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

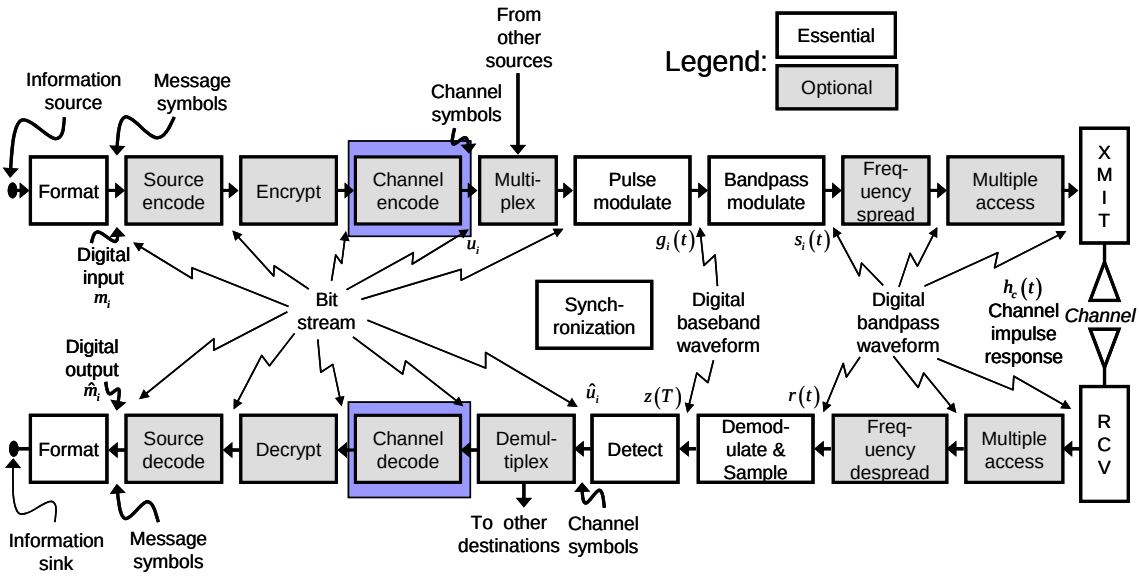
Web Page: http://temple.jkbeard.com

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 1



1 EE521 Course End-Game

Plans for completion of EE521 for Spring 2006 follow.

1.1 Remaining Class Sessions

- Today, April 12, 2006 – Cleanup of Chapter 7, examples of convolutional codes; setting up BER computation in your term project.
- April 19, 2006 – 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 Chapter 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.

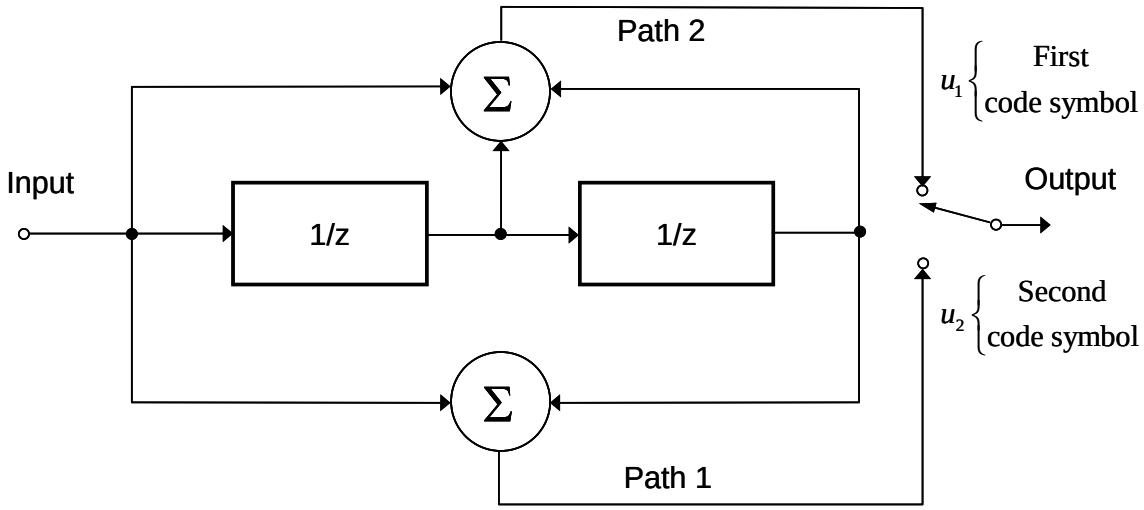


Figure 1 Block Diagram for Rate ½, K=3 Convolutional Encoder

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.

