DE_8/R_8E_8/... \tag{10.158}$$

These partition chains are summarized in the partition towers and accompanying tables in Tables 10.12 and 10.13. These partitionings are also used extensively in four- and eight-dimensional trellis codes, as discussed in the next section. Figure 10.29 is a partition tree showing the specific labels for a four-dimensional partitioning with Ungerboeck labeling. Each of the cosets of the original constellation in Figure 10.29 can be written as (possibly unions of) Cartesian products of two-dimensional cosets in the partitioning of the Z^2 lattice. This section lists those Cartesian products for reference:

$$A_4^0 = A_2^0 \otimes A_2^0 \tag{10.159}$$

$$B_4^0 = \left(B_2^0 \otimes B_2^0\right) \bigcup \left(B_2^1 \otimes B_2^1\right) \tag{10.160}$$

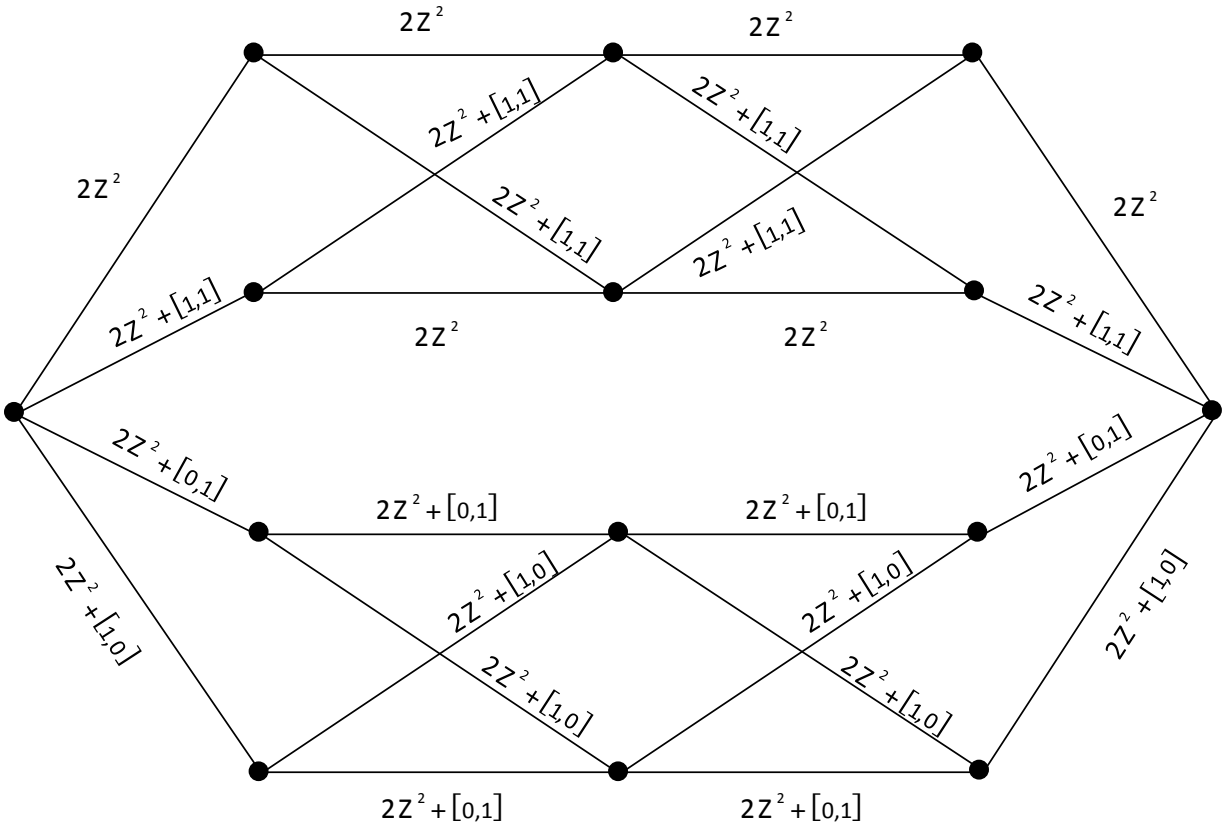


Figure 10.28: 2D Trellis for the E_8 lattice.

v_i		$d^2_{min}(\Lambda)$	$\mathcal{V}(\Lambda)$	$\gamma_f(\Lambda)$ dB
–	Z^8	1	1	0
v_0	D_8	2	2	2.27
v_1	$(D_4)^2$	2	4	1.5
v_2	DE_8	2	8	.73
v_3	E_8	4	16	3
v_4	R_8D_8	4	32	2.27
v_5	$R_8(D_4)^2$	4	64	1.5
v_6	R_8DE_8	4	128	.73
v_7	R_8E_8	8	256	3

Table 10.13: Eight-Dimensional Partition Tower and Lattice Parameters

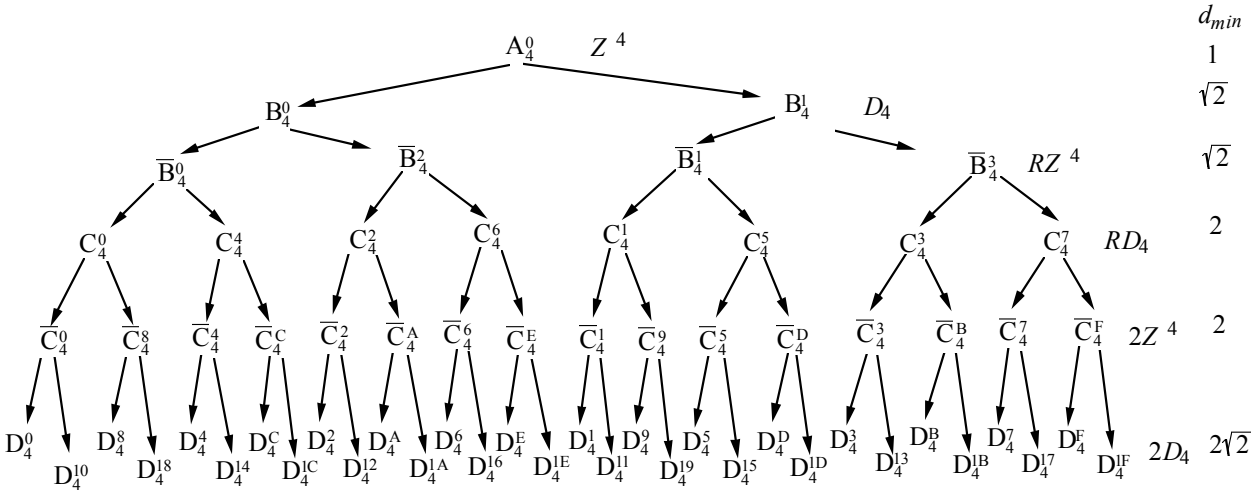


Figure 10.29: Four-dimensional partition tree with Ungerboeck labeling - indices of $\bar{C}$ and of D are in hexadecimal.

$$B_4^1 = \left(B_2^0 \otimes B_2^1\right) \bigcup \left(B_2^1 \otimes B_2^0\right) \tag{10.161}$$

$$\bar{B}_4^0 = B_2^0 \otimes B_2^0 \tag{10.162}$$

$$\bar{B}_4^2 = B_2^1 \otimes B_2^1 \tag{10.163}$$

$$\bar{B}_4^1 = B_2^0 \otimes B_2^1 \tag{10.164}$$

$$\bar{B}_4^3 = B_2^1 \otimes B_2^0 \tag{10.165}$$

$$C_4^0 = \left(C_2^0 \otimes C_2^0\right) \bigcup \left(C_2^2 \otimes C_2^2\right) \tag{10.166}$$

$$C_4^4 = \left(C_2^0 \otimes C_2^2\right) \bigcup \left(C_2^2 \otimes C_2^0\right) \tag{10.167}$$

$$C_4^2 = \left(C_2^1 \otimes C_2^1\right) \bigcup \left(C_2^3 \otimes C_2^3\right) \tag{10.168}$$

$$C_4^6 = \left(C_2^1 \otimes C_2^3\right) \bigcup \left(C_2^3 \otimes C_2^1\right) \tag{10.169}$$

$$C_4^1 = \left(C_2^0 \otimes C_2^1\right) \bigcup \left(C_2^2 \otimes C_2^3\right) \tag{10.170}$$

$$C_4^5 = \left(C_2^0 \otimes C_2^3\right) \bigcup \left(C_2^2 \otimes C_2^1\right) \tag{10.171}$$

$$C_4^3 = \left(C_2^1 \otimes C_2^0\right) \bigcup \left(C_2^3 \otimes C_2^2\right) \tag{10.172}$$

$$C_4^7 = \left(C_2^1 \otimes C_2^2\right) \bigcup \left(C_2^3 \otimes C_2^0\right) \tag{10.173}$$

$$\bar{C}_4^0 = C_2^0 \otimes C_2^0 \tag{10.174}$$

$$\bar{C}_4^8 = C_2^2 \otimes C_2^2 \tag{10.175}$$

$$\bar{C}_4^4 = C_2^0 \otimes C_2^2 \tag{10.176}$$

$$\bar{C}_4^C = C_2^2 \otimes C_2^0 \tag{10.177}$$

$$\bar{C}_4^2 = C_2^1 \otimes C_2^1 \tag{10.178}$$

$$\bar{C}_4^A = C_2^3 \otimes C_2^3 \tag{10.179}$$

$$\bar{C}_4^6 = C_2^1 \otimes C_2^3 \tag{10.180}$$

$$\bar{C}_4^E = C_2^3 \otimes C_2^1 \tag{10.181}$$

$$\bar{C}_4^1 = C_2^0 \otimes C_2^1 \tag{10.182}$$

$$\bar{C}_4^9 = C_2^2 \otimes C_2^3 \tag{10.183}$$

$$\bar{C}_4^5 = C_2^0 \otimes C_2^3 \tag{10.184}$$

$$\bar{C}_4^D = C_2^2 \otimes C_2^1 \tag{10.185}$$

$$\bar{C}_4^3 = C_2^1 \otimes C_2^0 \tag{10.186}$$

$$\bar{C}_4^B = C_2^3 \otimes C_2^2 \tag{10.187}$$

$$\bar{C}_4^7 = C_2^1 \otimes C_2^2 \tag{10.188}$$

$$\bar{C}_4^F = C_2^3 \otimes C_2^0 \tag{10.189}$$

$$D_4^0 = (D_2^0 \otimes D_2^0) \bigcup (D_2^4 \otimes D_2^4) \tag{10.190}$$
$$D_4^{10} = (D_2^0 \otimes D_2^4) \bigcup (D_2^4 \otimes D_2^0) \tag{10.191}$$
$$D_4^8 = (D_2^2 \otimes D_2^2) \bigcup (D_2^6 \otimes D_2^6) \tag{10.192}$$
$$D_4^{18} = (D_2^2 \otimes D_2^6) \bigcup (D_2^6 \otimes D_2^2) \tag{10.193}$$
$$D_4^4 = (D_2^0 \otimes D_2^2) \bigcup (D_2^4 \otimes D_2^6) \tag{10.194}$$
$$D_4^{14} = (D_2^0 \otimes D_2^6) \bigcup (D_2^4 \otimes D_2^2) \tag{10.195}$$
$$D_4^C = (D_2^2 \otimes D_2^0) \bigcup (D_2^6 \otimes D_2^4) \tag{10.196}$$
$$D_4^{1C} = (D_2^2 \otimes D_2^4) \bigcup (D_2^6 \otimes D_2^0) \tag{10.197}$$
$$D_4^2 = (D_2^1 \otimes D_2^1) \bigcup (D_2^5 \otimes D_2^5) \tag{10.198}$$
$$D_4^{12} = (D_2^1 \otimes D_2^5) \bigcup (D_2^5 \otimes D_2^1) \tag{10.199}$$
$$D_4^A = (D_2^3 \otimes D_2^3) \bigcup (D_2^7 \otimes D_2^7) \tag{10.200}$$
$$D_4^{1A} = (D_2^3 \otimes D_2^7) \bigcup (D_2^7 \otimes D_2^3) \tag{10.201}$$
$$D_4^6 = (D_2^1 \otimes D_2^3) \bigcup (D_2^5 \otimes D_2^7) \tag{10.202}$$
$$D_4^{16} = (D_2^1 \otimes D_2^7) \bigcup (D_2^5 \otimes D_2^3) \tag{10.203}$$
$$D_4^E = (D_2^3 \otimes D_2^1) \bigcup (D_2^7 \otimes D_2^5) \tag{10.204}$$
$$D_4^{1E} = (D_2^3 \otimes D_2^5) \bigcup (D_2^7 \otimes D_2^1) \tag{10.205}$$

$$D_4^1 = (D_2^0 \otimes D_2^1) \bigcup (D_2^4 \otimes D_2^5) \tag{10.206}$$
$$D_4^{11} = (D_2^0 \otimes D_2^5) \bigcup (D_2^4 \otimes D_2^1) \tag{10.207}$$
$$D_4^9 = (D_2^2 \otimes D_2^3) \bigcup (D_2^6 \otimes D_2^7) \tag{10.208}$$
$$D_4^{19} = (D_2^2 \otimes D_2^7) \bigcup (D_2^6 \otimes D_2^3) \tag{10.209}$$
$$D_4^5 = (D_2^0 \otimes D_2^3) \bigcup (D_2^4 \otimes D_2^7) \tag{10.210}$$
$$D_4^{15} = (D_2^0 \otimes D_2^7) \bigcup (D_2^4 \otimes D_2^3) \tag{10.211}$$
$$D_4^D = (D_2^2 \otimes D_2^1) \bigcup (D_2^6 \otimes D_2^5) \tag{10.212}$$
$$D_4^{1D} = (D_2^2 \otimes D_2^5) \bigcup (D_2^6 \otimes D_2^1) \tag{10.213}$$
$$D_4^3 = (D_2^1 \otimes D_2^0) \bigcup (D_2^5 \otimes D_2^4) \tag{10.214}$$
$$D_4^{13} = (D_2^1 \otimes D_2^4) \bigcup (D_2^5 \otimes D_2^0) \tag{10.215}$$
$$D_4^B = (D_2^3 \otimes D_2^2) \bigcup (D_2^7 \otimes D_2^6) \tag{10.216}$$
$$D_4^{1B} = (D_2^3 \otimes D_2^6) \bigcup (D_2^7 \otimes D_2^2) \tag{10.217}$$
$$D_4^7 = (D_2^1 \otimes D_2^2) \bigcup (D_2^5 \otimes D_2^6) \tag{10.218}$$
$$D_4^{17} = (D_2^1 \otimes D_2^6) \bigcup (D_2^5 \otimes D_2^2) \tag{10.219}$$
$$D_4^F = (D_2^3 \otimes D_2^0) \bigcup (D_2^7 \otimes D_2^4) \tag{10.220}$$

$$D_4^{1F} \quad = \quad \left(D_2^3 \otimes D_2^4\right) \bigcup \left(D_2^7 \otimes D_2^0\right) \tag{10.221}$$

In order to more completely understand the entries in the four-dimensional coding tables, we also summarize some partitioning results in four-dimensions in Table 10.14.

	A_4^0							
d_{min} w.r.t. A_4^0	1							
N_e w.r.t. A_4^0	8							
	B_4^0				B_4^1			
d_{min} w.r.t. B_4^0	$\sqrt{2}$				1			
N_e w.r.t. B_4^0	24				8			
	$\bar{B}_4^0$		$\bar{B}_4^2$		$\bar{B}_4^1$		$\bar{B}_4^3$	
d_{min} w.r.t. $\bar{B}_4^0$	$\sqrt{2}$		$\sqrt{2}$		1		1	
N_e w.r.t. $\bar{B}_4^0$	8		16		4		4	
	C_4^0	C_4^4	C_4^2	C_4^6	C_4^1	C_4^5	C_4^3	C_4^7
d_{min} w.r.t. C_4^0	2	$\sqrt{2}$	$\sqrt{2}$	$\sqrt{2}$	1	1	1	1
N_e w.r.t. C_4^0	24	8	8	8	2	2	2	2
	$\bar{C}_4^0$	$\bar{C}_4^4$	$\bar{C}_4^2$	$\bar{C}_4^6$	$\bar{C}_4^1$	$\bar{C}_4^5$	$\bar{C}_4^3$	$\bar{C}_4^7$
d_{min} w.r.t. $\bar{C}_4^0$	2	$\sqrt{2}$	$\sqrt{2}$	$\sqrt{2}$	1	1	1	$\sqrt{3}$
N_e w.r.t. $\bar{C}_4^0$	8	4	4	4	2	2	2	8
	$\bar{C}_4^8$	$\bar{C}_4^C$	$\bar{C}_4^A$	$\bar{C}_4^E$	$\bar{C}_4^9$	$\bar{C}_4^D$	$\bar{C}_4^B$	$\bar{C}_4^F$
d_{min} w.r.t. $\bar{C}_4^0$	2	$\sqrt{2}$	$\sqrt{2}$	$\sqrt{2}$	$\sqrt{3}$	$\sqrt{3}$	$\sqrt{3}$	1
N_e w.r.t. $\bar{C}_4^0$	16	4	4	4	8	8	8	2
	D_4^0	D_4^4	D_4^2	D_4^6	D_4^1	D_4^5	D_4^3	D_4^7
d_{min} w.r.t. D_4^0	$2\sqrt{2}$	$\sqrt{2}$	$\sqrt{2}$	$\sqrt{2}$	1	1	1	$\sqrt{3}$
N_e w.r.t. D_4^0	24	2	2	2	1	1	1	4
	D_4^8	D_4^C	D_4^A	D_4^E	D_4^9	D_4^D	D_4^B	D_4^F
d_{min} w.r.t. D_4^0	2	$\sqrt{2}$	$\sqrt{2}$	$\sqrt{2}$	$\sqrt{3}$	$\sqrt{3}$	$\sqrt{3}$	1
N_e w.r.t. D_4^0	8	2	2	2	4	4	4	1
	D_4^{10}	D_4^{14}	D_4^{12}	D_4^{16}	D_4^{11}	D_4^{15}	D_4^{13}	D_4^{17}
d_{min} w.r.t. D_4^0	2	$\sqrt{2}$	$\sqrt{2}$	$\sqrt{2}$	1	1	1	$\sqrt{3}$
N_e w.r.t. D_4^0	8	2	2	2	1	1	1	4
	D_4^{18}	D_4^{1C}	D_4^{1A}	D_4^{1E}	D_4^{19}	D_4^{1D}	D_4^{1B}	D_4^{1F}
d_{min} w.r.t. D_4^0	2	$\sqrt{2}$	$\sqrt{2}$	$\sqrt{2}$	$\sqrt{3}$	$\sqrt{3}$	$\sqrt{3}$	1
N_e w.r.t. D_4^0	8	2	2	2	4	4	4	1

Table 10.14: Four-Dimensional Partitioning

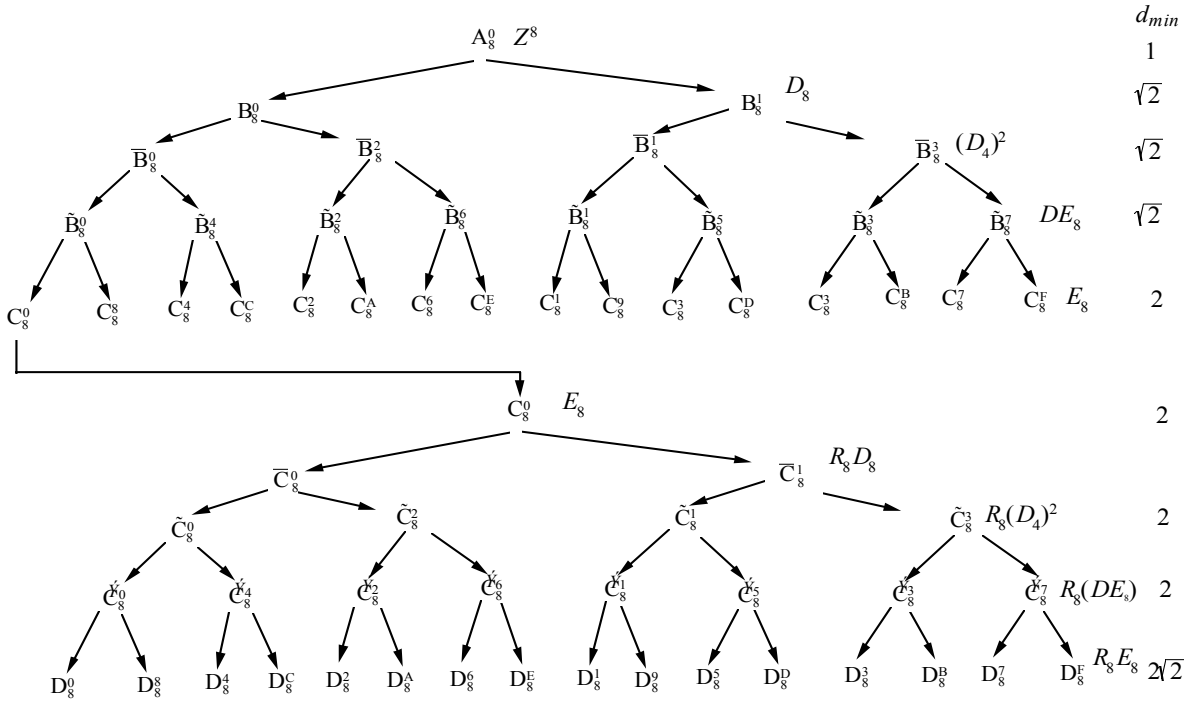


Figure 10.30: Eight-dimensional partition tree with Ungerboeck labeling.

The subscript on a partition refers to the dimension of the underlying coset code, while the superscript refers to the coset index with an Ungerboeck labeling.

Figure 10.30 is a partition tree showing the specific labels for a eight-dimensional partitioning with Ungerboeck labeling. Each of the cosets of the original constellation in Figure 10.30 can be written as (possibly unions of) Cartesian products of two-dimensional cosets in the partitioning of the Z^4 lattice. Those Cartesian products are (for reference):

$$A_8^0 = A_4^0 \otimes A_4^0 \tag{10.222}$$
$$B_8^0 = \left(B_4^0 \otimes B_4^0\right) \bigcup \left(B_4^1 \otimes B_4^1\right) \tag{10.223}$$
$$B_8^1 = \left(B_4^0 \otimes B_4^1\right) \bigcup \left(B_4^1 \otimes B_4^0\right) \tag{10.224}$$
$$\bar{B}_8^0 = B_4^0 \otimes B_4^0 \tag{10.225}$$
$$\bar{B}_8^2 = B_4^1 \otimes B_4^1 \tag{10.226}$$
$$\bar{B}_8^1 = B_4^0 \otimes B_4^1 \tag{10.227}$$
$$\bar{B}_8^3 = B_4^1 \otimes B_4^0 \tag{10.228}$$

$$\tilde{B}_8^0 = \left(\bar{B}_4^0 \otimes \bar{B}_4^0\right) \bigcup \left(\bar{B}_4^2 \otimes \bar{B}_4^2\right) \tag{10.229}$$
$$\tilde{B}_8^4 = \left(\bar{B}_4^0 \otimes \bar{B}_4^2\right) \bigcup \left(\bar{B}_4^2 \otimes \bar{B}_4^0\right) \tag{10.230}$$
$$\tilde{B}_8^2 = \left(\bar{B}_4^1 \otimes \bar{B}_4^1\right) \bigcup \left(\bar{B}_4^3 \otimes \bar{B}_4^3\right) \tag{10.231}$$
$$\tilde{B}_8^6 = \left(\bar{B}_4^1 \otimes \bar{B}_4^3\right) \bigcup \left(\bar{B}_4^3 \otimes \bar{B}_4^1\right) \tag{10.232}$$
$$\tilde{B}_8^1 = \left(\bar{B}_4^0 \otimes \bar{B}_4^1\right) \bigcup \left(\bar{B}_4^2 \otimes \bar{B}_4^3\right) \tag{10.233}$$
$$\tilde{B}_8^5 = \left(\bar{B}_4^0 \otimes \bar{B}_4^3\right) \bigcup \left(\bar{B}_4^2 \otimes \bar{B}_4^1\right) \tag{10.234}$$

$$\tilde{B}_8^3 = (\bar{B}_4^1 \otimes \bar{B}_4^0) \bigcup (\bar{B}_4^3 \otimes \bar{B}_4^2) \quad (10.235)$$

$$\tilde{B}_8^7 = (\bar{B}_4^1 \otimes \bar{B}_4^2) \bigcup (\bar{B}_4^3 \otimes \bar{B}_4^0) \quad (10.236)$$

$$\begin{aligned}
C_8^0 &= (C_4^0 \otimes C_4^0) \bigcup (C_4^4 \otimes C_4^4) \bigcup (C_4^2 \otimes C_4^2) \bigcup (C_4^6 \otimes C_4^6) \\
C_8^8 &= (C_4^0 \otimes C_4^4) \bigcup (C_4^4 \otimes C_4^0) \bigcup (C_4^2 \otimes C_4^6) \bigcup (C_4^6 \otimes C_4^2) \\
C_8^4 &= (C_4^0 \otimes C_4^2) \bigcup (C_4^4 \otimes C_4^6) \bigcup (C_4^2 \otimes C_4^0) \bigcup (C_4^6 \otimes C_4^4) \\
C_8^C &= (C_4^0 \otimes C_4^6) \bigcup (C_4^4 \otimes C_4^2) \bigcup (C_4^2 \otimes C_4^4) \bigcup (C_4^6 \otimes C_4^0) \\
C_8^2 &= (C_4^1 \otimes C_4^1) \bigcup (C_4^5 \otimes C_4^5) \bigcup (C_4^3 \otimes C_4^3) \bigcup (C_4^7 \otimes C_4^7) \\
C_8^A &= (C_4^1 \otimes C_4^5) \bigcup (C_4^5 \otimes C_4^1) \bigcup (C_4^3 \otimes C_4^7) \bigcup (C_4^7 \otimes C_4^3) \\
C_8^6 &= (C_4^1 \otimes C_4^3) \bigcup (C_4^5 \otimes C_4^7) \bigcup (C_4^3 \otimes C_4^1) \bigcup (C_4^7 \otimes C_4^5) \\
C_8^E &= (C_4^1 \otimes C_4^7) \bigcup (C_4^5 \otimes C_4^3) \bigcup (C_4^3 \otimes C_4^5) \bigcup (C_4^7 \otimes C_4^1) \\
C_8^1 &= (C_4^0 \otimes C_4^1) \bigcup (C_4^4 \otimes C_4^5) \bigcup (C_4^2 \otimes C_4^3) \bigcup (C_4^6 \otimes C_4^7) \\
C_8^9 &= (C_4^0 \otimes C_4^5) \bigcup (C_4^4 \otimes C_4^1) \bigcup (C_4^2 \otimes C_4^7) \bigcup (C_4^6 \otimes C_4^3) \\
C_8^5 &= (C_4^0 \otimes C_4^3) \bigcup (C_4^4 \otimes C_4^7) \bigcup (C_4^2 \otimes C_4^1) \bigcup (C_4^6 \otimes C_4^5) \\
C_8^D &= (C_4^0 \otimes C_4^7) \bigcup (C_4^4 \otimes C_4^3) \bigcup (C_4^2 \otimes C_4^5) \bigcup (C_4^6 \otimes C_4^1) \\
C_8^3 &= (C_4^1 \otimes C_4^0) \bigcup (C_4^5 \otimes C_4^4) \bigcup (C_4^3 \otimes C_4^2) \bigcup (C_4^7 \otimes C_4^6) \\
C_8^B &= (C_4^1 \otimes C_4^4) \bigcup (C_4^5 \otimes C_4^0) \bigcup (C_4^3 \otimes C_4^6) \bigcup (C_4^7 \otimes C_4^2) \\
C_8^7 &= (C_4^1 \otimes C_4^2) \bigcup (C_4^5 \otimes C_4^6) \bigcup (C_4^3 \otimes C_4^0) \bigcup (C_4^7 \otimes C_4^4) \\
C_8^F &= (C_4^1 \otimes C_4^6) \bigcup (C_4^5 \otimes C_4^2) \bigcup (C_4^3 \otimes C_4^4) \bigcup (C_4^7 \otimes C_4^0)
\end{aligned}$$

For the reader's and designer's assistance, Table 10.15 lists the distances and nearest neighbor counts for eight-dimensional partitioning

10.5.2 Multidimensional Trellis Codes

This section returns to the coset-code encoder, which is re-illustrated in Figure 10.31. Typically, $r_G = 1$, although there are a few (mainly impractical) codes for which $r_G > 1$. Thus, signal expansion is over many dimensions, and thus there is a smaller constellation expansion over any particular dimension. However, as determined in Section 10.5.1, more levels of partitioning will be necessary to increase distance on the parallel transitions defined by Λ' and its cosets to an attractive level. The lower signal expansion per dimension is probably the most attractive practical feature of multi-dimensional codes. Constellation expansion makes the coded signals more susceptible to channel nonlinearity and carrier jitter in QAM. Constellation expansion can also render decision-directed (on a symbol-by-symbol basis) timing and carrier loops, as well as the decision-feedback equalizer, sub-desirable in their performance due to increased symbol-by-symbol error rates.

Multidimensional codes can be attractive because the computation required to implement the Viterbi Detector is distributed over a longer time interval, usually resulting in a slight computational reduction (only slight, as we shall see that parallel transition resolving in the computation of the branch metrics becomes more difficult). One particularly troublesome feature of multidimensional trellis codes is, however, a propensity towards high nearest neighbor counts, with the resultant significant decrease in γ_f to $\tilde{\gamma}_f$.

Subsection 10.5.2 introduces a few simple examples of multidimensional trellis codes. Subsection 10.5.2 lists the most popular 4 dimensional codes in tabular form, similar to the tables in Section 10.4.3. Subsection 10.5.2 lists the most popular 8 dimensional codes.

	A_8^0							
d_{min} w.r.t. A_8^0	1							
N_e w.r.t. A_8^0	16							
	B_8^0				B_8^1			
d_{min} w.r.t. B_8^0	$\sqrt{2}$				1			
N_e w.r.t. B_8^0	132				16			
	$\bar{B}_8^0$		$\bar{B}_8^2$		$\bar{B}_8^1$		$\bar{B}_8^3$	
d_{min} w.r.t. $\bar{B}_8^0$	$\sqrt{2}$		$\sqrt{2}$		1		1	
N_e w.r.t. $\bar{B}_8^0$	48		64		8		8	
	$\tilde{B}_8^0$	$\tilde{B}_8^4$	$\tilde{B}_8^2$	$\tilde{B}_8^6$	$\tilde{B}_8^1$	$\tilde{B}_8^5$	$\tilde{B}_8^3$	$\tilde{B}_8^7$
d_{min} w.r.t. $\tilde{B}_8^0$	$\sqrt{2}$	$\sqrt{2}$	$\sqrt{2}$	$\sqrt{2}$	1	1	1	1
N_e w.r.t. $\tilde{B}_8^0$	16	32	32	32	4	4	4	4
	C_8^0	C_8^4	C_8^2	C_8^6	C_8^1	C_8^5	C_8^3	C_8^7
d_{min} w.r.t. C_8^0	2	$\sqrt{2}$	$\sqrt{2}$	$\sqrt{2}$	1	1	1	$\sqrt{3}$
N_e w.r.t. C_8^0	240	16	16	16	2	2	2	2
	C_8^8	C_8^C	C_8^A	C_8^E	C_8^9	C_8^D	C_8^B	C_8^F
d_{min} w.r.t. C_8^0	$\sqrt{2}$	$\sqrt{2}$	$\sqrt{2}$	$\sqrt{2}$	1	1	1	1
N_e w.r.t. C_8^0	16	16	16	16	2	2	2	2

Table 10.15: Eight-Dimensional Partitioning

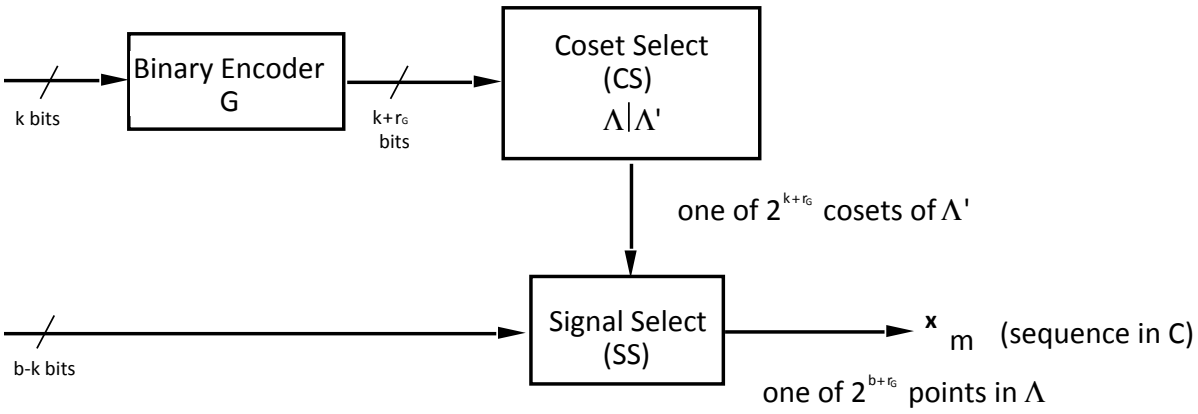


Figure 10.31: The coset-code encoder.

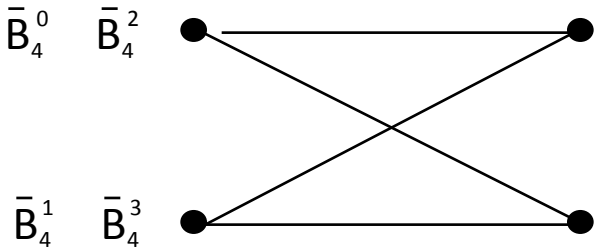


Figure 10.32: Trellis for 2-state, rate 1/2, 4D code.

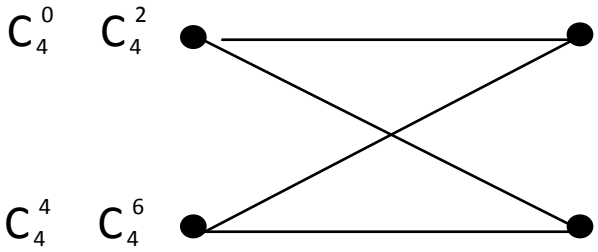


Figure 10.33: Trellis for 2-state, rate 1/2, 4D Code, based on D_4.

Multidimensional Trellis Code Examples

EXAMPLE 10.5.1 (2-state, rate 1/2, 4D Code) The code uses $\Lambda = Z^4$ and $\Lambda' = R_4 Z^4$. The two-state trellis is shown in Figure 10.32. The labels for the various subsets are as indicated in the tables of Section 10.5.1. The minimum distance is given by $d^2_{min} = 2$, and $r_{\bar{C}} = 0 + \frac{1}{4}$. Thus, the fundamental gain is

$$\gamma_f = \frac{2}{2^{2 \cdot 1/4}} = \sqrt{2} \ \ (1.5 \text{ dB}) \quad . \tag{10.237}$$

This gain is no better than the D_4 gain, which required no states. However, $\bar{N}_e(D_4) = 6$, whereas for this code $\bar{N}_e = 2$, so the effective gain for the D_4 lattice code is about 1.2dB, while it is a full 1.5dB for this 2-state trellis code.

EXAMPLE 10.5.2 (2-state, rate 1/2, 4D Code, based on D_4) The code uses $\Lambda = D_4$ and $\Lambda' = R_4 D_4$. The two-state trellis is shown in Figure 10.33. The labels for the various subsets are as indicated on the tables in Section 10.5.1. The minimum distance is given by $d^2_{min} = 4$, and $r_{\bar{C}} = \frac{1}{4} + \frac{1}{4}$.[5] Thus, the fundamental gain is

$$\gamma_f = \frac{4}{2^{2 \cdot 1/2}} = 2 \ \ (3.01 \text{ dB}) \quad . \tag{10.238}$$

This gain is better than the D_4 gain, which required no states. However, $\bar{N}_e = 22 \quad (= \frac{24+8\times 8}{4})$, so the effective gain is about 2.31 dB for this code.

EXAMPLE 10.5.3 (Wei's 8-state, rate 2/3, 4D Code) The code uses $\Lambda = Z^4$ and $\Lambda' = R_4 D_4$. The 8-state trellis is shown in Figure 10.34. The labels for the various subsets are as indicated on the tables in Section 10.5.1. The minimum distance is given by $d^2_{min} = 4$, and $r_{\bar{C}} = 0 + \frac{1}{4}$. Thus, the fundamental gain is

$$\gamma_f = \frac{4}{2^{2 \cdot 1/4}} = 2^{1.5} \ \ (4.52 \text{ dB}) \quad . \tag{10.239}$$

[5]Note that $r_{\bar{\Lambda}} = 1/4$ for $\Lambda = D_4$.

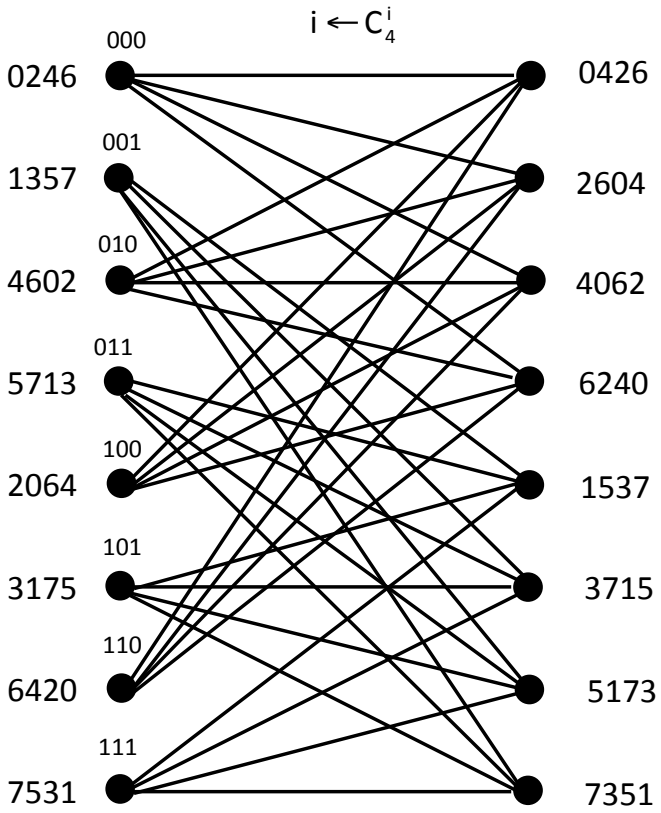


Figure 10.34: Trellis for Wei's 8-state, rate 2/3, 4D Code (same as 8-state 2 dimensional Ungerboeck trellis, except branch index of i stands for coset C_4^i)

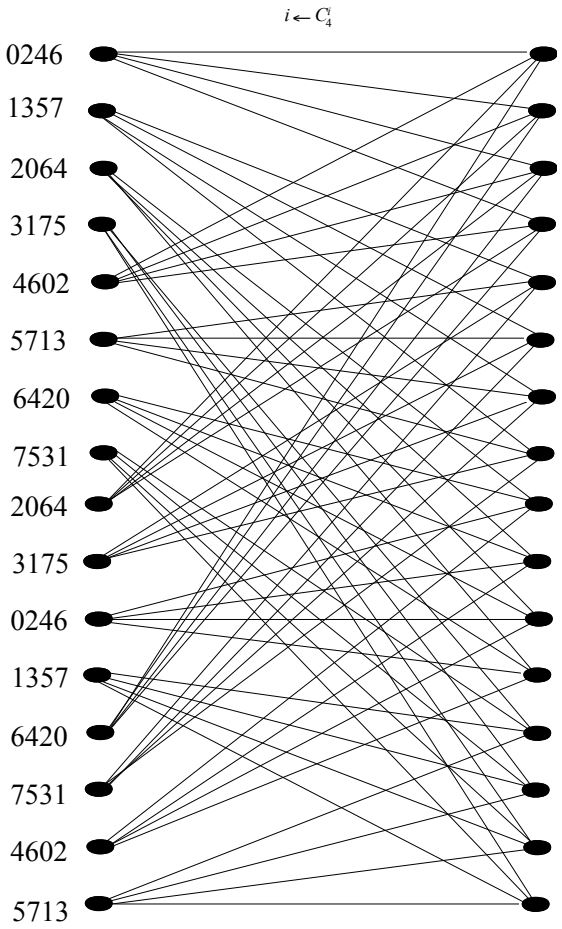


Figure 10.35: Trellis for Wei's 16-state, rate 2/3, 4D code.

