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

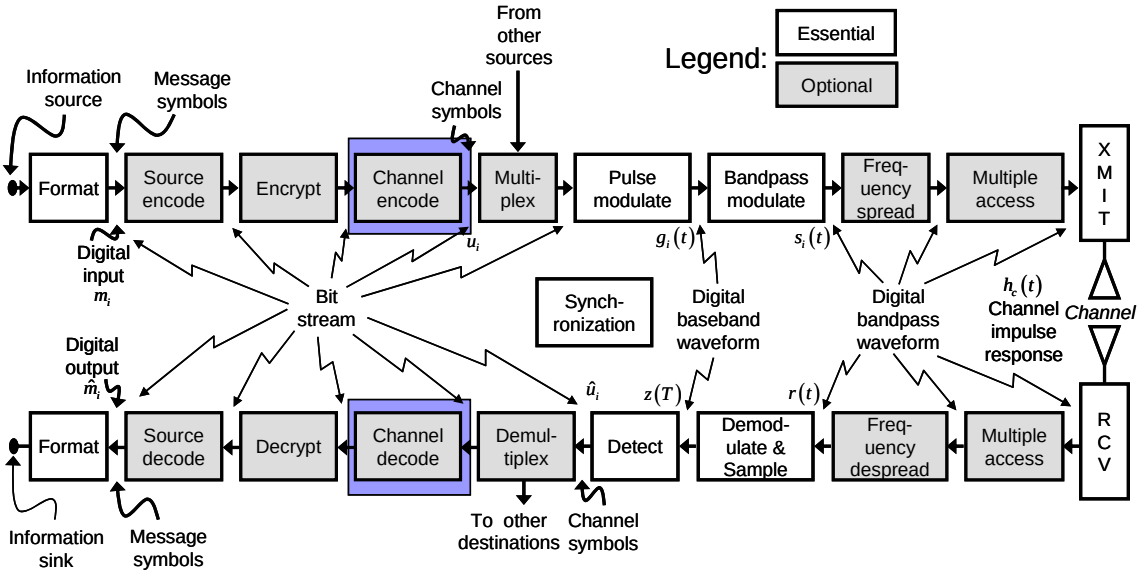
Web Page: http://temple.jkbeard.com

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 Quiz 2 Problems

1.1 Problem 1

SystemView Problem

- Use a frequency sweep signal, from 30 Hz to 3000 Hz..
- Add noise, initially at 17 dB SNR.
- Digitize to a serial bit stream with these tokens:
    - Quantize to 8 bits, using the quantizer from the function library. Make sure that the input voltage range accommodates the voltage of the input, plus noise.
    - Use the sampling operator to sample at the engineering Nyquist rate for the signal chosen.
    - Use a custom algebraic function to turn the sampled, quantized output into a Symbol. This will require that the signal be shifted in level and scaled to become an unsigned integer.
    - Use the symbol-to-bit converter from the Comms library to convert the symbol to a bit stream.
- Convert the bit stream back to a symbol using the bit-to-symbol converter from the comms library.
- Use a custom function to re-scale the symbol to the original scale of the signal plus noise.
- Display the output in a system sink.

See example to be posted after next term project milestone.

1.2 Problem 2

1.2.1 Part A

a) For coherent detection of BPSK in an AWGN channel, what is the BER for Eb/N0 of 5 dB?

b) Repeat for Eb/N0 of 10 dB.

c) Repeat for Eb/N0 of 15 dB.

1.2.2 Part B

Using the results from Part A, calculate the probability of message error for an uncoded 12-bit word in an AWGN channel with a Eb/N0 of 5 dB.

1.2.3 Part C

Using the results from Part A, calculate the probability of message error for a (24,12) linear block code implemented to correct bit errors in one and two bits, e.g. t=2, in an AWGN channel with an Eb/N0 of 5 dB.

1.2.4 Response

From Sklar's Table 4.1 page 219, gives the equations for probability of bit error for four binary modulation schemes. The entry for PSK is

$$P_B = Q\left(\sqrt{\frac{2E_b}{N_0}}\right)$$

Using this equation with the Excel function NORMDIST(x) to compute $Q(x)$ results in

Table 1 PB vs. E_b/N_0 for Problem 2A

E_b/N_0, dB	E_b/N_0	P_B
5	3.16227766	0.005953885
10	10	3.87556E-06
15	31.6227766	8.88178E-16

Using the P_B for 5 dB, we need to know what the probability of word error P_W is for a 12-bit word. This is, as explained in Sklar's Section 6.3,

$$P_W = \sum_{j=2}^{12} \binom{12}{j} \cdot p^j \cdot (1-p)^{2-j}$$

where p is the probability of bit error P_B. Again using Excel, we can prepare the table below.

