EE521 Analog and Digital Communications

Instructor: James K Beard, PhD **Office:** Ft. Washington 115

Email:. jkbeard@temple.edu, jkbeard@comcast.net

Office Hours: Wednesdays 5:00 PM to 6:00PM

Location: Ft. Washington 107 **Time:** Wednesdays 6:00 PM – 8:30 PM

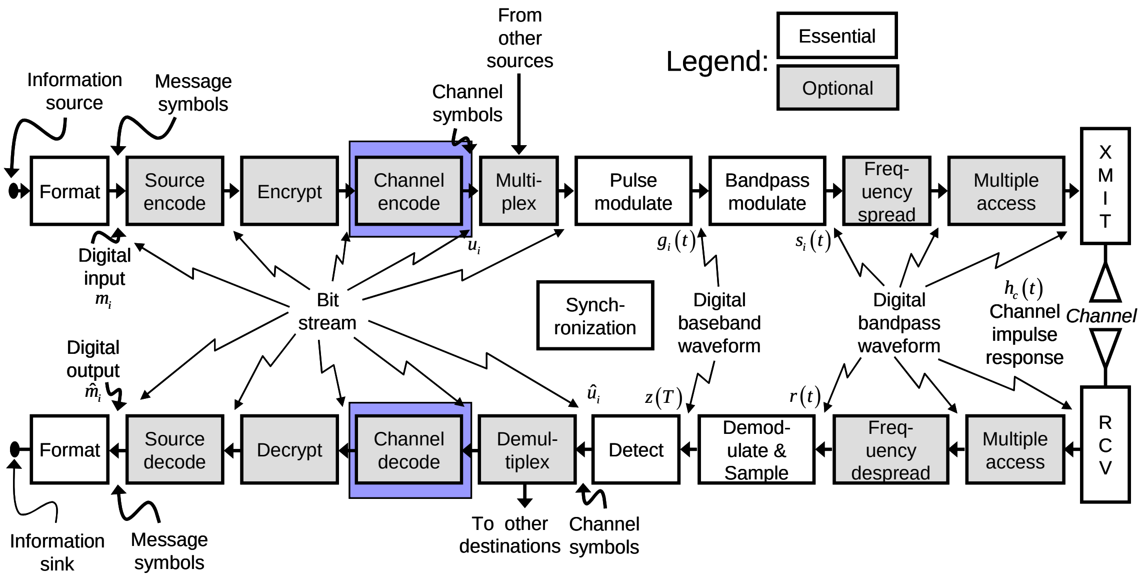
Web Page: http://temple.jkbeard.com

Texts:

- Bernard Sklar, Digital Communications, Second Edition, Prentice Hall P T R, 2001 (2004 printing), ISBN 0-13-084788-7
- Digital Communication Systems Using SystemVue, by Dr. Silage, ISBN 1-58-450850-7

Today's Topics

Channel Coding: Part 2



1 EE521 Course End-Game

Plans for completion of EE521 for Spring 2006 follow.

1.1 Remaining Class Sessions

- Today, April 19, 2006 – Cleanup of Chapter 7, examples of convolutional codes; setting up BER computation in your term project, overview of Chapter 9

1.2 *Critical Dates*

- April 26, 2006 – TERM PROJECT DUE. You will need to do a demo. Bring the file; I will run it on my laptop. You will also need your final report and a short presentation. You will have a maximum of 20 minutes total.
- May 3, 2006 – STUDY DAYS; no scheduled class but I will come for a review and study session with you if you like.
- May 10, 2006 – FINAL EXAM here in this room at the usual class time.

1.3 *EE551*

We will address Sklar's Chapters 5 and 8 and the remainder of Chapter 9 in addition to the remainder of selections from Sklar as given in the Synopsis. See the EE551 course web page on http://temple.jkbeard.com for the synopsis.