This gain is yet better. However, $\bar{N}_e = 22$ $(= \frac{24+8\times 8}{4})$, so the effective gain is about 3.83 dB for this code.

EXAMPLE 10.5.4 (Wei's 16-state, rate 2/3, 4D Code) The code uses $\Lambda = Z^4$ and $\Lambda' = R_4 D_4$. The 16-state trellis is shown in Figure 10.35. The minimum distance is given by $d^2_{min} = 4$, and $\bar{r_C} = \frac{1}{4} + 0$. Thus, the fundamental gain is

$$\gamma_f = \frac{4}{2^{2\cdot 1/4}} = 2^{1.5} \ \ (4.52 \text{ dB}) \quad . \tag{10.240}$$

This gain is the same as for 8 states. However, $\bar{N}_e = 6$, so the effective gain is about 4.2 dB for this code, which is better than the 8-state code. This code is the most commonly found code in systems that do use a 4-dimensional trellis code and has been standardized as one option in the CCITT V.fast code for 28.8 Kbps (uncompressed) voiceband modems (with minor modification, see Section 10.7 and also for Asymmetric Digital Subscriber Line (ADSL) transceivers. There is a 32-state 4-dimensional Wei code with $\bar{N}_e = 2$, so that its effective gain is the full 4.52dB.

EXAMPLE 10.5.5 (4-state, rate 2/4, 8D Code) The code uses $\Lambda = E_8$ and $\Lambda' = R_8 E_8$. The two-state trellis is shown in Figure 10.36. The minimum distance is given by

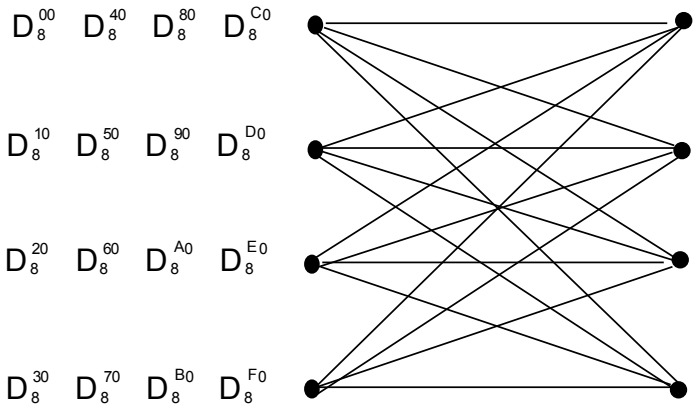


Figure 10.36: Trellis for 4-state, rate 2/4, 8D Code

$d_{min}^2 = 8$, and $\bar{r_C} = \frac{4}{8} + \frac{2}{8} = .75$.[6] Thus, the fundamental gain is

$$\gamma_f = \frac{8}{2^{2 \cdot 3/4}} = 2^{1.5} \ \ (4.52 \text{ dB}) \quad . \tag{10.241}$$

This gain is better than for any other 4-state code presented so far. However, $\bar{N}_e = 126$ $(= \frac{240+3(16\times16)}{8})$, so the effective gain is about 3.32 dB for this code, which is still better than the 4-state 2-dimensional 3 dB code with effective gain 3.01 dB.

EXAMPLE 10.5.6 (Wei's 16-state, rate 3/4, 8D Code) The code uses $\Lambda = Z^8$ and $\Lambda' = E_8$. The 16-state trellis is shown in Figure 10.37. The labels for the various subsets are as indicated on the tables in Section 10.5.1. The minimum distance is given by $d_{min}^2 = 4$, and $\bar{r_C} = \frac{1}{8} + 0 = .125$. Thus, the fundamental gain is

$$\gamma_f = \frac{4}{2^{2 \cdot 1/8}} = 2^{1.75} \ \ (5.27 \text{ dB}) \quad . \tag{10.242}$$

This gain is better than for any other 16-state code studied. However, $\bar{N}_e = 158$, so the effective gain is about 3.96 dB. There is a 64-state Wei code that is also $\gamma_f = 5.27$ dB, but with $\bar{N}_e = 30$, so that $\tilde{\gamma}_f = 4.49$ dB. This 64-state code (actually a differentially phase-invariant modification of it) is used in some commercial (private-line) 19.2kbps voiceband modems.

4D Code Table

Table 10.16 is similar to Tables 10.6 and 10.8, except that it is for 4 dimensional codes. Table 10.16 lists up to rate 4/5 codes. The rate can be inferred from the number of nonzero terms h_i in the table. The design specifies Λ and Λ' in the 4D case, which also appears in Table 10.16. $\bar{N}_1$ and $\bar{N}_2$ are not shown for these codes. $\bar{N}_1$ and $\bar{N}_2$ can be safely ignored because an increase in d_{min} in the tables by 1 or 2 would be very significant and the numbers of nearest neighbors would have to be very large on paths with $d > d_{min}$ in order for their performance to dominate the union bound for P_e.

8D Code Table

Table 10.17 is similar to Tables 10.6 and 10.8, except that it is for 8 dimensional codes. Table 10.17 lists up to rate 4/5 codes. The rate can again be inferred from the number of nonzero terms h_i in the table.

[6] Note that $\bar{r_\Lambda} = 4/8$ for $\Lambda = E_8$.

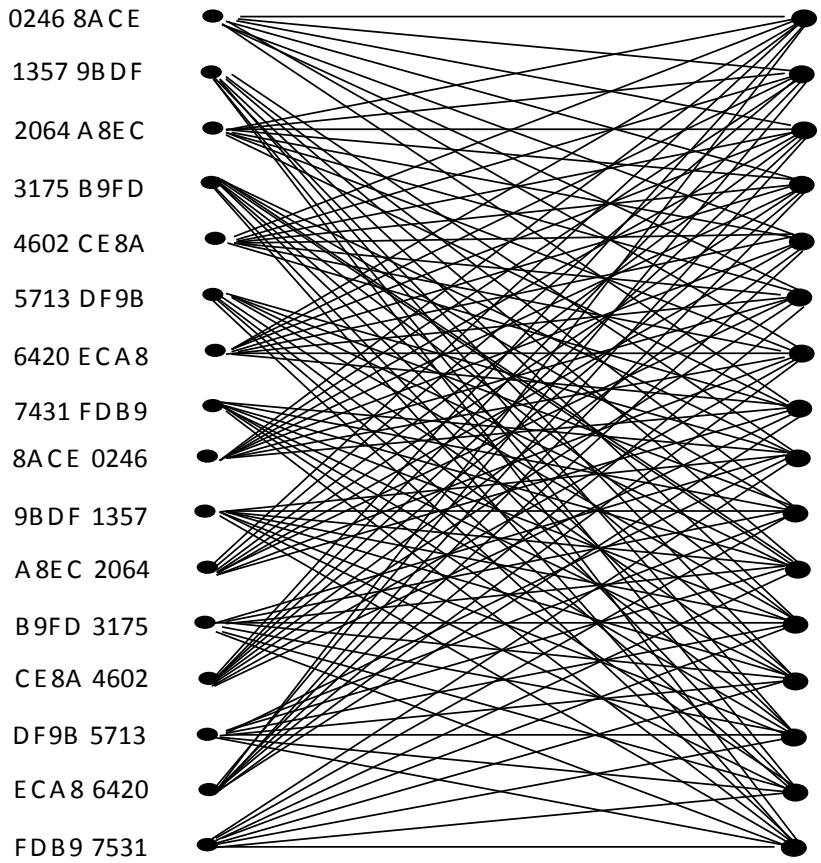


Figure 10.37: Trellis for Wei's 16-state, rate 3/4, 8D code.

Λ	Λ'	2^ν	h_4	h_3	h_2	h_1	h_0	d^2_{min}	γ_f	(dB)	$\bar{N}_e$	$\tilde{\gamma}_f$	$\bar{N}_D$
Z^4	R_4D_4	8	–	–	02	04	11	4	$2^{3/2}$	4.52	22	3.82	22
D_4	$2D_4$	16	–	10	04	02	21	6	3	4.77	88	3.88	76
Z^4	R_4D_4	16	–	–	14	02	21	4	$2^{3/2}$	4.52	6	4.20	36
Z^4	$2Z^4$	32	–	30	14	02	41	4	$2^{3/2}$	4.52	2	4.52	122
D_4	$2D_4$	64	–	050	014	002	121	6	3	4.77	8	4.37	256
Z^4	$2D_4$	64	050	030	014	002	101	5	$\frac{5}{\sqrt{2}}$	5.48	36	4.65	524
Z^4	$2D_4$	128	120	050	022	006	203	6	$\frac{6}{\sqrt{2}}$	6.28	364	4.77	1020

Table 10.16: Four-Dimensional Trellis Codes and Parameters

Λ	Λ'	2^ν	h_4	h_3	h_2	h_1	h_0	d^2_{min}	γ_f	(dB)	$\bar{N}_e$	$\tilde{\gamma}_f$	$\bar{N}_D$
E_8	R_8E_8	8	–	10	04	02	01	8	$2^{7/4}$	5.27	382	3.75	45
E_8	R_8E_8	16	–	10	04	02	21	8	$2^{7/4}$	5.27	158	4.01	60
Z^8	E_8	16	–	10	04	02	21	4	$2^{7/4}$	5.27	158	4.01	52
E_8	$2E_8$	16	r=4/8	-?-	-?-	-?-	-?-	16	4	6.02	510	4.42	342
E_8	R_8E_8	32	–	30	14	02	61	8	$2^{7/4}$	5.27	62	4.28	90
Z^8	E_8	32	–	10	04	02	41	4	$2^{7/4}$	5.27	62	4.28	82
E_8	R_8E_8	64	–	050	014	002	121	8	$2^{7/4}$	5.27	30	4.49	150
Z^8	E_8	64	–	050	014	002	121	4	$2^{7/4}$	5.27	30	4.49	142
Z^8	R_8D_8	128	120	044	014	002	101	8	$2^{7/4}$	5.27	14	4.71	516
E_8	$2E_8$	256	r=4/8	-?-	-?-	-?-	-?-	16	4	6.02	30	5.24	1272

Table 10.17: Eight-Dimensional Trellis Codes and Parameters

Table 10.17 specifies Λ and Λ' in the 8D case. $\bar{N}_1$ and $\bar{N}_2$ are also not shown for these codes, although the author suspects that $\bar{N}_1$ could dominate for the Z^8/R_8D_8 128-state code.

10.6 Theory of the Coset Code Implementation

This section investigates the simplification of the implementation of both the encoder and the decoder for a multidimensional trellis code.

10.6.1 Encoder Simplification

The general coset-code encoder diagram is useful mainly for illustrative purposes. Actual implementations using such a diagram as a guideline would require excessive memory for the implementation of the coset select and signal select functions. In practice, the use of two-dimensional constituent subsymbols in Z^2 is possible, even with codes that make use of more dense multidimensional lattices. All the lattices discussed in this Chapter and used are known as **binary lattices**, which means that they contain $2Z^N$ as a sublattice. With such binary lattices, additional partitioning can enlarge the number of cosets so that $2Z^N$ and its cosets partition the original lattice Z^N upon which the constellation was based. The number of additional binary partitions needed to get from Λ' to $2Z^N$ is known as the **informativity** of Λ', $k(\Lambda')$. This quantity is often normalized to the number of dimensions to get the **normalized informativity** of the lattice, $\bar{\kappa}(\Lambda) = k(\Lambda')/N$.

The addition of the extra partitioning bits to the coset code's convolutional encoder and coset selection creates a rate $[k + k(\Lambda')] \,/\, [k + k(\Lambda') + r_G]$ convolutional code that selects cosets of $2Z^N$. These cosets can then be trivially separated into two-dimensional cosets of $2Z^2$ (by undoing the concatenation). These 2D cosets can then can be independently specified by separate (reduced complexity and memory) two-dimensional signal selects. Wei's 16-state 4D code and 64-state 8D codes will illustrate this concept for multidimensional codes.

4D Encoder with rate 2/3, and Z^4/R_4D_4 An example is the encoder for a four-dimensional trellis code that transmits 10 bits per 4D symbol, or $b = 10$. The redundant extra bit from G ($r_G = 1$) requires a signal constellation with 2^{11} points in 4 dimensions. The encoder uses an uncoded constellation that is the Cartesian product of two 32CR constellations. 32CR is a subset of Z^2, and the uncoded constellation would be a subset of Z^4. A desirable implementation keeps the redundancy (extra levels) in the two 2D constituent sub-symbols as small as practically possible. The extension of 32CR to what is called 48CR appears in Figure 10.38. 16 new points are added at the lowest 2D energy positions outside the 32CR to form 48CR. The 32CR points are denoted as "inner" points and the 16 new points are denoted as "outer" points. The Cartesian product of the two 48CR (a subset of Z^4) with deletion of those points that correspond to (outer, outer) form the coded constellation. This leaves 2^{10} (inner, inner) points, 2^9 (inner, outer) points, and 2^9 (outer, inner) points for a total of $2^{10} + 2 \cdot 2^9 = 2^{10} + 2^{10} = 2^{11} = 2048$ points. The probability of an inner point occuring is 3/4, while the probability of an outer is 1/4. The **two-dimensional** symbol energy is thus

$$\mathcal{E}_{\boldsymbol{x}}(two-dimensional) = \frac{3}{4}E_{32CR} + \frac{1}{4}E_{outer} = .75(5) + .25(\frac{2 \cdot 12.5 + 12.5 + 14.5}{4}) = 7 \quad , \quad (10.243)$$

so that $\bar{\mathcal{E}}_{\boldsymbol{x}} = 7/2$. The shaping gain for 48CR is then

$$\gamma_s = \frac{2^{2(1/4)} \cdot (2^5 - 1)}{12 \cdot (7/2)} = \frac{43.84}{42} = 1.0438 = .19 \; dB \quad . \quad (10.244)$$

The shaping gain of 32CR has already been computed as .14dB, thus the net gain in shaping is only about .05dB - however, a nominal deployment of a two-dimensional code would have required 64 points, leading to more constellation expansion, which might be undesirable. Since the code G is rate 2/3, the 3 output bits of G are used to select one of the 8 four-dimensional cosets, $C_{4,0}$, ..., $C_{4,7}$, and the 8 remaining bits would specify which of the parallel transitions in the selected $C_{4,i}$ would be transmitted, requiring a look-up table with 256 locations for each coset. Since there are 8 such cosets, the decoder would nominally need $8 \times 256 = 2048$ locations in a look-up table to implement the mapping from bits to transmitted signal. Each location in the table would contain 4 dimensions of a symbol. The large memory requirement can be mitigated to a large extent by using the structure illustrated in Figure

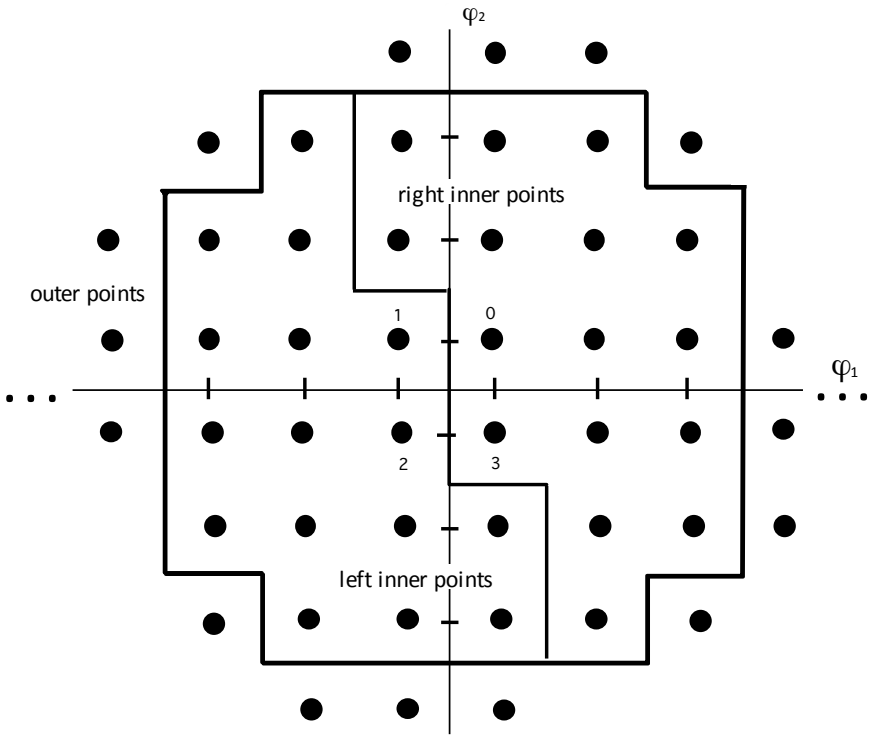


Figure 10.38: 48 Cross constellation.

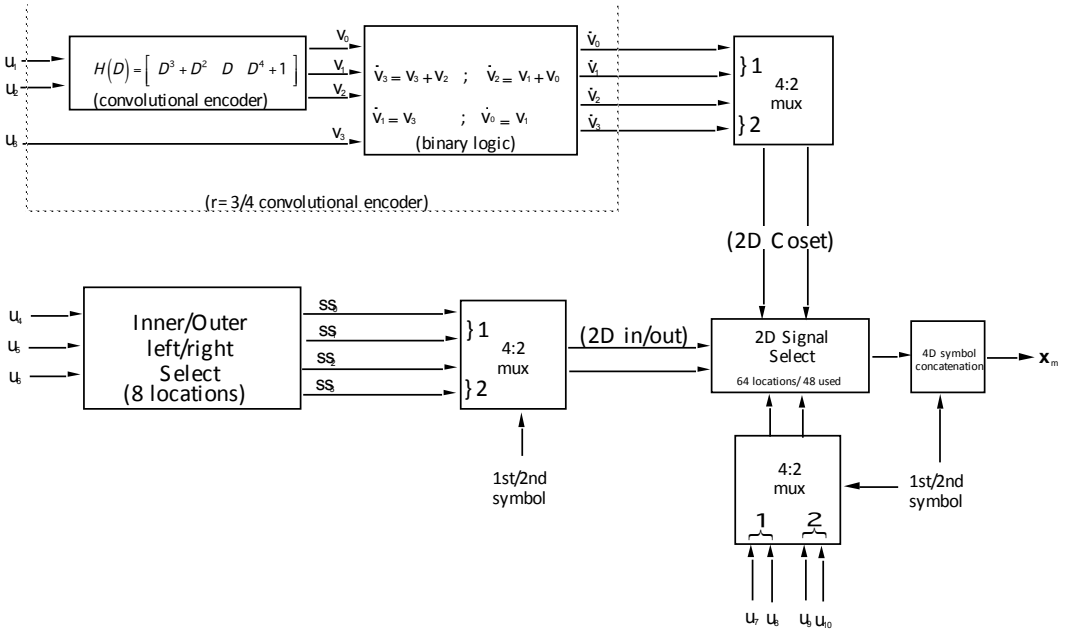


Figure 10.39: 4D coset-select implementation.

Input Bits			1^{st} 2D subsym.			2^{nd} 2D subsym.		
u_4	u_5	u_6	ss_1	ss_0	position	ss_3	ss_2	position
0	0	0	0	0	left-inner	0	0	left-inner
0	0	1	0	0	left-inner	0	1	right-inner
0	1	0	0	0	left-inner	1	0	outer
0	1	1	0	1	right-inner	0	0	left-inner
1	0	0	0	1	right-inner	0	1	right-inner
1	0	1	0	1	right-inner	1	0	outer
1	1	0	1	0	outer	0	0	left-inner
1	1	1	1	0	outer	0	1	right-inner

Table 10.18: Truth Table for 4D inner/outer selection (even b)

10.39. Because of the partitioning structure, each 4D coset of $C_{4,0}$ can be written as the union of two Cartesian products, for instance

$$C_4^0 = \left(C_2^0 \otimes C_2^0\right) \bigcup \left(C_2^2 \otimes C_2^2\right) \quad , \tag{10.245}$$

as in Section 10.5. Bit v_3 specifies which of these two Cartesian products $\left(C_2^0 \otimes C_2^0\right)$ or $\left(C_2^2 \otimes C_2^2\right)$ contains the selected signal. These two Cartesian products are now in the desired form of cosets in $2Z^4$. Thus, $k(R_4D^4) = 1$. The four bits v_0 ,..., v_3 can then be input into binary linear logic to form $\bar{v}_0$,..., $\bar{v}_3$. These four output bits then specify which 2D coset C_2^i, $i = 0, 1, 2, 3$ is used on each of the constituent 2D subsymbols that are concatenated to form the 4D code symbol. A clock of speed $2/T$ is used to control a 4:2 multiplexer that chooses bits $\bar{v}_0$ and $\bar{v}_1$ for the first 2D subsymbol and the bits $\bar{v}_2$ and $\bar{v}_3$ for the second 2D subsymbol within each symbol period. One can verify that the relations

$$\bar{v}_3 = v_2 + v_3 \tag{10.246}$$

$$\bar{v}_2 = v_0 + v_1 \tag{10.247}$$

$$\bar{v}_1 = v_3 \tag{10.248}$$

$$\bar{v}_0 = v_1 \tag{10.249}$$

will ensure that 4D coset, C_4^i, with i specified by $[v_2, v_1, v_0]$ is correctly translated into the two constituent 2D cosets that comprise that particular 4D coset.

The remaining input bits $(u_4, ..., u_{10})$ then specify points in each of the two subsymbols. The inner/outer/left/right select described in the Table 10.18 takes advantage of the further separation of each 2D coset into 3 equal-sized groups, left-inner, right-inner, and outer as illustrated in Figure 10.39. Note the left inner and right inner sets are chosen to ensure equal numbers of points from each of the 4 two-dimensional cosets. The outputs of this (nonlinear) bit map are then also separated into constituent 2D subsymbols by a multiplexer and the remaining 4 bits (u_7, u_8, u_9, u_{10}) are also so separated. The final encoder requires only 8 locations for the inner/outer selection and 64 locations (really only 48 are used because some of the combinations for the ss (signal select) bits do not occur) for the specification of each constituent subsymbol. This is a total of only 72 locations, significantly less than 2048, and only 2 dimensions of a symbol are stored in each location. The size of the 64-location 2D Signal Select is further reduced by Wei's observation that the 4 two-dimensional cosets can be generated by taking any one of the cosets, and performing sign permutations on the two dimensions within the subsymbol. Further, by storing an appropriate value (with signs) in a look-table, the necessary point can be generated by permuting signs according to the two bits being supplied for each 2D subsymbol from the CS function. Then the 2D signal selection would require only 16 locations, instead of 64. Then, the memory requirement would be (with a little extra logic to do the sign changes) 16+8=24 locations. We note $b \geq 6$ and b must be even for this encoder to work (while for $b < 6$ the encoder is trivial via look-up table with 2^{b+1} locations). For even $b > 6$, only the 2D Signal Select changes, and the memory size (using Wei's sign method to reduce by a factor of 4) is $4 \times 2^{\frac{b-6}{2}}$. For odd b, $b+1$ is even, the constellation is square, and the inner/outer/left/right partitioning of the constituent 2D subsymbols is unnecessary.

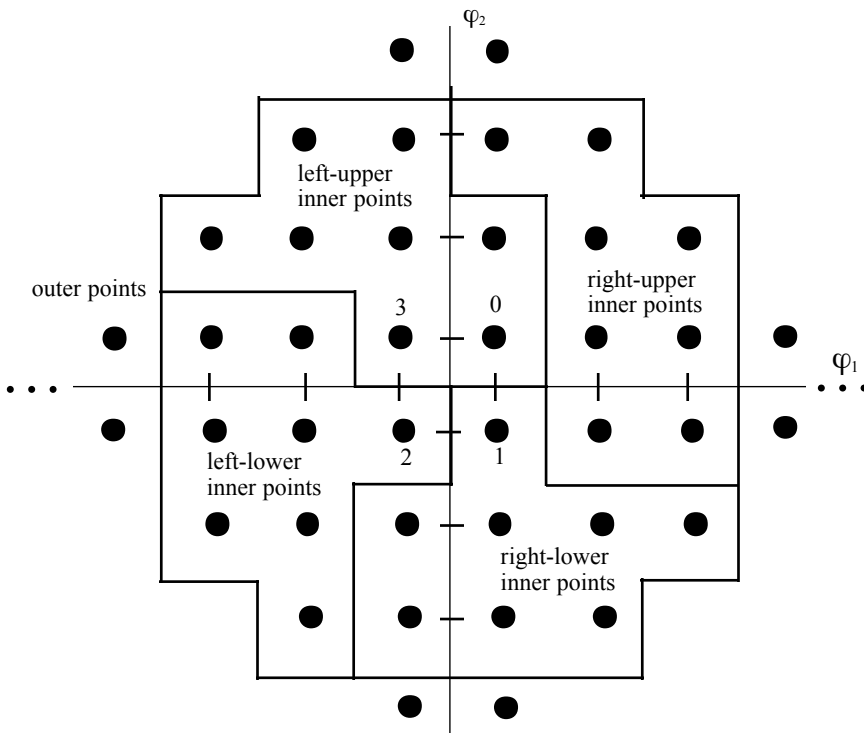


Figure 10.40: 40 Cross constellation.

8D Encoder with rate 3/4, and Z^8/E_8 An encoder for $\bar{b} = 2.5$ and a 64-state eight-dimensional trellis code transmits 20 bits per 8D symbol, or $b = 20$. The redundant extra bit from G ($r_G = 1$) requires a signal constellation with 2^{21} points in 8 dimensions. The uncoded constellation is the Cartesian product of four 32CR constellations. Note that 32CR is a subset of Z^2, and our uncoded constellation would be a subset of Z^8. A desirable implementation keeps the redundancy (extra levels) in the two 2D constituent sub-symbols as small as is practically possible. 32CR extends to what is called 40CR and shown in Figure 10.40. 8 new points are added at the lowest 2D energy positions outside the 32CR to form 40CR. The 32CR points are denoted as "inner" points and the 8 new points are denoted as "outer" points. The Cartesian product of the four 40CR (a subset of Z^8) with deletion of those points that have more than one outer point forms the 8-dimensional transmitted signal constellation. This leaves 2^{20} (in, in, in, in) points, 2^{18} (in, in, in, out) points, 2^{18} (in, in, out, in) points, 2^{18} (in, out, in, in) points, and 2^{18} (out, in, in, in) points for a total of $2^{20} + 4 \cdot 2^{18} = 2^{20} + 2^{20} = 2^{21} = 2,097,152$ points. The probability of an inner point occuring is 7/8, while the probability of an outer is 1/8. The **two-dimensional** symbol energy is thus

$$\mathcal{E}_{\boldsymbol{x}} = \frac{7}{8}E_{32CR} + \frac{1}{8}E_{outer} = .875(5) + .125(12.5) = 5.9375 \quad , \tag{10.250}$$

so that $\bar{\mathcal{E}}_{\boldsymbol{x}} = 5.9375/2$. The shaping gain for 40CR is then

$$\gamma_s = \frac{2^{2(1/8)} \cdot (2^5 - 1)}{12 \cdot (5.9375/2)} = \frac{36.8654}{35.625} = .15\ dB \quad . \tag{10.251}$$

The shaping gain of 32CR has already been computed as .14dB, thus the net gain in shaping is only about .01dB - however, a nominal deployment of a two-dimensional code would have required 64 points, leading to more constellation expansion, which may be undesirable. Since the code G is rate 3/4, the 4 output bits of G are used to select one of the 16 eight-dimensional cosets, C_8^0, ..., C_8^F, and the 17 remaining bits would specify which of the parallel transitions in the selected C_8^i would be transmitted, requiring a look-up table with $2^{17} = 131,072$ 8-dimensional locations for each coset. Since there are 16

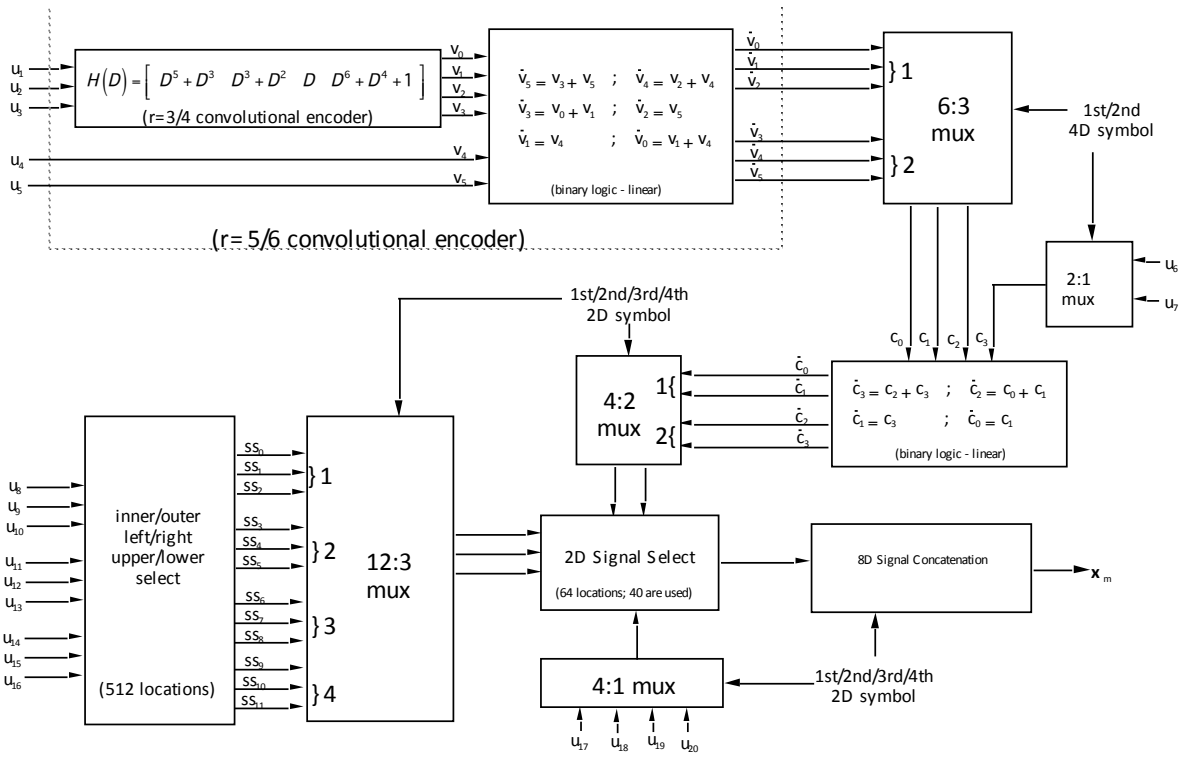


Figure 10.41: 8D Coset Select Implementation

ss_2	ss_1	ss_0	position
0	0	0	left-top-inner
0	0	1	left-bottom-inner
0	1	0	right-top-inner
0	1	1	right-bottom-inner
1	0	0	outer

Table 10.19: Truth Table for inner/outer selection (even b)

such cosets, the encoder would nominally need 2,097,152 8-dimensional locations in a look-up table to implement the mapping from bits to transmitted signal. The large memory requirement can be mitigated to a large extent by using the structure illustrated in Figure 10.41. From the partitioning structure, each 8D coset of C_8^0 can be written as the union of four Cartesian products, for instance

$$C_8^0 = \left(C_4^0 \otimes C_4^0\right) \bigcup \left(C_4^2 \otimes C_4^2\right) \bigcup \left(C_4^4 \otimes C_4^4\right) \bigcup \left(C_4^6 \otimes C_4^6\right) \quad , \tag{10.252}$$

and each of these Cartesian products is a R_4D_4 sublattice coset. Bits u_4 and u_5 specify which of these four Cartesian products contains the selected signal. The original convolutional encoder was rate 3/4, and it now becomes rate 5/6. The six bits v_0 ... v_5 can then be input into a linear circuit with the relations illustrated in Figure 10.41 that outputs six new bits in two groups of three bits each. The first such group selects one of the eight 4D cosets C_4^0 ... C_4^7 for the first 4D constituent sub-symbol and the second group does exactly the same thing for the second constituent 4D subsymbol. The encoder uses two more bits u_6 and u_7 as inputs to exactly the same type of linear circuit that was used for the previous four-dimensional code example. We could have redrawn this figure to include u_6 and u_7 in the convolutional encoder, so that the overall convolutional encoder would have been rate 7/8. Note that $k(E_8) = 4$ (u_4, u_5, u_6, and u_7). The nine bits u_8 ... u_{16} then are used to specify inner/outer selection abbreviated by Table 10.19. The 9 output bits of this inner/outer selection follow identical patterns for the other 2D constituent subsymbols. There are 512 combinations ($4^4 + 4 \cdot 4^3$), prohibiting more than

one outer from occuring in any 8D symbol.

The final encoder requires only 512 locations for the "inner/outer/left/right/upper/lower" circuit, and 64 locations (only 40 actually used) for the specification of each constituent subsymbol. This is a total of only 552 locations, significantly less than 2,097,152, and each is only two-dimensional. Further reduction of the size of the 64-location memory occurs by noting that the 4 two-dimensional cosets can be generated by taking any one of the cosets, and performing sign permutations on the two dimensions within the subsymbol. Then, the memory requirement would be (with a little extra logic to do the sign changes) 528 locations.

10.6.2 Decoder Complexity

The straightforward implementation of this maximum likelihood sequence detector for 4D and 8D codes can be very complex. This is because of the (usually) large number of parallel transitions between any two pairs of states. In one or two dimensions with (possibly scaled) Z or Z^2 lattices, the closest point to a received signal is found by simple truncation. However, the determination of the closest point within a set of parallel transitions that fall on a dense 4D or 8D lattice can be more difficult. This subsection studies such decoding for both the D_4 and E_8 lattices. A special form of the Viterbi Algorithm can also readily be used in resolving the closest point within a coset, just as another form of the Viterbi Algorithm is useful in deciding which sequence of multidimensional cosets is closest to the received sequence.

Decoding the D_4 Lattice By definition:

$$D_4 = R_4Z^4 \bigcup \left(R_4Z^4 + [0,1,0,1]\right) \tag{10.253}$$

$$= \left(R_2Z^2 \otimes R_2Z^2\right) \bigcup \left((R_2Z^2 + [0,1]) \otimes (R_2Z^2 + [0,1])\right) \quad , \tag{10.254}$$

which can be illustrated by the trellis in Figure 10.25. Either of the two paths through the trellis describes a sequence of two 2D subsymbols which can be concatenated to produce a valid 4D symbol in D_4. Similarly, upon receipt of a 4D channel output, the decoder determines which of the paths through the trellis in Figure 10.25 was closest. This point decoder can use the Viterbi Algorithm to perform this decoding function. In so doing, the following computations are performed:

trellis position	subsymbol (1 or 2)	adds (or compares)
R_2Z^2	1	1 add
R_2Z^2	2	1 add
$R_2Z^2 + [0,1]$	1	1 add
$R_2Z^2 + [0,1]$	2	1 add
middle states	–	2 adds
final state	–	1 compare
Total	1 & 2	7 ops

The R_4D_4 Lattice has essentially the same lattice as shown in Figure 10.42. The complexity for decoding R_4D_4 is also the same as in the D_4 lattice (7 operations).

To choose a point in each of the 8 cosets of $\Lambda' = R_4D_4$ in a straightforward manner, the Viterbi decoding repeats 8 times for each of the 8 cosets of R_4D_4. This requires 56 operations for the coset determination alone. However, 32 operations are sufficient: First, the two cosets of D_4 partition Z_4 into two parts, as illustrated in Figure 10.43. The 2 dimensional cosets $B_{2,0}$ and $B_{2,1}$ for both the first 2D subsymbol and the second 2D subsymbol are common to both trellises. Once the decoder has selected the closest point in either of these two cosets (for both first subsymbol and again for the second subsymbol), it need not repeat that computation for the other trellis. Thus, the decoder needs 7 computations to decode one of the cosets, but only 3 additional computations (2 adds and 1 compare) to also decode the second coset, leaving a total of 10 computations (which is less than the 14 that would have been required had the redundancy in the trellis descriptions for the two cosets not been exploited).

Returning to R_4D_4, the 8 trellises describing the 8 cosets, C_4^i, $i = 0, ..., 7$, used in a 4D (Z^4/R_4D_4) trellis code are comprised of 4 2D cosets (C_2^0, C_2^1, C_2^2 and C_2^3) for both the first 2D subsymbol and for the second 2D subsymbol. The decoding of these cosets (for both subsymbols) thus requires 8 additions. Then the decoding performs 3 computations (2 adds and 1 compare) for each of the 8 trellises

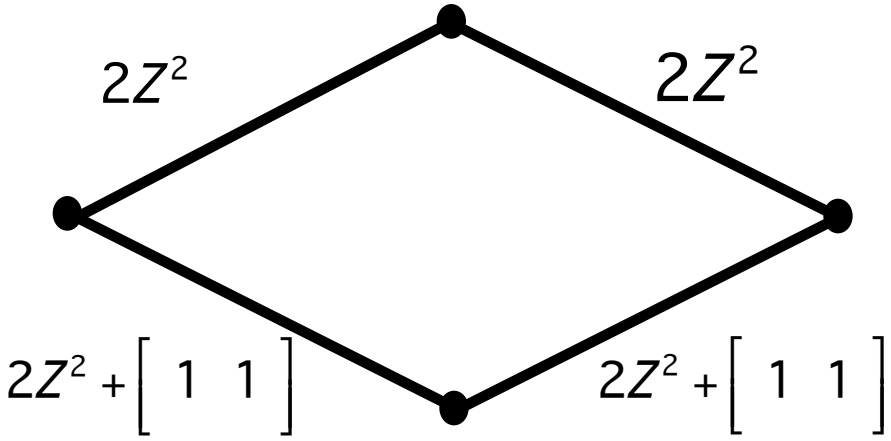


Figure 10.42: Trellis for the R_4D_4 Lattice

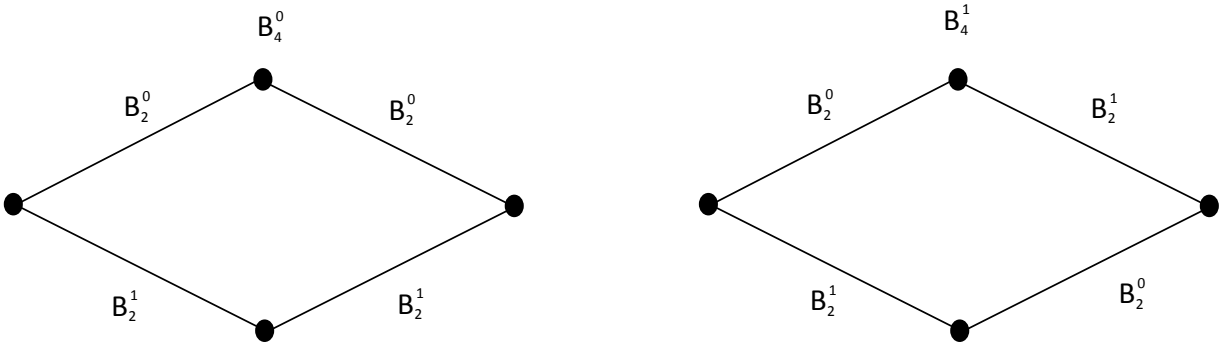


Figure 10.43: Trellis for the two cosets of the D_4 lattice

corresponding to the 8 cosets, requiring an additional 24 computations. Thus, the total computation required to decode all 8 cosets is 32 computations. Normalized to 1 dimension, there are 8 computations for all 8 cosets of the D_4 lattice.

We can now return to the complexity of decoding the entire coset code. The complexity of decoding using the Viterbi detector for the trellis code, after parallel transitions have been resolved is

$$N_D(trellis) = 2^\nu \left(2^k \text{ adds } + 2^k - 1 \text{ compares}\right) \quad . \tag{10.255}$$

The overall complexity $N_D(C)$ is the sum of the complexity of decoding the 2^{k+r_G} cosets of Λ' plus the complexity of decoding the sequences of multidimensional codewords produced by the trellis code.

EXAMPLE 10.6.1 (Wei's 16-state 4D code, with $r = 2/3$) This code is based on the partition Z^4/R_4D_4 and thus requires 32 operations to decode the 8 sets of parallel transitions. Also as $k = 2$ and $2^\nu = 16$, there are $16(4+3) = 112$ operations for the remaining 4D sequence detection. The total complexity is thus $112 + 32 = 144$ operations per 4D symbol. Thus $\bar{N}_D = 144/4 = 36$, which was the corresponding entry in Table 10.16 earlier.