Table 2 Calculation of P_W for Problem 2B

j	$\binom{12}{j}$	$p^j \cdot (1-p)^{12-j}$	$\binom{12}{j} \cdot p^j \cdot (1-p)^{12-j}$	Terms Used in P_W
0	1	0.930847184	0.930847184	
1	12	0.005575352	0.066904222	0.066904222
2	66	3.33938E-05	0.002203992	0.002203992
3	220	2.00014E-07	4.4003E-05	4.4003E-05
4	495	1.19799E-09	5.93006E-07	5.93006E-07
5	792	7.17543E-12	5.68294E-09	5.68294E-09
6	924	4.29776E-14	3.97113E-11	3.97113E-11

7	792	2.57416E-16	2.03874E-13	2.03874E-13
8	495	1.54181E-18	7.63194E-16	7.63194E-16
9	220	9.23471E-21	2.03164E-18	2.03164E-18
10	66	5.53117E-23	3.65057E-21	3.65057E-21
11	12	3.31292E-25	3.97551E-24	3.97551E-24
12	1	1.98429E-27	1.98429E-27	1.98429E-27
SUMS			1	**0.069152816**

Note that the first term in the sum dominates the total P_W. If we have a (24,12) code with a t of 2, we compute the probability of 3 or more bit errors out of 24 bits as

$$P_W = \sum_{j=3}^{24} \binom{24}{j} \cdot p^j \cdot (1-p)^{24-j}$$

Again using Excel, we produce the table below.

j	$\binom{24}{j}$	$p^j \cdot (1-p)^{24-j}$	$\binom{24}{j} \cdot p^j \cdot (1-p)^{24-j}$	**Terms Used in P_W**
0	1	0.866476479	0.866476479	
1	24	0.005189801	0.124555213	
2	276	3.10845E-05	0.008579335	
3	2024	1.86182E-07	0.000376833	0.000376833
4	10626	1.11515E-09	1.18496E-05	1.18496E-05
5	42504	6.67923E-12	2.83894E-07	2.83894E-07
6	134596	4.00055E-14	5.38459E-09	5.38459E-09
7	346104	2.39615E-16	8.29317E-11	8.29317E-11
8	735471	1.43519E-18	1.05554E-12	1.05554E-12
9	1307504	8.59611E-21	1.12394E-14	1.12394E-14
10	1961256	5.14868E-23	1.00979E-16	1.00979E-16
11	2496144	3.08382E-25	7.69767E-19	7.69767E-19
12	2704156	1.84707E-27	4.99477E-21	4.99477E-21
13	2496144	1.10631E-29	2.76151E-23	2.76151E-23
14	1961256	6.6263E-32	1.29959E-25	1.29959E-25

j	$\binom{24}{j}$	$p^j \cdot (1-p)^{24-j}$	$\binom{24}{j} \cdot p^j \cdot (1-p)^{24-j}$	**Terms Used in P_W**
15	1307504	3.96885E-34	5.18929E-28	5.18929E-28
16	735471	2.37716E-36	1.74833E-30	1.74833E-30
17	346104	1.42381E-38	4.92787E-33	4.92787E-33
18	134596	8.52799E-41	1.14783E-35	1.14783E-35
19	42504	5.10788E-43	2.17105E-38	2.17105E-38
20	10626	3.05939E-45	3.25091E-41	3.25091E-41
21	2024	1.83243E-47	3.70885E-44	3.70885E-44
22	276	1.09755E-49	3.02922E-47	3.02922E-47
23	24	6.5738E-52	1.57771E-50	1.57771E-50
24	1	3.93741E-54	3.93741E-54	3.93741E-54
SUMS			1	**0.000388972**

Again note that the first term dominates the sum. Since we cal tolerate two bit errors, we begin the sum with the third term, as opposed to beginning the sum with the second term. The probability of word error is smaller by approximately a factor of $1/P_B$.