2 Convolutional Codes

2.1.1 Concepts

Convolutional codes, at their simplest, are convolutional filters operating on data bit streams with binary arithmetic. As used in communications systems, multiple convolutional filters are run in parallel and their outputs are interleaved. A convolutional code is characterized by three integers:

- The number *n* is the number of convolutional filters in parallel.
- The number *k* is the number of bits fed in at a time.
- The parameter *K* is the memory depth of the convolutional filters and is called the *constraint length* of the code.

In practice *n* and *k* are small integers and *K* is set to satisfy the requirements of the code. The code rate is *k/n*, so the symbols in the equation are the same as those for the code rate for linear block codes. Below we will consider the simplest case, *k=1, n=2, K=3.*

2.1.2 Simple Convolutional Encoder Block Diagram

A block diagram for

$$\begin{aligned} k &= 1 \\ n &= 2 \\ K &= 3 \end{aligned} \tag{2.1}$$

is shown below. This is an actual, practical doe used in digital communications, and is the default convolutional code in the SystemView tokens for this code.

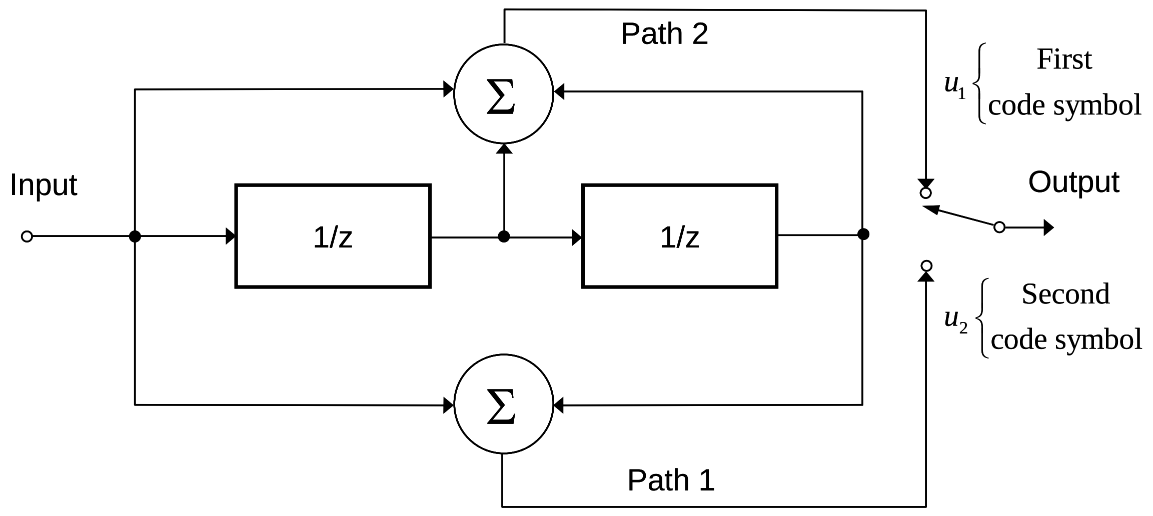


Figure 1 Block Diagram for Rate ½, K=3 Convolutional Encoder

Note that the constraint length *K* is one more than the number of delays in the diagram; the third register is implied in the hold function in the input.

2.1.3 State Logic Table

The state is defined as the bits held in the two shift registers of the shift register, and is denoted by either or both of the letters or the bits. The state naming convention is shown below in Table 1.

Table 1 State Definitions for Rate 1/2, K=3 Convolutional Encoder

State Name	Last Bit	Previous Bit	Designation
a	0	0	[0,0]
b	1	0	[1,0]
c	0	1	[0,1]
d	1	1	[1,1]

Logic for transitioning from one state to the next is illustrated by the table of states and the input bit. A table showing the logic for next message bits 0 and 1 for each of the four states is shown below as Table 2.