The rest of the $\bar{N}_D$ entries in the 4D table are computed in a similar fashion.

10.6.3 Decoding the Gossett (E_8) Lattice

The decoding of an E_8 lattice is similar to the decoding of the D_4 lattice, except that it requires more computation. It is advantageous to describe the decoding of the 16 cosets of E_8 in Z^8 in terms of the easily decoded 2D constituent symbols.

The E_8 lattice has decomposition:

$$\begin{aligned} E_8 &= R_8D_8 \bigcup (R_8D_8 + [0,1,0,1,0,1,0,1]) && (10.256)\\ &= (R_4D_4)^2 \bigcup (R_4D_4 + [0,0,1,-1])^2 && (10.257)\\ &\quad \bigcup (R_4D_4 + [0,1,0,1])^2 \bigcup (R_4D_4 + [0,1,1,0])^2 \quad , && (10.258) \end{aligned}$$

which uses the result

$$R_8D_8 = (R_4D_4)^2 \bigcup (R_4D_4 + [0,0,1,-1])^2 \quad . \tag{10.259}$$

The trellis decomposition of the E_8 in terms of 4D subsymbols as described by (10.258) is illustrated in Figure 10.27. The recognition that the 4D subsymbols in Figure 10.27 can be further decomposed as in Figure 10.25 leads to Figure 10.28. The coset leader $[1,-1]$ is equivalent to $[1,1]$ for the "C-level" cosets in two dimensions, which has been used in Figures 10.27 and 10.28.

The complexity of decoding the E_8 lattice requires an addition for each of the 4 cosets of R_2D_2 for each of the 4 2D subsymbols, leading to 16 additions. Then for each of the (used) 4 cosets of R_4D_4, decoding requires an additional (2 adds and 1 compare) leading to 12 computations for each 4D subsymbol, or 24 total. Finally the 4D trellis in Figure 10.27 requires 4 adds and 3 compares. The total is then 16+24+7=47 operations.

All 16 cosets of the E_8 lattice require 16 additions for the 4 2D C-level partitions, and 48 computations for both sets (first 4D plus second 4D) of 8 cosets of R_4D_4 that are used, and finally 16(7)=112 computations for decoding all sixteen versions of Figure 10.27. This gives a total of $N_D = 16 + 48 + 112 = 176$, or $\bar{N}_D = 22$.

EXAMPLE 10.6.2 (Wei's 64-state 8D code, with $r = 3/4$) This code is based on the partition Z^8/E_8 and requires 176 operations to decode the 16 cosets of E_8. Also with $k = 3$ and $2^\nu = 64$, there are $64(8+7) = 960$ operations for the remaining 8D sequence detection. The total complexity is thus $960 + 176 = 1136$ operations per 8D symbol. Thus $\bar{N}_D = 1136/8 = 142$, which was the corresponding entry in Table 10.17 earlier.

10.6.4 Lattice Decoding Table

Table 10.20 is a list of the decoding complexity to find all the cosets of Λ' in Λ.

Λ	Λ'	$\lvert\Lambda/\Lambda'\rvert$	N_D	$\bar{N}_D$
Z^2	D_2	2	2	1
Z^2	$2Z^2$	4	4	2
Z^2	$2D_2$	8	8	4
Z^2	$4Z^2$	16	16	8
Z^4	D_4	2	10	2.5
Z^4	R_4D_4	8	32	8
Z^4	$2D_4$	32	112	28
Z^4	$2R_4D_4$	128	416	104
D_4	R_4D_4	4	20	5
D_4	$2D_4$	16	64	16
D_4	$2R_4D_4$	64	224	56
Z^8	D_8	2	26	3.25
Z^8	E_8	16	176	22
Z^8	R_8E_8	256	2016	257
Z^8	$2E_8$	4096	29504	3688
D_8	E_8	8	120	15
D_8	R_8E_8	128	1120	140
D_8	$2E_8$	2048	15168	1896
E_8	R_8E_8	16	240	30
E_8	$2E_8$	256	2240	280

Table 10.20: Lattice Decoding Complexity for all Cosets of Selected Partitions

10.7 Shaping Codes

Shaping codes for the AWGN improve shaping gain up to a maximum of 1.53 dB (see also Problems 10.17 and 10.18). Following a review of this limit, this section will proceed to the 3 most popular shaping-code methods.

The shaping gain of a code is again

$$\gamma_s \triangleq \frac{\frac{V_{\tilde{\boldsymbol{x}}}^{2/N}}{\bar{\mathcal{E}}_{\tilde{\boldsymbol{x}}}}}{\frac{V_{\boldsymbol{x}}^{2/N}}{\bar{\mathcal{E}}_{\boldsymbol{x}}}} \quad . \tag{10.260}$$

Presuming the usual Z^N-lattice cubic reference, best shaping gain occurs for a spatially uniform distribution of constellation points within a hypersheric volume[7] . For an even number of dimensions $N = 2n$, an N-dimensional sphere of radius r is known to have volume

$$V(\text{sphere}) = \frac{\pi^n \cdot r^{2n}}{n!} \quad , \tag{10.261}$$

and energy

$$\bar{\mathcal{E}}(\text{sphere}) = \frac{1}{2}\left[\frac{r^2}{n+1}\right] \quad . \tag{10.262}$$

The asymptotic equipartition analysis of Chapter 8 suggests that if the number of dimensions goes to infinity, so that $n \to \infty$, then there is no better situation for overall coding gain and thus for shaping thant the uniform distribution of equali-size regions throughout the volume. For the AWGN, the marginal one- (or two-) dimensional input distributions are all Gaussian at capacity and correspond to a uniform distribution over an infinite number of dimensions.[8] Since the coding gain is clearly independent of region shape, then the shaping gain must be maximum when the overall gain is maximum, which is thus the uniform-over-hypershere/Gaussian case. Thus, the asymptotic situation provides an upper bound on shaping gain. The uncoded reference used throughout this text is again

$$\bar{\mathcal{E}}_{\boldsymbol{x}} = \frac{1}{12}\left(2^{2\bar{b}} - 1\right) \tag{10.263}$$

$$V_{\boldsymbol{x}}^{2/N} = 2^{2\bar{b}} \cdot 1 \quad . \tag{10.264}$$

Then, algebraic subsitution of (10.261) -(10.264) into (10.260) leads to the **shaping-gain bound**

$$\gamma_s \leq \frac{\pi r^2 (n!)^{-\frac{1}{n}} / \frac{r^2}{2(n+1)}}{12\left(1 - 2^{-2\bar{b}}\right)} = \frac{\pi}{6}\frac{(n!)^{-\frac{1}{n}} \cdot (n+1)}{1 - 2^{-2\bar{b}}} \quad . \tag{10.265}$$

Table 10.21 evaluates the formula in (10.265) for $b \to \infty$ while Table 10.22 evaulates this formula for $n \to \infty$. With $\bar{b}$ infinite, and then taking limits as $n \to \infty$ for the asymmptotic best case using Stirling's approximation (see Problems 10.17 and 10.18),

$$\lim_{n\to\infty} \gamma_s \leq \lim_{n\to\infty} \frac{\pi}{6} \cdot \frac{n+1}{(n!)^{\frac{1}{n}}} = \frac{\pi}{6} \lim_{n\to\infty} \frac{n+1}{\frac{n}{e}} = \frac{\pi \cdot e}{6} = 1.53 \text{ dB} \quad . \tag{10.266}$$

Equation (10.265) provides a shaping-gain bound for any finite b and n, and corresponds to uniform density of the 2^b points with a $2n$-dimensional sphere. Equation (10.266) is the overall asymptotically attainable bound on shaping gain.

[7] Inspection of (10.260)'s numerator observes that for any given radius, which is directly proportional to the square root of energy $\sqrt{\bar{\mathcal{E}}_{\tilde{\boldsymbol{x}}}}$, a sphere has largest volume and thus largest number of points if they can be uniformly situated spatially.

[8] A simple Gaussian proof sketch: A point component in any dimension can be obtained by linear projection, which has an infinite number of terms, thus satisfying the central limit theorem in infinite dimensions. Thus all the components' (or marginal) distributions are Gaussian. QED.

b	$\gamma_s(n \to \infty)$
∞	1.53 dB
4	1.52 dB
3	1.46 dB
2	1.25 dB
1	0.28 dB
0	0 dB

Table 10.21: Shaping gain limits for an infinite number of dimensions.

$n(N)$	$\gamma_s(b \to \infty)$
1 (2)	0.20 dB
2 (4)	0.46 dB
3 (6)	0.62 dB
4 (8)	0.73 dB
12 (24)	1.10 dB
16 (32)	1.17 dB
24 (48)	1.26 dB
32 (64)	1.31 dB
64 (128)	1.40 dB
100 (200)	1.44 dB
500 (64)	1.50 dB
1000 (2000)	1.52 dB
10000 (20000)	1.53 dB

Table 10.22: Shaping gain limits for an infinite number of constellation points.

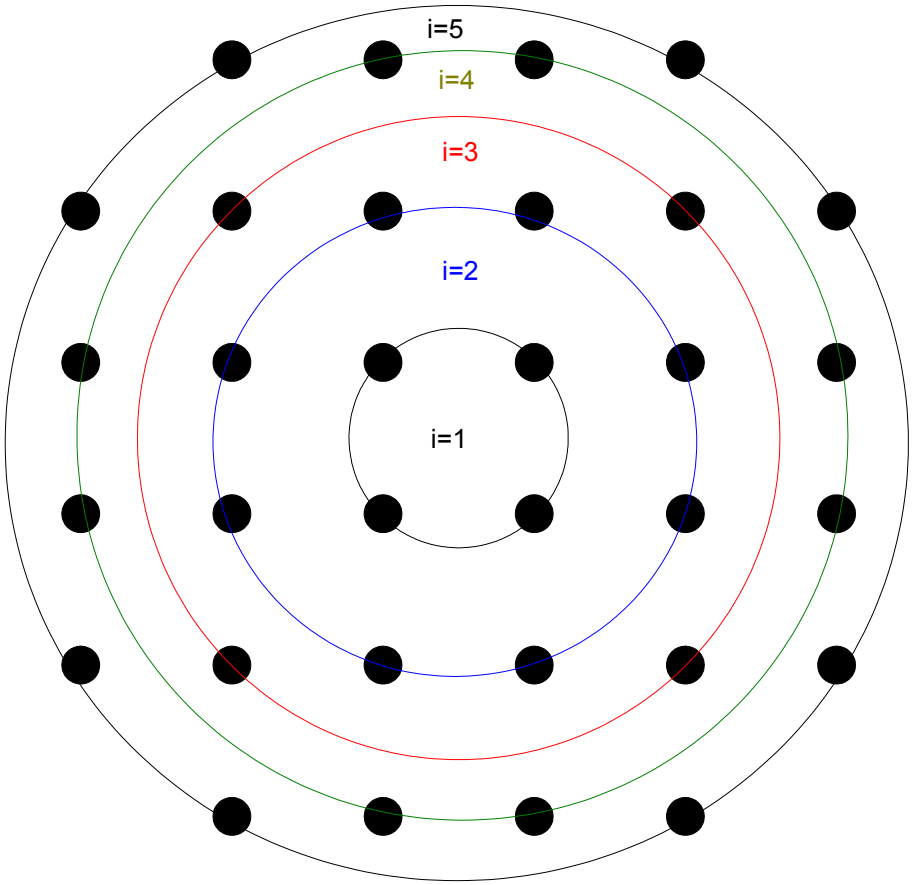


Figure 10.44: Shell Construction Illustration for 32SH (=32CR).

10.7.1 Non-Equiprobable Signaling and Shell Codes

Shell constellations were introduced by former EE379 student Paul Fortier in his PhD dissertation. They are conceptually straightforward and partition any constellation into shells (or rings). Each "shell" is a group of constellation points contained on a shell (or circle in 2D) as in Figure 10.44. Shell (SH)constellations have all points within a group (or shell) at constant energy. These shell groups are indexed by $i = 1, ..., G$, and each has energy $\mathcal{E}_i$ and contains M_i points. The probability that a point from group i is selected by an encoder is p_i. For the example of Figure 10.44, there are 5 shell groups with two-dimensional energies (for $d = 2$) $\mathcal{E}_1 = 2$, $\mathcal{E}_2 = 10$, $\mathcal{E}_3 = 18$, $\mathcal{E}_4 = 26$, $\mathcal{E}_5 = 34$. Each contains $M_1 = 4$, $M_2 = 8$, $M_3 = 4$, $M_4 = 8$, and $M_5 = 8$ points respectively. The shells have respective probabilities $p_1 = 1/8$, $p_2 = 1/4$, $p_3 = 1/8$, $p_4 = 1/4$ and $p_5 = 1/8$ if each point in the constellation is equally likely to occur. The average energy of this constellation can be easily computed as

$$\mathcal{E}_{\boldsymbol{x}} = \sum_{i=1}^{G} p_i \cdot \mathcal{E}_i = \left[\frac{2}{8} + \frac{10}{4} + \frac{18}{8} + \frac{26}{4} + \frac{34}{4}\right] = 20 \quad , \tag{10.267}$$

with consequent shaping gain

$$\gamma_s = \frac{(1/6)31 \cdot d^2}{20} = \frac{31}{30} = 0.14 \text{ dB} \quad . \tag{10.268}$$

The rectangular-lattice constellation (a coset of $2Z^2$) in 2D that uses the least energy for 32 points is 32CR=32SH, because it corresponds to using the shells with the least successive energies first.

128CR does not have such a least-energy property. A 128SH constellation is more complicated. The 128-point shell constellation 128SH constellation uses 17 shells of energies (number of points) (for $d = 2$) of 2 (4), 10 (8), 18 (4), 26 (8), 34 (8), 50 (12), 58 (8), 74 (8), 82 (8), 90 (8), 98 (4) , 106 (8) , 123 (8), 130 (16), 146 (8), 162 (4), and 170 (4 of 16 possible). The 128SH uses the points $(\pm 9, \pm 9)$ that do not appear in 128CR. The shaping gain of 128SH is .17 dB and is slightly higher than the .14 dB of 128CR. Problem 10.26 discusses a 64SH constellation (which is clearly not equal to 64SQ).

The number of bits per dimension can approach the entropy per dimension with well-designed input buffering (and possibly long delay) as in Chapter 8. While a constellation may have a uniform density over N dimensions, the marginal distribution for one- or two-dimensional constituent subsymbols need not be uniform. Sometimes this is used in shaping to effect large-dimensional γ_s with unequal probabilities in a smaller number of dimensions. The entropy per symbol of any constellation with point probabilities p_{ij} for the j^{th} point in the i^{th} group upper bounds the bits per one-or-two-dimensional, or any finite-dimensional, symbol

$$b \le H = \underbrace{\sum_{i=1}^{G}}_{\text{shells}} \quad \underbrace{\sum_{j=1}^{M_i}}_{\text{pts in shell}} \quad p_{ij} \log_2 \left(\frac{1}{p_{ij}} \right) \quad . \tag{10.269}$$

Equation (10.269) holds even when the groups are not shells. When all points are equally likely, b is simply computed in the usual fashion as

$$b = H = \sum_{i=1}^{G} \sum_{j=1}^{M_i} \frac{1}{M} \log_2 (M) = \log_2(M) = \log_2 \left(\sum_i M_i \right) \quad . \tag{10.270}$$

When each point within a group has equal probability of occuring (but the group probability is not necessarily the same for each group as in the 32CR and 128SH examples above), $p_{ij} = \frac{p_i}{M_i}$, the same H can be written

$$H = \sum_{i=1}^{G} \frac{p_i}{M_i} \sum_{j=1}^{M_i} \log_2(\frac{M_i}{p_i}) = \sum_{i=1}^{G} p_i \log_2(\frac{M_i}{p_i}) \quad , \tag{10.271}$$

which if $b_i \triangleq \log_2(M_i)$, then becomes

$$H = \sum_{i=1}^{G} p_i \log_2 \left(\frac{1}{p_i} \right) + \sum_{i=1}^{G} p_i \cdot b_i = H(\boldsymbol{p}) + \sum_{i=1}^{G} p_i \cdot b_i \quad , \tag{10.272}$$

where $H(\boldsymbol{p})$ is the entropy of the group probability distribution. good shaping code design would cause $b = H$, as is evident later. Equation (10.272) supports the intuition that the over all data rate is the sum of the data rates for each group plus the entropy of the group probability distribution.

The one-dimensional slices of the constellation should favor points with smaller energy; that is smaller energies have a higher probability of occurrence. For instance in 32CR, a one-dimensional slice has $p_{\pm 1} = \frac{12}{32}$, $p_{\pm 3} = \frac{12}{32}$, but $p_{\pm 5} = \frac{8}{32}$. As the dimensionality $N \to \infty$, the enumeration of shells can be very tedious, but otherwise follows the two-dimensional examples above. Furthermore as $N \to \infty$ the one-dimensional distribution of amplitudes (for large numbers of points) approaches Gaussian[9] with very large amplitudes in any dimension having low probability of occurrence. Such low probability of occurrence reduces energy of that dimension, thus increases shaping gain. The non-uniform distribution in a single (or two) dimensions suggests that a succession of two-dimensional constellations viewed as a shaping-code codeword might have extra points (that is more than $2^{2\bar{b}}$, as also in trellis codes) with points in outer shells (or groups) having lower probability than an equally likely $2^{-2\bar{b}}$ in one dimension. For instance, a 32CR constellation might be used with $\bar{b} = 2 < 2.5$, but with points in the outer shells at much lower probability of occurence than points in the inner shell when viewed over a succession of $N/2$ 2D-constellations as one large shaping codeword. Then, the two-dimensional "M" in equation (10.272)

[9]The result if viewed in terms of 2D constellations is that the 2D complex amplitudes approach a complex Gaussian distribution.

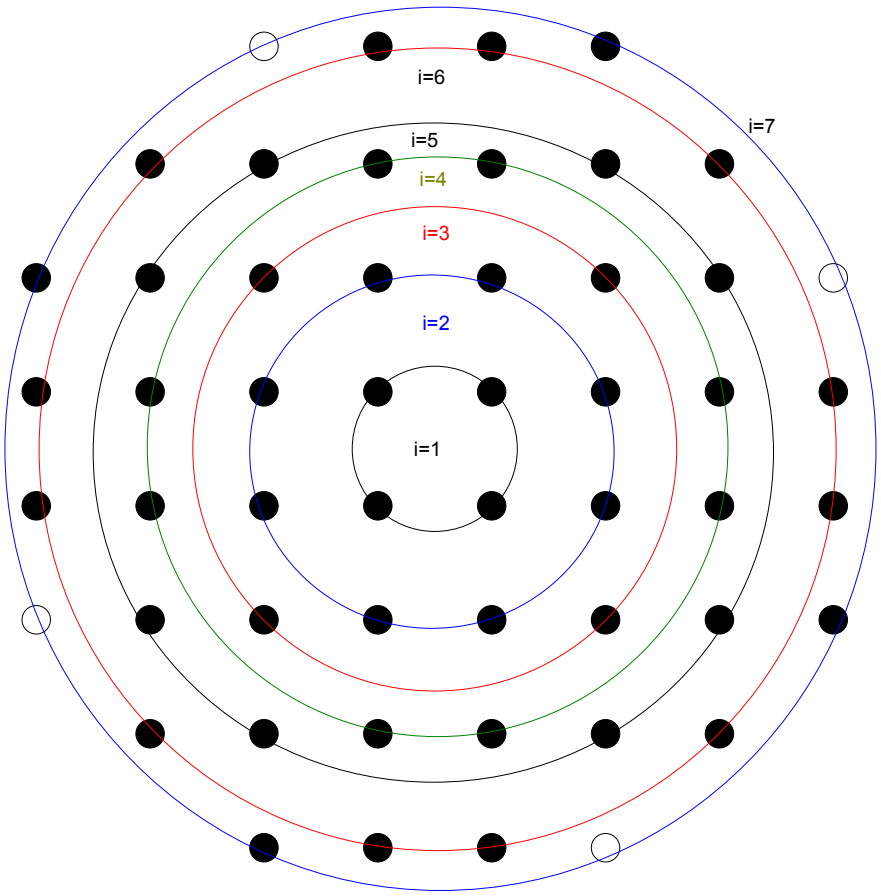


Figure 10.45: 48SH Constellation (with extra 4 “empty” points for 52SH).

is greater than $2^{2\bar{b}}$, but Equation (10.272) still holds with b directly computed even though $p_{ij} \neq \frac{1}{M}$, but $p_{ij} = p_i \cdot \frac{1}{M_i}$. The number of points in two dimensions is not the $\left(\frac{N}{2}\right)^{th}$ root of the true N-dimensional M. Indeed,

$$b = H(\boldsymbol{p}) + \sum_{i=1}^{G} p_i \cdot \log_2(M_i) \tag{10.273}$$

is a good equation to use since it avoids the confusion on M applying to a larger number of dimensions than 2.

Such was the observation of Calderbank and Ozarow in their **Nonequiprobable Signalling (NES)**. In NES, A 2D constellation is expanded and viewed as a part of a larger multi-dimensional constellation. INES chooses outer groups/shells less frequently than inner groups (or less frequently than even $\frac{M_i}{2^{2\bar{b}}}$).

A simple example of nNES was the 48CR constellation used for 4D constellations earlier in this chapter. In that constellation, the 32 inner two-dimensional points had probability of occurence 3/4 while the 16 outer points had probability only 1/4. While this constellation corresponds also to a 2D shell constellation (see Figure 10.45), the probabilities of the various groups are not simply computed by dividing the number of points in each group by 48. Furthermore,the constellation is not a 4D shell, but does have a higher shaping gain than even the two-dimensional 128SH. In one dimension, the probabilities of the various levels are

$$p_{\pm 1} = \frac{12}{32} \cdot \frac{3}{4} + \frac{4}{16} \cdot \frac{1}{4} = \frac{11}{32} \tag{10.274}$$

$$p_{\pm 3} = \frac{12}{32} \cdot \frac{3}{4} + \frac{2}{16} \cdot \frac{1}{4} = \frac{10}{32} \tag{10.275}$$

$$p_{\pm 5} = \frac{8}{32} \cdot \frac{3}{4} + \frac{4}{16} \cdot \frac{1}{4} = \frac{8}{32} \tag{10.276}$$

$$p_{\pm 7} = 0 + \frac{6}{16} \cdot \frac{1}{4} = \frac{3}{32} \quad . \tag{10.277}$$

Equations (10.274 -(10.276) illustrate that large points in one dimension have low probability of occurrence. This 48CR constellation can be used to transmit 5.5 bits in two dimensions, emphasizing that $2^{5.5} < 48$ so that the points are not equally likely. An astute reader might note that 4 of the points in the outer most 2D shell in Figure 10.45 could have been included with no increase in energy to carry an additional $\frac{1}{4} \cdot \frac{4}{16} \cdot 1 = \frac{1}{16}$ bit in two dimensions. The energy of 48CR (and of thus of 52CR) is

$$\mathcal{E}_{\boldsymbol{x}} = \frac{3}{4}20 + \frac{1}{4}\left(\frac{50 \cdot 12 + 58 \cdot 4}{16}\right) = 28 \quad , \tag{10.278}$$

leading to a shaping gain of for $b = 5.5$ of

$$\gamma_s = \frac{(1/6) \cdot 2^{5.5} \cdot 4}{28} = .23 \text{ dB} \quad , \tag{10.279}$$

and for 52CR's $b = 5.625$ of .42 dB. The higher shaping gain of the 52SH simply suggests that sometimes best spherical approximation occurs with a number of points that essentially covers all the points interior to a hypersphere and is not usually a nice power of 2.

The 48CR/52CR example is not a shell constellation in 4D, but does provide good shaping gain relative to 2D because of the different probabilities (3/4 and 1/4) of inner and outer points. Essentially, 48CR/52CR is a 4D shaping method, but uses 2D constellations with unequal probabilities. The four-dimensional probability of any of the 4D 48SH/52SH points is still uniform and is $2^{-4\bar{b}}$. That is, only the 2D component constellations have non-uniform probabilities. Again, for large number of constellation points and infinite dimensionality, the uniform distribution over a large number of dimensions with an average energy constraint leads to Gaussian marginal distributions in each of the dimensions.

For the 48CR/52CR or any 4D inner/outer design with an odd number of bits per symbol, Table 10.18 provided a nonlinear binary code with 3 bits in and 4 bits out that essentially maps input bits into two partitions of the inner group or the outer group. The left-inner and right-inner partitioning of Table 10.18 is convenient for implementation, but the essence of that nonlinear code is the mapping of inputs

into a description of "inner" group (call it "0") and "outer" group (call it "1"). This general description would like as few 1's (outers) per codeword as possible, with the codewords being (0,0), (1,0), and (0,1) but never (1,1). For even numbers of bits in four dimensions, Table 10.18 was not necessary because there was no attempt to provide shaping gain – in Section 10.6, the interest was coding gain and the use of inner/outer constellations there was exclusively to handle the "half-bit" constellations, coincidentally providing shaping gain. However, the concept of the 3 codewords (0,0), (1,0), (0,1) remains if there are extra points in any 2D constellation beyond $2^{2\bar{b}}$. The set of 2D constellation points would be divided into two groups, inner and outer, where the size of the outer group M_1 is most easily exactly[10] 1/2 the size of the inner group M_0 if the probabilities of the groups are to remain 3/4 and 1/4 respecively. This type of view of course begs the question of generalization of the concept.

Calderbank and Ozarow essentially generalized this "low maximum number of 1's" concept in an elegant theory that is simplified here for design. This text calls these "NE" (Non-Equal probability) codes. NE codes require the partitioning of the constellation into equal numbers of points in each group. For a binary (nonlinear) code that spans n dimensions[11] where no more than m ones occur, the NE shaping code rate is easily

$$2\bar{b} = \frac{1}{n} \log_2 \left\{ \sum_{j=0}^{m} \binom{n}{j} \right\} \quad , \tag{10.280}$$

and the probability of a 0 in any position is given by

$$p_0 = \frac{\sum_{j=0}^{m} \binom{n}{j} \cdot (n-j)}{n \cdot \sum_{j=0}^{m} \binom{n}{j}} \quad . \tag{10.281}$$

The probability of a 1 is then easily $p_1 = 1 - p_0$. Such an NE code is labelled $B_{n,m}$ for "binary" code with n positions and no more than m of these positions containing 1's. The 1's correspond to the outer group of points in a 2D constellation, while the 0's correspond to inner points in the same constellation. Such a binary code then specifies which points are inner or outer. The numerator of the code rate in (10.280) is unlikely to be an integer, so an efficient implementation of the mapping into codewords may have to span several successive codewords. Alternatively, the least integer in the numerator can instead be chosen. For instance, a $B_{12,3}$ code has 299 codewords and thus rate $\frac{\log_2(299)}{12} = .6853$ and might likely be rounded down to $\frac{8}{12}$ so that 8 bits enter a non-linear memory (look-up table like that of Table 10.18) while 12 bits leave. In such a code in 12 successive 2D symbols, no more than 3 symbols would be outer points. The shaping gain of such a code is easily and exactly computed from the p_0 in (10.281) as

$$\gamma_s = \frac{\frac{1}{6}\left(2^{2\bar{b}-1}\right) \cdot d^2}{p_0 \cdot \mathcal{E}_0 + (1-p_0) \cdot \mathcal{E}_1} \leq 1.53 \text{ dB} \; . \tag{10.282}$$

Calderbank and Ozarow provide a number of contellation-size independent approximations for various large block lengths and large numbers of constellation points in their work, but this author has found that direct calculation of the exact gain is probably more covenient, even though it requires knowledge of the exact constellation and inner-outer energies.

EXAMPLE 10.7.1 (8D 40CR and 44SH constellation) The 8D 40CR constellation of Figure 10.40 is also a 40SH constellation in 2D, but is not an 8D shell constellation. The 4 points $(\pm 5, \pm 5)$ were previously not used, so they are added here to form 44SH, which is divided into two equal groups of 22 points as shown in Figure 10.46. The nonlinear $B_{4,1}$ code rate is

$$2\bar{b} = (1/4) \cdot \log_2(1+4) = \frac{log_2(5)}{4} = .5805 \quad . \tag{10.283}$$

[10] Please note we've re-indexed the groups starting with 0 in this case.

[11] these dimensions correspond to 2D sub-constellation points in a $2n$-dimensional shaping-code symbol or codeword.

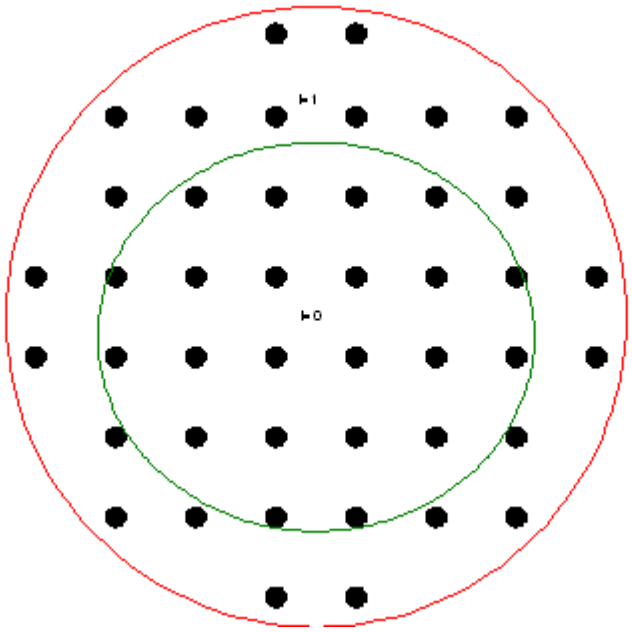


Figure 10.46: 44SH Constellation.

The probability of an inner point with straightforward use of the $B_{4,1}$ NE is computed as

$$p_0 = \frac{1 \cdot 4 + 4 \cdot 3}{4 \cdot (1+4)} = \frac{4}{5} \quad , \tag{10.284}$$

and thus the outer point has probability 1/5. The overall shaping-code rate is then

$$\log_2(22) + \mathcal{H}(.2) = 4.4594 + .7219 = 5.1814 \text{ bits per 2D symbol} \quad . \tag{10.285}$$

The shaping gain requires the two energies, so

$$\mathcal{E}_0 = \frac{4 \cdot 2 + 8 \cdot 10 + 4 \cdot 18 + 6 \cdot 26}{22} = \frac{316}{22} = 14.3636 \tag{10.286}$$

$$\mathcal{E}_1 = \frac{2 \cdot 26 + 8 \cdot 34 + 12 \cdot 50}{22} = \frac{872}{22} = 39.6364 \quad , \tag{10.287}$$

and

$$\gamma_s = \frac{(1/6)(2^{5.1814} - 1) \cdot 4}{.8 \cdot 14.3636 + .2 \cdot 39.6364} = \frac{23.5243}{19.4182} = 1.2115 = .83 \text{ dB} \quad . \tag{10.288}$$

This compares favorably with the original use in trellis coding where the shaping gain was only .15 dB. Table 10.22 has a bound for 8 dimensional shaping that is .7 dB, an apparent contradiction with the gain in this example. However, those bounds are for infinite numbers of points in the constellation and do not consider the nuances in detail of specific constellations, so .8dB (which is still less than the maximum of 1.53 dB, which always holds for any number of points) is not unreasonable, especially when one considers that a code rate of .5805 would effectively need to be implemented over a large number of dimensions anyway (thus, the example could not truly be called "8-dimensional" because the bit-fractioning process itself can add shaping gain as in Section 10.6's 40CR example earlier.) Unfortunately, realization constraints will reduce this shaping gain. For instance, an implementation might choose nonlinear code rate of $\bar{b} = .5625 = 9/16$. Since there are 5 possible codewords ([0000], [1000], [0100], [0010], [0001]) per symbol, 4 successive codewords would have $5^4 = 625$ possibilities of which $2^9 = 512$ with the least energy would be used, reducing code rate to 9/16. Of the 625 possibilities, 500 will have a 0 in any particular position and $125 > 113$ would have a 1

in that same particular position. There are 256 codewords with four 1's in the orginal code. 113 of these can be deleted. The probability that a 1 is in the same particular position after deletion would then be .2(113/256)=.0883.

$$\log_2(22) + \mathcal{H}(.0883) = 4.4594 + .4307 = 4.89\text{bits per 2D symbol} \quad . \tag{10.289}$$

The shaping gain would then be a lower value of

$$\gamma_s = \frac{(1/6)(2^{4.89}-1)\cdot 4}{.9117\cdot 14.3636 + .0883\cdot 39.6364} = \frac{19.102}{16.5947} = 1.15 = .60 \text{ dB} \quad , \tag{10.290}$$

and effectively corresponds to a 32-dimensional realization and is below the bound for both 8 and 32 dimensions. The original 8D trellis coded system by coincidence had .15 dB of shaping gain simply because the 40CR constellation was closer to a circle than a square. Interestingly enough, with no increase in energy, that constellation could have transmitted another 4 points and thus $(1/8)(1/2)(1) = 1/16$ bit/2D. The shaping gain would thus increase by $(2^{5.25+.0625}-1)/(2^{5.25+.0625}-1) = 1.0455 = = .19$ dB, to become then .15+.19 = .34 dB. Clearly the earlier system with trellis code is easier to implement in terms of look-up tables, but NE codes do have a gain as this example illustrates.

The above example illustrates that at least when groups are chosen in cocentric fashion that NE provides good shaping gain in effectively a few dimensions with reasonable complexity. Calderbank and Ozarow found some limitation to the use of binary codes. So for large number of constellation points, they found shaping gains for binary $B_{n,m}$ codes to be limited to about .9 dB for up to $n = 20$. An obvious extension is to use more groups, but then the code is no longer binary. NE codes with 4 cocentric groups make use of 2 binary codes for the each of the bits in a 4-ary label to the group, so for a concatenation of two $B_{m,k}$ codes

$$0 \rightarrow 00 \tag{10.291}$$
$$1 \rightarrow 01 \tag{10.292}$$
$$2 \rightarrow 10 \tag{10.293}$$
$$3 \rightarrow 11 \tag{10.294}$$

The probabilities of the 4 groups are then

$$p_{00} \rightarrow p_1 \cdot p_2 \tag{10.295}$$
$$p_{01} \rightarrow (1-p_1)\cdot p_2 \tag{10.296}$$
$$p_{10} \rightarrow (1-p_2)\cdot p_1 \tag{10.297}$$
$$p_{11} \rightarrow (1-p_1)\cdot(1-p_2) \quad . \tag{10.298}$$

Energy and shaping gain then follow exactly as in general with 4 groups contributing to all calculations. An extra .2 dB for sizes up to $n = 20$ for a total shaping gain of 1.1 dB was found by Calderbank and Ozarow for 4 groups.

Essentially, the designer needs to compute exact gain for particular constellations, but generally Calderbank and Ozarow's NE codes will provide good shaping gain for reasonable constellation sizes.

10.7.2 Voronoi/Block Shaping

Forney's **Voronoi constellations** have shaping boundaries that approximate a hypersphere in a finite number of dimensions. These boundaries are scaled versions of the decision (or "Voronoi") regions of dense lattices. The closeness of the approximation depends on the lattice selected to create the constellation boundary – very good lattices for coding will have decision regions that approach hyperspheres, and thus can also provide good boundaries for constellation design[12]. This subsection describes a procedure for encoding and decoding with Voronoi shaping regions and also enumerates the possible shaping

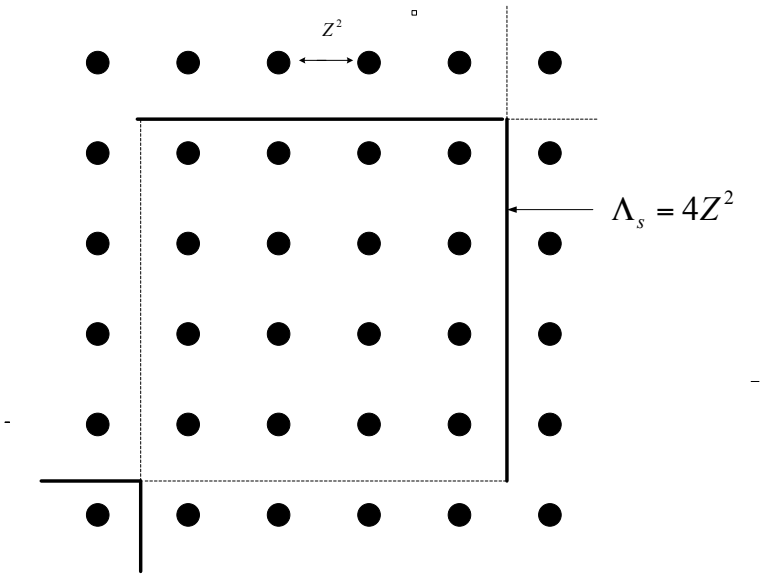


Figure 10.47: Tesselation of two-dimensional space by shaping lattice $4Z^2$ with constellation based on $Z^2 + (\frac{1}{2}, \frac{1}{2})$

gains for some reasonable and common lattice choices. This subsection prepares the reader for the generalization to trellis shaping in Subsection 10.7.3.

Since nearly all the codes in this chapter (and the next) are based on (cosets of) origin-centered Z^N lattices (usually coded or uncoded sequences of one- or two-dimensional sub-symbols), the objective will be to cirumscribe such points by the origin-centered larger Voronoi shaping region of some reasonable scaled-by-power-of-two lattice. The Voronoi shaping regions of such a scaled lattice will **tesselate** the N-dimensional space into mutually disjoint regions. Tesselation means that the union of all the Voronoi regions is the entire space, and so there are no interstial gaps between the Voronois shaping regions. Points on Voronoi-region boundaries need to be assigned in a consistent way so that each such region is closed on some of the faces of the Voronoi region and is open on the others. Each such Voronoi shaping region should essentially "look the same" in terms of the center of the region relative to which boundaries are open and closed. A two-dimensional example appears in Figure 10.47. Figure 10.47 shows a Voronoi region of $4Z^2$, which is a square of side 4. The closed boundaries (solid lines) are the upper and right-most faces, while the open boundaries (dashed lines) are the lower and left-most faces. Such two-dimensional squares with closed-open boundaries precisely tesselate two-dimensional space. Each contains 16 points from $Z^2 + (\frac{1}{2}, \frac{1}{2})$. In general, the number of included points is the ratio of the volume of the Voronois shaping region to the volume of the underlying constellation lattice's decision region, $(4^2/1^2) = 16$ in Figure 10.47.

In general, one-half the boundaries should be closed and one-half should be open. The closed boundaries should be a connected set of points (that is any point on the closed part of the boundary can be connected to any other point on the closecd part of the boundary by passing through other boundary points that are included in the closed part of the boundary). Thus, the use of top and bottom for the closed boundaries and left/right for the open boundaries is excluded from consideration. Constellation points on a closed face of the shaping lattice's Voronoi region are included, while those on an open face are not included if such a situation arises (it does not arise in Figure 10.47 because of the coset offset of $(\frac{1}{2}, \frac{1}{2})$). In some cases including the rest of this section, it may be easier to add coset offsets as the last step of encoding, and so situations of points on boundaries may occur prior to such a last coset-offset

[12]There are limiting definitions of lattice sequences that do approach hyperspheres and thus capacity, but the practical interest in such systems is for a smaller number of dimensions like 4, 8, 16, 24, or 32.

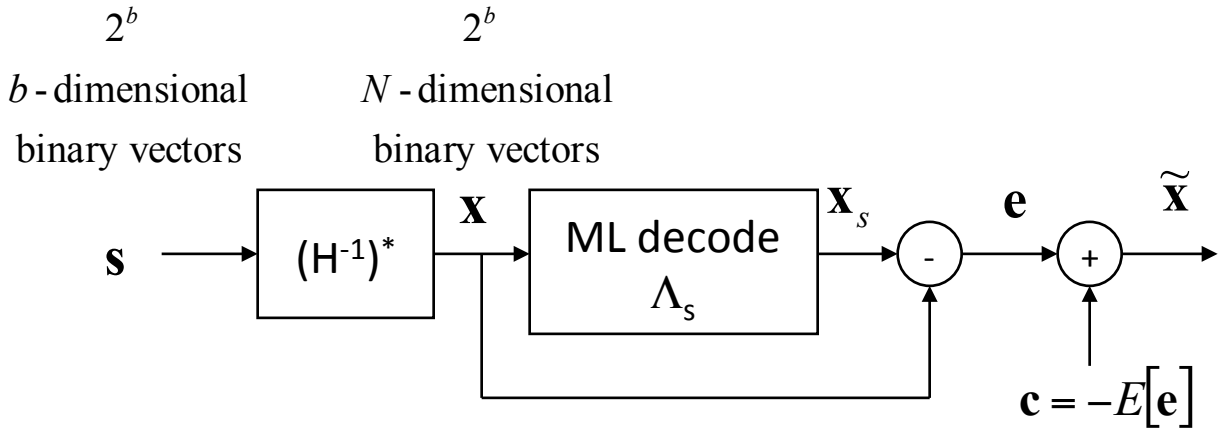


Figure 10.48: Voronoi encoder for a shaping code.

step.