1.3 Problem 3

This problem will be concerned with the (8,4) linear block code determined by the generator matrix *G*:

0	1	1	1	1	0	0	0
1	0	1	1	0	1	0	0
1	1	0	1	0	0	1	0
1	1	1	0	0	0	0	1

1.3.1 Part A

a) Is this a systematic code? Why?
b) Use Excel or other aid and compile a codeword list.
c) What is d_{min}?
d) What is the maximum number of correctable errors *t*?

1.3.2 Part B

a) Draw the parity portion of the generator matrix.
b) Draw the transpose of the parity check matrix, H^T.

c) Compute the syndrome for the received code word (1, 0, 0, 1, 1, 0, 1, 0)

1.3.3 Response

The right for columns of the generator matrix is I_4 so we have a systematic code. The parity portion of the generator matrix is the leftmost four columns. The Hamming weight of all the codewords is 4 except the zero codeword and its complement, which have Hamming weights of 0 and 8, respectively, so d_{min} is 4, and t is 2. The parity check matrix is given by the table below.

Table 3 Transposed Parity Check Matrix for Problem 3

1	0	0	0
0	1	0	0
0	0	1	0
0	0	0	1
0	1	1	1
1	0	1	1
1	1	0	1
1	1	1	0

The syndrome for the specified received code word is (0, 0, 1, 1).

1.4 Problem 4

Here we will be concerned with a (15,5) systematic cyclic code. The generator polynomial is

$$g(x) = 1 + x + x^2 + x^5 + x^8 + x^{10}$$

We have a message polynomial

$$m(x) = 1 + x^2 + x^3$$

For this systematic cyclic code, the parity bits are the lower order portion of the polynomial.

1.4.1 Part A

Write an equation for the code polynomial that places the parity polynomial *p(x)* in the lower orders of the code polynomial, and places the message polynomial in the higher orders of the code polynomial.

1.4.2 Part B

Find the parity polynomial.

1.4.3 Part C

Write the codeword polynomial

1.4.4 Response

Sklar's Section 6.7.3 discusses cyclic coding in systematic form. For an (n,k) cyclic code, the *k* message bits are posed as an order *k-1* polynomial, multiplied by x^{n-k} to make an order *n-1* polynomial with the lower order *n-k* coefficients zero. The message polynomial, multiplied by x^{n-k}, is divided by the generator polynomial (which is of order *n-k*) to provide a quotient and a remainder polynomial of order *n-k-1*. The coefficients of the remainder polynomial are the parity bits, and provide the lower order *n-k* bits of the codeword. Thus the codeword is

$$c(x) = p(x) + x^{n-k} \cdot m(x)$$
$$= \sum_{i=0}^{n-k-1} p_i \cdot x^i + x^{n-k} \sum_{j=0}^{k-1} m_j \cdot x^j$$

where the parity polynomial *p(x)* is the remainder in the division

$$\frac{x^{n-k} \cdot m(x)}{g(x)} = q(x) + \frac{p(x)}{g(x)}$$

Using the example given in the lecture of March 29, and the Excel file that is posted on the web site http://temple.jkbeard.com on the Files page, we can perform this division with Excel, resulting in the table below.

Table 4 Polynomial Division for Problem 4

Division Process		Yes/No															
p0	x^10*m		0	0	0	0	0	0	0	0	0	0	1	0	1	1	**0**
	x^4*g	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	Sum		0	0	0	0	0	0	0	0	0	0	1	0	1	**1**	0
	x^3*g	1	0	0	0	1	1	1	0	0	1	0	0	1	0	1	
	Sum		0	0	0	1	1	1	0	0	1	0	1	1	**1**	0	
	x^2*g	1	0	0	1	1	1	0	0	1	0	0	1	0	1		
	Sum		0	0	1	0	0	1	0	1	1	0	0	**1**	0		
	x^1*g	1	0	1	1	1	0	0	1	0	0	1	0	1			
	Sum		0	1	0	1	0	1	1	1	1	1	**0**	0			
	x^0*g	0	0	0	0	0	0	0	0	0	0	0	0				
	Sum		0	1	0	1	0	1	1	1	1	1	0				
	Quotient		0	1	1	1	0										
	Remain der		0	1	0	1	0	1	1	1	1	1					

The codeword is the remainder followed by the message, (0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0). These are the coefficients of the codeword polynomial, beginning with the x^0 coefficient.

2 EE521 Course End-Game

Here we lay out the plans for completion of EE521 for Spring 2006

2.1 Remaining Class Sessions

- Today, April 5, 2006 – Introduction to Convolutional Codes
- 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

2.2 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.