Table 2 State Transition Logic for Rate 1/2, K=3 Convolutional Encoder

Inp bit	Last bit	Prev bit	State	Next u1	Next u2	Trellis	Next State
0	0	0	a [0,0]	0	0	(0,0)	a [0,0]
0	0	1	c [0,1]	1	1	(1,1)	a [0,0]
0	1	0	b [1,0]	1	0	(1,0)	c [0,1]
0	1	1	d [1,1]	0	1	(0,1)	c [0,1]
1	0	0	a [0,0]	1	1	(1,1)	b [1,0]
1	0	1	c [0,1]	0	0	(0,0)	b [1,0]
1	1	0	b [1,0]	0	1	(0,1)	d [1,1]
1	1	1	d [1,1]	1	0	(1,0)	d [1,1]

2.1.4 The State Diagram

The number of states for three bits is 2^3 or eight, but we need to characterize the states by the bits available from the two filters. The additional degree of freedom is accounted for by “remembering” one bit previous to the current message bit. The result is that there are four ways to enter and exit each of four states, not two ways to enter and exit each of eight states. A state diagram for a rate 1/3, K=3 convolutional code is shown below.

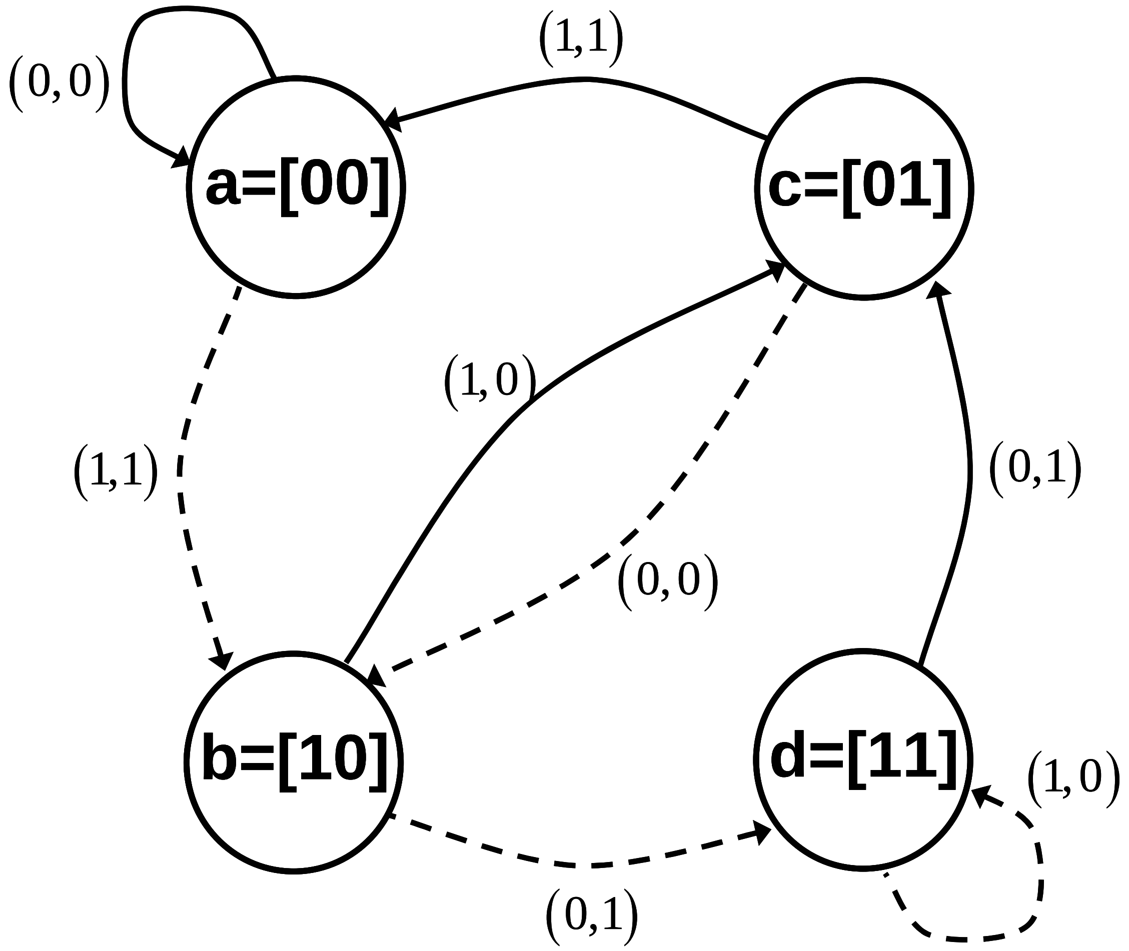


Figure 2 State diagram for rate 1/2, K=3 Convolutional Code

2.1.5 Tree Diagram