While the concept is easily pictured in two-dimensions, the encoding of points in multiple dimensions nominally might require a large table-look-up operation as well as a difficult search to find which points are in/out of the Voronoi region of the selected shaping lattice of size $M \cdot V(Z^N)$. Figure 10.48 provides a simplified implementation that uses a maximum-likelihood search in the encoder. The ML search (say implemented by Viterbi algorithm, as for instance for the lattices studied earlier in Section 10.5 of this chapter) finds that point in the shaping lattice $\boldsymbol{x}_s$ that is closest to an input point $\boldsymbol{x}$ in Z^N. Any such closest point has an error that is within the Voronoi/decision region. Thus, the error is transmitted, since it must be within the desired shaping region. As long as the input constellation points $\boldsymbol{x}$ to such a search are reasonably and carefully chosen, then the output errors $\boldsymbol{e}$ will be in a one-to-one correspondence with those input points. Thus as in Figure 10.49, decoding of $\boldsymbol{e}$ by any true ML decoder for $\tilde{\boldsymbol{x}} - \boldsymbol{c}$ in a receiver can be inverted to the corresponding input.

Since the Voronoi regions tesselate the complete space, a square region when $2^{2\bar{b}}$ is even and a rectangular region of sides $2^{2\lfloor \frac{\bar{b}}{2} \rfloor}$ and $2^{2\lfloor \frac{\bar{b}}{2} \rfloor + 1}$ when $2^{2\bar{b}}$ is odd, will suffice for $\boldsymbol{x}$ in two dimensions. For more general M, care must be exercised to ensure that two input vectors do not correspond to the same error vector $\boldsymbol{e}$ at the ML search output. Such care is the subject of Subsection 10.7.2 below. With such care, the ML decoder invertibly maps easily generated inputs $\boldsymbol{x}$ into "error" vectors for transmission in the Voronoi shaping region. Thus, the shaping gain will be that of the Voronoi shaping region, presumably good because the shaping lattice is well chosen. The volume of the shaping lattice must be at least M times the volume of the underlying code constellations, typically $V(Z^N) = 1$, making $V(\Lambda_s) = M$, where M is the number of points in the N-dimensional constellation. The number of these points may exceed 2^b in situations where a code is already in use. However, the shaping is independent of such coding. Thus, convenience of notation for the rest of this section presumes that $b = \log_2(M)$ where M is the number of points in the N-dimensional constellation. So as far as shape-encoding is concerned, M is simply the total number of points in the constellation including any earlier-introduced redundancy for coding. The final coset addition for centering (making the mean value zero) the constellation is also shown in Figure 10.48. The added vector $\boldsymbol{c}$ should be the negative of the mean value of the points in the constellation corresponding to the values of $\boldsymbol{e}$.

EXAMPLE 10.7.2 (R_2D_2 simple shaping) Figure 10.50 illustrates the shaping of 3 bits in a rectangular pattern on the grid of Z^2 into the shape of R_2D_2. The points with squares and circles are the original 8 points $\boldsymbol{x}$ in a rectangular pattern that is simple to create. By tesselating the two-dimensional space with the diamond R_2D_2 shapes of volume 8 (with right-sided boundaries included and left-sided boundaries not included), it is clear the original points do not fit the shape. However, by taking the error of any point with respect to the center of the closest region, the points map as shown as the dark circles $\boldsymbol{e}$. Four points

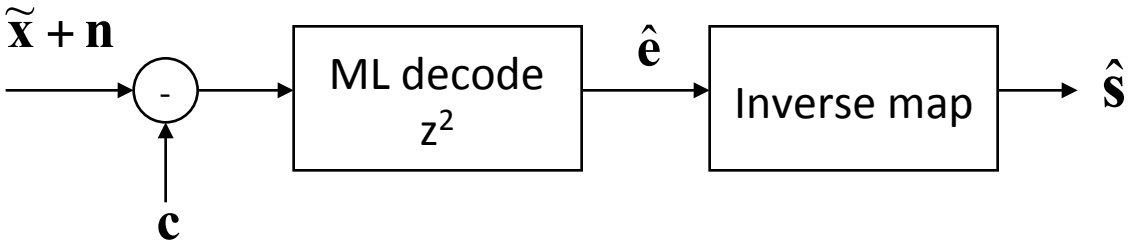


Figure 10.49: Decoding of AWGN with shaping inversion included.

remain in the original positions, but the other 4 move as a result of the ML search and consequent transmission of the corresponding error vector. A point on a boundary has error vector with respect to the center of the Voronoi region that has that point on the closed boundary. There is no shaping gain in this example, but the concept of the mapping into Voronoi regions is clear. The 8 points in the shaped constellation are darkened. Those points have characteristics

$\boldsymbol{e}$ point	$\bar{\mathcal{E}}_{\boldsymbol{e}}(i)$	$m_{\boldsymbol{e}}(i)$
$\frac{1}{2}[\pm 1,\ \pm 1]$	$\frac{1}{4}$	$[0,\ 0]$
$\frac{1}{2}[3,\ \pm 1]$	$\frac{5}{4}$	$[\frac{3}{2},\ 0]$
$\frac{1}{2}[1,\ \pm 3]$	$\frac{5}{4}$	$[\frac{1}{2},\ 0]$
average	$\frac{3}{4}$	$[\frac{1}{2},\ 0]$

Since the constellation after shaping has non-zero mean, energy can be reduced by subtracting that non-zero mean from all points, so $\boldsymbol{c} = -[\frac{1}{2},\ 0]$. Then the energy is

$$\mathcal{E}_{\boldsymbol{x}} = \frac{3}{4} - \frac{1}{2}\left\{\left(\frac{1}{2}\right)^2 + 0^2\right\} = \frac{3}{4} - \frac{1}{8} = \frac{5}{8} \quad , \tag{10.299}$$

which is the same energy as an 8SQ constellation and so has no shaping advantage. The shaping gain with respect to the common reference Z^2 is also $\frac{(8-1)}{12} \cdot \frac{8}{5} = \frac{14}{15}$ or about -0.1 dB. Thus R_2D_2, while easy for illustration , is not a particularly good lattice for shaping.

Encoder Simplification for Voronoi Shaping

A key concept in Voronoi-encoding simplification, cleverly constructed by Forney, makes use of the observation in Appendix C that (scaled) binary lattices tesselate Z^N. Essentially every point in Z^N is in some coset of the binary shaping lattice Λ_s if $Z^N/2^m\Lambda_S/2^mZ^N$ where $m \geq 1$. In Appendix C, $m = 1$. In such an $m = 1$ case, the shape of Λ_s may be desirable, but its volume may not equal M. If constellations have a power of two number of points, then the factor 2^m represents the additional enlargement (scaling) necessary of the desired Voronoi shaping region to cover the M points in N dimensions. This subsection first describes encoder implementation with $m = 1$, and then proceeds to $m > 1$.

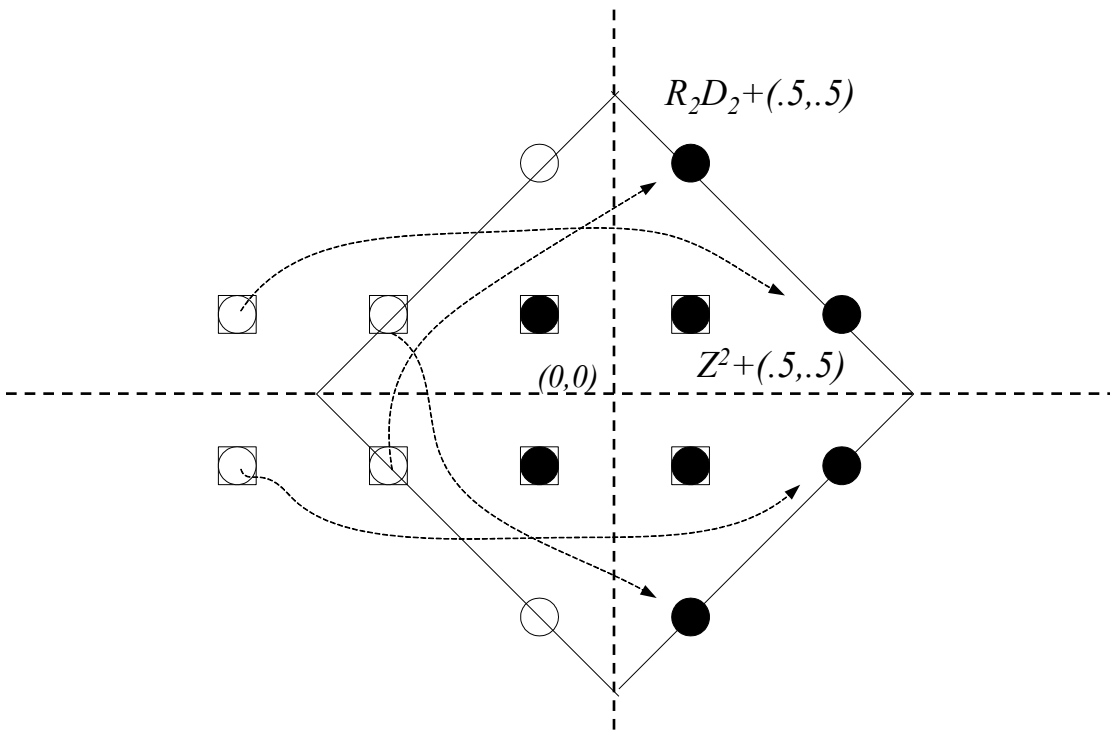


Figure 10.50: Shaping 3 input bits in Z^2 into an R_2D_2 boundary.

Appendix C notes that every point in Z_N can be constructed by some point in the partitioning (and shaping) binary lattice Λ_s plus some coset offset. A binary code with linear binary block encoder G_s and corresponding binary-block parity matrix H_s defines a binary lattice Λ_s as in Appendix C. The order of the partition is

$$| Z^N/\Lambda_s | = 2^b = M \quad . \tag{10.300}$$

Then, as in Appendix C, the distinct 2^b cosets of Λ_s whose union is Z^N are in a one-to-one relationship with the 2^b distinct syndrome vectors of the binary code (such syndrome vectors being generated by taking any of the 2^b binary b-row-vectors and postmultiplying by the parity matrix H_s^*). Since these binary-code syndrome vectors are easily generated as any or all of the 2^b binary b vectors, then they are equivalent to an easily generated input as in Figure 10.51. The transformation by the inverse-transpose parity matrix generates a set of distinct N-vector cosets that when added to any and all points in Λ_s generate Z_N. These coset vectors now in Z_N represent the distinct error vectors with respect to the closest points in Λ_s modulo 2, but are not necessarily the error vectors of minimum Euclidean norm. These inputs $\boldsymbol{x}$ are now assured of each mapping in a one-to-one relationship with one of the 2^b distinct syndromes/error-vectors $\boldsymbol{e}$. The ML search simply re-maps them into the Voronoi shaping region of minimum total energy. The channel with any receiver decoding is eventually viewed as hard coding to one of 2^b possible N-dimensional $\boldsymbol{e}$ values. The original binary syndrome vectors (viewed as the b information bits here) is recovered in the receiver by multiplying $\hat{\boldsymbol{x}}$ by H^* after executing a modulo-2 operation. The modulo-2 operation precedes the syndrome re-generation because the vectors $\boldsymbol{e}$ are equivalent modulo 2 to the original $\boldsymbol{x}$ vectors, and then the binary arithmetic of the parity matrix can be directly applied. The decoder is shown in Figure 10.52 for $m = 1$.

The situation with $m > 1$ corresponds to $(m-1)N + k$ bits. The coset-indicator outputs in Figure 10.53 are multiplied by 2^{m-1} and added to a binary N-vector $\boldsymbol{u}_{m-2}$ of N bits and multiplying it by 2^{m-2} and then adding it in turn to another vector of N bits $\boldsymbol{u}_{m-1}$ multiplied by 2^{m-2} ... and adding it to the last binary vector of N bits unscaled. Essentially , the scaling by decreasing powers of 2 on the integer vector $\boldsymbol{x}$'s components allows the contributions at each stage to represent the "least significant

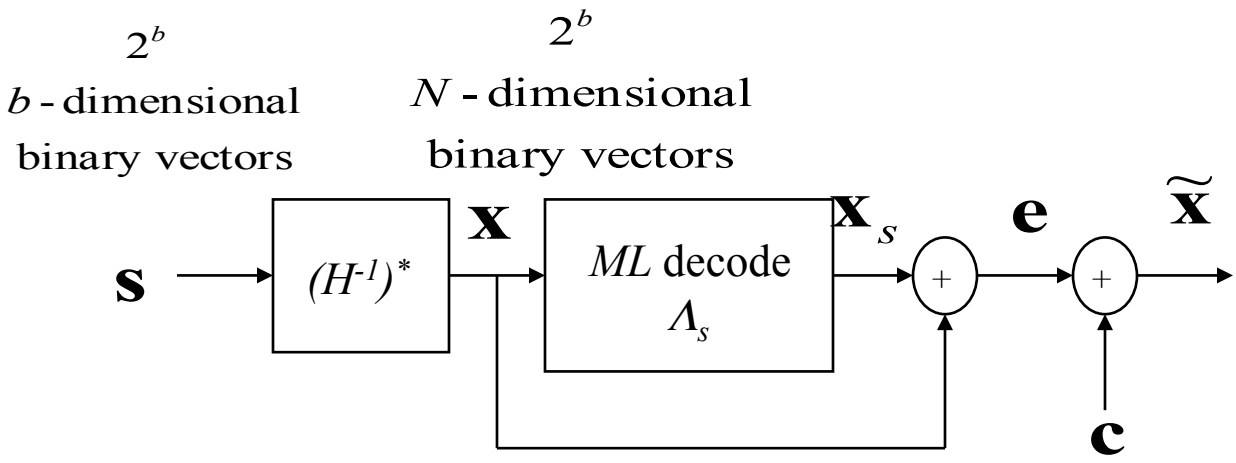


Figure 10.51: Voronoi encoder for shaping code with syndrome generation of input vectors.

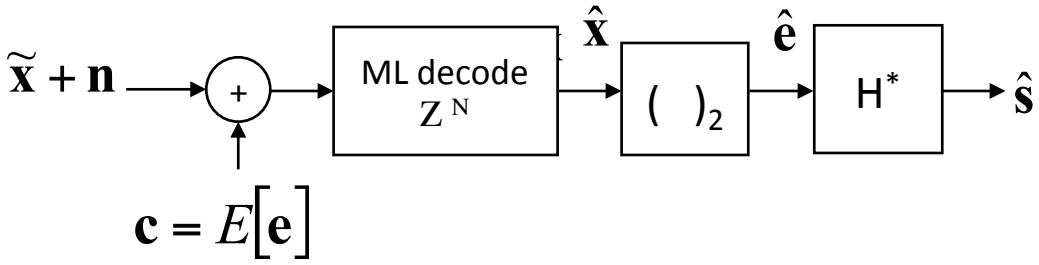


Figure 10.52: Voronoi decoder for shaping code with syndrome generation of input vectors.

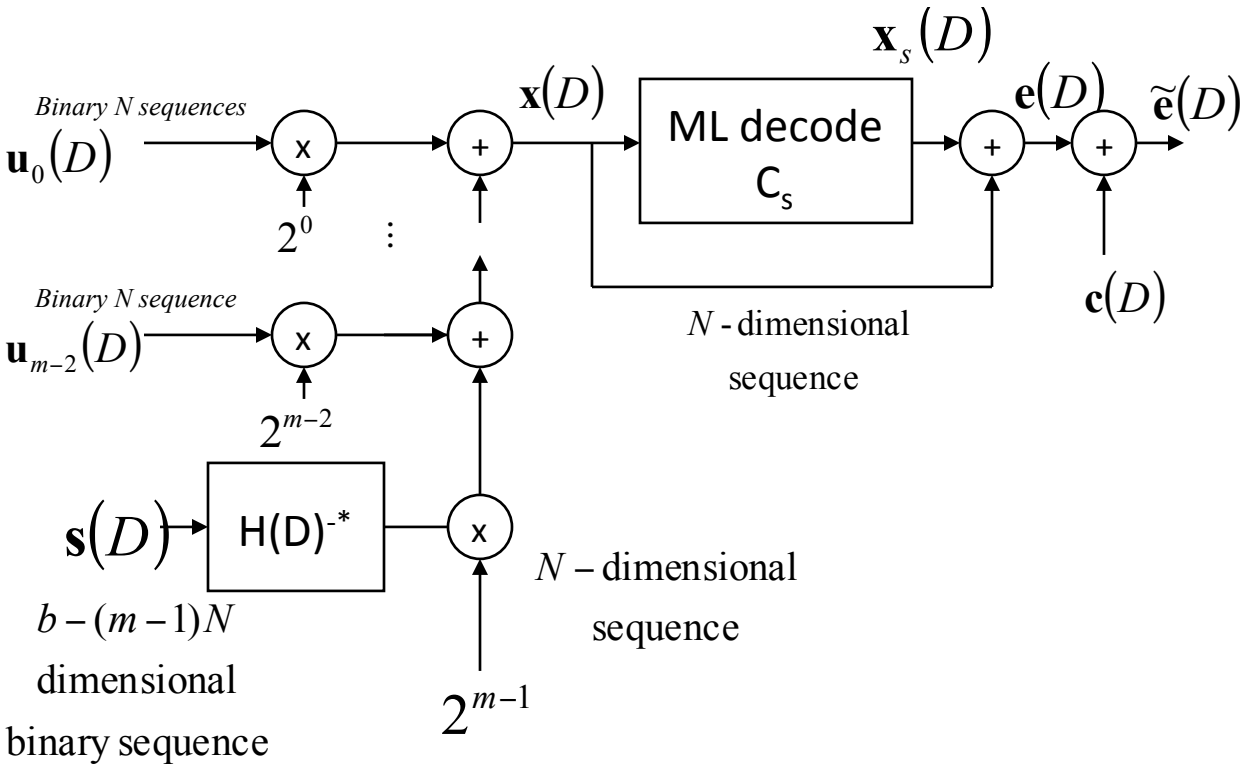


Figure 10.53: Voronoi encoder for shaping code with syndrome generation of input vectors with scaling and $m > 2$.

contributions" with respect to values scaled with a larger power of 2. For decoding in Figure 10.54, a modulo-2 operation recovers the upper binary N-vector $\hat{\boldsymbol{u}}_0$ from the ML decoder output integer, while after removal of those decided bits with integer subtraction, the remaining N-dimensional vector can be processed modulo 2^2, then its components divided by 2^1. Continuing through stage index i, the next remaining N-dimensional vector is produced by an operation of modulo 2^{i+1} and the components divided by $2^{i=1}$ to produce $\hat{\boldsymbol{u}}_i$, and so forth until the last remaining N-dimensional vector is modulo 2^m and devided by 2^{m-1} prior to multiplication by the parity transpose for deciding the syndrome bits of the encoder.

A drawback of the encoder is that only integer multiples (that is $m-1$) of N bits are allowed to be added to the k-dimensional syndrome vector of bits, severely limiting bit rates when N is large. If some intermediate number of bits corresponds to the desired data rate, then this number of bits will fall between two values for m. If some fractional value of $m = 1 + \frac{p}{p+q} > 1$ of N-bit blocks is desired, then p successive block-packets are used with $m-1 = p$, followed by q packets with $m-1 = q$. The shaping gain is assumed to be the same for both designs, although a slight deviation related to the number of points might alter the actual shaping gain. The decoder is altered correspondingly.

EXAMPLE 10.7.3 (Continuing R_2D_2 shaping example) For the earlier 8-point example, the generator matrix is $G = [1\ 1]$ and the parity matrix is also $H = [1\ 1]$. An acceptable left inverse is $H^{-*} = [1\ 0]$. However, this H is for D_2, so that R_2D_2 has $m = 2$. Thus, a single bit $\boldsymbol{s}$ times $[1\ 0]$ is scaled by $2^1 = 2$ before adding it to the 4 possible binary 2-vector values of $\boldsymbol{u}_0$ in Figure 10.55. The constellation produced is not exactly the same as in Figure 10.47, but is equivalent. The decoder in Figure 10.56 adds back the mean $[\frac{1}{2}\ 0]$ coset offset and does ML slicing for the offset of the constellation shown. The 8 points at the output then processed modulo 2 to produce the bits u_2 and u_1. These bits are then subtracted from the ML slicing output and will produce either the points $[0\ 0]$ when $u_3 = 0$ or will produce one of the points $[-2\ 0]$ or $[0\ -2]$ when $u_3 = 1$. The encoder multiplication

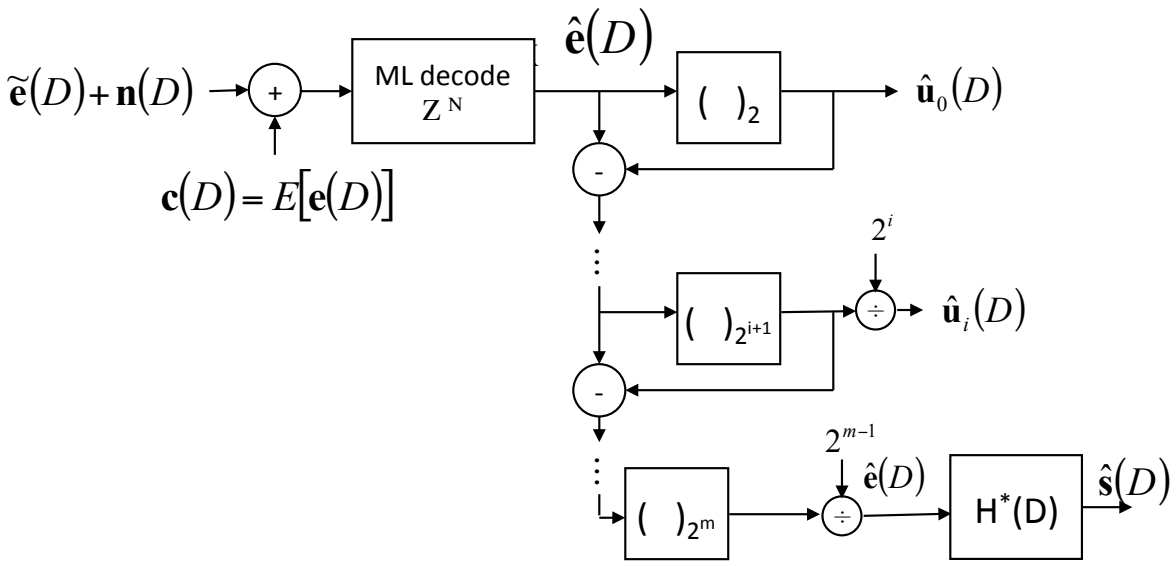


Figure 10.54: Voronoi decoder for shaping code with syndrome generation of input vectors with $m > 2$.

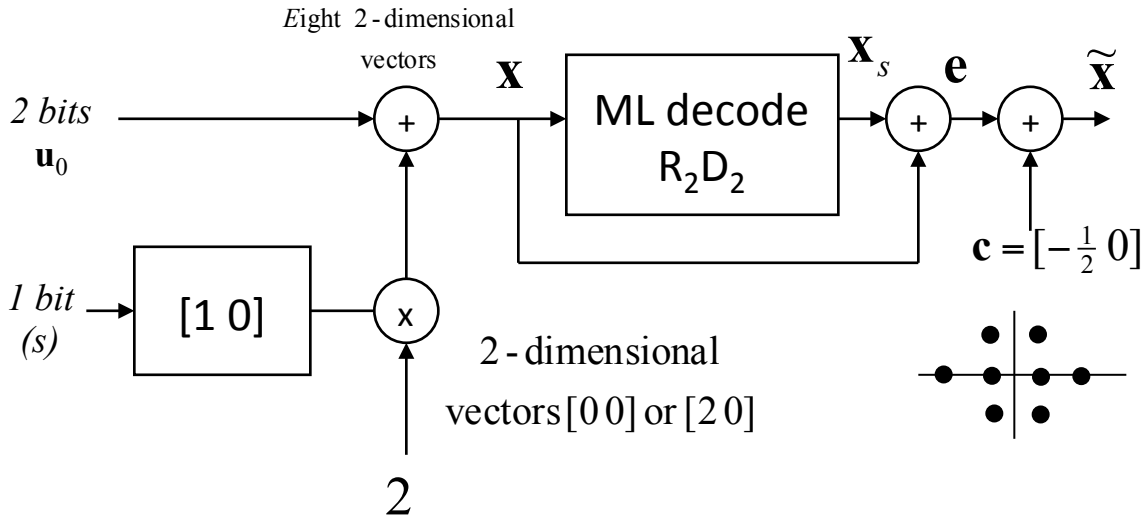


Figure 10.55: Voronoi encoder for example.

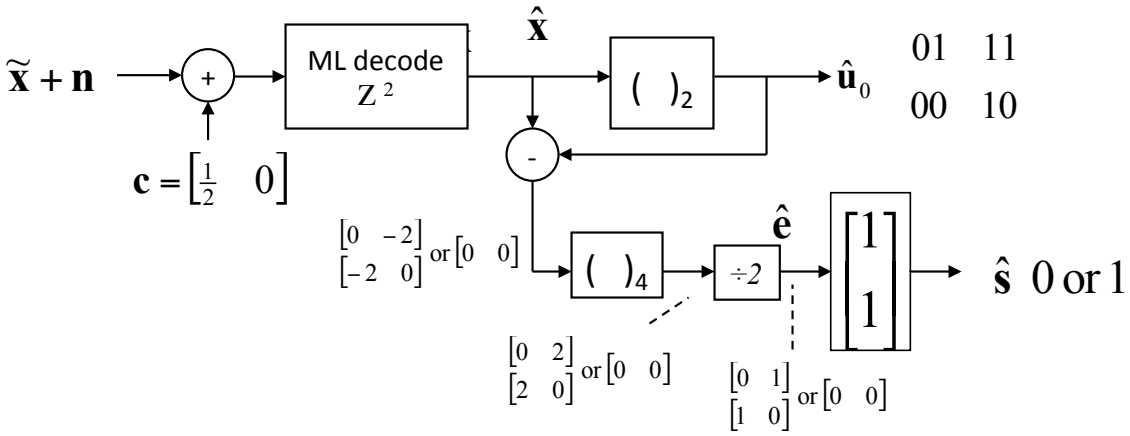


Figure 10.56: Voronoi decoder for the R_2D_2 shaping code example.

by 2 is inverted by first processing these two points modulo 4 (that is by modulo-2^{m+1}, and then dividing by 2. The final step is adding the two dimensions (since $H = [1\ 1]$) to produce 0+0=0 when $u_3 = 0$ and 1+0=1 when $u_3 = 1$.

Voronoi Shaping Code Design

The shaping gain of a Voronoi shaping code is precisely computed only if b is known as

$$\gamma_s = \frac{2^{2\bar{b}}}{12\mathcal{E}_{\Lambda_s}(1-2^{-2\bar{b}})} \quad , \tag{10.301}$$

but for large values of b, the asymptotic gain can be listed and approximated by the continuous approximation that replaces the term $2^{2\bar{b}}/\mathcal{E}_{\Lambda_s}$ by

$$2^{2\bar{b}}/\mathcal{E}_{\Lambda_s} \approx \frac{\int_{V(\Lambda_s)} \|\boldsymbol{x}\|^2 dV}{\int_{V(\Lambda_s)} dV} \quad . \tag{10.302}$$

Table 10.7.2 enumerates the shaping gain for some well-known lattices.

The gains for the Barnes-Wall Lattice Λ_{16} and the Leech Lattice Λ_{24} are provided to get an idea of the increase in shaping gain with N. The ML-search for these structures may be excessive for the gain provided and trellis shaping in Subsection 10.7.3 provides an easier alternative. For the dual of the D_8 lattice

$$H_{D_8^\perp} = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{10.303}$$

2^ν	code rate	γ_s (dB)	$H(D)$	$H(D)^{-*}$
2	1/2	.59	$[1+D\ 1]$	$[\ 0\ 1]$
4	1/2	.97	$[1+D+D^2\ 1+D^2]$	$[D\ 1+D]$
8	1/2	1.05	$[1+D^2+D^3\ 1+D+D^3]$	$[1+D\ D]$
8	$D_8^\perp$	.47	see (10.303)	see (10.304)
8	E_8	.65	see (10.305)	see see (10.306)
16	Λ_{16}	.86	not provided	not provided
24	Λ_{24}	1.03	not provided	not provided

with transmitter inverse

$$H_{D_8^\perp}^{-*} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \end{bmatrix} . \tag{10.304}$$

For the implementation of the self-dual E_8 shaping

$$H_{E_8} = \begin{bmatrix} 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 \end{bmatrix} \tag{10.305}$$

and

$$H_{E_8}^{-*} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix} \tag{10.306}$$

10.7.3 Trellis Shaping

Voronoi shaping extends naturally to the use of trellis codes or lattice-sequences to define a shaping region from which constellation points must be selected. This concept was introduced by Forney and called Trellis Shaping. This method can achieve shaping gains of over 1 dB with relatively low complexity. The ML search becomes a semi-infinite trellis search (Viterbi algorithm) and thus implies infinite delay. Thus, the search is terminated after an appropriate number of stages. There may thus be a small probability of sending the wrong sequence and thus of the receiver detecting correctly that wrong sequence. By

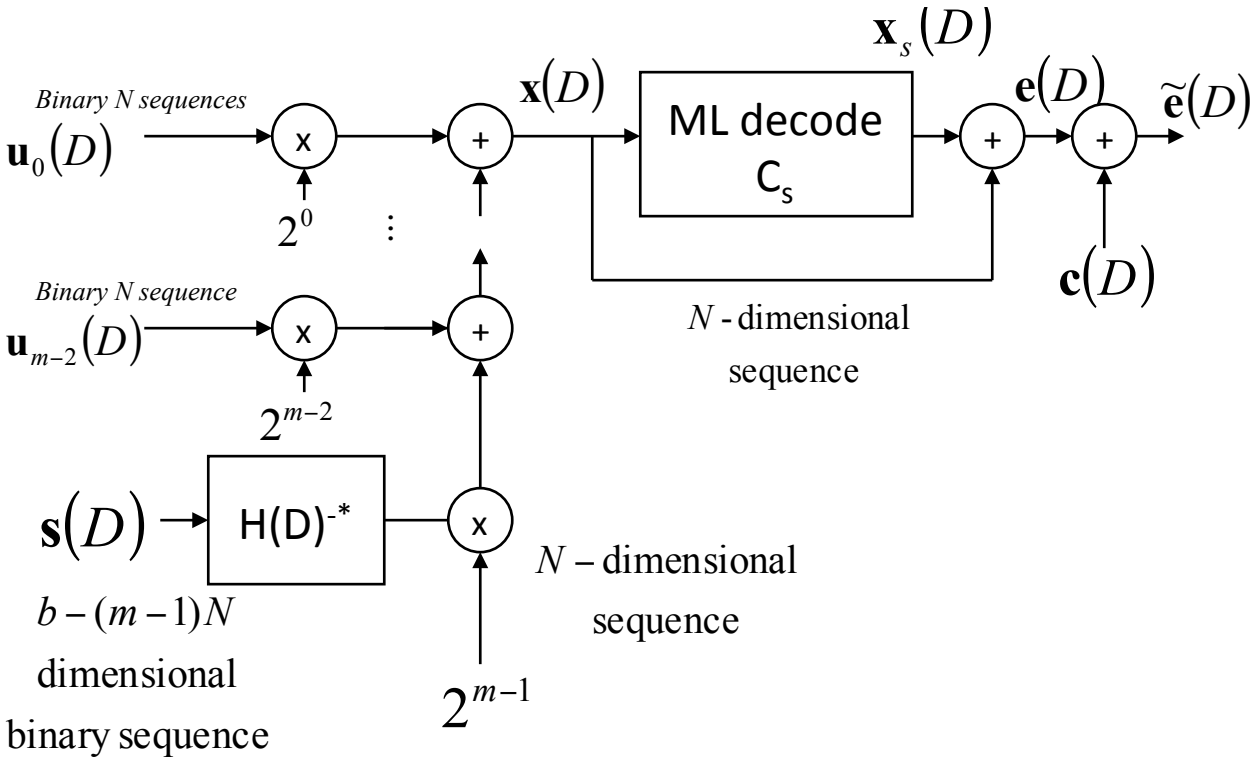


Figure 10.57: Trellis-shaping encoder for shaping code with syndrome generation of input vectors with scaling and $m \geq 2$.

using feed-back-free parity realization in the receiver for recovery of the "syndrome" sequence, any such errors with finite small non-zero probability are limited to a finite run (of course the transmit shaping code needs to be non-catastrophic also).

Calculation of the exact transmit energy basically is achieved via computer simulation in trellis shaping as it appears no known method for computing these gains exists (even for infinite number of points in the constellation). Figure 10.57 shows the encoder diagram, which works for $N \geq 2$ and $m \geq 2$ (the cases of smaller numbers of bits and dimensions is considered later in this subsection). When $N = 2$, then odd integer numbers of bits can be handled directly and an even integer number of bits requires the upper most encoding path choose only the sequences (0,0) or (1,1) instead of all 4 binary 2-tuples. For $N > 2$, then the granularity to integer bits is achieved in a similar manner as Voronoi shaping codes (as long as m is large enough). As with Voronoi codes, duals of good codes (lattices) tend to be best because the redundancy introduced is least (that is, a potentially desired event is H^{-*} is $(b-[m-1]N) \times (b-[m-1]N+1)$ so that only a doubling in constellation size is necessary; however, E_8 is a self dual and quadruples constellation size and has good shaping gain.)

Figure 10.58 illustrates the corresponding receiver. Again, the parallels with the Voronoi situation are clear. The sequences are processed N dimensions at a time. The $H(D)$ matrix should be chosen to be feed-back free to avoid error propagation.

Table 10.23 lists the two-dimensional trellis-shaping codes and their simulated gains (along with the delay in the ML search and b). These gains are all for 7 bits/symbol and a 256-point square constellation for $\boldsymbol{x}$ in Figure 10.57. The gains do not increase substantially for higher numbers of bits per dimension, but will reduce for lower numbers of bits per dimension. Feed-back-free parity matrices have been provided, but there are many generalized left inverses. In the last two codes, inverses with feedback were used (because they're easy to determine). Feedback in the transmitter is acceptable (but not in the receiver). However, the zeros in an inverse with 0 feedback reduces the redundancy in the two-dimensional constellation (fewer points can be selected in any two-dimensional symbol because of the

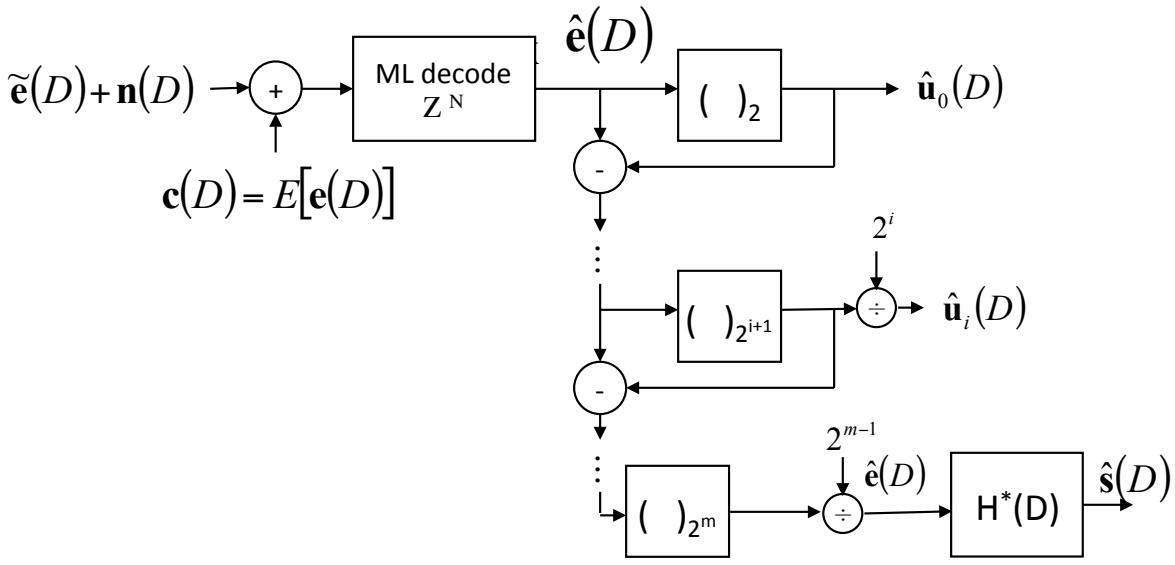


Figure 10.58: Trellis-shaping decoder for shaping code with syndrome generation of input vectors with scaling and $m \geq 2$.

zeros).

The drawback of the encoder is that only multiples (that is $m-1$) of $N=2$ bits is somewhat less restrictive with trellis-shaping codes than with Voronoi codes. However, the same method for fraction m values as in the Voronoi section can be used. Four and eight dimensional trellis shaping codes were investigated by Forney, but the shaping gains are less (although constellation expansion is less also). One-dimensional trellis shaping is capable of higher gains, but typically quadruples the number of points in one dimension, which may be unacceptable from an implementation standpoint. The additional gain is about .2 dB, but requires more states (and ML decoders than run at double the speed). Thus, the two-dimensional shaping codes seem preferred when trellis shaping may be used.

2^ν	γ_s (dB)	$H(D)$	$H(D)^{-*}$	delay
2	0.59	$[1+D \;\; 1]$	$[1 \;\; D]$	8
4	0.97	$[1+D^2 \;\; 1+D+D^2]$	$[D \;\; 1+D]$	26
8	1.05	$\begin{bmatrix} 1+D^2+D^3 & 1+D+D^3 \end{bmatrix}$	$\begin{bmatrix} 1+D & D \end{bmatrix}$	34
8	1.06	$\begin{bmatrix} 1+D^3 & 0 & D \\ 0 & 1+D^3 & D^2 \end{bmatrix}$	$\begin{bmatrix} \frac{1}{1+D^3} & 0 & 0 \\ 0 & \frac{1}{1+D^3} & 0 \end{bmatrix}$	42
16	1.14	$\begin{bmatrix} 1+D+D^4 & 0 & D+D^2+D^3 \\ 0 & 1+D+D^4 & D^2 \end{bmatrix}$	$\begin{bmatrix} \frac{1}{1+D+D^4} & 0 & 0 \\ 0 & \frac{1}{1+D+D^4} & 0 \end{bmatrix}$	100?

Table 10.23: Simulated gains and parity matrices for 2-dimensional trellis shaping at $b = 7$.

10.8 Block Codes

Block codes are a special case of convolutional codes when $\nu = 0$, as defined in Section 10.1. The inputs and the outputs of the encoder will be elements of a finite field (in this text exclusively a Galois Field, see Appendix A). There is a vast body of knowledge on block codes, their structure, properties, and efficient decoding on discrete memoryless channels. As with convolutional codes, this textbook is concerned largely with use of block codes, their enumeration, encoder and decoder implementations, and their performance analysis. There are numerous excellent textbooks dedicated to block codes like those of Wilson, of Lin and Costello, and of Blahut to which the reader is referred for more comprehensive treatments. This text views them as component tools in the design of the basic physical and link-layer digital transmission.

While there is much elegant theory of block codes, perhaps the most widely used codes are a special cyclic class of block codes known as Reed-Solomon codes. These are very powerful codes with high dmin and low redundancy that are used on DMC's to drive a nominal probability of bit error on the inner channel such as $\bar{P}_b = 10^{-6}$ to essentially zero. Such codes find good use in applications from wireline and wireless transmission to disk storage. Such block codes drive random bit errors to essentially zero, achieving then Shannon's promise of arbitrarily low probability of error in systems that desire or need essentially zero errors. Typically, block codes' symbol length N is much, much larger than in convolutional codes[13], albeit with $\nu = 0$. Some digital transmission applications (for instance digitized voice or video) are relatively insensitive to occasional errors, and block codes may not be used. However, the retrieval of a file from a disk with even 1 bit error in the recovered file could lead to a very unhappy file user. Similarly transmission of a file or message over the internet with a few errors in it could have catastrophic consequences for the communicants. Even though in the internet case, the transmission-control protocol TCP allows for detection and retransmission of errored packets, such retransmission can lead to delays, large intermediate memory requirements, and low throughput so reduction of error rates to acceptably small levels may be the function of the block code. Then the upper layer retransmission methods will have an easier design.