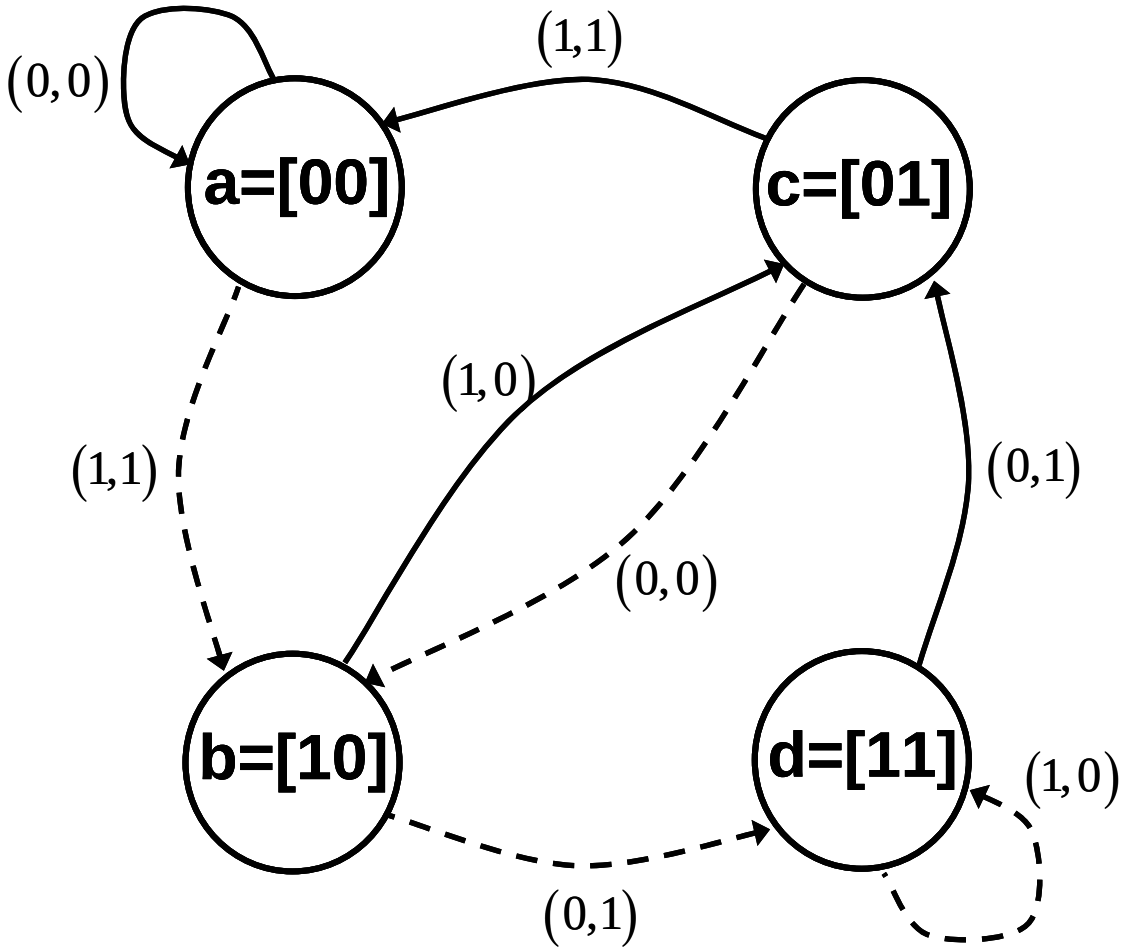


Figure 2 State diagram for rate 1/2, K=3 Convolutional Code

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.

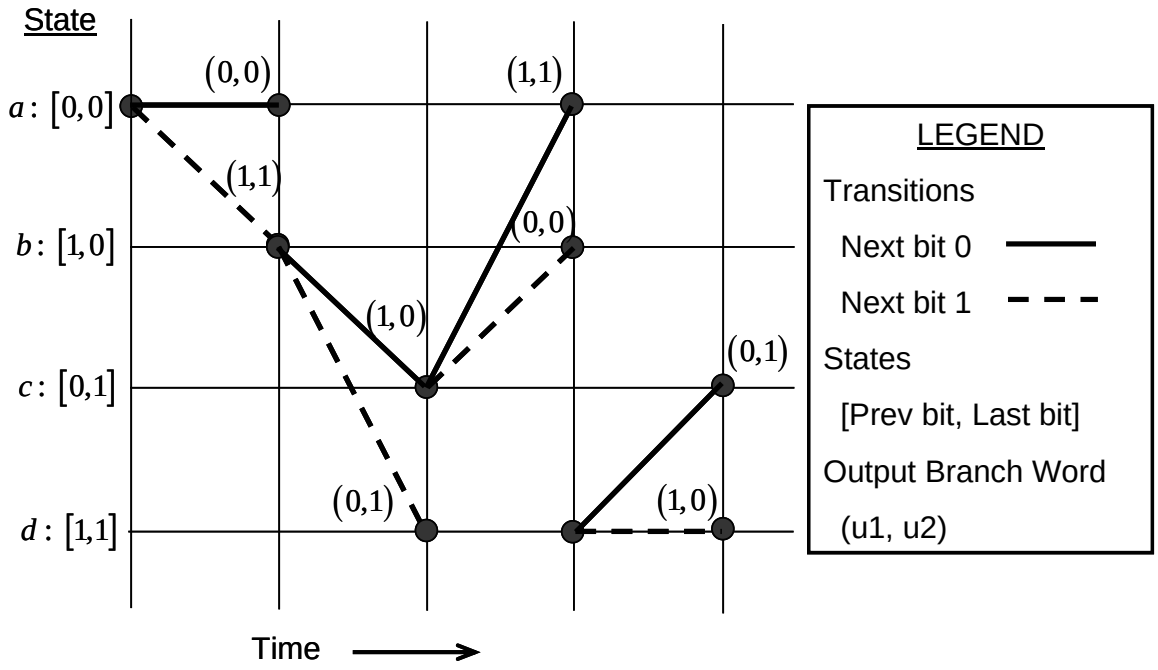


Figure 3 Encoder trellis diagram for a rate 1/2, K=3 Convolutional Encoder

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.