A tree diagram splits each time a new bit comes in, so that each branch completely characterizes the message. For *k* message bits, the tree has 2^k branches. Each branc can be labeled with the states, current bit, and last bit, as is the state diagram, and in a sense it represents an unfolding of the state diagram. See Slkar, Figure 7.6 page 392 for a tree diagram for a rate ½, K=3 code.

2.1.6 The Trellis Diagram

The trellis diagram provides a way to unfold the state diagram while keeping the states on one row; as such it is re-folded. Its principal advantage over the state diagram is that clocks of bits into the encoder are on a linear axis. A trellis diagram for a rate ½, K=3 code is shown below.

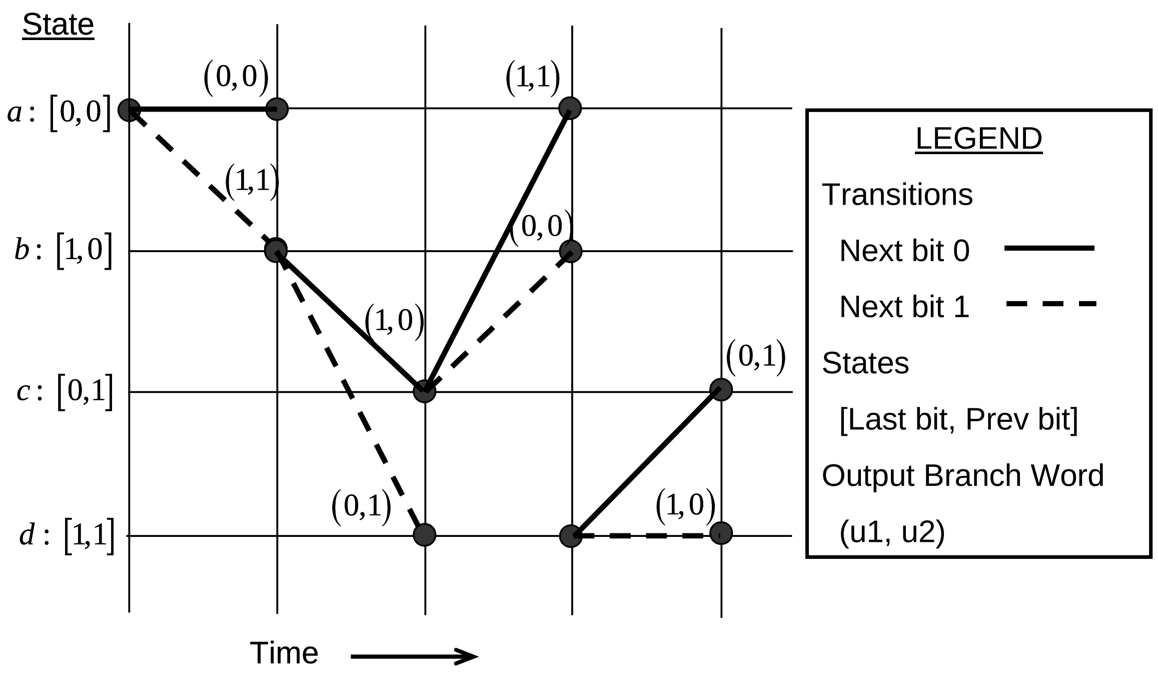


Figure 3 Encoder trellis diagram for a rate 1/2, K=3 Convolutional Encoder

2.2 Decoding

The mathematics of the decoding process is complex and not well represented in most tutorial textbooks, including Sklar. We provide a high-level treatment here to provide insight and defer the mathematics for now.