Reed Solomon block codes are often used with hard-decoding outside an inner coding system that may use soft decoding (see Chapter 11 for concatenated codes). They easily have blocks of hundreds to thousands of bits or Galois Field symbols to gain high efficiency ($k/n \to 1$) and large minimum distance, but straightforward maximum-likelihood decoding is computationally complex. Thus, the block codes of greatest interest are those that have structures that simplify enormously such decoding wihtout sacrificing too much efficiency. Similarly, there are channels that defy stationary or Gaussian descriptions that frustrate the types of ML receivers elsewhere in this text,leaving patches of errors that somehow need to be "cleaned up." modeling this inner frustrated system as a DMC then enables the outer block code to expunge the errors, no matter what their cause, although possibly this will lead to code rates k/n that are much less than one but nevertheless necessary to ensure quality communication.

Subsection 10.8.1 investigates the performance analysis of block codes on a DMC. There are some basic bounds on minimum distance versus code rate and block length that appear along with some useful methods to compute overall bit and symbol error probabilities as a function of the DMC (or most often BSC) error probability. Subsection 10.8.2 progresses to cyclic codes and their special "rotationally invariant" structure that simplifies many aspects of very powerful codes' implementations. The main codes discussed as special cases are Bose C... H... (BCH) and Reed Solomon codes, the latter of which can have very efficient encoder and decoder implementations as in Subsection 10.8.3 and 10.8.4 respectively. Subsection 10.8.5 then investigates selection of such codes to meet a system requirement.

10.8.1 Block Code Performance Analysis

10.8.2 Cyclic Codes

Cyclic codes have good distance properties and simplified implementations because they exploit the finite-field equivalent (on a memoryless channel) of circulant matrices that were discussed in Chapter 4 on

[13]Note the use of N instead of lower-case n has returned for block codes because the ambiguity with $N = \infty$ for convolutional codes has been removed.

DMT and OFDM transmission methods. Just as FFT's or Fourier Transforms allow large simplification there, certain equivalents in a finite field simplify when all appears periodic. Blahut's classic textbook on Error-Correction Codes does indeed comprehensively pursue this avenue of cyclic-transform connection and the reader is deferred there for full understanding of such structure. The present development will consider only some essential points with the aim of code use rather than structural elegance.

For convolutional codes, it was convenient to use a placeholder D to correspond to successive symbol time positions in an infinite-length code word. For cyclic block codes, N will typically be large aso the placeholder D will be reintroduced (even though with $\nu = 0$ in a convolutional code, there is no D) to correspond to successive time positions within the finite-length codeword. Thus,

$$v(D) = v_{N-1}D^{N-1} + vN - 2D^{N-2} + ... + v_1 D + v_0 \quad . \tag{10.307}$$

The individual elements are scalars in some Galois Field (rather than vectors as in the convolutional code); however Galois Extension Field scalar elements coincidentally often constructed from binary (or p-ary)vectors themselves. The usage should be clear from the context whether semi-infinite-length convolutional codewords or finite-length block codewords are intended. This entire section focuses entirely on the latter, while all other previous sections focus only on the former.

Definition 10.8.1 (Cyclic Code) *A code $\mathcal{C}$ is cyclic if any (and all) cyclic shifts of codewords are also codewords:*

$$\textit{if } v(D) \in \mathcal{C} \textit{ , then } \left(D^i v(D)\right)_{(D^N-1)} \in \mathcal{C} \quad . \tag{10.308}$$

Modulo $D^N - 1$ simply corresponds to replacing D^N by 1 in a polynomial where higher-order polynomial terms $D^{i \geq n}$ (and those correspond to terms outside the block length of interest anyway). This also corresponds to circular right shift, an operation easily undertaken in implementations. Thus, $,D^i$ means circular right shift i times. If $i < 0$, then left circular shift by i positions is implied, which produces the same result as circular right shift by $N - i$ positions.

This text is only interested in linear cyclic block codes, and it is possible to write any linear block codeword as

$$v(D) = u(D) \cdot g(D) \tag{10.309}$$

where $g(D)$ is a generator polynomial. Linear cyclic codes have a special generator form.

Theorem 10.8.1 (Cyclic Code Generator) *A linear cyclic code has a generator $g(D)$ that is the unique greatest common divisor (GCD) of all codewords.*

Proof: The GCD of all codewords must have degree less than $n-1$ because no codeword has a degree that exceeds $n-1$. Further, the unique GCD of any set of polynomials is always (from basic algebra) a linear combination of those polynomials so

$$\begin{aligned} g(D) &= \sum_i a_i(D) \cdot v_i(D) & (10.310) \\ &= \left(\sum_i a_i(D) \cdot v_i(D) \right)_{(D^N-1)} & (10.311) \\ &= \sum_i \left(a_i(D) \cdot v_i(D) \right)_{(D^N-1)} \quad , & (10.312) \end{aligned}$$

which is a linear combination of codewords of a linear code and so therefore $g(D)$ is a codeword itself. Clearly then $u(D) \cdot g(D)$ is a linear combination of codewords for all inputs $u(D)$ and thus all linear cyclic block code codewords are so generated. **QED**.

If $g(D)$ is of degree $P \leq N-1$, which means[14] that $g_P = 1$ and $g_0 \neq 0$, then possible inputs $u(D)$ must be of degree $K-1$ where $N = K + P$ and the code rate is $r = K/N$. There are thus P redundant "parity" symbols per codeword.

[14] $g(D)$ can always be scaled so that $g_P = 1$ without changing the code.

By writing

$$v'(D) = \left(D^{K+1} \cdot g(D)\right)_{(D^N-1)} = [g_{P-1} \ldots g_1\ g_0\ 0 \ldots 0\ 1] \tag{10.313}$$

$$= D^{K+1} cdot g(D) - \left(D^N - 1\right) \tag{10.314}$$

so then

$$v'(D) - D^{K+1} cdot g(D) = -\left(D^N - 1\right) \tag{10.315}$$

$$\text{multiple of} g(D) = \left(D^N - 1\right) \tag{10.316}$$

and the generator $g(D)$ is also a factor of $D^N - 1$. Thus, finding the factors of $D^N - 1$ is one way to design cyclic codes.

Furthermore, setting

$$h(D) = \frac{D^N - 1}{g(D} \tag{10.317}$$

defines a form of a "parity" polynomial because $v(D) \cdot h(D) = u(D) \cdot (D^N - 1)$, which is zero modulo $D^N - 1$. The implementation of such a parity polynomial with cyclic shift registers appears in Subsection 10.8.4.

10.8.3 Reed Solomon Encoder Implementations

10.8.4 Reed Solomon Decoder Implementations

10.8.5 Block Code Selection

Exercises - Chapter 10

10.1 *Our first convolutional encoder.*

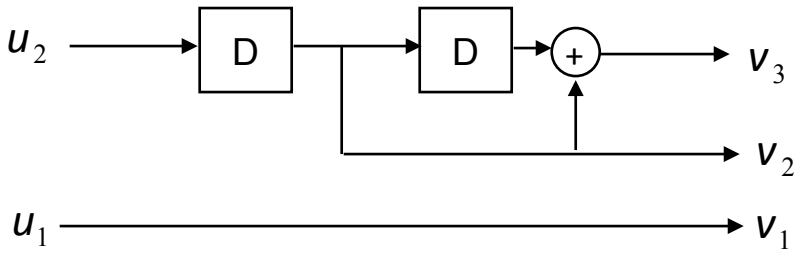


Figure 10.59: A convolutional encoder

a. (1 pt) Determine the rate and the constraint length of the convolutional encoder shown in Figure 10.59.

b. (3 pts) Determine the generator matrix $G(D)$ and the parity matrix $H(D)$ for the encoder.

c. (3 pts) Draw the trellis diagram. Please use the same notation and conventions as used in the trellis diagrams in the notes. *Hint:* Because u_1 is not stored as a state, you will have multiple paths between states. That is, you can have two different outputs that both have the same origin state and the same destination state but differ in the value of u_1.

10.2 *Systematic encoders, or encoding with feedback.*

a. (3 pts) For the convolutional encoder shown in figure 10.60, we have,

$$G(D) = \left[\begin{array}{ccc} 1+D^2 & D^2 & 1+D+D^2 \end{array} \right]$$

and one possible choice of $H(D)$ is (Remember that $G(D)H^*(D)$=0),

$$H(D) = \left[\begin{array}{ccc} D & 1 & D \\ 1+D & D & 1 \end{array} \right]$$

(Verify this!)
Convert $G(D)$ to a systematic generator matrix $G_s(D)$. Show that

$$C(G) = C(G_s).$$

Hint: Can you use the $H(D)$ matrix to show this?

b. (3 pts) Implement the systematic encoder $G_s(D)$. (i.e., draw a flow diagram.) The systematic encoder will involve feedback.

10.3 *Convolutional Code Analysis.*
A convolutional code is described by the trellis shown in Figure 10.61. The state label is u_{k-1}. The upper state is 0 and the lower state is 1. Other labellings follow the conventions in the text. There are 2 outputs; v_2 and v_1.

a. (2 pt) Determine the rate and the constraint length of the convolutional encoder corresponding to this trellis. *Hint*: Can you see from the trellis that the encoder has only one input u?

b. (2 pts) Determine the generator matrix $G(D)$ for the code.

c. (2 pts) Draw a circuit that realizes $G(D)$.

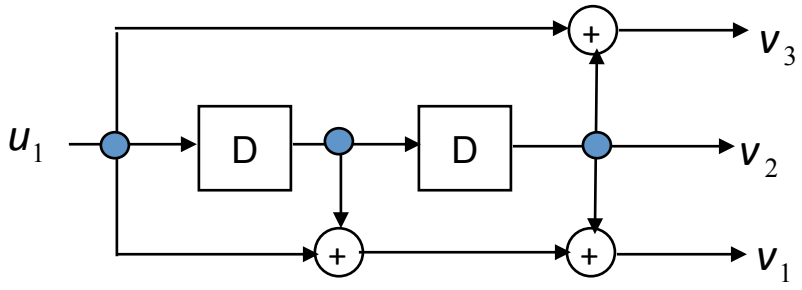


Figure 10.60: Another convolutional encoder.

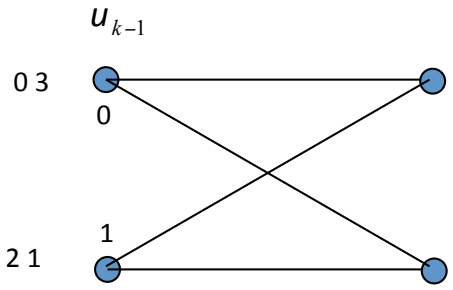


Figure 10.61: Trellis corresponding to a convolutional encoder.

d. (1 pt) Find d_{free}.

e. (3 pts) Find the extended transfer function $T(W, L, I)$ for the code. How many codewords are there, each of weight 3? How many input bit errors occur in an output error event of length l?

10.4 *A catastrophy.*

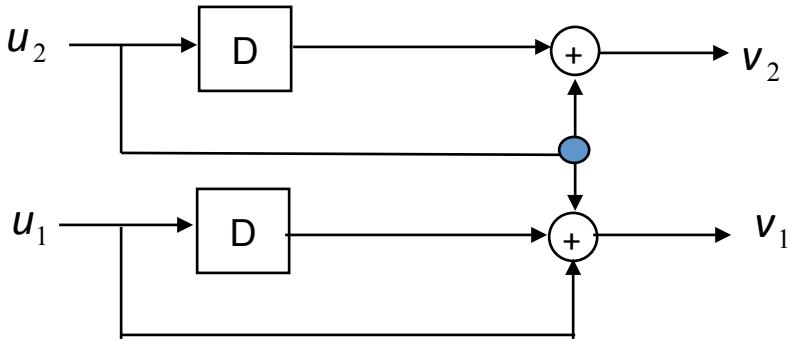


Figure 10.62: A rate 1 convolutional code.

a. (5 pts) Draw an extended transfer function flow diagram (i.e. include W, L, and I weights.) for the rate 1 (!) convolutional code shown in Figure refcv3. We will use this diagram only to see if there are any pathological loops causing the code to be catastrophic. Thus, you don't have to draw the paths from the "in" state 00 or paths to the "out" state 00. This should make your drawing task a little easier.

b. (2 pts) Specify the loops and their corresponding gains which show that this code is catastrophic.

c. (2 pts) Give *two* examples of pairs of input sequences which have an infinite Hamming distance between them but produce corresponding output sequences (using the convolutional encoder above) that have only a finite Hamming distance between them.

d. (1 pt) A non-catastrophic rate 1 convolutional code cannot have a d_{free} larger than 1. Give an example of a *simple* rate 1 convolutional code which achieves $d_{free} = 1$ and is not catastropic. *Hint:* By *simple*, we mean *very simple*.

10.5 *Coding Gain*

Assume we are using a binary transmission system on an AWGN channel which has $SNR = 6.5dB$.

a. (1 pt) Determine the P_e for uncoded binary PAM on this channel.

b. (2 pts) Suppose that we're allowed to double our symbol rate (and hence, the bandwidth), but we cannot increase transmit power. With these constraints, we need to transmit at the same data rate R as in part (a). Use the tables provided in Section 10.2 to find the code which provides the largest free distance (among those on the tables) of all the codes which do not reduce the rate r by more than a factor of two, and which require no more than six delay elements for implementation. Give $G(D)$ for the code.

c. (2 pts) What is the coding gain of this code? Use ing soft decoding, approximate the P_e of the coded system. How much did coding reduce our P_e?

10.6 *Convolutional Code Design.*

A baseband PAM ISI channel with Gaussian noise is converted to an AWGN channel, by using a ZF-DFE, operating at a symbol rate $1/T = 10$ kHz. It is calculated that $SNR_{ZF-DFE} = 7.5$ dB.

a. (2 pts) For the AWGN channel produced by the DFE, what is the capacity $\overline{C}$ and C?

b. (1 pt) Find the P_e for uncoded 2-PAM transmission on this channel.

c. (5 pts) The P_e produced by uncoded PAM (part (b)) is intolerable for data transmission. We would like to use convolutional coding to reduce the P_e. However, since we're using a DFE that has been designed for a specific symbol rate, we cannot increase the symbol rate (ie. bandwidth). Nor are we allowed to increase the transmitted signal power. Further, we have a limit on the decoder complexity for the convolutional code, which is represented mathematically as,

$$\frac{1}{k} \cdot [2^{\nu}(2^k + 2^k - 1) + 2^n] \quad \leq \quad 320$$

where k, n, ν have the usual meaning for a convolutional code. Under these constraints, design a convolutional encoder, using the convolutional coding tables of Section 10.2, so as to achieve the highest data rate R for $P_e < 10^{-6}$. What is this maximum R?
Hint: Use the soft decoding approximation to calculate P_e and remember to account for N_e.

d. (2 pts) Draw the systematic encoder and modulator for the convolutional code designed in part (c).

10.7 *Concatenated Convolutional Code Design - 19 pts - Midterm 1997*

A baseband AWGN has SNR = 8 dB for binary uncoded transmission with symbol rate equal to clock rate of 1 Hz. (Recall that the clock rate of an AWGN channel can be varied when codes are used.)

a. What is the capacity C of the AWGN channel in bits/sec? (1 pt)

b. Design a convolutional code with $r = 3/4$ bits per dimension that achieves the lowest probability of error using one of the codes listed earlier in this chapter, assuming a soft MLSD decoder. This coded system should have the same data rate as the uncoded system. (4 pts) ("Design" means for you to provide an acceptable generator matrix $G(D)$ and to compute the corresponding probability of symbol error.)

c. Model the system in part b as a binary symmetric channel (BSC) and assume that N_b for the code you selected is 2. Provide the value $p =?$ (1 pt)

d. For the BSC in part c, Design a convolutional code of rate $r = 2/3$ that achieves the lowest probability of symbol error using codes listed in Section 10.2 (4 pts)

e. What is the combined data rate R of the "concatenation" of the codes you designed in parts b and d? (2 pt)

f. Return to the uncoded transmission system and now design an $r = 1/2$ convolutional code with soft decoding that has lowest probability of error based on codes in Chapter 7. This coded system should have the same information rate, R, as the concatenated code above in part e. (Hint - think carefully about what clock rate you would use.) (4 pts)

g. Compare the system in part f to the system in part d. What does this comparison tell you about concatenation of codes? (3 pts)

10.8 *Coset codes – an ineffective design.*

You will show that simple uncoded modulation, and a simplistic application of convolutional codes to QAM, are both (ineffective) special cases of coset codes.

a. (2 pts) For the special case of *uncoded* (379A) 4-QAM modulation, specify k, b, and the order of the partition.

b. (1 pt) With $\bar{\mathcal{E}}_x = 1$, compute d_{min}, the minimum distance between two valid signals for the encoder of the previous part. Recall that for QAM, we have

$$d = \sqrt{\frac{12\bar{\mathcal{E}}_x}{4^{\bar{b}} - 1}}.$$

c. (2 pts) For the case of 16-QAM modulation preceeded by a rate 2/4 convolutional code, specify k, b, and the order of the partition. Note that the situation described here is simply a convolutional code cascaded with a modulator. It is not a 'normal' coset code.

d. (1 pt) Keeping $\bar{\mathcal{E}}_x = 1$, compute d_{min} for the 16-QAM constellation.

e. (2 pts) Continuing with the encoder of the previous two parts, assume that for the convolutional encoder of the coset code, we use two copies of the 4 state, rate 1/2 code given in Table 7.1 which has $d_{free} = 5$ (We use 2 copies so that there are 2 inputs and 4 outputs overall). Let's make the following assumptions,

- Assume soft decoding of the coset code at the receiver (so that the d^2 metric is relevant).
- Assume that the d^2_{min} for the coset code occurs when the transmitted sequence of symbols is $0, 0, 0, 0, 0, ...$ and the detected sequence is $0, 3, 2, 3, 0, ...$ (see figure below for labelling conventions). Note that these sequences are consistent with the rate 1/2 convolutional code trellis. Note also that the minimum distance of the 16-QAM constellation is as found in part (d).

Show that with these assumptions, the minimum distance between two valid signals for this encoder is exactly the same as that found in part (b). Thus, this simple example of a 'coset' code does not provide any coding gain. This shows that one has to carefully design coset codes in order to get coding gain.

10.9 *32-CR trellis code.*

Here, we examine a coset code where the trellis error distance is less than the intra-coset error distance. This code has $\Lambda = 32$-CR (with minimum distance between the lattice points $d = 1$) and

$$G(D) = \begin{bmatrix} 1 & 0 & \frac{D^2}{1+D^3} \\ 0 & 1 & \frac{D}{1+D^3} \end{bmatrix}.$$

Note that this $G(D)$ is examined in detail in Subsection 10.4.2.

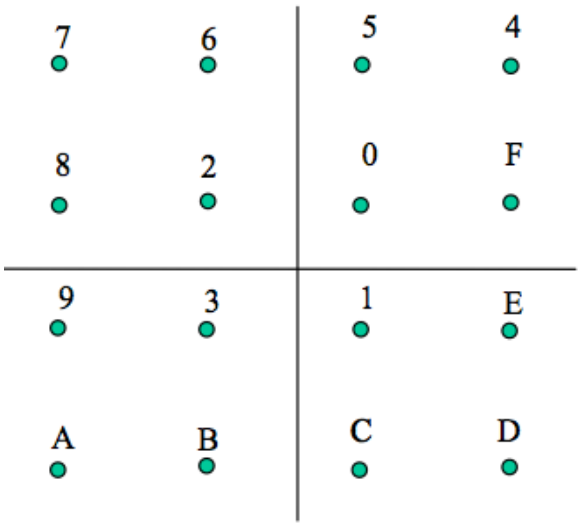


Figure 10.63: 16-QAM used in the coset code.

a. (2 pts) Draw the encoder. Explicitly draw the convolutional coder. *Hint:* Most of the encoder can be copied from the relevant diagram in the course reader!

b. (3 pts) Draw the 32-CR constellation and give the Ungerboeck labeling of the points (i.e. $\{D0, \ldots, D7\}$). Label each point by its associated bit pattern $(v_4, v_3, v_2, v_1, v_0)$ also. *Important:* To make sure that we have a common notation, ensure that the labeling you have on the inner 16 points of the 32-CR constellation is the same as the Ungerboeck labeling for the 16-QAM constellation of Subsection 10.4.2.

c. (3 pts) We need to find the minimum distance for the code. As a first step, fill in the following table with the minimum distance between the various cosets. Some of the entries have been filled in for you.

	$D0$	$D1$	$D2$	$D3$	$D4$	$D5$	$D6$	$D7$
$D0$	$2\sqrt{2}$	1	$\sqrt{2}$					
$D1$								
$D2$								
$D3$								
$D4$								
$D5$								
$D6$								
$D7$								

d. (3 pts) Now conclude that the trellis distance (ie. distance between a sequence of cosets) and intra-coset distance (ie. the parallel transitions distance) for our coset code are exactly the trellis distance and intra-coset distance of the code in Subsection 10.3.2.

e. (3 pts) Compute the shaping gain, coding gain, and fundamental gain of our coset code.

10.10 *A one dimensional trellis code.*

Consider the trellis code $G(D) = [a_0 \oplus a_1 D \quad a_2 \oplus a_3 D]$ with an M-PAM constellation. Let d be the minimum distance of the M-PAM constellation.

a. (4 pts) For $M = 8$; what are the values of b, r_G and k. Draw the signal constellation and give the Ungerboeck labeling for the required level of partitioning (label so that the most negative point of the constellation is included in the $C0$ coset). What is the intra-coset error distance? Repeat for $M = 4$. What is the intra-coset error distance?

b. (4 pts) For the convolutional encoder $G(D) = [1 \;\; 1 \oplus D]$ and $M = 4$, find the trellis error distance using the zero input (top branch) as reference. This is not the minimum distance of the trellis code. Try to find a pair of sequences that are at a smaller distance. What is the true value of the minimum distance of the code? *Moral of the problem:* Even though the $G(D)$ is a *linear* convolutional encoder, yet the d_{min} of the trellis encoder cannot be evaluated just by comparing with the all zero-coset sequence.

10.11 *Capacity and Design. 14 pts - Midterm 2003*

An AWGN channel has SNR=18 dB with uncoded PAM transmission at a symbol rate of 10 MHz. For parts a-d, a symbol rate of 10 MHz is maintained. The data rate is 20 Mbps.

a. What is the capacity of this channel in bits per second? (2 pts)

b. What is $\bar{b}$? (1 pt)

c. What is the probability of symbol error for uncoded PAM transmission at the data rate of 20 Mbps? (2 pts)

d. Design a trellis-coded system that achieves $\bar{P}_e \leq 10^{-6}$, showing (5 pts total)

   (i) $\bar{P}_e$ calculation. (1 pt)

   (ii) show the constellation and bit labels for each point (2 pts)

   (iii) show $G(D)$ circuit implementation in systematic form (2 pts)

e. Could a convolutional code with hard decoding be used directly somehow to achieve the same $\bar{P}_e$? If so, how and what is the smallest number of states that could be used? (4 pts)

10.12 *Coset code design for HDTV.*

A digital satellite channel with $SNR = 17.5$ dB uses two dimensional symbols (QAM) with a symbol rate of 35 MHz to send 140 Mbps digital HDTV signals.

a. (2 pts) How many information bits per 2-D symbol are required to send the HDTV signals? What 2-D constellation would typically be used to send this many bits (uncoded)? What 2-D constellation would typically be used to send the same number of information bits in a coset code?

b. (1 pt) What is the P_e obtained by uncoded transmission using the constellation found in the previous part?

c. (2 pts) What $\tilde{\gamma}_f$ is needed for a coset code to achieve $P_e \leq 10^{-6}$?

d. (3 pts) Find the lowest complexity code that will achieve $P_e \leq 10^{-6}$. Find the associated P_e, parity matrix, and systematic encoder matrix.

e. (3 pts) Draw the encoder circuit and the Ungerboek labeled constellation.

Note: A satellite channel typically has no ISI. Therefore, no DFE is required, and so, there is no problem of error propagation.

10.13 *Coset code decoding.*

Consider a one dimensional trellis code with 2 uncoded bits being coded to 3 bits and with $G(D) = [1 + D^2 \;\; D]$. The coded constellation is illustrated in Figure 1 with labeling (v_2, v_1, v_0).

Figure 10.64: Coded PAM constellation.

a. (3 pts) Find the sequence of constellation points produced by the coset code described above, given the following sequence of input bits. Assume that the initial state of the convolutional coder is **00**.

01 11 10 00 11 11

Each pair of input bits in the above sequence is written as $u_2\ u_1$.
Follow the notation in the text - in particular, note that $v_2 = u_2$ here (refer to fig. 8.14 in the text).
Note : Hopefully, you've recognised that the $G(D)$ for this code is the Ungerboeck 4-state, rate 1/2 convolutional coder. The trellis for this can be found in the text.

b. (2 pts) The constellation points found in the previous part are sent over a noisy channel producing the following points at the receiver:

-1.02 3.25 3.44 -2.45 1.6 4.2

Discuss the complexity of a maximum likelihood decoder for this sequence. Specifically, show that the complexity will be $N_D = 2^{\nu+k}(2 - 2^{-k})$. What is the numerical value of N_D for our coset code?

c. (5 pts) Decode the received sequence using MLSD. Assume that the initial state is **00**.

10.14 *Sum constraints.*

The constraints we derive below are not likely to come up in the future. However, proving them will deepen your understanding of the lattices involved. *Hint*: Every time you impose an even or odd constraint on the sum of components of the N-dimensional points in a lattice (coset), you are reducing the effective number of possible points to half. *E.g.*: if you impose $x_1 + x_2 = even$ in Z^2 then you get D_2.

a. (2 pts) Show that (x_1, x_2) is an element of $D_2 = R_2 Z^2$ iff $x_1 + x_2$ is even.

b. (1 pt) For the rest of the problem we will be interested in eight-dimensional points $x = (x_1, x_2, \ldots, x_8)$. We will specifically be interested in showing that certain 8-D lattices can be described in terms of constraints on the four pair sums

$$\sum_{i=k}^{k+1} x_i \qquad \text{for} \quad k \in \{1, 3, 5, 7\}.$$

Show that $x \in R_8 Z^8$ iff all four pair sums are even.

c. (2 pts) Show that $x \in DE_8$ iff either all four pair sums are even or all four pair sums are odd.

d. (2 pts) Show that $x \in {D_4}^2$ iff

$$p_1 p_2 p_3 p_4 \in \{eeee, oooo, eeoo, ooee\}$$

where p_i refers to the i^{th} pair sum and e and o indicate that the pair sum is even or odd. Restate this as a constraint on two quad-sums.

e. (2 pts) Show that $x \in D_8$ iff an even number of the four pair sums are odd. Restate this as a constraint on the sum of the eight terms.

f. (Optional, 0 pt) We derived sum constraints for

$$Z^8/D_8/{D_4}^2/DE_8/R_8Z^8.$$

It is often more useful to replace R_8Z^8 with E_8 because of its superior distance properties. Give a sum constraint for E_8 that provides an if and only if relationship. (You need not prove your relationship.)

10.15 *8D is too easy.*

In this problem, we'll test our understanding of lattices by considering those with more than 8 dimensions.

a. (2 pts) Construct D_{16} using D_8. What sum constraint do the elements in D_{16} satisfy?
Hint: Basically, extend the idea of lattices D_2, D_4, D_8 to define the 16 dimensional lattice D_{16}.

b. (2 pts) Consider the 16D constellation formed by choosing points from 8 separate 4QAM constellations. The four points in the 4QAM are $x_0 = (1, 1)$, $x_2 = (-1, -1)$, $x_1 = (1, -1)$ and $x_3 = (-1, 1)$. For the point $[x_1\ x_2\ x_3\ x_2\ x_*\ x_0\ x_0\ x_0]$ in D_{16}, what are the possible values of x_*?
Hint: Use a modification of the result in part (a). The modification is required because, strictly speaking, the 4-QAM constellation given above is not part of the Z^2 lattice.

c. (1 pt) What is the value of γ_f for D_{16}.

d. (2 pts) Consider now the case D_{2^M}, where M is a positive integer. What sum constraint do the elements in D_{2^M} satisfy? What is the value of γ_f?

e. (1 pt) What problem do we encounter, as far as lattice gain is concerned, as M grows?

10.16 *A Lattice Code based on the E_8 lattice.*

The code consists of a sequence of points chosen from the E_8 lattice. There is no redundancy used (ie. $r_G = 0$). Thus, this is a simple example of a lattice code. It was shown in the text that this code has a fundamental gain $\gamma_f = 3$ dB.
Note that each E_8 lattice point is actually transmitted as a sequence of four 2D QAM symbols.

a. (2 pts) Assume that there are $b = 12$ input bits $u_1, ..., u_{12}$. Each block of these 12 bits has to be encoded into one E_8 lattice point. The first step in encoding is to select a sequence of four 2D cosets to be transmitted. How many of the input bits are used to specify this sequence of 2D cosets?
Hint : Look at the 2D trellis for the E_8 lattice shown in the text.

b. (2 pts) Argue that, in order to transmit the 12 bits of information using this lattice code, each of the four 2D symbols should be selected from a 16 SQ-QAM constellation.

c. (2 pts) The 16 QAM constellation with Ungerboeck labeling is shown in Figure 10.65. Denote each symbol by the pair **(x,y)**. Assuming that the symbol **(0.5,-0.5)** is in the sub-lattice $2Z^2$ (ie. the origin has been appropriately shifted), redraw the 2D trellis for the E_8 lattice, such that the 2D cosets are labeled $C0, C1, C2, C3$.
Hint : The $2Z^2$ label in a branch of the trellis is replaced by the $C0$ label, etc.

d. (3 pts) Which of these 2D sequences represent valid E_8 lattice points?

(i) **(0.5,-0.5)**,**(0.5,-0.5)**,**(-1.5,1.5)**,**(0.5,1.5)**

(ii) **(1.5,0.5)**,**(-0.5,0.5)**,**(0.5,-0.5)**,**(1.5,0.5)**

(iii) **(0.5,0.5)**,**(-0.5,1.5)**,**(0.5,-1.5)**,**(1.5,-0.5)**

Hint : Use the trellis drawn in part (c).

e. (1 pt) An E_8 lattice point is transmitted (as a sequence of four 16 QAM symbols) over an AWGN channel. At the channel output, the following noisy sequence is obtained **(0.6,-0.5),(-**
A (sub-optimal) 2D symbol-by-symbol detector would select the nearest 16 QAM point for each of the above received symbols. Decode the above sequence in this sub-optimal manner to get a sequence of four 16 QAM symbols. Does this sequence represent a point in the E_8 lattice?

f. (4 pts) Decode the received sequence in part (e) optimally, using MLSD, to get the transmitted E_8 lattice point.
Hint : Use the 2D trellis found in part (c).

10.17 *Shaping gain limit.*

In this problem we will show that the limit for γ_s is actually 1.53 dB. For this we will calculate $V_x^{2/N}/\bar{\mathcal{E}}$ of the N-dimensional (where N is even) square Amplitude Modulated Constellation (ie. a generalization of SQ-QAM for N dimensions), and compare it to the $V_x^{2/N}/\bar{\mathcal{E}}$ for the most power efficient signal strategy in N-dimensions, which is to choose points uniformly from within an N-dimensional sphere centered at the origin. Note that V_x refers to the total volume of the constellation (ie. product of the Voronoi region volume and the number of constellation points). It is assumed that there are a large number of points in both constellations.

a. (2 pts) Show that for the N-dimensional Amplitude Modulation, we get the approximate formula

$$\frac{V^{2/N}}{\bar{\mathcal{E}}} = 12$$

This is our reference constellation.
Hint : Recollect the $\bar{\mathcal{E}}$ formula for such a constellation. It would be helpful to note that PAM,QAM,TAM, etc. follow a similar $\bar{\mathcal{E}}$ expression.

b. (2 pts) Now, we need to calculate the same quantity for the spherical constellation. To this end, note that the volume of an N-dimensional sphere is $\pi^k r^{2k}/k!$, where $N = 2k$. Using the continuous approximation for constellation power evaluation

$$\bar{\mathcal{E}} = \frac{1}{NV(\Lambda)} \int_{r\in\Lambda} r^2 dV(r)$$

to evaluate the $\bar{\mathcal{E}}$ for the N-dimensional spherical constellation, we get

$$\bar{\mathcal{E}} = \frac{R^2}{2(k+1)},$$

where R is the radius of the sphere. Verify this for $N = 2$.

c. (3 pts) Calculate γ_s as a function of k. Verify that the limit as $k \to \infty$ is $\frac{\pi e}{6}$ which is 1.53 dB.
Hint : You may need to use the Stirling approximation formula for factorials,

$$k! \approx \sqrt{2\pi k}\left(\frac{k}{e}\right)^k \qquad \text{for large k}$$

10.18 *Sphere bound on fundamental gain. (Due in part to Forney)*

In this problem, we investigate the bound on the fundamental gain γ_f.

a. (1 pt) The absolute minimum volume that could be associated with $d_{min} = 2r$ in an N-dimensional lattice is the volume of the N-dimensional hyper-sphere of radius r. Why? Note that we cannot actually achieve a volume this small with a real lattice.

b. (1 pt) Use the observations of the previous part to provide an upper bound for the fundamental gain achievable by a lattice of dimension $N = 2k$. This is called the 'sphere bound'. Recollect that the volume of an $N = 2k$ dimensional hyper-sphere with radius r is

$$V = \frac{\pi^k}{k!} r^{2k}.$$

c. (2 pts) As mentioned in part (a), this bound cannot be achieved by real lattices. Compare (in dB) the fundamental gain of the best lattices in 4 and 8 dimensions (that you've seen) with the sphere bound on the best possible fundamental gain in 4 (i.e., 1.5 dB for the D_4 lattice) and 8 (i.e., 3 dB for the E_8 lattice) dimensions.

d. (1 pt) Use Stirling's formula,

$$k! \approx \sqrt{2\pi k} \left(\frac{k}{e}\right)^k \qquad \text{for large k}$$

to conclude that the sphere bound on fundamental gain grows without bound as k increases.

e. (1 pt) While the sphere bound is only a bound, it is true that the actual fundamental gain also grows without bound as k increases. This would seem to contradict the fact that for $P_e = 10^{-6}$, the gap to capacity for uncoded transmission is only 9 dB. Explain this apparent contradiction.

10.19 *Multi-dimensional trellis code.*

The 2-state 4D code of Figure 10.25 is used to send $b = 8$ bits/4D-symbol. We will implement this 4D system using successive 2D transmissions.

a. (3 pts) Find an appropriate 2D constellation to use for our implementation. Your 2D constellation will have four 2D cosets, C0, C1, C2, C3. Label the points in your constellation so that the point in the upper right quadrant closest to the origin is C0 and the point in the upper left quadrant closets to the origin is C1.

b. (3 pts) Draw an encoder that generates each 4D output as two successive 2D outputs.

c. (4 pts) Assume the 2D constellation has scaled and shifted so that it has minimum distance 1 and the points are in $Z^2 + (\frac{1}{2}, \frac{1}{2})$. Three 4D symbols are transmitted after the state was known to be 0. The 4D received values are:

(.56, .45, 1.45, -1.55), (-.5, -2.5, -.6, 1.5), (.45, -.55, .5, -1.6)

Using the coset labelling of (a), find the minimum squared error sequence of 4D cosets. Also find the associated sequence of 2D cosets. Give the overall squared error of the minimum squared error sequence. How close was the nearest alternative to this squared error?