2.3 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.

3 Convolutional Codes

From Sklar Chapter 7, *Channel Coding: Part 2*. Convolutional codes are currently the most common codes used in commercial cell phones. We will look at some simple examples here.

3.1 Convolutional Encoding

3.1.1 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.*

3.1.2 Simple Convolutional Encoder Block Diagram

A block diagram for

$$\begin{aligned} k &= 1 \\ n &= 2 \\ K &= 3 \end{aligned} \tag{3.1}$$

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.

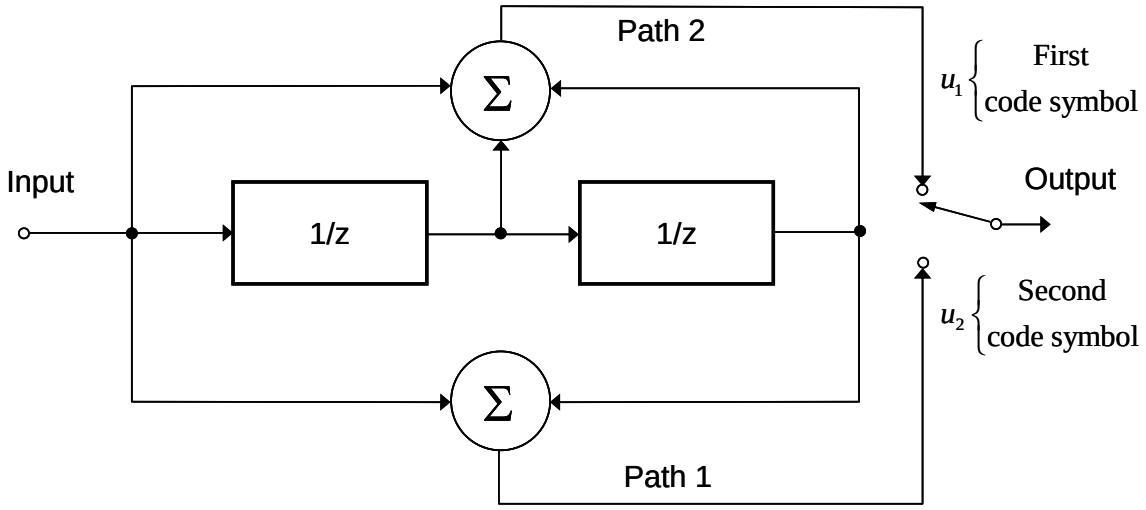


Figure 1 Block Diagram for Rate ½, K=3 Convolutional Encoder

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.

3.1.3 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 5.

Table 5 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 6.

Table 6 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]

3.1.4 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.