Note that the code rate of *k/n* is achieved only for very long streams of data, and that if the data is blocked, an extra $k \cdot K$ bits must be allowed to clock out the memory of the convolutional filters, and the actual code rate falls slightly below *k/n*.

2.2.1 Maximum Likelihood Decoding

The probability density function is, in the context of mathematical statistics, sometimes called the likelihood function. We can write the probability density function for the received signal in an AWGN channel, given the input codeword, as the product of the Gaussian distribution functions of the noises out of the matched filter for each bit, with means as expected from each bit of the codeword. We can compute this probability density function, with the codeword bit as the mean. Writing it algebraically with the codeword bits as parameters allows us to find the combination of codeword bits that maximizes this probability density function. This is the concept of the maximum likelihood decoder.

The logarithm of the likelihood function is almost always taken to simplify analysis and implementation, particularly when the measurement noise is Gaussian. The result is a least-squares fit, weighted by the E_b/N_0 for each bit. Since E_b/N_0 is usually constant over a message block, the maximum likelihood decoder is effectively a least-squares fit of valid codewords to observed received data. This effectively means that the decoded codeword is the one with the smallest Hamming distance to the received codeword.

A message of *M* bits will result in a codeword of length

$$C = (M + K - 1) \cdot \left(\frac{1}{\text{code rate}} \right) \tag{2.2}$$

The maximum likelihood decoding criteria is posed as follows. We define the message bit string of *M* bits as

$$\underline{m} = [m_1, m_2 \ldots m_M] \tag{2.3}$$

and the resulting codeword of *C* bits as

$$\underline{u} = [u_1, u_2 \ldots u_C] \tag{2.4}$$

then we can characterize the probability density function, or likelihood function, of the received codeword of an AWGN channel for a given input codeword as

$$p(\underline{ur}|\underline{u}) = p(\underline{ur}|\underline{m}) = \prod_{j=1}^{C} \frac{1}{\sqrt{2\pi} \cdot \sigma} \cdot \exp\left(-\frac{1}{2} \cdot \frac{(ur_j - u_j)^2}{\sigma^2} \right) \tag{2.5}$$

where the variance σ^2, including normalization for signal amplitude out of the correlator, is given by

$$\sigma^2 = \frac{N_0}{2E_b} \tag{2.6}$$

The maximum likelihood decoder selects the message which, when its codeword is used in (2.5) along with the received codeword $\underline{ur}$, produces the largest likelihood function.

The logarithm of the likelihood function is

$$llf\,(\underline{ur}|\underline{m}) = \frac{C}{2} \cdot \ln(2\pi \cdot \sigma^2) + \frac{1}{2\sigma^2} \cdot \sum_{j=1}^{C} (ur_j - u_j)^2 \tag{2.7}$$

which we will use in decoding decision processes.

2.2.2 Hard versus Soft Decisions

When the received codeword provides a quantization of the receive signal instead of simply providing one-bit threshold decisions of zero or one, this additional data can be provided to a maximum likelihood decoding algorithm. In (2.5), we use the received value of $\underline{ur}_j$, quantized to two or three bits, instead of a threshold decision for each bit – that is, just one bit. Allowing three bits of quantization can provide a BER for the maximum likelihood detection scheme that is approximately the same as that for hard decoding of the received codeword for an E_b/N_0 about 2 dB higher.

2.2.3 Decoding with a State Table

We continue the rate ½, K=3 code as an example. We begin by a decoder state of a, or [0,0]. We proceed by two bits of the message code and inferring the next bit and the next state.

Table 3 Decoding State Table for Rate 1/2, K=3 Code

State	Code	Next bit	Next State
a [0,0]	(0,0)	0	a [0,0]
a [0,0]	(1,1)	1	b [1,0]
b [1,0]	(0,1)	1	d [1,1]
b [1,0]	(1,0)	0	c [0,1]
c [0,1]	(0,0)	1	b [1,0]
c [0,1]	(1,1)	0	a [0,0]
d [1,1]	(0,1)	0	c [0,1]
d [1,1]	(1,0)	1	d [1,1]

Decoding with the table is equivalent to use of the state diagram or trellis to decode a sequence of code word pairs. Note that when an encoder is in a particular state, equivalent to a state of the encoder, it expects only two of four possible codeword bit pairs.