10.20 *Integer bit constellation restriction with trellis codes - 11 pts.*

An AWGN has an SNR of 23 dB when the symbol rate is $1/T = 1$ MHz. QAM transmission is used with the restriction that the modulator can only implement SQ QAM constellations with an integer number of bits (that is only 4QAM, 8 SQ, 16 QAM, 32 SQ, etc. are allowed).

a. What is the maximum bit rate with uncoded transmission for which the probability of symbol error is less than 10^{-6}? (1 pt)

b. The encoder for the 4D 16-state Wei code is available for use. Theoretically with no restrictions on constellation design, what is the maximum bit rate that could be achieved when this code is used? What is the lower bit rate that is achieved when only the SQ constellations above can be used and each constellation point appears on average with the same probability? (2 pts)

c. Design a look-up table that could be inserted between the Wei encoder output and the SQ-QAM modulator that would allow 6 Mbps transmission and possibly different probabilities for the different constellation points. Then design a feedforward encoder, table, and constellation that can be used and show labels for points in the constellation. (3 pts)

d. Suppose instead that the modulator can be time-varying in that constellations of different SQ sizes can be sent on subsequent symbol periods, allowing both number of levels transmitted to change and the distance between points on any given transmitted symbol to vary (which means energy can vary from symbol to symbol in time). Now redesign avoiding the use of the look table, and compute the new coding gain for this case. Was much lost? (3 pts)

e. Often in transmission, a system designer may be presented with an even number of parallel AWGN transmission channels with the same symbol rate that have different SNR. Application of a code individually on each channel leads to longer system/decoder delay, so application of a code across the parallel channels is desirable. How might a 4D trellis code be applied in this case? (1 pt)

f. Does the modulator as described present any difficulty for 1D constellations? (hint: we are looking for a one sentence answer not a reiteration of this entire problem.) (1 pt)

10.21 *Basic Trellis Code Design - 8 pts - Final 1997*

An AWGN channel has SNR=16.4 dB. QAM modulation is to be used on this channel with symbol rate 5 MHz. The probability of symbol error, P_e, needs to be less than 10^{-7}.

a. What is the highest data rate than can be achieved with uncoded QAM transmission? (1 pt)

b. What is the capacity of this channel? (1 pt)

c. Design a system with a 2-dimensional trellis code that achieves a data rate of 20 Mbps and still meets the probability of symbol error requirement. Write a systematic convolutional encoder matrix $G(D)$ and show a labeled constellation with octal notation and the least significant bits indicating the coset index. (4 pts)

d. Assume that an add or compare operation takes one instruction on a programmable processor being used to implement the maximum likelihood decoder for your design in part c. How many instructions per second must the computer be able to execute to implement your design? (2 pts)

10.22 *Design with a popular code - 10 pts - Final 1997*

We are to use a popular 16-state four-dimensional code on an AWGN with $b = 6$ over the $N = 4$ dimensions.

a. How many points would you use in a 2D coded constellation? (1 pt)

b. Show your constellation (1 pt). (hint: don't label cosets yet)

c. What is the minimum SNR required on the AWGN to achieve a symbol error probability of less than 10^{-6}? (1 pt)

d. What are the four-dimensional offset vectors $\boldsymbol{g}_i$, $i = 0, , 3$ for a linear implementation of the coset selection/offset? Cite any potential problems with this linear implementation. (2 pts)

e. For your constellation in part b and offsets in part d, label the two-dimensional cosets and use the modulo-4 integer addition (presumably implemented more simply than real-valued vector addition) as in the table below to implement the labeling of your constellation. This mod-4 addition is to be applied independently to the value of the transmitted symbol in each of the two dimensions. Show in your implementation diagram the position where the constant $[-.5, -.5]$ should be added to each two-dimensional constituent subsymbol. (3 pts).

value	0	1	2	3
mod-4 value	0	1	2	-1

Remember to recompute energy.

f. The fundamental gain of this code can be increased by 1 dB for certain values of b without increase in complexity. Find these values. (hint: minimum distance for trellis codes is the smallest of the parallel transisition distance and the distance between two sequences that span more than one symbol). (2 pts)

10.23 *Code Complexity - 12 pts - Final 1997*
An E8 lattice is used alone as an 8-dimensional code when $b = 20$ on an AWGN.

a. How many points would appear in two dimensional constituent symbols? (1 pt)

b. What is the decoder complexity, N_D , if the Viterbi algorithm is used according to the trellis in Figure 10.27? (1 pt)

c. Suppose instead that a ML decoder was used that simply computed the sum squared differences between each possible codeword and the received sequence. What would be the complexity of this decoder? Compare this answer to your answer in part b. (1 pt)

Let us suppose that a new "superstar" trellis code has been found based on the partition Z8/2E8 , which has an effective fundamental coding gain of 7.1 dB, and fundamental coding gain of 7.3 dB, with and minimum distance paths dominating in determining effective coding gain. This code has 8192 states. This code also is used for transmission with $b = 20$. The eight-dimensional constellation for the superstar has double the number of points that would appear in an uncoded 8-dimensional constellation.

d. What is the rate of the convolutional code inside the trellis code, $r_G =?$ (1 pt)

e. What 2D constellation should be used? (1 pt)

f. How many subsets would partition this 2D constellation and what letter in the alphabet should we use to describe them? (1 pt)

g. What is the normalized complexity of the Viterbi decoder for this code,$\bar{N}_D = ?$ (2 pts)

h. Suppose at any stage of Viterbi decoding of the trellis corresponding to the convolutional code, only 33 survivors are kept (those with the 33 smallest cost functions). Approximate the new maximum normalized complexity be for this suboptimum decoder? Compare to the answer in part g. (2 pts)

i. Given that large noise samples are rare with Gaussian noise, how much would you expect the suboptimum decoder in part h to lose in performance with respect to the full Viterbi detector characterized by the complexity in part g? (2 pts)

10.24 *Combined Convolutional and Trellis Code Design - 10 pts - Final 1997* A combination of a trellis code and a rate-3/4 convolutional code are used on a bandlimited channel with a $SNR_{MMSE-DFE,U}$ = 8.4 dB. The number of information bits/dimension transmitted is $\bar{b} = .75$. The system must have probability of bit error less than 10^{-7}. Assume a Tomlinson precoder will be used to avoid error propagation, but the loss of such precoding needs to be included in your design.

a. Which code, trellis or convolutional, must be the inner code, assuming no iterative decoding? (1 pt)

b. What is the lowest probability of bit error that the one-dimensional trellis code alone can achieve (at $\bar{b} = 1$) with trellis codes known to you in EE379B? You may assume a symbol error leads to an average of 5 bit errors with your choice of code. (4 pts)

c. Model the system in part b as a binary symmetric channel (BSC). Provide the value $p =?$ (1 pt)

d. For the BSC in part c, Compute the probability of bit error for the best convolutional code that you know from EE379B, again assuming that one symbol error in the decoder corresponds to 5 bit errors. (2 pts)

e. How might you further reduce probability of error? (2 pts)

10.25 *Lost Charger (10 pts) – Final 2006*

Your PDA nominally uses (uncoded) 16 QAM to transfer data at 10 Mbps to a local hub over a wireless, stationary AWGN channel probability of symbol error 10^{-6}. In transferring data, 100 mW of transmit power uses 500 mW of total PDA battery power, and you may assume that the ratio of 5:1 consumed/transmitted power holds over a range of a factor 4 above or below 100 mW. Unfortunately, youve lost your battery charger and must transfer a 10 Mbyte file (your exam) sometime within the next 3 hours to the internet for grading, or get an F otherwise, at the same sufficiently low probability of error at the fixed symbol rate of nominal use. Your battery has only 1.5 Joules of energy left (1 Watt = Joule/second). You may assume that the binary logic for a convolutional code, as well as any coset and point-selection hardware, uses negligible energy in a trellis encoder.

a. Design a 2-dimensional trellis encoded system that allows you to transmit the file, showing the encoder, constellation, and labeling at the same symbol rate. (5 pts)

b. The hub receiver is a separate device that would have consumed 0.1 Joules of energy for nominal uncoded transmission, assuming $N_D = 1$. How much energy will it consume for your design in part a? (2 pts)

c. Suppose the symbol rate could be changed. What is the minimum energy that any code could use to transmit this (10 Mbyte) file? Equivalently what is the largest size file that could be transferred with your battery (1.5 Joules)? (3 pts)

10.26 *SH Diagrams - 11 pts*

Section 10.7 describes A 128 shell constellation for $d = 2$.

a. Draw this constellation showing all 17 shells. (4 pts)

b. How many shells does a 64SH constellation have? (2 pts)

c. Draw a 64SH constellation and show the shells (3 pts)

d. What is the shaping gain of 64SH? (2 pts)

10.27 *Hexagonal Shells - 9 pts*

Consider hexagonal packing and shaping gains in this problem.

a. For a constellation based on the A_2 hexagonal lattice, how many shells are necessary for a 24-point constellation? (4 pts)

b. What is the average energy of the 24SH constellation based on $A - 2$? (3 pts)

c. What is the total coding gain of this constellation? (1 pt)

d. For equal numbers of constellation points, do rectangular or hexagonal points have more shells? (1 pt)

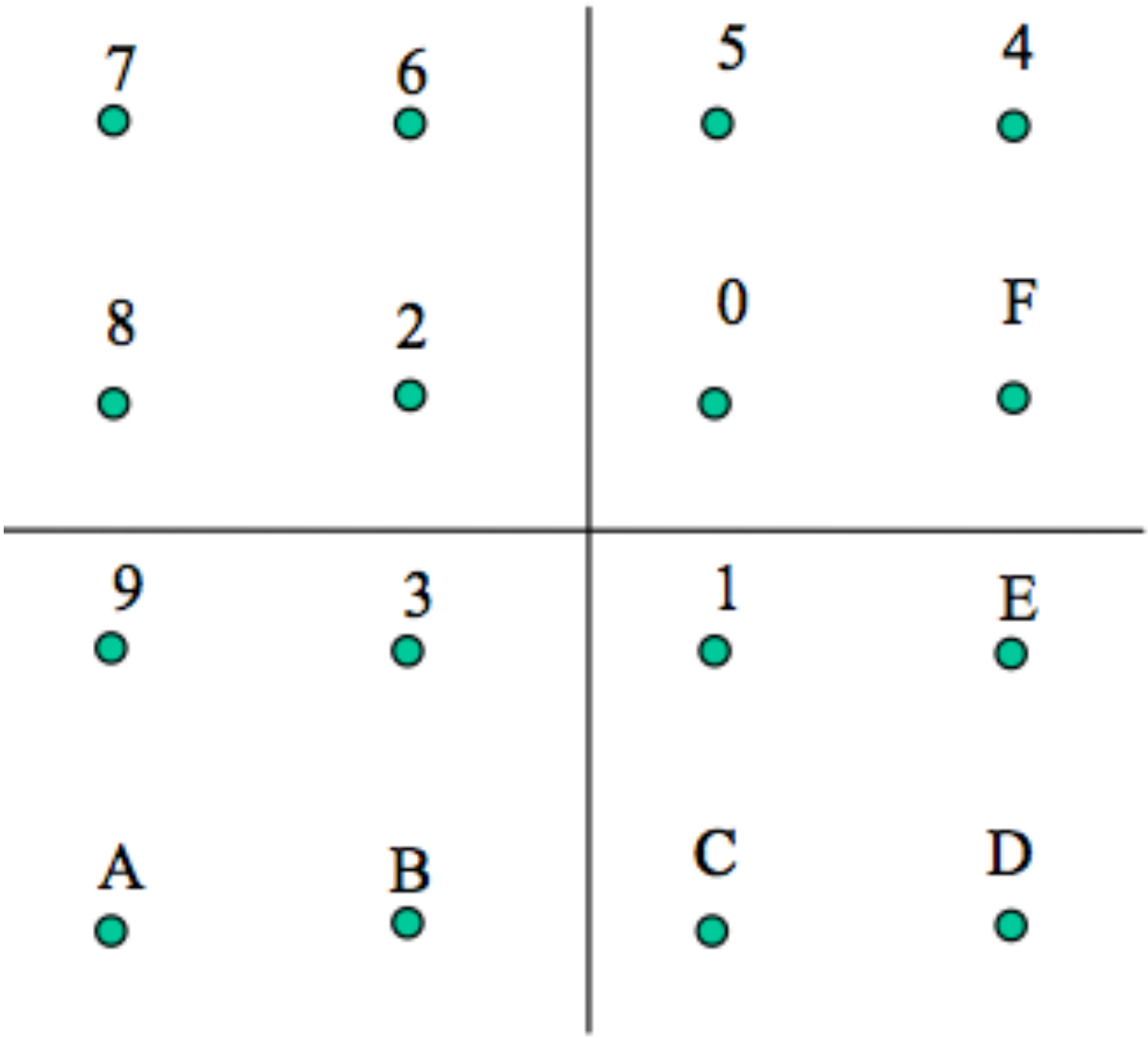


Figure 10.65: 16 QAM with Ungerboeck labeling.

Appendix A

Finite-Field Algebra Review

Coding theory uses the concepts of finite fields, algebras, groups and rings. This brief appendix concisely reviews the basics of these topics.

A.1 Groups, Rings, and Fields

A group is a set of objects, that is closed under an operation addition, associative over that same operation, and for which an identity and inverse exist in the group. More formally,

> **Definition A.1.1 (Group)** *A* **group** S *is a set, with a well-defined operation for any two members of that set, call it* **addition** *and denote it by* $+$*, that satisfies the following four properties:*
>
> *a.* **Closure** $\forall\, s_1, s_2 \in S$*, the sum* $s_1 + s_2 \in S$*.*
> *b.* **Associative** $\forall\, s_1, s_2, s_3 \in S$*,* $s_1 + (s_2 + s_3) = (s_1 + s_2) + s_3$*.*
> *c.* **Identity** *There exists an identity element* 0 *such that* $s + 0 = 0 + s = s$*,* $\forall\, s \in S$*.*
> *d.* **Inverse** $\forall\ s \in S$*, there exists an inverse element* $(-s) \in S$ *such that* $s + (-s) = (-s) + s = 0$*.*

The identity element 0 is unique. When the group also exhibits the **commutative property**, $s_1 + s_2 = s_2 + s_1$, the group is said to be an **Abelian** group. A **subgroup** is a subset of S that satisfies all the properties of a group.

A ring is an Abelian group with the additional operation of multiplication, such that closure and associativity also hold for multiplication, and that multiplication distributes over addition. More formally,

> **Definition A.1.2 (Ring)** *A* **ring** R *is an Abelian group, with the additional well-defined operation for any two members of that set, call it* **multiplication** *and denote it by* $\cdot$ *(or by no operation symbol at all), that satisfies the following three properties:*
>
> *a.* **Closure for multiplication** $\forall\, r_1, r_2 \in R$*, the product* $r_1 \cdot r_2 \in R$*.*
> *b.* **Associative for multiplication** $\forall\, r_1, r_2, r_3 \in R$*,* $r_1 \cdot (r_2 \cdot r_3) = (r_1 \cdot r_2) \cdot r_3$*.*
> *c.* **Distributive** $\forall\, r_1, r_2, r_3 \in R$*, we have* $r_1 \cdot (r_2 + r_3) = r_1 \cdot r_2 + r_1 \cdot r_3$ *and* $(r_1 + r_2) \cdot r_3 = r_1 \cdot r_3 + r_2 \cdot r_3$*.*

A ring often has a **multiplicative identity** denoted by 1, and if multiplication is commutative, the ring is called a **commutative ring**. Any element of a ring R, call it r, for which a multiplicative inverse $1/r$ also exists in R is called a **unit** or **prime**. A field is a ring that defines division:

> **Definition A.1.3 (Field)** *A* **field** F *is a ring, with the additional operation of division, the inverse operation to multiplication, denoted by* $/$*. That is for any* $f_1, f_2 \in F$*, with* $f_2 \neq 0$*, then* $f_1/f_2 = f_3 \in F$*, and* $f_3 f_2 = f_1$*.*

A somewhat weaker version of division occurs in what is known as the **integral domain**, which is a ring with the following additional property: $f_1 \cdot f_2 = f_1 \cdot f_3$ implies $f_2 = f_3$ if $f_1 \neq 0$.

A field may contain a finite or infinite number of member objects. A field of interest in this chapter is the finite field with two elements $GF(2) = \{0, 1\}$, with addition defined by $0 + 0 = 0$, $0 + 1 = 1$, and $1 + 1 = 0$, multiplication defined by $0 \cdot 0 = 0$, $0 \cdot 1 = 0$, and $1 \cdot 1 = 1$. The only unit or prime in $GF(2)$ is 1.

Another example of a field is $F(D)$ defined in Section 10.1, the ratios of all binary polynomials in D, where multiplication and division are defined in the obvious way, with modulo 2 addition.

Vector spaces are used often in this text in other Chapters and there often refer to vectors of real or complex numbers. More generally, and specifically, in coding, the vector space can have elements in any field, in particular a fine field.

Definition A.1.4 (Vector Space) *An n-dimensional* **Vector Space** *V over a field F contains elements called vectors $\boldsymbol{v} = [v_{n-1}, ..., v_0]$, each of whose components v_i $i = 0, ..., n-1$ is itself an element in the field F. The vector space is closed under addition (because the field is) and also under scalar multiplication where $f_i \cdot \boldsymbol{v} \in V$ for any element $f_i \in F$ where*

$$f_i \boldsymbol{v} = [f_i \cdot v_{n-1}, ..., f_i \cdot v_0] \ . \tag{A.1}$$

The vector space captures the commutativity, associativity, zero element (vector of all zero components), and additive inverse of addition and multiplication (by scalar of each element) of the field F. Similarly, the mulitplicative identity is the scalar $f_i = 1$. A set of J vectors is linearly independent if

$$\sum_{j=1}^{J} f_j \cdot \boldsymbol{v}_j = 0 \tag{A.2}$$

necessarily implies that

$$f_j = 0 \ \ \forall j \quad . \tag{A.3}$$

A.2 Galois Fields

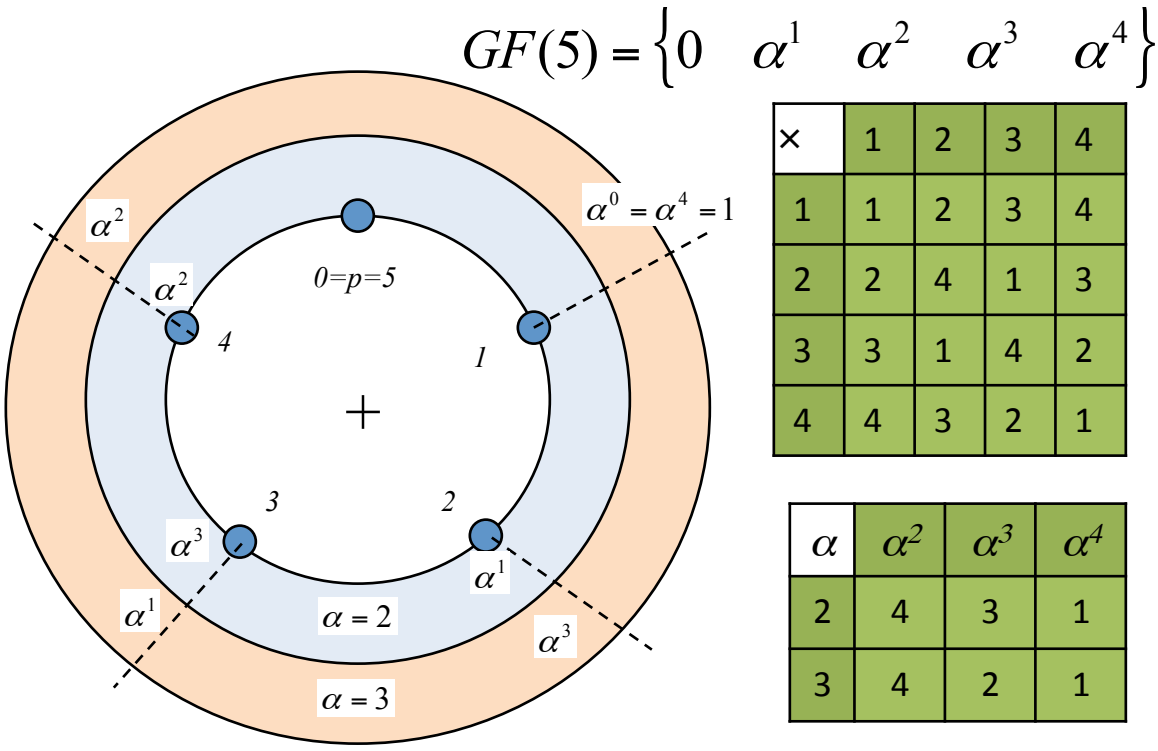
Galois Fields are essentially based on arithmetic modulo a prime number p (or a power of a prime number p^m as to be shown shortly). The elements of a Galois field can be written

$$\mathrm{GF}(p) = \{0, 1, ..., p-1\} \quad . \tag{A.4}$$

The simplest Galois Field GF(2)uses binary arithmetic, where the prime is $p = 2$ and the elements are 0 and 1. Addition and subtraction reduce to binary "exclusive or," multiplication is binary "and," while division for non-zero elements is trivially 1/1=1. Figure A.1 illustrates a less trivial example for $p = 5$, or GF(5). In Figure A.1, a modulo-arithmetic circle illustrates addition, consistent with this text's previous use of modulo addition. Addition corresponds to moving clockwise around the circle while subtraction is counter clockwise. A multiplication table also appears in Figure A.1. This choice of multiplication definition simply multiplies integers and then takes the result modulo 5. Each row or column of this symmetric multiplication table contains each element of GF(5) only once, which means that the reciprocal of an element is the column index for the entry 1 in the corresponding row of that element. (Division then occurs by multiplying by the reciprocal.) This reciprocal is the unique multiplicative inverse needed for a field.

Lemma A.2.1 (GF(p)) *The elements 0, 1, ... $p-1$ of GF(p) form a field under addition and multiplication modulo p, where p is prime.*

Proof: *Closure of addition, subtraction (additive inverse), closure of multiplication, and the zero and identity element (1) follow trivially from the properties of integers, as do commutative and associative properties (all modulo p). Division and the multiplicative inverse do*



×	1	2	3	4
1	1	2	3	4
2	2	4	1	3
3	3	1	4	2
4	4	3	2	1

α	α^2	α^3	α^4
2	4	3	1
3	4	2	1

Figure A.1: Illustration of GF(5).

not trivially follow. If a multiplication table is formed, each row and column will contain all the non-zero elements exactly one time: This completeness of a row (or column) follows from observing that multiplication of an element $0 < \alpha \leq p-1$ *by two distinct elements* $0 < a_1 \leq p-1$ *and* $0 < a_2 < a_1$ *cannot create the same result modulo p, for if they did*

$$\alpha \cdot a_1 = p \cdot d_1 + r \tag{A.5}$$
$$\alpha \cdot a_2 = p \cdot d_2 + r \quad . \tag{A.6}$$

Equivalently,

$$\alpha(a_1 - a_2) = p(d_1 - d_2) > 0 \text{ and } = (0)_p \quad . \tag{A.7}$$

For such an equality to hold true, noting that $\alpha < p$ *and also* $0 < a_1 - a_2 < p$ *since* a_1 *and* a_2 *are distinct, then neither the first or second term on the left can equal p and both are less than p. This necessarily implies that* $0 < d_1 - d_2 \leq \min(\alpha, (a_1 - a_2))$ *and thus*

$$\frac{\alpha(a_1 - a_2)}{d_1 - d_2} = p \tag{A.8}$$

a factorization of a prime number and a contradiction. Thus, each element can occur only once in each row and column if p is prime. **QED.**

Galois Fields often equivalently represent their nonzero elements by power of a primitive element α

$$\text{GF}(p) = \{0, 1, \alpha^1, \alpha^2, ..., \alpha^{p-2}\} \quad . \tag{A.9}$$

Figure A.1 illustrates the use of both $\alpha = 2$ and $\alpha = 3$ to generate GF(5). The successive powers of each such **primitive element** (primative elements are prime numbers in this case) visits each and every non-zero GF5 element exactly once before $\alpha^{p-1} = 1$, and $\alpha^4 = 1$ in both cases. The element $\alpha = 4$

does not generate the entire field and instead generates $\{0, 1, \alpha = 4\}$, a subfield of 3 elements (which is GF(3) where addition is refined with these symbols as $1 + 1 = \alpha$, $\alpha + \alpha = 1$, and $1 + \alpha = 0$ essentially redefining the symbol 4 to be 2. This "symbol interpretation" of the Galois Field is often good to keep in mind as addition and multiplication need not necessarily be defined in a direct correspondence with integer addition and multiplication (even though this example has done so outside of this observation on GF(3) as a subfield of GF(5) for the elements 0, 1, and 4. While the multiplication table in Figure A.1, and thus multiplication, is invariant to the use of the prime α as 2 or 3.

More generally for GF(p), the multiplicative identity is $\alpha^0 = 1$ for all elements $\alpha \in$ GF(p). The $(p-1)^{th}$ power of any element must always be unity, $\alpha^{p-1} = 1$ since the field contains p elements and thus a non-zero element's powers must repeat some nonzero value once a maximum $p-1$ non-zero elements have been generated. This value that first repeats must be 1 by the uniqueness of the inverse already established. From the uniqueness of the rows and columns of the multiplication table, any prime-integer $\alpha > 1$ in GF(p) is a primitive element and can be selected for the value of α to generate all the other nonzero GF(p) elements $\alpha^0 = 1$, α^1, α^2, ..., α^{p-2}. A non-prime integer has the liability of being the product of primes, so that each movement corresponding to another multiplication by α in going from α^i to α^{i+1} actually corresponds to multiplication by each element in this non-prime's factorization (and so the repeating of the sequence occurs earlier because there are more steps implicit in this multiplication (or in fact more than one multiplication) and so "we get to 1 faster."

The use of the notation α^i is useful for multiplication because the exponents can be added, so

$$\alpha^i \cdot \alpha^j = \alpha^{(i+j)_{(p-1)}} \quad . \tag{A.10}$$

Adding of exponents is executed mod $p-1$ because $\alpha^{p-1} = 1 \; \forall \alpha \in$ GF(p). Further, division is executed by multiplying by the inverse $\alpha^{-i} = \alpha^{p-1-i}$ so simple addition on exponents allows all multiplication and division within the field. Storage of each element's index as a power of α and its inverse's index as a power of α, along with simple mod-p addition or subtraction allows all computation with minimal computational effort. If addition is thus viewed as trivial, then perhaps the circle in Figure A.2 is more useful. Each multiplication by α consistently refers to rotation by 90 degrees clockwise in the figure (and division by α is rotation by 90 degrees in the opposite direction).

The use of Galois Fields is perhaps most useful when extended to vectors of elements (typically vectors of bits). Such a Galois Field is denoted GF(p^m) where p is prime. This field has p^m distinct elements. The elements of GF(p^m) are m-dimensional vectors or m-tuples of GF(p) elements. Typically $p = 2$ so these become vectors of bits, typically with $m = 8$ (so bytes in a digital system). However, any prime value of p and any positive integer m defines a Galois Field. It is convenient in such fields to think of an element as represented by a polynomial

$$\alpha(D) = \alpha_0 + \alpha_1 \cdot D + \alpha_2 \cdot D^2 + ... + \alpha_{m-1} D^{m-1} \tag{A.11}$$

where $\alpha \in$ GF(p). The powers of the variable D are used to represent positions within the vector, and multiplication by D corresponds to a shift of the elements (dealing with D^m will be addressed shortly). Addition in GF(p^m) is simple polynomial addition modulo p, or equivalently add the elements of like power modulo p.

Multiplication is based on GF(p) multiplication of polynomial coefficients to create new coefficients in GF(p) but extends the concept of modulo arithmetic for a polynomial to be modulo also a prime polynomial so that $D^{i \geq m}$ can be replaced by lower positive powers of D and thus map to one of the $p^m - 1$ nonzero elements in the field. This concept of multiplication differs significantly from typical binary multiplication in digital computers where the multiplication of two 8-bit quantities would produce a 16-bit result. In Galois fields, multiplication of two 8-bit quantities produces another 8-bit quantity. The mapping back into the original field also requires removal of all multiples of a primitive polynomial $p(D)$ and retaining just the remainder, which is often called "multiplication modulo $p(D)$". A primitive polynomial is a polynomial of degree m that cannot be factored into the product of lower-order polynomials of degree 1 or larger in $GF(p)$. A primitive polynomial or element is similar to the prime number p in the simpler GF(p) arithmetic. In straightforward GF(p) multiplication of two polynomials, multiples of $p(D)$ must be removed or if

$$\alpha(D) \cdot \beta(D) = d(D) \cdot p(D) + r(D) \quad , \tag{A.12}$$

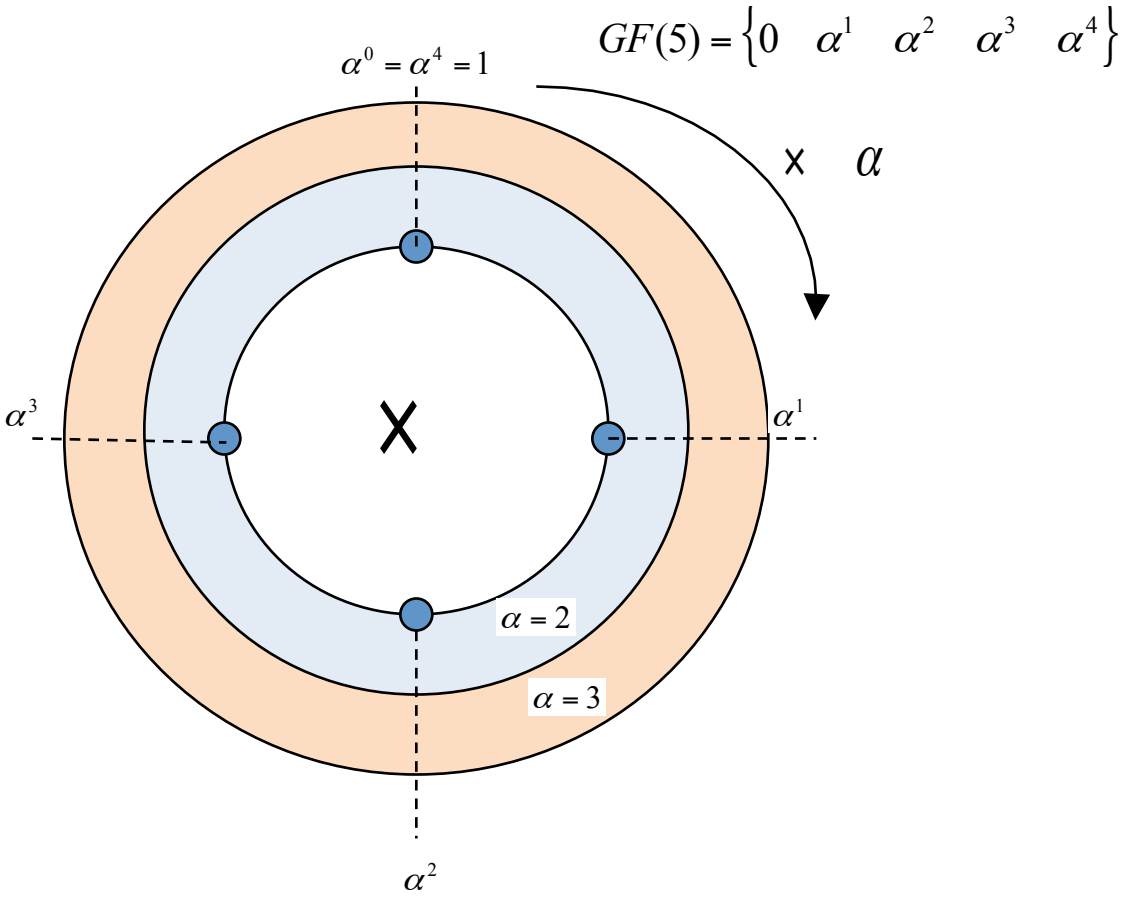


Figure A.2: Multiplicative generation of nonzero elements in GF(5).

where $r(D)$ is the remainder of degree $m-1$ or less, then

$$(\alpha(D)\cdot\beta(D))_{p(D)} = r(D) \quad . \tag{A.13}$$

It is possible to compute $r(D)$ and thus the product by polynomial long divison of $\alpha(D)\cdot\beta(D)$ by $p(D)$, but there are simpler methods to compute the result more quickly. Conceptually, such a simple method recognizes that setting $p(D)=0$ in the above equation determines $r(D)$ quickly.

As an example, for $p=2$ and $m=4$ (so "nibble" arithmetic in GF(16)), the polynomial $p(D) = 1+D+D^4$ cannot be factored, and so is a primitive polynomial that can be used to define multiplication. Multiplication of $1+D^3$ by $1+D^2$ leads to

$$(1+D^3)\cdot(1+D^2) = 1+D^2+D^3+D^5 \quad , \tag{A.14}$$

but $p(D)=0$ implies that $D^4 = 1+D$ and thus $D^5 = D+D^2$ so

$$(1+D^3)\cdot(1+D^2) = 1+D^2+D^3+D+D^2 = 1+D+D^3 \quad . \tag{A.15}$$

The multiplication was easily executed without need for long division. There are actually 16 polynomials and a table of multiplication can be created for GF(16). Such a multiplication table again has the property that each element of GF(p^m) appears only once in each row or column. The proof of this exactly follows the earlier proof. Thus, any non-zero element has a multiplicative inverse, thus defining division and confirming that this group is a field. Another viewd of multiplication enumerates the elements

$$\mathrm{GF}(p^m) = \{0, 1, \alpha, \alpha^2, \alpha^3, ..., \alpha^{p^m-2}\} \quad , \tag{A.16}$$

where α is any **primitive element** that cannot be factored as a polynomial in GF(p). Following identically the earlier development $\alpha^{p^m-1} = 1$ for all nonzero α in GF(p^m). For the case of $m>1$, there is no circular addition diagram like in Figure A.1 because modulo addition is done in each component of the vector of polynomial. However, multiplication is preserved.

A simple example is GF(4) with primitive polynomial $1+D+D^2 = p(D)$ or equivalently $D^2 = 1+D$. The four elements are GF(4)=$\{0, 1, D, 1+D\}$ with $D^2 = 1+D$ and $D(1+D)=1$. Each element is its own additive inverse, and addition is trivial. However, care should be taken to avoid thinking that $1+3$ is equal to zero (since 4 mod 4 is zero). In GF(4) 1+3 = 2 , or $1+(1+D) = D$.

Matlab has a finite field facility that allows simle execution of Galois Field arithmetic. For example the following sequence generates the same multiplication table as in Figure A.1:

```
>> A=gf(ones(4,1)*[0 1 2 3],2)
 A = GF(2^2) array. Primitive polynomial = D^2+D+1 (7 decimal)
 Array elements =

          0          1          2          3
          0          1          2          3
          0          1          2          3
          0          1          2          3

>> B=gf([0 1 2 3]'*ones(1,4),2)
 B = GF(2^2) array. Primitive polynomial = D^2+D+1 (7 decimal)
 Array elements =

          0          0          0          0
          1          1          1          1
          2          2          2          2
          3          3          3          3

>> A.*B
 ans = GF(2^2) array. Primitive polynomial = D^2+D+1 (7 decimal)
```

```
Array elements =
          0          0          0          0
          0          1          2          3
          0          2          3          1
          0          3          1          2
```

A slightly more complicated example is GF(16) with primitive polynomial $P(D) = D^4 + D + 1$ so (using hexadecimal notation)

$$(5{\cdot}B)_{p(D)} = \left((D^2+1)\cdot(D^3+D+1)\right)_{D^4+D+1} = \left(D^5+D^3+D^2+D^3+D+1\right)_{p(D)} = D(D{+}1){+}D^2{+}D{+}1 = 1 \quad , \tag{A.17}$$

so $B = 5^{-1}$ in GF(16). A multiplication table for GF(16) can be found (using Matlab as):

```
A=gf(ones(16,1)*[0:15],4);
B=gf([0:15]'*ones(1,16),4);
A.*B
ans = GF(2^4) array. Primitive polynomial = D^4+D+1 (19 decimal)

 0  0  0  0  0  0   0   0   0   0   0   0   0   0   0   0
 0  1  2  3  4  5   6   7   8   9  10  11  12  13  14  15
 0  2  4  6  8 10  12  14   3   1   7   5  11   9  15  13
 0  3  6  5 12 15  10   9  11   8  13  14   7   4   1   2
 0  4  8 12  3  7  11  15   6   2  14  10   5   1  13   9
 0  5 10 15  7  2  13   8  14  11   4   1   9  12   3   6
 0  6 12 10 11 13   7   1   5   3   9  15  14   8   2   4
 0  7 14  9 15  8   1   6  13  10   3   4   2   5  12  11
 0  8  3 11  6 14   5  13  12   4  15   7  10   2   9   1
 0  9  1  8  2 11   3  10   4  13   5  12   6  15   7  14
 0 10  7 13 14  4   9   3  15   5   8   2   1  11   6  12
 0 11  5 14 10  1  15   4   7  12   2   9  13   6   8   3
 0 12 11  7  5  9  14   2  10   6   1  13  15   3   4   8
 0 13  9  4  1 12   8   5   2  15  11   6   3  14  10   7
 0 14 15  1 13  3   2  12   9   7   6   8   4  10  11   5
 0 15 13  2  9  6   4  11   1  14  12   3   8   7   5  10
```

Alternately, we could enumerate all 15 powers of a primitive element such as 2:

```
A=gf(gf((2*ones(16,1)'),4).^[0:15],4)
A = GF(2^4) array. Primitive polynomial = D^4+D+1 (19 decimal)

 1  2  4  8  3  6  12  11  5  10  7  14  15  13  9  1
\begin{verbatim}
where each element appears once until the element 1 repeats itself in the 16th position.  Further, t
\begin{verbatim}
gf(ones(1,16)./A,4)
 1  9 13 15 14  7  10   5 11 12   6   3   8   4  2  1
```

which is simply the reversed sequence of powers of the primitive element. Thus, essentially 16 (4-bit) elements could be stored in 8 bytes of memory, adjoined with a high-speed 4-bit adder (for the index) along with simple 4-places of binary addition to create an exceptionally high-speed and low-cost GF(16) general purpose arithmetic implementation. Even for GF(256), only 256 bytes of storage and a simple 8-bit adder (plus eight 1-bit adders) are sufficient for high-speed very efficient arithmetic. When one of the elements is fixed, as is often the case in coding, much further simplification yet is possible, as in a later subsection.

Primitive elements of GF(p^m) cannot be factored when viewed as polynomials with coefficients in GF(p). Such primitive elements are however be roots of higher-degree polynomials with coefficients in

GF(p), for instance such a primitive element α always satisfies $f(D=\alpha) = \alpha^{p^m-1} - 1 = 0$ and $f(D)$ in this case has coefficients (1 and -1) in GF(p). The polynomial with GF(p) coefficients of minimum degree for which $f_\alpha(D) = 0$ when $D = \alpha$ is called the **minimal polynomial** of the primitive element α. Clearly $f_\alpha(D)$ must be a factor of $D^{p*m} - 1$.

Conjugates

The polynomial $(\alpha + \beta)^p$ has binomial expansion

$$(\alpha+\beta)^p = \alpha^p + \left[\sum_{k=1}^{p-1} \binom{p}{k} \alpha * p - k\beta^k\right] + \beta^p \quad . \tag{A.18}$$

With mod p arithmetic where p is prime, then

$$\binom{p}{k} = p \cdot \frac{(p-1)\cdots(p-k+1)}{k!} = 0 \bmod p \tag{A.19}$$

because this quantity is an integer and since $k < p$ cannot divide p, it must divide the remaining factors. Then the quantity is an integer multiple of p and thus zero. This then means that

$$(\alpha+\beta)^p = \alpha^p + \beta^p \tag{A.20}$$

for all prime p in modulo-p arithmetic, and thus for arithmetic in GF(p^m) for any $m \geq 1$. The statement is obvious for $p = 2$ of course. Further, then it is true by induction that

$$(\alpha+\beta)^{p^i} = \alpha^{p^i} + \beta^{p^i} \tag{A.21}$$

since it holds for $i = 1$ already, and raising (A.21) to the p^{th} power corresponds to $i \to i+1$ and then

$$(\alpha+\beta)^{p^{i+1}} = \left(\alpha^{p^i} + \beta^{p^i}\right)^p = \left(\alpha^{p^i}\right)^p + \left(\beta^{p^i}\right)^p = \alpha^{p^{i+1}} + \beta^{p^{i+1}} \quad . \tag{A.22}$$

The successive powers of a primitive element α^{p^i} for $i = 0, ...r-1$ are called its **conjugates**. Since GF(p^m) contains a finite number of elements, eventually this set must repeat and that occurs for the first r for which $\alpha^{p^r} = 1$. The set $\{\alpha, \alpha^p, ..., \alpha^{p^{r-1}}\}$ is called the conjugacy class of the primitive element α. Each primitive element has its own conjugacy class, and these classes are mutually exclusive (if not mutually exclusive one the two primitive elements has the other as a factor, a contradiction of their being prime elements).

All elements in a conjugacy class are roots of the corresponding class' minimal polynomial. This is easily proved by noting

$$\begin{aligned}
[f_\alpha(D=\alpha)]^{p^j} &= 0^p = 0 && \text{(A.23)}\\
&= \left[f_0 + f_1\cdot\alpha + ... + f_j\cdot\alpha^j\right]^{p^j} && \text{(A.24)}\\
&= f_0^{p^j} + f_1^{p^j}\cdot\alpha^p + ... + f_j^{p^j}\cdot\alpha^{p^j} && \text{(A.25)}\\
&= f_0 + f_1\cdot\alpha^p + ... + f_j\cdot\alpha^{p^j} && \text{(A.26)}\\
&= f_\alpha(\alpha^{p^j}) && \text{(A.27)}
\end{aligned}$$

so α^{p^j} is also a root. The minimal polynomial has minimum degree and contains all the elements in the conjugacy class so that degree is equal to the number of elements in the conjugacy class, namely

$$f_\alpha(D) = (D-\alpha)\cdot(D-\alpha^p)\cdot....\cdot(D-\alpha^{r-1}) \ \ l. \tag{A.28}$$

There is a minimal polynomial for each of the primitive elements (and all products of primitive elements, generating thus the entire non-zero portion of the Galois Field). Since all the non-zero elements are determined by

$$D^{p^m-1} - 1 = 0 \tag{A.29}$$

then

$$D^{p^m-1} - 1 = (D-1)\cdot\prod_\alpha f_\alpha(D) \quad . \tag{A.30}$$

Appendix B

Various Results in Encoder Realization Theory

B.1 Invariant Factors Decomposition and the Smith Canonical Forms

Our goal in this appendix is to present a procedure that will eventually generate the invariant factors decomposition of Section B.2, or equivalently the **Smith canonical form** of the generator matrix $G(D)$. This form is again:

$$G = A\Gamma B \quad , \tag{B.1}$$

where we have dropped the placeholder D for notational brevity in this appendix.

The procedure is:

a. Extract the least-common multiple, $\phi(D)$ of any denominator terms by forming $\tilde{G}(D) = \phi(D)G(D)$, thus making the elements of $G(D)$ polynomials in $f[D]$.

b. Use row and/or column switches to move nonzero element with lowest degree (in D) to the (1,1) position, and *keep track of the operations.*

c. Use the (1,1) entry together with row and column operations to eliminate (if possible) all other elements in the first row and column. Use only polynomials in D; you may not be able to eliminate the element, but still try to eliminate as much of it (in terms of D^k quantities) as possible. *Keep track of the operations here also.*

   Example:

$$\begin{bmatrix} D & \cdots & (D^2) \times \text{row 1} + \text{row 2} = D^3 + (1 + D^3) = 1 \\ (1 + D^3) & & \text{in (1,1) spot. – can't do better.} \end{bmatrix} \tag{B.2}$$

d. If (1,1) entry is now not of the lowest degree in D, return to 2.

e. Since the (1,1) entry is now of lowest degree, repeat 2 except now move *next lowest* degree term in D to (2,2). Then repeat 3.

This process, although potentially tedious, *will* produce the matrix of invariants. It is the same procedure as is used to find the Smith-Macmillan form of a transfer matrix for a MIMO linear system.