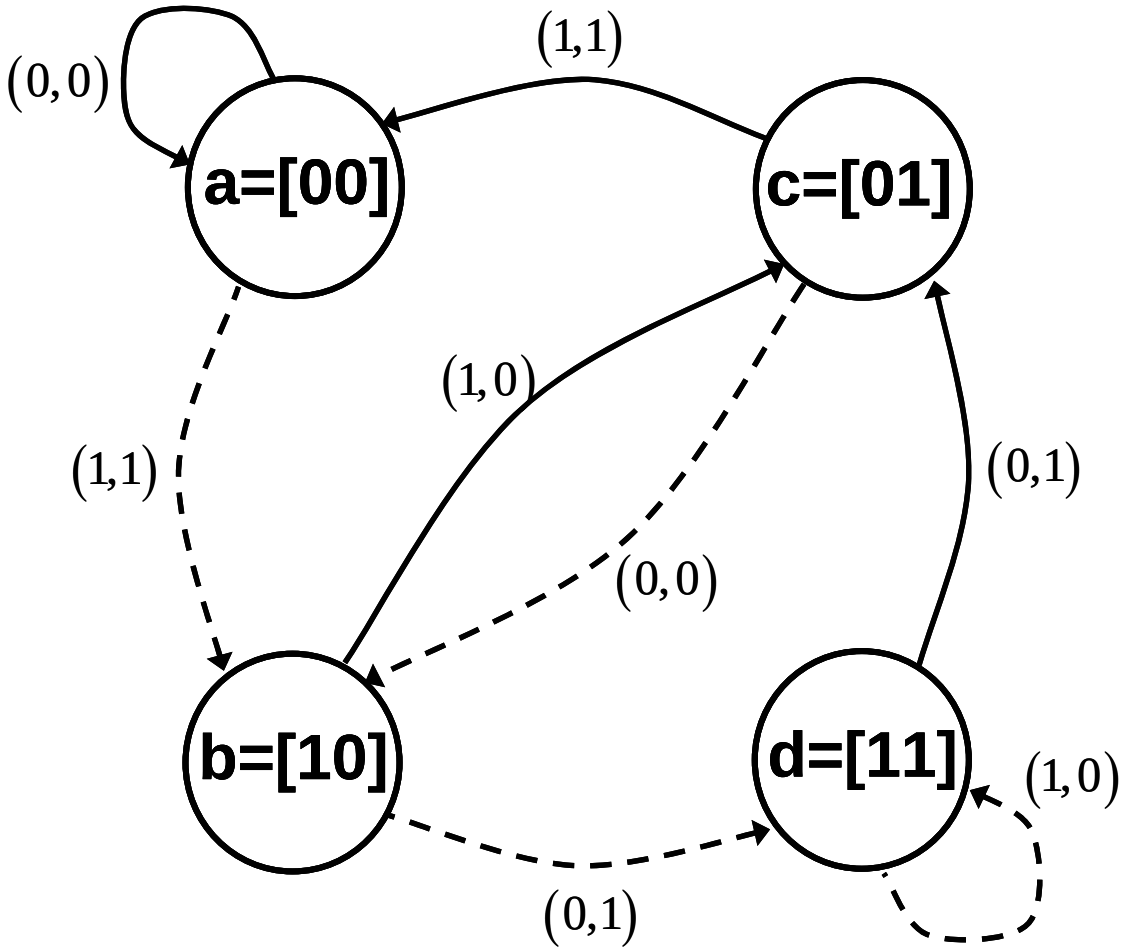


Figure 2 State diagram for rate 1/2, K=3 Convolutional Code

3.1.5 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.

3.1.6 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.

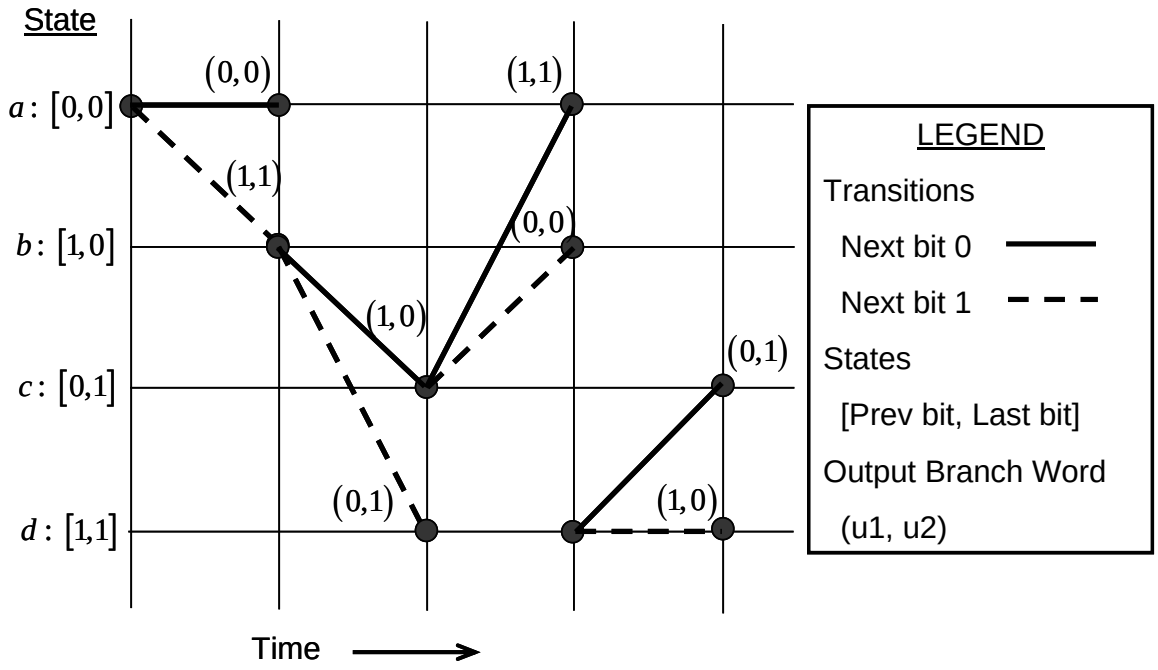


Figure 3 Encoder trellis diagram for a rate 1/2, K=3 Convolutional Encoder

3.2 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.