2.2.2 Hard versus Soft Decisions

When the received codeword provides a quantization of the receive signal instead of simply thresholding it to provide a zero or one, this additional data can be provided to a maximum likelihood decoding algorithm. 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 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.

3 Probability of Word Error

This material is a refocus of material from material presented in passing in Sklar's section 6.3 as part of the topic of word errors for parity codes.

3.1 Probability that k of N Bit Errors will Occur

If the probability of bit error is designated *p* and we have *M* bits in a data word (we don't care if it is a token, character, or a collection of bits here), then the number of ways that k errors can occur in *M* bits is given by the binomial distribution:

$$P\left(k\middle|M\right)=\binom{M}{k}\cdot p^{k}\cdot\left(1-p\right)^{M-k} \tag{2.2}$$

where $\binom{M}{k}$ is the binomial coefficient, or the number of combinations of possible ways that the *k* errors may be distributed among the *M* bits,

$$\binom{M}{k}=\frac{M!}{k!\cdot\left(M-k\right)!} \tag{2.3}$$

3.2 Probability of Word Error Without ECC

For a binary word of length *M* without error-correcting coding, including parity check coding with parity bits, the probability that there will be at least one error is

$$P\left(k>0\middle|M\right)=\sum_{j=1}^{M}\binom{M}{k}\cdot p^{j}\cdot\left(1-p\right)^{M-j}$$
$$=1-\left(1-p\right)^{M} \tag{2.4}$$

where, in the last line, we have taken advantage of the fact that the probability that there are zero or more errors is one, and subtracted the probability of zero errors from one.

3.3 Probability of Word Errror With ECC

The probability that the number of bit errors will be less than, or equal to the number *t* of correctable bit errors is

$$P\left(k>t\right)=\sum_{j=t+1}^{M}\binom{M}{j}\cdot p^{j}\cdot\left(1-p\right)^{M-j} \tag{2.5}$$

This is the binomial distribution. Numerical results for the binomial distribution can be obtained by direct computation with Excel or another computer program when *M* is not too large. For larger *M*, you might want to use something like the exeuctiable binary linked from the EE521 web site that uses the incomplete beta function to compute the binomial distribution.

4 Principles of Modeling and Simulation using Menus of System Blocks

Implementation of real systems from the standpoint of feasibility often means looking at COTS modules that will be used to assemble most parts of the receiver. Use of VLSI or FPGAs from large foundries like LSI, Inc. often means that you select from among available macrocells and define the message paths among them. SystemView and other very high-level modeling and design software implements this basic premise by providing a selection of modules that represent COTS components from RF or DSP houses, or as FPGA macrocells. Specific support for TI TMS320C600 series data processors and Xilinx FPGAs is advertised by SystemView, Mentor Graphics, and other modeling software vendors.

Modeling using these tools is based on implementing a high-fidelity simulation that reproduces the actions of digital hardware bit-by-bit. Simulation of analog hardware is more complicated mathematically but is based on simulating the actions of linear hardware in either the time domain (impulse responses) or frequency domain.

The most fundamental issue in the use of system blocks selected from a finite menu is determination of the available system blocks that will perform the functions that you need. Specifications of your functions will include interface requirements such as input and output data format and functional requirements such as dynamic range, latency, and bandwidth. To achieve these specifications, you must observe the requirements of each system block. In SystemView, these requirements include, but are not necessarily limited to:

- Function or processing type – analog, digital, numeric, logical, etc.

- Input data format – analog, zero-one digital, minus one-plus one digital, numerical with signed integers, numerical with unsigned integers, etc.
- Output data format
- Data rate – note that when multiple inputs are used, the data rates must be the same in most modules

To achieve these ends, SystemView provides a number of tokens that can provide data interface, and which represent FPGA macrocells or other available high level functions. These include the quantizer token, which can provide outputs of either floating point or signed integer as a setup option, the token-to-bits and bits-to-token converters, the sampling tokens, and others.

Analog signals have a voltage range. SystemView analog source tokens have an amplitude default of one volt peak-to-peak. Pulse trains have default output voltages of zero volts and one volt. You may choose to use voltages that look more like TTL voltages, such as 0.7 volts and 2.4 volts to represent TTL logic voltages at their specified limits.

We will continue this discussion with SystemView projected and using the whiteboard.

5 Assignment

5.1 Reading

See SystemView manual Appendix A and the application notes mentioned last time for BER computation and curves.

5.2 Homework

See SystemView.

5.3 SystemView

Add noise to your Term Project baseline.