Having kept track of the row and column operations, we have

$$L_n \cdots L_2 L_1 \tilde{G} R_1 R_2 \cdots R_M = \Gamma \tag{B.3}$$

($\tilde{G}$ is ϕG). Therefore,

$$\tilde{G} = (\cdots L_2 L_1)^{-1} \Gamma (R_1 R_2 \cdots)^{-1} = A\Gamma B \tag{B.4}$$

where $A = (L_i \cdots L_2 L_1)^{-1} = L_1^{-1} L_2^{-1} \cdots L_i^{-1}$ and $B = (R_1 R_2 \cdots R_j)^{-1} = R_j^{-1} R_{j-1}^{-1} \cdots R_1^{-1}$.
Notes:

a. $(L_i \cdots L_2 L_1)^{-1}$ and $(R_1 R_2 \cdots R_j)^{-1}$ exist because each L_i and R_i must be individually unimodular, with determinant 1; so the determinants of the products $(\cdots L_2 L_1)$ and $(\cdots R_1 R_2)$ are also 1; so the inverses exist and all elements are polynomials in D, provided we used only polynomials (and not polynomial fractions) in 3.

b. In $G = A\Gamma B = (L_1^{-1} \cdots L_i^{-1})\Gamma(R_j^{-1} \cdots R_1^{-1})$ A and B are not unique; what one comes up with depends upon the order of operations in 2-5. But Γ is unique for G.

c. Matrix inversions when elements are binary-coefficient polynomials are easy - just calculate the determinants of the adjoints, since the denominator determinant is 1 (re: Cramer's Formula for inverses).

EXAMPLE B.1.1 (Example 10.4 of Section 10.1) Let

$$G(D) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & \frac{D^2}{1+D^3} \\ 0 & 0 & 1 & \frac{D}{1+D^3} \end{bmatrix} = \frac{1}{1+D^3} \begin{bmatrix} 1+D^3 & 0 & 0 & 0 \\ 0 & 1+D^3 & 0 & D^2 \\ 0 & 0 & 1+D^3 & D \end{bmatrix} . \quad \text{(B.5)}$$

So we begin with

$$\tilde{G}(D) = \begin{bmatrix} 1+D^3 & 0 & 0 & 0 \\ 0 & 1+D^3 & 0 & D^2 \\ 0 & 0 & 1+D^3 & D \end{bmatrix} \quad \text{(B.6)}$$

The term of lowest degree is D, so swap rows 1 and 3; columns 1 and 4: i.e.

$$\begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix} \tilde{G}(D) \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} = L_1 \tilde{G} R_1 \quad \text{(B.7)}$$

$$= \begin{bmatrix} D & 0 & 1+D^3 & 0 \\ D^2 & 1+D^3 & 0 & 0 \\ 0 & 0 & 0 & 1+D^3 \end{bmatrix} \quad \text{(B.8)}$$

Note that

$$L_1 = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix} \text{ swaps rows 1 \& 3 and} \quad \text{(B.9)}$$

$$R_1 = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \text{ swaps columns 1 \& 4} \quad \text{(B.10)}$$

Now try to eliminate the remaining entries in column 1

$$\begin{bmatrix} D & 0 & 1+D^3 & 0 \\ D^2 & 1+D^3 & 0 & 0 \\ 0 & 0 & 0 & 1+D^3 \end{bmatrix} \quad \text{(B.11)}$$

$\Rightarrow$ row 2 $\rightarrow$ row 2 + D row 1

$$L_2 = \begin{bmatrix} 1 & 0 & 0 \\ D & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \text{(B.12)}$$

which gives

$$\begin{bmatrix} D & 0 & 1+D^3 & 0 \\ 0 & 1+D^3 & D(1+D^3) & 0 \\ 0 & 0 & 0 & 1+D^3 \end{bmatrix} . \tag{B.13}$$

Now eliminate entries in row 1, by $\Rightarrow$ column 3 $\rightarrow$ column 3 + D^2 (not $\frac{D}{1+D^3}$, as we must have a $f[D]$ matrix) $\times$ column 1

$$R_2 = \begin{bmatrix} 1 & 0 & D^2 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{B.14}$$

produces

$$\begin{bmatrix} D & 0 & 1 & 0 \\ 0 & 1+D^3 & D(1+D^3) & 0 \\ 0 & 0 & 0 & 1+D^3 \end{bmatrix} \tag{B.15}$$

Move the (1,3) element to (1,1), since it is now the nonzero element with lowest degree in D.
column 1 $\leftrightarrow$ column 3

$$R_3 = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{B.16}$$

gives

$$\begin{bmatrix} 1 & 0 & D & 0 \\ D(1+D^3) & 1+D^3 & 0 & 0 \\ 0 & 0 & 0 & 1+D^3 \end{bmatrix} \tag{B.17}$$

$\Rightarrow$ row 2 $\rightarrow$ row 2 + D(1+D^3) $\times$ row 1

$$L_3 = \begin{bmatrix} 1 & 0 & 0 \\ D(1+D^3) & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{B.18}$$

and
$\Rightarrow$ column 3 $\rightarrow$ column 1 $\cdot D$ + column 3

$$R_4 = \begin{bmatrix} 1 & 0 & D & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{B.19}$$

gives

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1+D^3 & D^2(1+D^3) & 0 \\ 0 & 0 & 0 & 1+D^3 \end{bmatrix} \tag{B.20}$$

The (1,1) term is "done". We add D^2 times column 2 to column 3

$$R_5 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & D^2 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{B.21}$$

gives

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1+D^3 & 0 & 0 \\ 0 & 0 & 0 & 1+D^3 \end{bmatrix} \tag{B.22}$$

The (2,2) element is also "done"; then swap columns 3 and 4, i.e.

$$R_6 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{B.23}$$

gives

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1+D^3 & 0 & 0 \\ 0 & 0 & 1+D^3 & 0 \end{bmatrix} = \Gamma \tag{B.24}$$

Therefore

$$L_3 L_2 L_1 \tilde{G} R_1 R_2 R_3 R_4 R_5 R_6 = \Gamma \tag{B.25}$$

$$L_3 L_2 L_1 = L = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & D^4 \\ 1 & 0 & 0 \end{bmatrix} \tag{B.26}$$

Note order $L_3\ L_2\ L_1$ preserves order of operations done on $\tilde{G}$.

$$A = L^{-1} = \begin{bmatrix} 0 & 0 & 1 \\ D^4 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix} \tag{B.27}$$

$$R_1 R_2 R_3 R_4 R_5 R_6 = R \tag{B.28}$$

$$B = R^{-1} = \begin{bmatrix} 0 & 0 & D^3 & D \\ 0 & 1 & D^4 & D^2 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & D^2 & 1 \end{bmatrix} \tag{B.29}$$

so $\tilde{G} = A\Gamma B$

$$= \begin{bmatrix} 0 & 0 & 1 \\ D^4 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1+D^3 & 0 & 0 \\ 0 & 0 & 1+D^3 & 0 \end{bmatrix} \begin{bmatrix} 0 & 0 & D^3 & D \\ 0 & 1 & D^4 & D^2 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & D^2 & 1 \end{bmatrix} \tag{B.30}$$

Bringing back the $\frac{1}{1+D^3}$ factor,

$$G = \begin{bmatrix} 0 & 0 & 1 \\ D^4 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} \frac{1}{1+D^3} & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 0 & 0 & D^3 & D \\ 0 & 1 & D^4 & D^2 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & D^2 & 1 \end{bmatrix} = A\Gamma B \tag{B.31}$$

- Note that we have different unimodulars than in Section **??** (ok)

- the hardest part above was collecting L_i, R_i; but was really more tedious than hard.

EXAMPLE B.1.2 (Example 10.1.2 in Section 10.1 continued)

$$G(D) = \begin{bmatrix} 1 & D & 0 \\ D^2 & 1 & D \end{bmatrix} \tag{B.32}$$

a. col 2 $\Rightarrow$ col 2 + D $\times$ col 1

$$R_1 = \begin{bmatrix} 1 & D & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{B.33}$$

gives

$$\begin{bmatrix} 1 & 0 & 0 \\ D^2 & 1+D^3 & D \end{bmatrix} \tag{B.34}$$

b. row 2 $\Rightarrow$ row 2 + D^2 $\times$ row 1

$$L_1 = \begin{bmatrix} 1 & 0 \\ D^2 & 1 \end{bmatrix} \tag{B.35}$$

gives

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1+D^3 & D \end{bmatrix} \tag{B.36}$$

c. col 2 $\Rightarrow$ col 3

$$R_2 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \tag{B.37}$$

gives

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & D & 1+D^3 \end{bmatrix} \tag{B.38}$$

d. col 3 $\Rightarrow$ col 3 + D^2 $\times$ col 2

$$R_3 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & D^2 \\ 0 & 0 & 1 \end{bmatrix} \tag{B.39}$$

gives

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & D & 1 \end{bmatrix} \tag{B.40}$$

e. col 2 $\Rightarrow$ col 3

$$R_4 = R_2 \text{ gives } \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & D \end{bmatrix} \tag{B.41}$$

f. col 3 $\Rightarrow$ col 3 + D $\times$ col 2

$$R_5 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & D \\ 0 & 0 & 1 \end{bmatrix} \tag{B.42}$$

gives

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} = \Gamma \tag{B.43}$$

$$L = L_1 = \begin{bmatrix} 1 & 0 \\ D^2 & 1 \end{bmatrix} \tag{B.44}$$

$$A = L_1^{-1} \tag{B.45}$$

$$R = R_1R_2R_3R_4R_5 = \begin{bmatrix} 1 & D & D^3 \\ 0 & 1 & D \\ 0 & D^2 & 1+D^3 \end{bmatrix} \tag{B.46}$$

$$B = R^{-1} = \begin{bmatrix} 1 & D & 0 \\ 0 & 1+D^3 & D \\ 0 & D^2 & 1 \end{bmatrix} \tag{B.47}$$

Therefore

$$G = \begin{bmatrix} 1 & 0 \\ D^2 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & D & 0 \\ 0 & 1+D^3 & D \\ 0 & D^2 & 1 \end{bmatrix} \tag{B.48}$$

B.2 Canonical Realizations of Convolutional Encoders

Section 10.1 earlier discussed equivalent encoders, for which the (output) codes were the same. Any code can be realized by canonical encoders that have several useful properties:

a. $G(D)$ has a feedback-free right inverse, $G^{-1}(D)$, an $n \times k$ matrix, such that $G(D)G^{-1}(D) = I$, that is, all entries of $G^{-1}(D)$ are in $F[D]$.

b. $G(D)$ is feedback free ($G(D)$ also has all entries in $F[D]$).

c. ν is a minimum, that is $\nu = \mu$.

The structure of such encoders and the algebra to translate a non-canonical encoder into canonical form are studied in this Section. The invariant factors decomposition of an encoder matrix $G(D)$ will be central to the reduction of any encoder to canonical form. The existence of and examples of this IV decomposition will be studied in Subsection B.2.1. The minimal encoder has a minimum number of delay elements are used in the realization. A minimal encoder will need to be delay preserving (i.e. input sequences are not delayed), degree preserving (i.e., an interesting dual property to delay preservation for sequences running backwards in time), and non-catastrophic – Subsection B.2.2 studies these properties and proves the equivalence of these properties to a minimal encoder. The construction of such minimal encoders is nontrivial and requires a two step procedure. The first step is the construction of the basic encoder, which is developed in Subsection B.2.3. The second step that produces the minimal encoder from a basic encoder is developed in Section B.2.4. Systematic encoders can also be found when the property that the encoder be feedback free is eliminated. The canonical systematic encoder with feedback is constructed in Subsection B.2.5.

B.2.1 Invariant Factors

The invariant factors for $G(D)$ are analogous to the singular values of a real matrix. These factors specify some of the most important structural aspects of the convolutional code. All entries in $G(D)$ will be elements in the set (ring) of all binary polynomials in $F[D]$, that is no feedback. Any $G(D)$ with entries in $F_r(D)$ can be transformed to another feedback-free encoder by premultiplying the original generator by $A = \phi(D)I$ ($A \in F(D)$), where $\phi(D)$ is the least common multiple of all the feedback polynomials used in the original generator. This premultiplication will not change the codewords. When all elements of $G(D)$ are in $F[D]$, there exists an **invariant factors decomposition** of $G(D)$. The ensuing development drops the D from notation to simplify notation.

Theorem B.2.1 (Invariant Factors Theorem) *Let G be a $k \times n$ matrix of polynomials in $F[D]$, then*

$$G = A\Gamma B \tag{B.49}$$

where

a. A and B are square $k \times k$ and $n \times n$ $F[D]$ matrices with unit determinants $|A| = 1$ and $|B| = 1$, respectively, that is they are **unimodular**.

b. A^{-1} and B^{-1} are also square $F[D]$ matrices.

c. Γ is a $k \times n$ matrix with the following structure:

$$\Gamma = \begin{bmatrix} \gamma_1 & 0 & \dots & 0 & 0 & \dots & 0 \\ 0 & \gamma_2 & \dots & 0 & 0 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \gamma_k & 0 & \dots & 0 \end{bmatrix} \quad . \tag{B.50}$$

d. $\gamma_i \in f[D]$ are the unique **invariant factors** *for $G(D)$ and*

$$\gamma_i = \frac{\Delta_i}{\Delta_{i-1}} \quad . \tag{B.51}$$

e. Δ_i *is the greatest common divisor (GCD) of all determinants of* $i \times i$ *submatrices of* $G(D)$*, with* $\Delta_0 = 1$*.*

f. γ_i *divides* γ_{i+1}*, that is* γ_i *is a factor of* γ_{i+1}*.*

Proof: Any A (B) with $|A| = 1$ can be represented as a product of matrices that is equivalent to a series of elementary row (column) operation (examples of the procedure generally described in this proof appear in Appendix B), each such operation being in one of the following two forms:

- interchange rows (columns)
- linear combination of rows (columns) such that $|A| = 1$ ($|B| = 1$).

i). If G is not already diagonal, let α and β be elements in the same column where α does not divide β (if it always did, there would have to be a β in some other column that could be added to the present column to make β not divisible by α, or the dimensionality would be less than k, which violates our original restriction that $G(D)$ be of rank k; or the code would be a rate $1/n$ code that can be trivially put into invariant-factors form). Let $\Delta = GCD(\alpha, \beta)$ be the greatest common divisor of α and β.
ii). Then the procedure finds x and y, both in $F[D]$, such that $\alpha x + \beta y = \Delta$, and does elementary column and/or row interchanges (always unimodular) and one additional unimodular transformation of the form

$$\begin{bmatrix} x & y & 0 \\ -\frac{\beta}{\Delta} & \frac{\alpha}{\Delta} & 0 \\ 0 & 0 & I \end{bmatrix} \begin{bmatrix} \alpha & \\ \beta & \\ & \end{bmatrix} = \begin{bmatrix} \Delta & \\ 0 & \\ & \end{bmatrix} \tag{B.52}$$

iii). IF Δ divides all other elements in the matrix (for instance if $\Delta = 1$, as it often does), then zero the remainder of the first row and column with

$$\begin{bmatrix} 1 & 0 \\ \frac{\beta}{\Delta} & 1 \end{bmatrix} \begin{bmatrix} \Delta \\ \beta \end{bmatrix} \quad or \quad [\Delta \;\; \beta] \begin{bmatrix} 1 & \frac{\beta}{\Delta} \\ 0 & 1 \end{bmatrix} \tag{B.53}$$

type transformations.
OTHERWISE, repeat i). and ii) until this occurs (see also Appendix B).

Eventually this step will produce a matrix of the form

$$\Gamma_1 = \begin{bmatrix} \gamma_1 & 0 \; ... \; 0 \\ 0 & \\ \vdots & G_1 \\ 0 & \end{bmatrix} \quad . \tag{B.54}$$

iv). Repeat this process recursively for G_1, then G_2, ... G_k.

Results 1-6 follow from the construction.**QED**.

As an example of a matrix that qualifies as a unimodular A matrix:

$$A = \begin{bmatrix} 1 & 1+D \\ 1 & D \end{bmatrix} \; ; \; |A| = D + D + 1 = 1 \tag{B.55}$$

$$A^{-1} = \begin{bmatrix} D & 1+D \\ 1 & 1 \end{bmatrix} \; ; \; |A^{-1}| = D + D + 1 = 1 \; . \tag{B.56}$$

EXAMPLE B.2.1 (4-state $r = 1/2$ code from Example 10.1.1) Returning to the earlier 4-state $G = [1 + D + D^2 \; 1 + D^2]$ example, $\Delta_0 = \Delta_1 = 1$, so $\gamma_1 = 1$:

$$G = \begin{bmatrix} 1 + D + D^2 & 1 + D^2 \end{bmatrix} \tag{B.57}$$

$$= [\,1\,] \cdot [\,1 \;\; 0\,] \begin{bmatrix} 1 + D + D^2 & 1 + D^2 \\ a(D) & b(D) \end{bmatrix} \tag{B.58}$$

$$= A \Gamma B \quad . \tag{B.59}$$

The values of $a(D)$ and $b(D)$ are such that $|B| = 1$. To get $|B| = 1$, let $a(D) = a_0 + a_1 D + a_2 D^2$ and $b(D) = b_0 + b_1 D + b_2 D^2$. Then

$$a(D)\left(1 + D^2\right) + b(D)\left(1 + D + D^2\right) = 1 \quad . \tag{B.60}$$

Equating terms in D^0 ... D^4 yields:

$$\begin{aligned}
D^0 &: \quad a_0 + b_0 = 1 \ \text{ or } \ a_0 = \bar{b}_0 \ \text{ , so let } b_0 = 0 \ \ a_0 = 1 && \text{(B.61)}\\
D^1 &: \quad a_1 + b_0 + b_1 = 0 \ \text{ or } \ a_1 = b_1 \ \text{ , so let } b_1 = 1 \ \ a_1 = 1 && \text{(B.62)}\\
D^2 &: \quad a_0 + a_2 + b_0 + b_1 + b_2 = 0 \ \text{ or } \ a_2 = b_2 \ \text{ , so let } b_2 = 0 \ \ a_2 = 0 && \text{(B.63)}\\
D^3 &: \quad a_1 + b_1 + b_2 = 0 \ \text{ , checks} && \text{(B.64)}\\
D^4 &: \quad a_2 + b_2 = 0 \ \text{ , checks} && \text{(B.65)}
\end{aligned}$$

So then,

$$a(D) = 1 + D \quad b(D) = D \tag{B.66}$$

is a valid solution for the B matrix, and

$$G = A\Gamma B = [\ 1\ ] \cdot [\ 1 \ \ 0\ ] \begin{bmatrix} 1 + D + D^2 & 1 + D^2 \\ 1 + D & D \end{bmatrix} \tag{B.67}$$

The invariant factor decomposition was readily obtained for this example. The complicated construction in the proof was not necessary. This is usually the case. However, sometimes a regular method of the form of the proof is necessary, and this method is illustrated in Appendix B.

EXAMPLE B.2.2 (8-state Ungerboeck code revisited) Continuing with the 8-state, $r = 2/3$, convolutional code of Example 10.1.2, $\Delta_0 = \Delta_1 = \Delta_2 = 1$, so $\gamma_1 = \gamma_2 = 1$ for the generator matrix

$$\begin{aligned}
G &= \begin{bmatrix} 1 & D & 0 \\ D^2 & 1 & D \end{bmatrix} && \text{(B.68)}\\
&= \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & D & 0 \\ D^2 & 1 & D \\ a(D) & b(D) & c(D) \end{bmatrix} && \text{(B.69)}
\end{aligned}$$

The quantities $a(D)$, $b(D)$, and $c(D)$ satisfy

$$c(D) + b(D) \cdot D + D\left[c(D) \cdot D^2 + D \cdot a(D)\right] = 1 \quad . \tag{B.70}$$

Let $c(D) = 1$ to satisfy the D^0 constraint. Then

$$b(D) \cdot D + D^3 + D^2 a(D) = 0 \tag{B.71}$$

Then a solution is $a(D) = 1$ and $b(D) = D^2 + D$, leaving

$$G = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & D & 0 \\ D^2 & 1 & D \\ 1 & D + D^2 & 1 \end{bmatrix} \tag{B.72}$$

The third example encoder cannot be decomposed by IVT directly because that 8-state, $r = 3/4$ encoder had feedback. This section extends the concept of invariant factors before proceeding to decompose G with feedback. Multiplication by $\phi(D)$, as described earlier, clears the denominators (eliminates feedback).

Extended Invariant Factors Let a more general $G(D)$ have rational fractions of polynomials as entries, that is the elements of $G(D)$ are in $F_r(D)$. Then,

$$\varphi G = A\tilde{\Gamma}B \tag{B.73}$$

where φ is the least common multiple of the denominators in G, thus permitting the invariant factors decomposition as shown previously on φG. Then

$$G = A\frac{\tilde{\Gamma}}{\varphi}B = A\Gamma B \quad , \tag{B.74}$$

where $\Gamma = \frac{\tilde{\Gamma}}{\varphi}$ is in $F_r(D)$, but A and B are still in $F[D]$. We let

$$\gamma_i = \frac{\alpha_i}{\beta_i} \quad , \tag{B.75}$$

where α_i and β_i are in $F[D]$. Since γ_i divides γ_{i+1}, then α_i divides α_{i+1} and β_{i+1} divides β_i.

EXAMPLE B.2.3 (8-state Ungerboeck code with feedback revisited) Returning to the third earlier example,

$$G = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & \frac{D^2}{1+D^3} \\ 0 & 0 & 1 & \frac{D}{1+D^3} \end{bmatrix} \tag{B.76}$$

Since $u_3(D) = v_4(D)$, the nontrivial portion of the encoder simplifies to have generator

$$G = \begin{bmatrix} 1 & 0 & \frac{D^2}{1+D^3} \\ 0 & 1 & \frac{D}{1+D^3} \end{bmatrix} \tag{B.77}$$

The LCM of the denominators is $\varphi = 1 + D^3$, so

$$\tilde{G} = \varphi G = \begin{bmatrix} 1+D^3 & 0 & D^2 \\ 0 & 1+D^3 & D \end{bmatrix} \quad . \tag{B.78}$$

The GCD's for $\tilde{G}$ are

$$\tilde{\Delta}_0 = \tilde{\Delta}_1 = 1 \quad , \quad \tilde{\Delta}_2 = 1 + D^3 \quad , \tag{B.79}$$

and

$$\tilde{\gamma}_1 = 1 \ , \ \tilde{\gamma}_2 = 1 + D^3 \ . \tag{B.80}$$

So,

$$\tilde{\Gamma} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1+D^3 & 0 \end{bmatrix} \tag{B.81}$$

It is desirable for the bottom row of $\tilde{G}$ to be proportional to $1 + D^3$. Adding D times the bottom row to the top row and interchanging rows accomplishes this proportionality:

$$\tilde{G}_1 = A_1\tilde{G} = \begin{bmatrix} 0 & 1 \\ 1 & D \end{bmatrix}\tilde{G} = \begin{bmatrix} 0 & 1+D^3 & D \\ 1+D^3 & D(1+D^3) & 0 \end{bmatrix} \quad . \tag{B.82}$$

Then

$$\tilde{G}_1 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1+D^3 & 0 \end{bmatrix}\begin{bmatrix} 0 & 1+D^3 & D \\ 1 & D & 0 \\ a & b & c \end{bmatrix} \tag{B.83}$$

For unimodular B,

$$D \cdot c + (1 + D^3) \cdot c + D(b + aD) = 1 \tag{B.84}$$

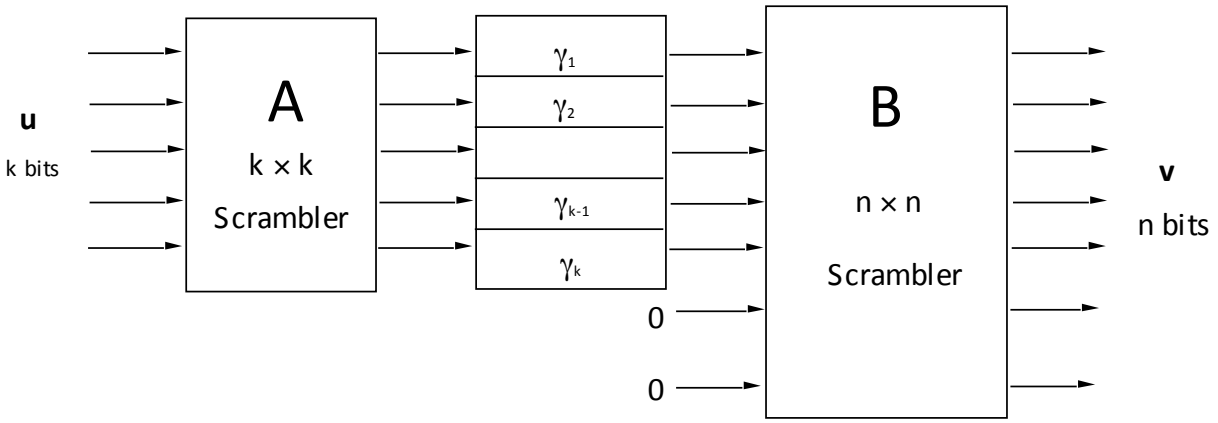


Figure B.1: Scrambler interpretation of IVT.

and $c = 1 + D$, $b = 1$, $a = D(1 + D)$ is a solution. Then, $\tilde{G} = A_1^{-1}\tilde{G}_1$

$$\tilde{G} = \begin{bmatrix} D & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1+D^3 & 0 \end{bmatrix} \begin{bmatrix} 0 & 1+D^3 & D \\ 1 & D & 0 \\ D(1+D) & 1 & 1+D \end{bmatrix} \tag{B.85}$$

or that

$$G = \begin{bmatrix} D & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} \frac{1}{1+D^3} & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 0 & 1+D^3 & D \\ 1 & D & 0 \\ D(1+D) & 1 & 1+D \end{bmatrix} \tag{B.86}$$

The rate 3/4 G with feedback is

$$G = \begin{bmatrix} 1 & 0 & 0 \\ 0 & D & 1 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \frac{1}{1+D^3} & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1+D^3 & D \\ 0 & 1 & D & 0 \\ 0 & D(1+D) & 1 & 1+D \end{bmatrix} . \tag{B.87}$$

The true IVT form requires reversal of the first and second diagonal entries to:

$$G = \begin{bmatrix} 0 & 1 & 0 \\ D & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} \frac{1}{1+D^3} & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 0 & 0 & 1+D^3 & D \\ 1 & 0 & 0 & 0 \\ 0 & 1 & D & 0 \\ 0 & D(1+D) & 1 & 1+D \end{bmatrix} . \tag{B.88}$$

Scrambler Interpretation of the Invariant Factors Decomposition of G A physical interpretation of the Invariant Factors Decomposition is illustrated in Figure B.1. The input $\boldsymbol{u}_k$ can be any k-dimensional sequence of bits and since A is nonsingular, its output can also be any k-dimensional sequence - all the A matrix does is "scramble" the input bits, but otherwise does not affect the code. The same can be said of the Γ matrix. Thus the code generator vectors are thus the first k rows of B, call it G_b. This code is in $F[D]$ and is thus free of feedback, and is equivalent to the original code. Additionally since $B^{-1} \in F[D]$, the first k columns of B^{-1} constitute a feedback-free inverse for G_b.

The effect of additional unitary factors in the matrix B only effects the labeling of the n-dimensional axes of the codewords, but otherwise the critical code parameters, the distances between codewords and the number of codewords at each distance, remain the same.

Tests for a Noncatastrophic Code The following theorem describes two equivalent tests for a noncatastrophic test that derive from the invariant factors decomposition (and avoid searching the state transition diagram for distance-zero loops):

Theorem B.2.2 (Catastrophic Tests) *The following three statements are equivalent:*

a. *An encoder corresponding to a $k \times n$ generator $G(D)$ is noncatastrophic.*
b. *The numerators α_i, $i = 1, ..., k$ of the invariant factors γ_i are of the form D^m, $m \geq 0 \in \mathcal{Z}$, (powers of D).*
c. *The greatest common divisor of the $k \times k$ determinants of $\varphi G(D)$ is equal to D^δ for some $\delta \geq 0$.*
d. *$G^{-1}(D)$ is feedback free.*

Proof:
(1) $\Rightarrow$ (2): By contradiction, assume $\alpha_k \neq D^m$, then let $\boldsymbol{u}(D) = \gamma_k^{-1} e_k A^{-1}$, where $e_k = [0\ ,...,\ 0\ 1]$, which is an input sequence of necessarily infinite weight. Then, $\boldsymbol{u}(D)G(D) = e_k B$ is of finite weight (since $B \in F[D]$), and thus by contradiction $\alpha_k = D^m$.
(2) $\Rightarrow$ (3): The proof follows directly from the definition of α_i.
(3) $\Rightarrow$ (4): $G^{-1} = B^{-1}\Gamma^{-1}A^{-1}$. Since B^{-1}, A^{-1} have all elements in $F[D]$, they are feedback free, and since $\alpha = D^m$, then Γ^{-1} is also feedback free.
(4) $\Rightarrow$ (1): If we take any finite weight code sequence $\boldsymbol{v}(D)$ and apply to the inverse invariant factors, $\boldsymbol{v}(D)B^{-1}\Gamma^{-1}A^{-1} \in F[D]$, then we must have a corresponding input of finite weight.
QED.

B.2.2 Minimal Encoders

A minimal encoder, $G(D) \in f[D]$, has a minimum number of delay elements in the obvious realization. A minimal encoder is generated by finding an equivalent encoder that has the property that all its elements are in $F[D]$. A more physical interpretation of the minimal encoder is that it preserves the length (noncatastrophic, finite $\rightarrow$ finite, infinite $\rightarrow$ infinite), as well as degree and delay of a sequence. The justification for this assertion will become more clear as this section proceeds.

A precise definition of a **delay-preserving** convolutional encoder follows:

Definition B.2.1 (Delay-Preserving Generator) *A* **delay-preserving generator** *is such that that for any $\boldsymbol{u}(D) \in \mathcal{F}^k$, $del(\boldsymbol{u}) = del(\boldsymbol{u}G)$.*

The constant matrix $G_0 = G(0)$ corresponding to the zeroth power of D in $G(D)$ must be nonzero for delay to be preserved. More formally:

Lemma B.2.1 *The following are equivalent statements for an encoder $G(D)$ in $F[D]$, or $G(D) = \sum_{m=0}^{\nu} G_m D^m$:*

a. *$G(D)$ preserves the delay of Laurent sequences, $del(\boldsymbol{u}) = del(\boldsymbol{u}G)\ \forall\ \boldsymbol{u}(D) \in \mathcal{F}^k$.*
b. *The $k \times n$ matrix of constant coefficients, G_0, has rank k.*
c. *$G(D)$ has a causal right inverse, $G^{-1}(D)$, in $F(D)$.*

Proof:
(1) $\Rightarrow$ (2): By contradiction, we assume that the rank of G_0 is less than k. Then there exists a constant sequence $\boldsymbol{u}(D) = \boldsymbol{u}_0$ such that $\boldsymbol{u}_0 G_0 = 0$. Then $0 = del(\boldsymbol{u}_0) \neq del(\boldsymbol{u}_0 G(D)) = 1$ is a contradiction, so that G_0 must be of full rank k.
(2) $\Rightarrow$ (3): We augment the rank-k G_0 to an $n \times n$ constant matrix $\bar{G}_0$ that is of rank n, by adding constant rows to the bottom of G_0. Then, we can also add these same rows to the bottom of $G(D)$ to get $\bar{G}(D)$, which can be written as $\bar{G}(D) = \bar{G}_0 + D\bar{G}'(D)$, where $\bar{G}'(D)$ is some $n \times n$ matrix with entries in $F[D]$. $\bar{G}(D)$ has a simple matrix inverse given by $\bar{G}^{-1}(D) = \left(I_n + D\bar{G}_0^{-1}\bar{G}'(D)\right)^{-1}\bar{G}_0^{-1}$, which is a matrix with no factors of D in the

denominator, so it must therefore be causal (starts at time zero or after). Thus, the first k columns of $\tilde{G}^{-1}(D)$ form a causal right inverse for $G(D)$. (Note the last $(n-k)$ columns form a causal parity matrix $H(D)$ also).
(3) $\Rightarrow$ (1): If $G(D)$ has a causal right inverse $G^{-1}(D)$, then $\boldsymbol{u}(D)G(D)G^{-1}(D) = \boldsymbol{u}(D)$. Since $G^{-1}(D)$ is causal, it cannot reduce the delay, so that $del(\boldsymbol{u}) = del(\boldsymbol{u}G)$.
QED.

Study the degree-preserving property of a generator $G(D)$ "reverses time" by defining

$$\tilde{G}(D^{-1}) \triangleq \begin{bmatrix} D^{-\nu_1} & 0 & ... & 0 \\ 0 & D^{-\nu_2} & ... & 0 \\ . & . & ... & . \\ 0 & 0 & ... & D^{-\nu_k} \end{bmatrix} G(D) \quad , \tag{B.89}$$

which can be written as $\tilde{G}(D^{-1}) = \sum_{m=0}^{\nu} \tilde{G}_m D^{-m}$. The quantity ν_i is the same for both $G(D)$ and $\tilde{G}(D^{-1})$, $i = 1, ..., k$, and the entries in $\tilde{G}(D^{-1})$ are in $F[D^{-1}]$, if the entries in $G(D)$ are in $F[D]$.

Lemma B.2.2 *The following are equivalent statements for an encoder $\tilde{G}(D^{-1})$ in $F[D^{-1}]$, for which $\tilde{G}(D) = \sum_{m=0}^{\nu} \tilde{G}_m D^{-m}$, as above:*

a. $\tilde{G}(D^{-1})$ preserves the degree of anti-Laurent sequences, $deg(\boldsymbol{u}) = deg(\boldsymbol{u}\tilde{G})$ $\forall$ $\boldsymbol{u}(D) \in F((D^{-1}))^k$.

b. The $k \times n$ matrix of constant coefficients, $\tilde{G}_0$, has rank k.

c. $\tilde{G}(D^{-1})$ has an anticausal right inverse, $\tilde{G}^{-1}(D^{-1})$, in $F(D^{-1})$.

Proof: Essentially the same as for Lemma B.2.1.

Definition B.2.2 (Degree-Preserving Generator) *A* **degree-preserving generator** *is such that that for* **all** *finite-length $\boldsymbol{u}(D)$,*

$$deg(\boldsymbol{u}G) = \max_{1 \le j \le k} [deg(\boldsymbol{u}_j) + \nu_j] \quad . \tag{B.90}$$

$G(D)$ is degree preserving if $\tilde{G}$ preserves the degree of anti-Laurent sequences, as in Lemma B.2.2.
The conditions necessary for a minimal encoder are:

Theorem B.2.3 (Minimal Encoders) *An encoder $G(D)$ is minimal if and only if all three of the following conditions are met:*

a. $G(D)$ is delay preserving (or G_0 has rank k).

b. $G(D)$ is degree preserving (or $\tilde{G}_0$ has rank k).

c. $G(D)$ is non-catastrophic.

Proof:
First we show that a minimal encoder must satisfy the 3 conditions listed in the theorem:
condition 1: (by contradiction). Assume the rank of G_0 is less than k, then $\exists$ $\boldsymbol{u}_0$, a constant vector, such that $\boldsymbol{u}_0 G_0 = 0$. For those rows of $G(D)$ corresponding to nonzero elements of the constant vector $\boldsymbol{u}_0$, let j correspond to the the one with largest degree, ν_j. We can replace row j of $G(D)$ by any linear combination of rows in $G(D)$ that includes a nonzero multiple of row j. The choice of this linear combination to be $D^{-1}\boldsymbol{u}_0 G(D)$ reduces the degree of this row and yet still produces an equivalent encoder. Thus, the original encoder could not have been minimal, and by contradiction, $G(D)$ must be delay preserving.
condition 2: (by contradiction). Assume the rank of $\tilde{G}_0$ is less than k, then $\exists$ $\boldsymbol{u}_0$, a constant vector, such that $\boldsymbol{u}_0 \tilde{G}_0 = 0$. For those rows of $\tilde{G}(D)$ corresponding to nonzero elements of the constant vector $\boldsymbol{u}_0$, let j correspond to the the one with largest degree, ν_j. Row j in $G(D)$ can be replaced by any linear combination of rows in $G(D)$ that includes a nonzero multiple

of row j. Let us define an input $\tilde{\boldsymbol{u}}(D) = [u_{k,0}D^{\nu_j-\nu_1}\ u_{k-1,0}D^{\nu_j-\nu_2}\ ...\ u_{1,0}D^{\nu_j-\nu_k}]$, whose elements are in $F[D]$. The linear combination $\tilde{\boldsymbol{u}}(D)G(D)$ is an n-vector in $F[D]$ of degree no more than $\nu_j - 1$. The replacement of row j in $G(D)$ by this n-vector reduces the complexity of the encoder $G(D)$, but still maintains an equivalent encoder. Thus, the original encoder could not have been minimal, and by contradiction, $G(D)$ must also be degree preserving.
condition 3: (by contradiction). Let $\boldsymbol{u}(D)$ be some infinitely long sequence that produces a finite-length output $\boldsymbol{v}(D) = \boldsymbol{u}(D)G(D)$. Then we can write $\boldsymbol{v}(D) = \sum_{m=r}^{s} \boldsymbol{v}_m D^m$. If some of the nonzero elements of $\boldsymbol{u}(D)$ were of finite length, then they could only affect a finite number of output codewords (since $G(D) \in F[D]$) so that they could not change the output $\boldsymbol{v}(D)$ to be of infinite length (or weight). Thus, we can ignore all nonzero elements of $\boldsymbol{u}(D)$ that have finite weight or length. For those rows of $G(D)$ corresponding to nonzero elements of the constant vector $\boldsymbol{u}_0 = \boldsymbol{u}(D)\mid_{D=0}$, again let j correspond to the the one with largest degree, ν_j. The successive removal of finitely many terms $\boldsymbol{u}_r D^r$, $\boldsymbol{u}_{r+1}D^{r+1}, ...\boldsymbol{u}_\tau D^\tau$ by altering the input sequence produces a corresponding output codeword $\boldsymbol{v}'(D)$ such that $deg(\boldsymbol{v}') < \nu_j$ (that is "the denominator" of $\boldsymbol{u}(D)$ must cancel the "numerator" of $G(D)$ exactly at some point, so that degree is reduced). The replacement of row j by this codeword produces an encoder with lower complexity, without changing the code. Thus, the original encoder could not have been minimal, and by contradiction, a minimal encoder must also be non-catastrophic.

Second, we prove that the 3 conditions are sufficient to ensure a minimal encoder. When conditions 1,2, and 3 apply, $len(\boldsymbol{g}_i) = deg(\boldsymbol{g}_i) - del(\boldsymbol{g}_i) + 1 = \nu_i + 1$ ($\boldsymbol{g}_i(D)$ is the row vector corresponding to the i^{th} row of $G(D)$), since $del(\boldsymbol{g}_i) = 0$. Also, $len(\boldsymbol{u}G) = \max_{1\le j\le k}[len(\boldsymbol{u}_j) + \nu_j]$. Since the minimal encoder, call it G_{min}, is equivalent to G, we have that $G_{min}(D) = AG(D)$, where A is an invertible matrix with entries in $F(D)$. We call the ij^{th} entry (row i, column j) $a_{ij}(D)$, and the i^{th} row of A, $\boldsymbol{a}_i(D)$. Then, for any row i of $G_{min}(D)$

$$len(\boldsymbol{g}_{min,i}) = len(\boldsymbol{a}_i G) = \max_{1\le j\le k}[len(a_{ij}) + \nu_j] \ge \max_j(\nu_j) \tag{B.91}$$

from which we infer by summing over i, that $\mu \ge \nu$; but since G_{min} is already minimal, $\mu = \nu$, and therefore $G(D)$ is minimal. **QED.**

B.2.3 Basic Encoders

Basic coders are used as an intermediate step on the way to deriving a minimal encoder.

Definition B.2.3 (Basic Encoder Properties) *An encoder is a* **basic encoder** *if*

a. *$G(D)$ preserves delay for any $\boldsymbol{u}(D) \in \mathcal{F}^k$.*
b. *If $\boldsymbol{v}(D) \in (F[D])^n$ and finite weight, then $\boldsymbol{u}(D) \in (F[D])^k$ and $w_H(\boldsymbol{u})$ is finite (that is the code is not catastrophic).*

A minimal encoder is necessarily basic. A basic encoder is minimal if the encoder preserves degree. Through the following theorem, a basic encoder has $\alpha_k = 1$, the numerator of the last invariant factor γ_k is one). $\alpha_k = 1$ also implies that $\alpha_i = 1\ i = 1, ..., k$.