3.2.1 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.

3.2.2 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.

3.2.3 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.

4 Assignment

4.1 Reading

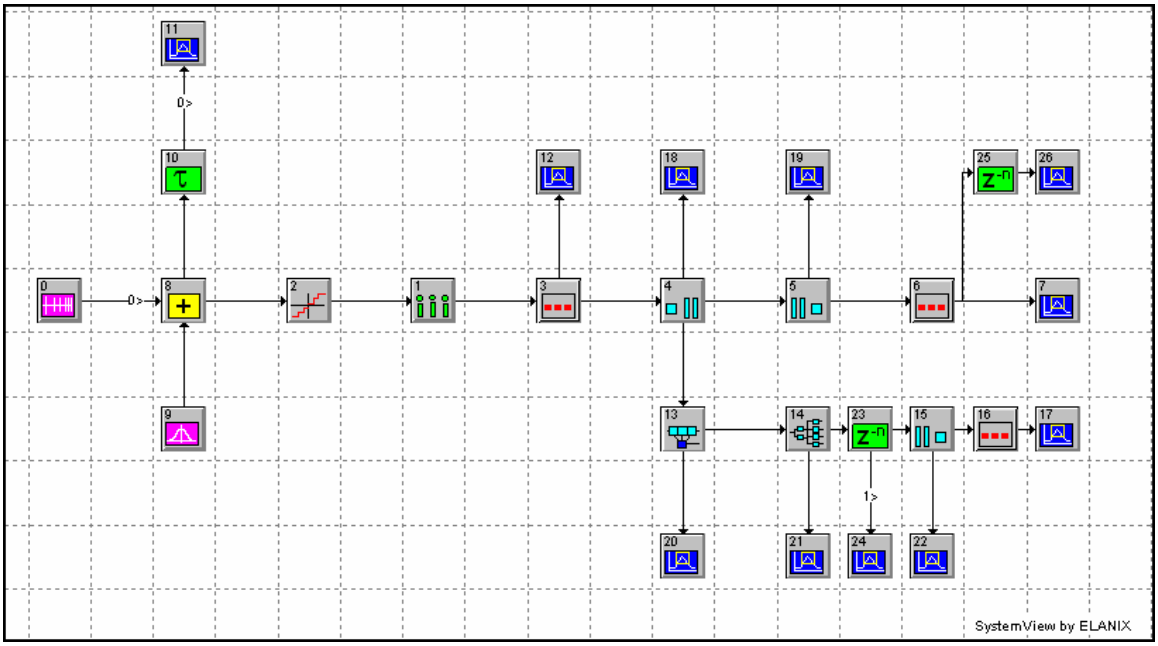
Sklar, Chapter 7, Sections 7.1-7.3, pages 382-408.

4.2 Homework

Take-home quiz for 75% of your grade for Quiz 2. The SystemView problem from last time is 25% of your grade.

4.3 SystemView

Add a simple convolutional code-decode to your Term Project baseline. Use the default convolutional code in the Communications Encode/Decode collection. The convolutional encoder is labeled “Cnv Coder” and the decoder is labeled “Cnv dCode.” Your system may look something like this:



Note that there is a delay, Token 23, that is required for the bits-to-symbol converter to work. The function of this delay is to align the 8-bit sequences in the bit stream out of the decoder with what is required by the bit-to-symbol decoder, which simply counts from the first symbol.

We have added a second output from the original data, token 26, and added a delay, token 25, so that this output is time-aligned with the output from the convolutional encoder-decoder path, token 17. You will need this to determine when bit errors occur in the next step.

When this is complete, we will add noise to the input to the convolutional decoder, token 14, and measure the bit error rate (BER). For information on BER, see the SystemView online documentation of the BER token in the Communications library, Processors tab. I will provide input on how you may approach BER computation. You will be required to find one BER for a specific single SNR in your demo. If you choose to run a BER curve, and the specified BER is within the limits of your curve, that is sufficient, but not required. Some examples, rice_tst.svu, blk_tst.svu, golay.svu, and cnvl.svu are available in your SystemView Program Documents folder. Also, as is a more elaborate example, BER_9pde3_srx32_RRC_ud_conv.svu is also there. There is an excellent but complex application note, AN-107, on computing BER curves using SystemView.