2.2.4 Recognizing and Handling Bit Errors

When a codeword bit pair is not one of the two possibilities expected, this indicates that a bit error has occurred. For example, if the state is a [0,0] and the codeword pair (0,1) is received, then one of the two codeword bits is in error. The probability that this will occur is $p \cdot (1-p)$ where *p* is the bit error rate. If two bit errors occur the error will be unrecognized; the probability of this is p^2.

When a bit error is recognized, both paths are followed. This continues until the same state is again achieved. The path with the fewest detected bit errors between the first detected bit error and the merging of the states is selected that has the fewest detected bit errors.

Unless the two convolution polynomials were chosen such that they share a common factor, propagation of the states with correct code pairs will always result in the paths merging with the same state at some point. See Sklar's discussion of *catastrophic error* in Section 7.4.3, pages 414-415 for a discussion of this. Codes of this type are called catastrophic codes, and as the name suggests, these are not used except as examples of erroneously formulated codes.

The maximum number of bit pairs that can be traversed before the states merge is a measure of the error-correcting capability of the code. Tracing paths that result in return to a starting state from a particular state is shown in Sklar to be identical for any codeword sequence, so that an analysis for a codeword sequence consisting of all zeros suffices. An analysis of such paths in terms of the number of bit errors required in the codeword is done in Sklar section 7.4.1 pages 408-413, in which our simple rate ½, K=3 convolutional code is shown to have a equivalent d_{min} of 5.

2.2.5 The Viterbi Decoding Algorithm

The Viterbi decoding algorithm is an implementation of the maximum likelihood decoder that avoids the complexity of a full search over all possible codewords, even for soft decoding. It works by elimination of non-viable possibilities as it goes along, a process called *pruning* in the theory of algorithmic complexity. Pruning usually is an *ad hoc* process that causes some minor compromise with optimality but the Viterbi algorithm for decoding convolutional codes has been shown to retain full optimality when certain constraints are met.

The Viterbi convolutional decoding algorithm always carries both paths, along with incremental log likelihood score functions computed according to (2.7). Whenever two paths enter the same state, the one with the best score is kept and the other discarded. Since there are four states, four paths are kept after each bit pair is decoded.

3 Care and Feeding of the BER Counter Token in SystemView

3.1 Setup

The BER counter token is in the SystemView Communications library is in the Optional Libraries collection. It is labeled as the BER token in the Processors folder. It has two inputs, four setup parameters, and three outputs. In addition, the sample rate of the input data is provided to it as a clock tick by SystemView – as a software module, it is called at this rate. The SystemView sample rate is used for converting offsets in seconds to offsets in sample times, if needed. The setup parameters are given below as Table 4.

Table 4 BER Counter Setup Parameters

Parameter	Default	Usage
Number of Trials	100	Number of bits per BER count in output 1
Threshold	0.5 V	Use default for bit range 0 to 1 v; for bit range -1 to +1, use 0 V
Offset	0	How long to wait before starting the count
Seconds or Samples	Seconds	Offset units; use bits or symbols for best control

There are two BER inputs, both bit streams. One of these must be derived form truth data; e.g., reference data that the BER counter will use to compare with data from the other input. The action of the BER counter is to perform an XOR between these two inputs and compute each of the three outputs.

The outputs are BER, as computed over the Number of Trials in the setup, a cumulative average of the BER, and a count of the total number of bit errors. These are repated in .