Theorem B.2.4 (Basic Encoders) *An encoder is basic if and only if $\alpha_k = 1$ in the invariant factors decomposition .*

Proof: First, we proof that if $\alpha_i = 1$, then the encoder is basic: When $\alpha_i = 1$, then $\gamma_i^{-1} = \beta_i \in F[D]$, and therefore $G^{-1} = B^{-1}\Gamma^{-1}A^{-1}$ has entries in $F[D]$ (that is a feedback-free inverse). Then for any finite weight $\boldsymbol{v}(D)$, then $\boldsymbol{v}(D)G^{-1}(D) = \boldsymbol{u}(D)$ must also be of finite weight, so that the encoder is non-catastrophic. If $G(D)$ is not delay preserving, then there exists a $\boldsymbol{u}_0$ such that $\boldsymbol{u}_0 G_0 = \boldsymbol{u}_0 A_0 \Gamma_0 B_0 = 0$, and at least one α_i must be zero. Thus, by contradiction, $G(D)$ is delay preserving.

Second, we proof that if the encoder is basic, then $\alpha_i = 1$. By contradiction, assume $\alpha_k \neq 1$, then either $\alpha_k = D^m$ $m > 0$ or $\alpha_k = 1 + \alpha'(D)$ with $\alpha'(D) \neq 0$. In either case, pick an input $\boldsymbol{u}(D) = \gamma_k^{-1}\epsilon_k A^{-1}$, where $\epsilon_k = [0\ ,...,0\ 1]$. If $\alpha_k = 1 + \alpha'$, then $w_H(\boldsymbol{u}) = \infty$, but $\boldsymbol{u}(D)G(D) = \boldsymbol{b}_k \in F[D]$, so the encoder is catastrophic. Thus, by contradiction in this case, then $\alpha \neq 1 + \alpha'$. If $\alpha_k = D^m$, and noting that the last row of A^{-1} must have at least one term with degree zero (otherwise $|A| \neq 1$), then $del(\boldsymbol{u}) = -m$, $m > 0$. However, the corresponding output $\boldsymbol{b}_k$ can have delay no smaller than 0, so by contradiction (with the delay-preserving condition) $m = 0$. Thus, as both possibilities for $\alpha_k \neq 0$ have been eliminated by contradiction, $\alpha_i = 1$. **QED**.

The following theorem shows that every convolutional encoder is equivalent to a basic encoder.

Theorem B.2.5 (Basic Encoder Equivalent) *Every encoder G is equivalent to a basic encoder.*

Proof: We use the invariant factors decomposition $G = A\Gamma B$ and denote the i^{th} row of B by $\boldsymbol{b}_i$. We note that for any particular codeword $\boldsymbol{v}_0$ that

$$\begin{aligned} \boldsymbol{v}_0 &= \boldsymbol{u}_0 G && \text{(B.92)} \\ &= (\boldsymbol{u}_0 A\Gamma)B && \text{(B.93)} \\ &= \sum_{i=1}^{n} (\boldsymbol{u}_0 A\Gamma)_i \, \boldsymbol{b}_i && \text{(B.94)} \\ &= \sum_{i=1}^{k} (\boldsymbol{u}_0 A\Gamma)_i \, \boldsymbol{b}_i && \text{(B.95)} \\ &= \boldsymbol{u}_1 G_0 && \text{(B.96)} \end{aligned}$$

since $(\boldsymbol{u}_0 A\Gamma)_i = 0 \ \forall \ i > k$. Since this is true for any $\boldsymbol{v}_0$, the original code is equivalent to G_0, a basic encoder (invariant factors decomposition is $I[I\ 0]B$).

Conversely, we can show G_0 is equivalent to the original encoder G by writing $\boldsymbol{v}_1 = \boldsymbol{u}_1 G_0 = [\boldsymbol{u}_1\ 0...0]\, B$. Thus $\boldsymbol{v}_1 = (\boldsymbol{u}_0 A\Gamma)B = \boldsymbol{u}_0 G$, where $\boldsymbol{u}_0 = \boldsymbol{u}_1 \Gamma^{-1} A^{-1}$. **QED**.

Essentially, the last theorem extracts the top k rows of B in the invariant factors decomposition to obtain a basic equivalent encoder. Also, the last $(n-k)$ columns of B^{-1} form a parity matrix for the code.

B.2.4 Construction of Minimal Encoders

The construction of the minimal encoder from the basic encoder is contained within the proof of the following result:

Theorem B.2.6 (Construction of the Minimal Encoder) *Every basic encoder G ($\in F[D]$) is equivalent to the minimal encoder G_{min} through a transformation A such that $|A| = 1$ (A is unimodular).*

Proof:
In order to obtain a minimal encoder from a basic encoder, we need the additional property that $\tilde{G}_0$ has rank k. If this property is not yet satisfied by the basic encoder, then we find $\boldsymbol{u}_0$, a constant vector, such that $\boldsymbol{u}_0 \tilde{G}_0 = 0$. Among all rows of $G(D)$ that correspond to nonzero components of $\boldsymbol{u}_0$, let j correspond to the one with largest degree. The replacement of $\boldsymbol{g}_j(D)$ by $\sum_{i=1}^{k} \boldsymbol{u}_{i,0} D^{\nu_j - \nu_i} \boldsymbol{g}_i(D)$ produces a new $\boldsymbol{g}_j$ that has degree less than ν_j. This replacement can be performed by a series of row operations with $|A| = 1$. This replacement process continues until the resulting $\tilde{G}_0$ has full rank k. **QED**.

When $\tilde{G}_0$ has full rank k, all $k \times k$ determinants in $\tilde{G}_0$ are nonzero. This is equivalent to stating that the maximum degree of all the $k \times k$ determinants of $G(D)$ is equal to μ, when $G(D)$ is minimal.

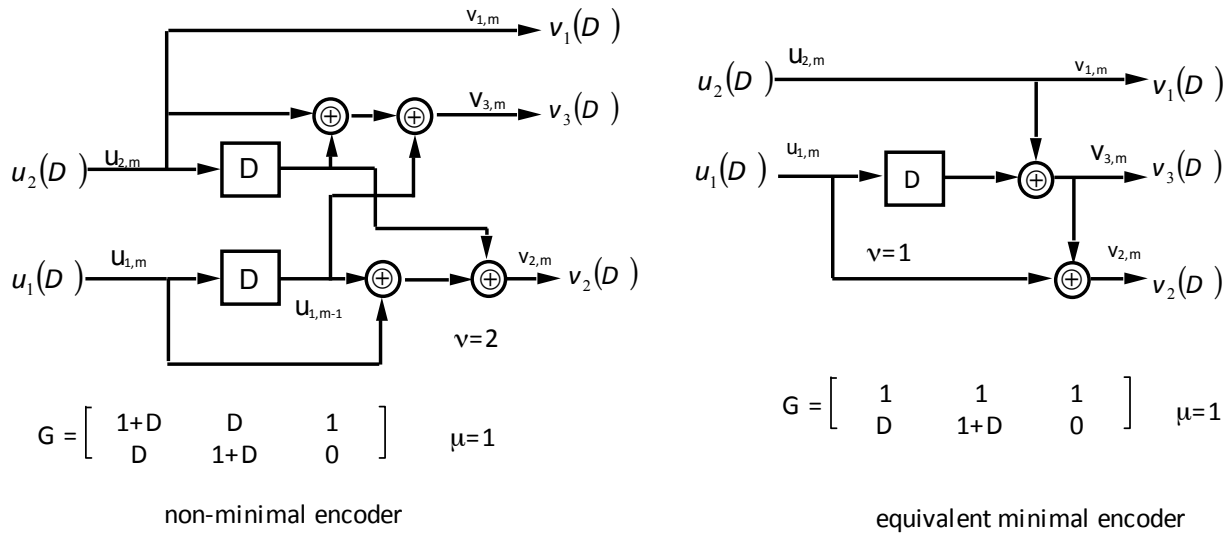


Figure B.2: Example 1 - a non-minimal encoder.

Since these determinants also are invariant to elementary row and column operations, then $\mu =$ max degree of the $k \times k$ determinants in any basic encoder. This also tells us the number of times $(\nu - \mu)$ that the reduction procedure in the previous proof may have to be applied before we obtain a full rank $\tilde{G}_0$. This later result ($k \times k$ determinants) is sometimes quicker to check than forming $\tilde{G}$ and finding $\tilde{G}_0$. The parity matrix forms a dual code, and if we also find the minimal realization of the dual code, then the degree of the minimal generator $H_{min}(D)$, which is also the parity matrix must be the same as the degree of $G_{min}(D)$.

As an example, consider the encoder in Figure B.2. There,

$$G = \begin{bmatrix} 1+D & D & 1 \\ D & 1+D & 0 \end{bmatrix} \quad . \tag{B.97}$$

For this encoder, $\nu = 2$, but that $\mu = 1$ (from rank of $\tilde{G}_0 = 1$, indicating that this basic encoder is not minimal. Thus,

$$\tilde{G} = \begin{bmatrix} 1+D^{-1} & 1 & D^{-1} \\ 1 & 1+D^{-1} & 0 \end{bmatrix} \quad , \tag{B.98}$$

and the linear combination $f = [1\ 1]$ (adding the rows) produces $f\tilde{G} = [D^{-1}\ D^{-1}\ D^{-1}]$. The replacement of $\tilde{G}$ by

$$\tilde{G} = \begin{bmatrix} D^{-1} & D^{-1} & D^{-1} \\ 1 & 1+D^{-1} & 0 \end{bmatrix} \quad , \tag{B.99}$$

and conversion to G_{min}, by multiplying by D, produces

$$G_{min} = \begin{bmatrix} 1 & 1 & 1 \\ D & 1+D & 0 \end{bmatrix} \quad . \tag{B.100}$$

The corresponding minimum equivalent encoder is also shown in Figure B.2.

As another example, consider

$$G = \begin{bmatrix} 1 & D & 0 \\ 0 & 1+D^3 & D \end{bmatrix} \quad , \tag{B.101}$$

which is basic, but $\nu = 4$ and $\mu = 3$. Note this example is illustrated in Appendix B, as the alternative "Smith Canonical Form" of Example (10.1.2) in this chapter. Then,

$$\tilde{G} = \begin{bmatrix} D^{-1} & 1 & 0 \\ 0 & 1 + D^{-3} & D^{-2} \end{bmatrix} \quad , \tag{B.102}$$

and again $f = [1\ 1]$ so that $f\tilde{G} = [D^{-1}\ D^{-3}\ D^{-2}]$ and thus our new $\tilde{G}$ is

$$\tilde{G} = \begin{bmatrix} D^{-1} & 1 & 0 \\ D^{-1} & D^{-3} & D^{-2} \end{bmatrix} \quad , \tag{B.103}$$

leaving a minimal encoder

$$G = \begin{bmatrix} 1 & D & 0 \\ D^2 & 1 & D \end{bmatrix} \quad , \tag{B.104}$$

with $\mu = \nu = 3$, which was the original encoder!

As another example uses the Smith Canonical Form for the last example of Appendix B:

$$G = \begin{bmatrix} 0 & 1 + D^3 & 0 & D \\ 1 & 0 & 0 & 0 \\ 0 & D^4 & 1 & D^2 \end{bmatrix} \quad , \tag{B.105}$$

which is non-minimal with $\nu = 7$ and $\mu = 3$.

Then

$$\tilde{G} = \begin{bmatrix} 0 & 1 + D^{-3} & 0 & D^{-2} \\ 1 & 0 & 0 & 0 \\ 0 & 1 & D^{-4} & D^{-2} \end{bmatrix} \quad , \tag{B.106}$$

and $f = [1\ 0\ 1]$ and $f\tilde{G} = [0\ D^{-3}\ D^{-4}\ 0]$. The new $\tilde{G}$ is then

$$\tilde{G} = \begin{bmatrix} 0 & 1 + D^{-3} & 0 & D^{-2} \\ 1 & 0 & 0 & 0 \\ 0 & D^{-3} & D^{-4} & 0 \end{bmatrix} \quad , \tag{B.107}$$

(where the factor D^{-3} could be divided from the last row by returning to G through multiplication by D^7, and then realizing that the encoder is still not minimal and returning by multiplying by D^4) and another $f = [1\ 0\ 1]$ with $f\tilde{G} = [0\ D^{-3}\ D^{-1}\ D^{-2}]$. The final $\tilde{G}$ is then

$$\tilde{G} = \begin{bmatrix} 0 & D^{-3} & D^{-1} & D^{-2} \\ 1 & 0 & 0 & 0 \\ 0 & 1 & D^{-1} & 0 \end{bmatrix} \quad , \tag{B.108}$$

leaving the final minimal encoder as

$$G = \begin{bmatrix} 0 & 1 & D^2 & D \\ 1 & 0 & 0 & 0 \\ 0 & D & 1 & 0 \end{bmatrix} \quad , \tag{B.109}$$

for which $\mu = \nu = 3$, which is not the same as our minimal encoder for this example in Section **??**, but is nevertheless equivalent to that encoder.

B.2.5 Canonical Systematic Realization

Every encoder is also equivalent to a canonical systematic encoder, where feedback may be necessary to ensure that the encoder realization is systematic. Systematic codes are (trivially) never catastrophic.

The canonical systematic encoder is obtained by following these 3 steps:

a. Find the minimal encoder

b. Every $k \times k$ determinant cannot be divisible by D (otherwise $G_0 = 0$, and the encoder would not be minimal) – find one that is not.

c. Premultiply G_{min} by the inverse of this $k \times k$ matrix.

As an example, we again return to Example (10.1.2), where

$$G = \begin{bmatrix} 1 & D & 0 \\ D^2 & 1 & D \end{bmatrix} \quad . \tag{B.110}$$

The first two columns are

$$M = \begin{bmatrix} 1 & D \\ D^2 & 1 \end{bmatrix} \quad , \tag{B.111}$$

with inverse

$$M^{-1} = \frac{\begin{bmatrix} 1 & D \\ D^2 & 1 \end{bmatrix}}{1 + D^3} \quad , \tag{B.112}$$

so that $G_{sys} = M^{-1} G_{min}$, or

$$G_{sys} = \frac{1}{1+D^3} \begin{bmatrix} 1+D^3 & 0 & D^2 \\ 0 & 1+D^3 & D \end{bmatrix} = \begin{bmatrix} 1 & 0 & \frac{D^2}{1+D^3} \\ 0 & 1 & \frac{D}{1+D^3} \end{bmatrix} \quad , \tag{B.113}$$

which is Example (10.4), ignoring the pass-through bit.

Our last example begins with the rate 3/4 encoder:

$$G = \begin{bmatrix} 0 & 1 & D^2 & D \\ 1 & 0 & 0 & 0 \\ 0 & D & 1 & 0 \end{bmatrix} \quad . \tag{B.114}$$

If we denote the left-most 3×3 matrix by M, then

$$M^{-1} = \frac{1}{1+D^3} \begin{bmatrix} 0 & 1+D^3 & 0 \\ 1 & 0 & D^2 \\ D & 0 & 1 \end{bmatrix} \quad . \tag{B.115}$$

Then

$$G_{sys} = \frac{1}{1+D^3} \begin{bmatrix} 0 & 1+D^3 & 0 \\ 1 & 0 & D^2 \\ D & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 1 & D^2 & D \\ 1 & 0 & 0 & 0 \\ 0 & D & 1 & 0 \end{bmatrix} \tag{B.116}$$

$$= \frac{1}{1+D^3} \begin{bmatrix} 1+D^3 & 0 & 0 & 0 \\ 0 & 1+D^3 & 0 & D \\ 0 & 0 & 1+D^3 & D^2 \end{bmatrix} \tag{B.117}$$

$$= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & \frac{D}{1+D^3} \\ 0 & 0 & 1 & \frac{D^2}{1+D^3} \end{bmatrix} \quad . \tag{B.118}$$

Appendix C

Lattices

The theory of coset codes depends heavily on the concept of a lattice:

> **Definition C.0.4 (Lattice)** *A* **lattice**, Λ, *is an N-dimensional group of points that is closed under addition in that the sum of any two points in the group is also a point in the group. Lattice addition is presumed to be real vector addition where used in this text.*
>
> *Any constellation that is a subset of Λ is also denoted by Λ (in a slight abuse of notation), and the number of constellation points in such a Λ is written $|\Lambda|$.*

Examples of lattices include, Z, the (one-dimensional) set of integers; Z^2, the two-dimensional set of all ordered-pairs of any two integers, $Z^2 \stackrel{\Delta}{=} \{(x_1, x_2) | x_1 \in Z,\ x_2 \in Z\}$; Z^N, the N-dimensional integer lattice, and D_2, a two-dimensional lattice that is formed by taking "every other point" from Z^2 (that is take the points where $x_1 + x_2$ is a even integer). Λ' is a sublattice of Λ, where a sublattice is defined as

> **Definition C.0.5 (Sublattice)** *A* **sublattice** Λ' *of a lattice Λ is an N-dimensional lattice of points such that each point is also a point in Λ.*

> **Definition C.0.6 (Coset of a Lattice)** *A* **coset of a lattice** *is an N-dimensional set of points, written $\Lambda + \boldsymbol{c}$, described by the translation*
>
> $$\Lambda + \boldsymbol{c} \stackrel{\Delta}{=} \left\{ \boldsymbol{x} \mid \boldsymbol{x}' + \boldsymbol{c} \ ;\ \boldsymbol{x}' \in \Lambda \ ,\ \boldsymbol{c} \in \mathcal{R}^N \right\} \quad , \tag{C.1}$$
>
> *where $\mathcal{R}^N$ is the N-dimensional set of vectors with real components.*

A sublattice Λ' **partitions** its parent lattice Λ into a group of cosets of Λ' whose union is λ. The partitioning is written Λ/Λ and the set of all cosets as $[\Lambda/\Lambda]$. The number of subsets is called the **order of the partition**, $|\Lambda/\Lambda'|$ and thus

$$\Lambda = \{\lambda + \boldsymbol{c} | \lambda \in \Lambda' \ ,\ \boldsymbol{c} \in [\Lambda/\Lambda]\} \quad . \tag{C.2}$$

A **partition chain** is formed by further partitioning of the sublattice into its sublattices and can be abbreviated:

$$\Lambda/\Lambda'/\Lambda'' \quad . \tag{C.3}$$

As a simple example of partitioning in two dimensions

$$Z^2/D_2/2Z^2/2D_2/4Z^4... \tag{C.4}$$

Each subsequent partition has order 2, that is there are two cosets of the sublattice to form the immediate parent lattice. Also, a four-way partition would be $Z^2/2Z^4$ which has $|Z^2/2Z^4| = 4$ and $\left[Z^2/2Z^4\right] = \{(0,0),\ (1,0),\ (0,1),\ (1,1)\}$. More sophisticated lattices in two and higher dimensions are introduced and used in Sections 10.5 and code6.

C.1 Elementary Lattice Operations

The **cartesian product** of one lattice with another has dimensionality equal to the sum of the dimensionality of the original two lattices and is formed (as was defined in Chapter 1) by taking all possible points in the first lattice and pairing them with each and every point in the second lattice. This is typically written

$$\Lambda_1 \otimes \Lambda_2 = \{(\lambda_1, \lambda_2) \mid \lambda_1 \in \Lambda_1 \quad , \quad \lambda_2 \in \Lambda_2\} \quad . \tag{C.5}$$

An important special case of the cartesian product for lattices is the so-called squaring construction:

Definition C.1.1 (Squaring Construction) *The* **squaring construction**, *performed on a lattice* Λ, *is given by*

$$\Lambda^2 \triangleq \Lambda \otimes \Lambda \quad , \tag{C.6}$$

that is, the (cartesian) product of Λ *with itself.*

Examples of the squaring construction are $Z^2 = Z \otimes Z$, $Z^4 = Z^2 \otimes Z^2$, and $Z^8 = Z^4 \otimes Z^4$. The cartesian product has a "distributive" property over set union:

$$\Big(A \bigcup B\Big) \otimes \Big(C \bigcup D\Big) = (A \otimes C) \bigcup (A \otimes D) \bigcup (B \otimes C) \bigcup (B \otimes D) \quad . \tag{C.7}$$

The **Rotation Operator** R_N is used to rotate a lattice when N is even.

Definition C.1.2 (Rotation Operator) *The rotation operator* R_2 *is defined by*

$$R_2 \triangleq \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \tag{C.8}$$

It is applied to a two-dimensional lattice Λ *in the following sense:*

$$R_2\Lambda = \left\{(x,y) \mid \begin{bmatrix} x \\ y \end{bmatrix} = R_2 \begin{bmatrix} \lambda_x \\ \lambda_y \end{bmatrix} \ni (\lambda_x, \lambda_y) \in \Lambda\right\} \quad . \tag{C.9}$$

A larger-dimensional rotation matrix is computed recursively according to

$$R_{2N} = \begin{bmatrix} R_N & 0 \\ 0 & R_N \end{bmatrix} \quad . \tag{C.10}$$

C.2 Binary Lattices and Codes

Binary lattices characterize most used coset codes. A lattice $\Lambda_(N,k)$ is a **binary lattice** if it can be partioned by the lattice $2Z^N$. In other words, a partition chain

$$Z^N/\Lambda_{(N,k)}/2Z^N \tag{C.11}$$

exists for a binary lattice. The overall partition $Z^N/2Z^N$ clearly has order $|Z^N/2Z^N| = 2^N$. When the subscript (N,k) is used, then

$$|Z^N/\Lambda_{(N,k)}| = 2^{N-k} = 2^{r_G} \tag{C.12}$$

$$|\Lambda_{(N,k)}/2Z^N| = 2^k \tag{C.13}$$

so that $r_G \triangleq N - k$. The number r_G is associated with the "number of parity bits" of a related binary code, just as r_G is used in Section 10.5 to characterize the number of parity bits in underlying convolutional codes. A binary lattice may be mapped to a binary code C with generator G and r_G parity bits so that N-dimensional binary codewords $\boldsymbol{v}$ are generated from any binary input k-tuple $\boldsymbol{u}$ according to

$$\boldsymbol{v} = \boldsymbol{u}G \quad . \tag{C.14}$$

Any point in the corresponding binary lattice, $\boldsymbol{\lambda}$, can be constructed as

$$\boldsymbol{\lambda} = \boldsymbol{v} + 2Z^N \quad . \tag{C.15}$$

Essentially then the 2^k distinct codewords of the binary code C are the set of coset leaders in the partition of the binary lattice $\Lambda_{(N,k)}$ by $2Z^N$. This may also be written as

$$\boldsymbol{\lambda} = \boldsymbol{v} + 2\boldsymbol{m} \quad , \tag{C.16}$$

where $\boldsymbol{m} \in Z^N$. The mapping to a binary code can provide a simple mechanism to generate points in the binary lattice. As discussed in Section 10.1, a **dual code** for the binary code is defined by a parity matrix H for the original code. The parity matrix H has the property for any codeword $\boldsymbol{v}$ in C that

$$\boldsymbol{v}H^* = 0 \ , \tag{C.17}$$

or the rows of the parity matrix are orthogonal to the codewords. The dimensionality of the matrix H is $(N-k) \times N = r_G \times N$ and defines the dual code $C^\perp$ such dual-code codewords are generated by any $(N-k)$ dimensional binary vector according to

$$\boldsymbol{v}^\perp = \boldsymbol{u}H^* \quad . \tag{C.18}$$

Any codeword in the dual code is orthogonal to all codewords in the original code and vice-versa:

$$\boldsymbol{v}^\perp \boldsymbol{v}^* = 0 \quad . \tag{C.19}$$

The dual code of a linear binary code is clearly also a linear binary code[1] and thus defines a binary lattice itself. This binary lattice is known as the **dual lattice** $\Lambda^\perp_{(N,k}$. There is a partition chain

$$Z^N / \Lambda^\perp_{(N,k} / 2Z^N \quad , \tag{C.20}$$

with

$$|Z^N / \Lambda^\perp_{(N,k}| = 2^k \tag{C.21}$$

$$|\Lambda^\perp_{(N,k} / 2Z^N| = 2^{N-k} \quad . \tag{C.22}$$

Then, the inner product of any $\boldsymbol{\lambda}$ point in the original lattice is orthogonal modulo-2 to any point in the dual lattice

$$\boldsymbol{\lambda}^\perp \boldsymbol{\lambda}^* = \boldsymbol{v}^\perp \boldsymbol{v}^* + 2 \cdot (\text{some integer vector}) \tag{C.23}$$

or thus

$$\left(\boldsymbol{\lambda}^\perp \boldsymbol{\lambda}^* \right)_2 = 0 \quad . \tag{C.24}$$

The notation $(\cdot)_2$ means modulo-2 on all components (so even integers go to zero and odd integers go to 1). The dual lattice has 2^{r_G} cosets in $2Z^N$. Decoding of any binary N-vector $\tilde{\boldsymbol{v}}$ for the closest vector in the code C often computes an $r_G = (N-k)$-dimensional vector **syndrome** $\boldsymbol{s}$ of $\tilde{\boldsymbol{v}}$ as

$$\boldsymbol{s} \stackrel{\Delta}{=} \tilde{\boldsymbol{v}} H^* \quad . \tag{C.25}$$

The syndrome $\boldsymbol{s}$ can be any of the 2^{r_G} possible r_G-dimensional binary vectors since the rank of H is r_G and $\tilde{\boldsymbol{v}}$ can be any binary vector. The syndrome concept can be generalized to the lattice $\Lambda_{(N,k)}$ for any N-dimensional vector in Z^N because

$$\boldsymbol{s} = \left(\tilde{\boldsymbol{\lambda}} H^* \right)_2 \quad . \tag{C.26}$$

An example of such a binary lattice in two dimensions is D_2, since $Z^2/D_2/2Z^2$. The corresponding binary code is a rate 1/2 code with $r_G = 1$ and codewords $\{(0,0), \ (1,1)\}$. This code is its own dual. The $2^{r_G} = 2$ syndromes are the one-dimensional vectors $\boldsymbol{s} \in \{0, 1\}$. These two one-dimensional

[1] If $\boldsymbol{v}_1^\perp$ and $\boldsymbol{v}_2^\perp$ are in $C^\perp$, then $\left(\boldsymbol{v}_1^\perp + \boldsymbol{v}_2^\perp \right) \boldsymbol{v}^* = 0$ and thus $\left(\boldsymbol{v}_1^\perp + \boldsymbol{v}_2^\perp \right)$ is also in $C^\perp$.

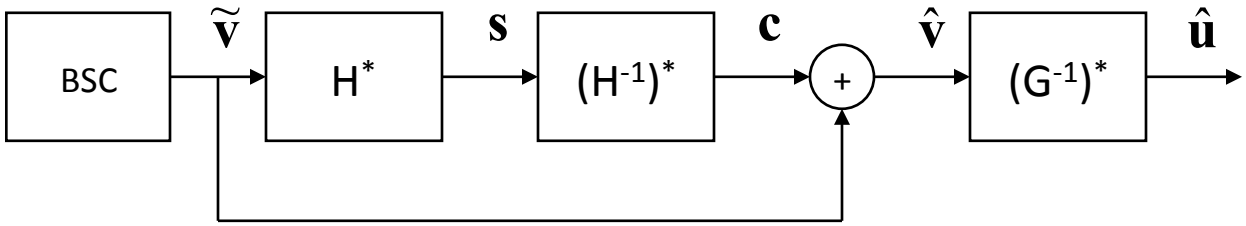


Figure C.1: Basic syndrome decoder.

syndromes can be found by taking any two-dimensional binary vector $\tilde{\boldsymbol{v}} \in \{(0,0),\ (0,1),\ (1,0),\ (1,1)\}$ and multiplying by $H^* = [1\ 1]^*$.

Any point in a coset of the partition set $\left[Z^N/\Lambda_{(N,k)}\right]$ can again be written as

$$\tilde{\boldsymbol{\lambda}} = \boldsymbol{c} + \boldsymbol{\lambda} \tag{C.27}$$

where $\boldsymbol{c} \in Z^2$. Thus, any "received" point in the same coset of $\Lambda_{(N,k)}$ will have the same syndrome, which is easily proven by writing

$$\boldsymbol{s} = (\boldsymbol{c} + \boldsymbol{\lambda})H^* = (\boldsymbol{c} + \boldsymbol{v} + 2\boldsymbol{m})H^* = (\boldsymbol{c}H^*)_2 \quad . \tag{C.28}$$

Thus, the 2^{r_G} cosets of $\Lambda_{(N,k)}$ in Z^N are enumerated by any and all of the 2^{r_G} possible distinct r_G-dimensinal binary syndrome vectors of the code C, which are simply all possible r_G-dimensional binary vectors.

The r_G-dimensional parity matrix H has a right inverse H^{-1} such that (binary operations implied)

$$HH^{-1} = I_{r_G} \quad . \tag{C.29}$$

Similarly by taking the transpose

$$(H^{-1})^* H^* = I_{r_G} \quad . \tag{C.30}$$

Thus the 2^{r_G} N-dimensional coset leaders can be enumerated according to

$$\boldsymbol{c} = \left(\boldsymbol{s}(H^{-1})^*\right)_2 \quad , \tag{C.31}$$

where $\boldsymbol{s}$ runs through all 2^{r_G} possible binary r_G-dimensional vectors. The syndromes correspond to the 2^{r_G} possible error vectors or offsets from actual transmitted codewords on a binary channel. After computing the syndrome, the corresponding binary vector $\boldsymbol{c}$ that satisfies $\boldsymbol{s} = \tilde{\boldsymbol{v}}H^*$ can be computed by Equation (C.31, then leading to $\hat{\boldsymbol{v}} = \tilde{\boldsymbol{v}} \oplus \boldsymbol{c}$. If the channel is a BSC, then $\hat{\boldsymbol{v}}$ is the ML estimate of the codeword (and indeed if the code is systematic, then the input is determined). For non-systematic codes, the relationship $\hat{\boldsymbol{u}} = \hat{\boldsymbol{v}}\left(G^{-1}\right)^*$ determines the ML estimate of the input on the BSC. Figure C.1 illustrates such binary block-code decoding. Syndromes are used in the shaping codes of Section 10.7 and in binary block codes of Section 10.8.

C.2.1 Association of lattices with binary codes

Partitionings of lattices can be associated with various binary codes, some of which are well known. This subsection lists some of those partitionings and associated codes. To describe a binary code, it will be indicated by an ordered triple (N, k, d_{free}). Thus, for instance, (4,3,2) would describe a binary code with $2^3 = 8$ codewords of length 4 bits each and a minimum Hamming distance between the closest 2 of 2 bit positions. A code with (N,k=N,1) must necessarily be uncoded and have free distance 1. A code with (N,0, ∞) has infinite free distance and only one codeword of all zeros.

One-dimensional partitioning of binary lattices

The one-dimensional partitioning chain

$$Z/2Z \tag{C.32}$$

is somewhat trivial and corresponds to the code (1,1,1).

Two-dimensional partitioning of binary lattices

The two-dimensional partitioning chain of interest is

$$Z^2/D_2/2Z^2 \tag{C.33}$$

making D_2 a binary lattice associated with the code (2,1,2). The codewords of this code are [0 0] and [1 1]. The lattice D_2 can be written as

$$D_2 = 2Z^2 + u_1 \cdot \underbrace{[1\ 1]}_{G_{(2,1,2)}} \tag{C.34}$$

where u_1 is the single input bit to the rate-1/2 linear binary code with generator $G_{D_2} = G_{(2,1,2)} = [1\ 1]$. Such a binary code, and thus the associated lattice, is its own dual so

$$H_{D_2} = G_{D_2} = G_{D_2^\perp} \tag{C.35}$$

and

$$H^{-*} = [0\ 1] \tag{C.36}$$

is an acceptable left inverse for H^*.

Four-dimensional parititioning of binary lattices

The four-dimensional partitioning chain of most interest in code is

$$Z^4/D_4/R_4Z^4/R_4D_4/2Z^4 \quad , \tag{C.37}$$

which is equivalent to the linear binary code partitioning chaing (each linear code is a linear binary-code subset of its parent in the chain)

$$\underbrace{(4,4,1)}_{Z^4}/\underbrace{(4,3,2)}_{D_4}/\underbrace{(4,2,2)}_{R_4Z^4}/\underbrace{(4,1,4)}_{R_4D_4}/\underbrace{(4,0,\infty)}_{2Z^4} \quad . \tag{C.38}$$

The generating matrices can easily be built from the bottom of the chain upward:

$$R_4D_4 = 2Z^4 + u_1 \cdot \underbrace{\begin{bmatrix} 1 & 1 & 1 & 1 \end{bmatrix}}_{(4,1,4)} \tag{C.39}$$

$$R_4Z^4 = 2Z^4 + [u_2\ u_1] \underbrace{\begin{bmatrix} 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix}}_{(4,2,2)} \tag{C.40}$$

$$D_4 = 2Z^4 + [u_3\ u_2\ u_1] \underbrace{\begin{bmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix}}_{(4,3,2)} \tag{C.41}$$

$$Z^4 = 2Z^4 + [u_4\ u_3\ u_2\ u_1] \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix} \tag{C.42}$$

Each successive step adds another row to the generator for the partitioning. The dual codes also form a "reverse" partition chain

$$\underbrace{(4,4,1)}_{Z^4} / \underbrace{(4,3,2))^{\perp}}_{(R_4D_4} / \underbrace{(4,2,2)}_{R_4Z^4} / \underbrace{(4,1,4)}_{D_4^{\perp}} / \underbrace{(4,0,\infty)}_{2Z^4} \quad . \tag{C.43}$$

The lattice $R_4D_4 = (D_2)^2$ is a self-dual and so its generator and parity matrix are the same. However,

$$D_4^{\perp} = R_4D_4 \tag{C.44}$$
$$(R_4D_4)^{\perp} = D_4 \tag{C.45}$$

and thus

$$H_{D_4} = G_{R_4D_4} \tag{C.46}$$
$$H_{R_4D_4} = G_{D_4} \quad . \tag{C.47}$$

Eight-dimensional parititioning of binary lattices

The eight-dimensional partitioning chain of most interest in code is

$$Z^8/D_8/(D_4)^2/DE_8/E_8/R_8D_8/(R_4D_4)^2/R_8DE_8/2Z_8 \quad , \tag{C.48}$$

which is equivalent to the linear binary code partitioning chaing (each linear code is a linear binary-code subset of its parent in the chain)

$$\underbrace{(8,8,1)}_{Z^8} / \underbrace{(8,7,2)}_{D_8} / \underbrace{(8,6,2)}_{(D_4)^2} / \underbrace{(8,5,2)}_{DE_8} / \underbrace{(8,4,4)}_{E_8} / \underbrace{(8,3,4)}_{R_8D_8} / \underbrace{(8,2,4)}_{(R_4D_4)^2} / \underbrace{(8,1,4)}_{R_8DE_8} / \underbrace{(8,0,\infty)}_{Z^8} \quad . \tag{C.49}$$

The generating matrices can easily be built from the bottom of the chain upward, starting this time with one generator for E_8 and working upward:

$$E_8 = Z^8 + [u_4\ u_3\ u_2\ u_1] + \underbrace{\begin{bmatrix} 0&1&0&1&0&1&0&1 \\ 1&0&1&0&1&0&1&0 \\ 1&1&0&0&1&1&0&0 \\ 1&0&0&1&0&1&1&0 \end{bmatrix}}_{(8,4,4)} \tag{C.50}$$

$$DE_8 = Z^8 + [u_5\ u_4\ u_3\ u_2\ u_1] \underbrace{\begin{bmatrix} 0&0&0&0&0&0&1&1 \\ 0&1&0&1&0&1&0&1 \\ 1&0&1&0&1&0&1&0 \\ 1&1&0&0&1&1&0&0 \\ 1&0&0&1&0&1&1&0 \end{bmatrix}}_{(8,5,2)} \tag{C.51}$$

$$(D_4)^2 = Z^8 + [u_6\ u_5\ u_4\ u_3\ u_2\ u_1] \underbrace{\begin{bmatrix} 0&1&0&1&0&0&0&0 \\ 0&0&0&0&0&0&1&1 \\ 0&1&0&1&0&1&0&1 \\ 1&0&1&0&1&0&1&0 \\ 1&1&0&0&1&1&0&0 \\ 1&0&0&1&0&1&1&0 \end{bmatrix}}_{(8,6,2)} \tag{C.52}$$

$$D_8 = Z^8 + [u_7\ u_6\ u_5\ u_4\ u_3\ u_2\ u_1] \underbrace{\begin{bmatrix} 0&0&0&1&0&0&0&1 \\ 0&1&0&1&0&0&0&0 \\ 0&0&0&0&0&0&1&1 \\ 0&1&0&1&0&1&0&1 \\ 1&0&1&0&1&0&1&0 \\ 1&1&0&0&1&1&0&0 \\ 1&0&0&1&0&1&1&0 \end{bmatrix}}_{(8,7,2)} \tag{C.53}$$

$$D_8 = Z^8 + [u_8\ u_7\ u_6\ u_5\ u_4\ u_3\ u_2\ u_1] \underbrace{\begin{bmatrix} 0&0&0&0&0&0&0&1 \\ 0&0&0&1&0&0&0&1 \\ 0&1&0&1&0&0&0&0 \\ 0&0&0&0&0&0&1&1 \\ 0&1&0&1&0&1&0&1 \\ 1&0&1&0&1&0&1&0 \\ 1&1&0&0&1&1&0&0 \\ 1&0&0&1&0&1&1&0 \end{bmatrix}}_{(8,8,1)} \tag{C.54}$$

These generators are not unique for any of the binary codes, but each provides a partitioning chain that corresponds to the sequence of binary codes shown, or more generally to binary linear codes with the correct parameters for a valid lattice partitioning chain. Thus, the rows need not match exactly those in the partitioning sections for multi-dimensional trellis codes and lattices. The Gosset lattice is its own dual. Also, R_8DE_8 is the dual of D_8, $(R_4D_4)^2$ would already have been known from four-dimensional partitioning to be the dual of $(D_4)^2$. Finally, R_8D_8 is the dual of DE_8.

C.2.2 16, 24, and 32 dimensional partitioning chains

Partitions in 16, 24, and 32 dimensions are sometimes used in advanced coset coding, but not in this chapter. They may be binary lattices or mod-4 lattices (meaning they have $4Z^N$ as a sub-lattice). However, it is possible to associate binary lattice partition chains with the code partitioning

$$(16,16,1)/(16,15,2)/..../(16,1,16)/(16,0,\infty) \quad . \tag{C.55}$$

Some special lattices sometimes used in coding are

$$D_{16} \triangleq 2Z^{16} + (16,15,2) \tag{C.56}$$

$$H_{16} \triangleq 2Z^{16} + (16,11,4) \tag{C.57}$$

$$\Lambda_{16} \triangleq 4Z^{16} + 2(16,15,2) + (16,5,8) \tag{C.58}$$

The lattices H_{16} (H for "half" lattice) and Λ_{16} are sometimes called 16-dimensional "Barnes-Wall" lattices and Λ_{16} has a coding gain 4.52 dB ($2^{1.5}$), while H_{16} has a coding gain of $2^{\frac{11}{8}}$=4.14 dB. D_{16} is a 16-dimensional checkerboard and has coding gain of $2^{7/8} = 2.63$ dB.

32 dimensional lattices follow the same binary linear code partitioning (now with 32 steps) with

$$D_{32} \triangleq 2Z^{32} + (32,31,2) \tag{C.59}$$

$$X_{16} \triangleq 2Z^{32} + (32,26,4) \tag{C.60}$$

$$H_{32} \triangleq 4Z^{32} + 2(32,31,2) + (32,16,8) \tag{C.61}$$

$$\Lambda_{32} \triangleq 4Z^{32} + 2(32,26,4) + (32,6,16) \quad . \tag{C.62}$$

These are all also (32-dimensional) Barnes-Wall lattices wit coding gains $2^{\frac{15}{16}}$=2.82 dB, $2^{\frac{13}{8}}$=4.89 dB, $2^{\frac{31}{16}}$=5.83 dB, and 4=6.02, dB respectively.

There is also a 24-dimensional series of partitions in the same fashion that is of particular interest because it contains a very special high-gain lattice Λ_{24} known as the Leech lattice

$$D_{24} \triangleq 2Z^{24} + (24, 23, 2) \tag{C.63}$$

$$X_{24} \triangleq 2Z^{24} + (24, 18, 2) \tag{C.64}$$

$$H_{24} \triangleq 4Z^{32} + 2(24, 23, 2) + (24, 12, 8) \tag{C.65}$$

$$\Lambda_{32} \triangleq 4Z^{24} + 2(24, 18, 4) + (24, 6, 16)' \quad . \tag{C.66}$$

the notation (24,6,16)' means the set of all binary linear combinations modulo 4 of a set of six generators wose coordinates are integers modulo 4. This not a binary lattice, but what is called a mod-4 lattice. These have coding gains $2^{\frac{11}{12}}$=2.76 dB, $2^{\frac{3}{2}}$=4.52 dB, $2^{\frac{23}{12}}$=5.77 dB, and 4=6.02, dB respectively.