Table 5 BER Counter Outputs

Output	Format	Remarks
BER	Nonnegative floating point; less than 1.0	BER over Number of Trials from setup parameters; reset each time the Number of Trials is reached
Cumulative Average	Nonnegative floating point; less than 1.0	Cumulative average from total errors and total data samples; best overall output for a run
Total Errors	Unsigned integer	Bit error count; can be used to trip a threshold and end a system loop

3.2 Usage

To use the BER counter, a bit stream is provided to one input as reference data, while the other is encoded, passed through a channel model, and decoded. The reference data must be delayed to time-align it with the reference data. The Auto-Correlate function in the SystemView Sink Calculator, Correlation and Convolution (Corr Conv) selections is good for doing this because the correlation peak will be at a time representing the synchronization mismatch between the inputs.

The BER as measured for a given run is provided by the Cumulative Average output.

The Number of Trials can be set to 1 without affecting the other outputs. This results in this output becoming binary, and will be a zero when the inputs agree and a one when they disagree.

3.3 Plotting BER curves

Plotting BER curves requires that the SystemView simulation run in a loop, with one E_b/N_0 for each loop. This can be done by using a gain token in the path of the noise added for the channel, and making the gain a function of a global parameter (see the Tools menu under Global Parameter Links for a setup window). The total number of errors can be used with a stop sink to terminate each loop.

There are special provisions for BER plots in the Sink Calculator. We will explore these in class.

4 Overview of Modulation and Coding Trade-Offs

4.1 The Shannon-Hartley Capacity Theorem the Shannon Limit, and Channel Efficiency

The basic resources in communications are bandwidth and signal to noise ratio. The Shannon channel capacity theorem tells us the maximum possible bit rate *C* through a channel with bandwidth W and signa-to-noise SNR is

$$C = W \cdot \log_2 \left(1 + SNR\right) \tag{4.1}$$

(Sklar equation (9.2) page 525). We relate SNR to E_b/N_0 by

$$\frac{E_b}{N_0} = SNR \cdot \frac{W}{C} \tag{4.2}$$

where we see the channel capacity *C* where we are accustomed to seeing the bit rate *R* (from Sklar's equation (9.5) page 527). Thus we were assuming that we were close to the Shannon limit with our E_b/N_0 that we were using. The parameter *R/C* is the channel efficiency, and it is through forward error correcting codes that we can achieve high channel efficiencies.

4.2 Matching the Waveform to the Channel

There is a minimum E_b/N_0 that can be used for communication, -1.6 dB, so significant levels of E_b/N_0 are used in practical systems. We see from (4.2) that this means that we must adhere to a reasonable match between our waveform bit rate *R* and the channel noise bandwidth *W*. Note that (4.2) breaks down when *W* is much larger than *C* or *R* – we don't have a blank check on increasing E_b/N_0. What we must do is to match our channel rate to *W*, at least very approximately. In terms of the message character rate R_M, the number of bits per token *K*, and the code rate *r*, the bit rate *R* is

$$R = \frac{R_M \cdot K}{r} \tag{4.3}$$

Often the main free parameter available to us is *K*, which amounts to a sound quality parameter in voice communications.

The number of bits per character is approximately $\log_2 \left(1 + SNR\right)$. For M-ary waveforms, we can move *K* from the bit rate equation (4.3) to the number of bits per character in our tradeoffs.

4.3 Entropy, Data Compression, and Coding

Entropy is a mathematical measure of the randomness of bits in a bitstream. When the probability of each bit in a bitstream being a one is 0.5 and there is no correlation between bits, we have maximum entropy in our message. Clearly a text stream and most binary files are not maximum entropy because both conditions are violated. Data compression schemes usually exploit regularities in the input data structure to produce higher entropy but smaller data blocks. Forward error-correcting codes nearly always increase the entropy of a data block so data compression to decrease the data block size and increase the entropy will effectively increase the channel capacity. The Shannon limit applies to maximum entropy data streams; if we are using anything less, we are effectively not using our channel capacity as effectively as we could.

5 Assignment

5.1 SystemView

Add noise to your Term Project baseline. Term Projects are due next time. Bring your file on a memory stick, or e-mail it to Dr. Beard by 3:00 PM. A report is also due. Be prepared to run your file, walk the class through its operation, discuss its inputs and outputs, and answer questions from the class and from the Instructor.