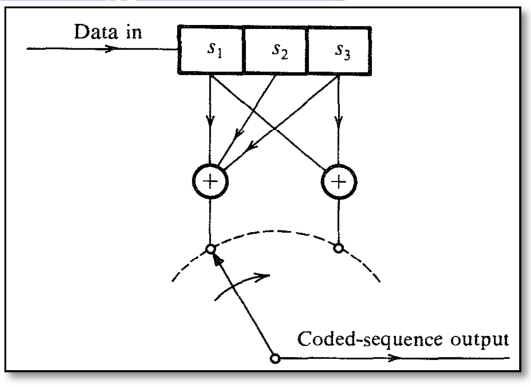
WEEK 14

CONVOLUTIONAL CODES

In a block code, the block of n code digits generated by the encoder in any particular time unit depends only on the block of k input data digits within that time unit. In a convolutional code, on the other hand, the block of n code digits generated by the encoder in a particular time unit depends not only on the block of k message digits within that time unit but also on the block of data digits within a previous span of N - 1 time units $(N > 1)$. For convolutional codes, k and n are usually small. Convolutional codes can be devised for correcting random errors, burst errors, or both. Encoding is easily implemented by shift registers. As a class, convolutional codes invariably outperform block codes of the same order of complexity.

A convolutional coder with constraint length N consists of an N-stage shift register and v modulo-2 adders. Figure A shows such a coder for the case of $N = 3$ and $v = 2$. The message digits are applied at the input of the shift register. The coded digit stream is obtained at the commutator output.

For an input digits 11010. Initially, all the stages of the register are clear; that is, they are in a 0 state. When the first data digit 1 enters the register, the stage s\ shows 1 and all the other stages (s_2 and s_3) are unchanged; that is, they are in a 0 state. The two modulo-2 adders show $v_x = 1$ and $v_2 = 1$. The commutator samples this output. Hence, the coder output is 11. When the second message bit 1 enters the register, it enters the stage s_1, and the previous 1 in s_1 is shifted to s_2. Hence, s_1 and s_2 both show 1, and s_3 is still unchanged; that is, it is in a 0 state. The modulo-2 adders now show 01. In the same way, we repeat this scenario till the end. Observe that each data digit influences N groups of v digits in the output (in this case three groups of two digits).



We cannot stop here, however. We add $N - 1$ number of 0's to the input stream (dummy data) to make sure that the last data digit (0 in this case) proceeds all the way through the shift register in order to influence the N groups of v digits. Hence, when the input digits are 11010, we actually apply 1101000 to the input of the shift register. It can be seen that when the last digit of the message stream enters the last digit of the message stream has passed through all the N stages of the register & the output code is :
11 01 01 00 10 11 00.
Thus, there are in all $n = (N + k - 1)v$ digits in the coded output for every k data digits.

Code tree, Trellis, and state diagram

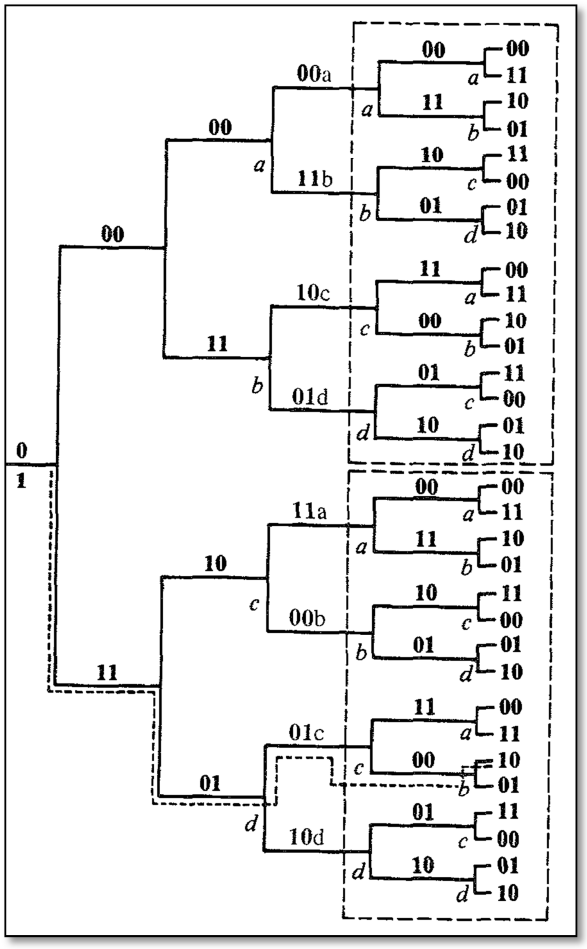
The structural properties of a convolutional encoder are shown in graphical form by using any one of three equivalent diagrams: code tree, trellis, and state diagram. We will use the convolutional encoder of Figure 10.13a as a running example to illustrate the insights that each one of these three diagrams can provide.

We begin the discussion with the *code tree* of Figure A. Each branch of the tree represents an input symbol, with the corresponding pair of output binary symbols indicated on the branch. The convention used to distinguish the input binary symbols 0 and 1 is as follows. An input 0 specifies the upper branch of a bifurcation, whereas input 1 specifies the lower branch. A specific *path* in the tree is traced from left to right in accordance with the input (message) sequence. The corresponding coded symbols on the branches of that path constitute the input (message) sequence. Consider, for example, the message sequence (11010) applied to the input of the encoder of Figure A. Following the procedure just described, we find that the corresponding encoded sequence is (11 01 01 00 10 11 00), which agrees with the first pairs of bits in the encoded sequence derived recently.

When a data bit enters the shift register (in stage s_1, the output bits are determined not only by the data bit in s_1, but by the two previous data bits already in stages s_3 and s_2. There are four possible combinations of the two previous bits (in s_3 and s_2)" 00, 01,10, and 11. We shall label these four states as [a, b, c, and d], respectively. Thus, when the previous two bits are 01 (s_3 = 0, s_2 = 1), the state is *b,* and so on.

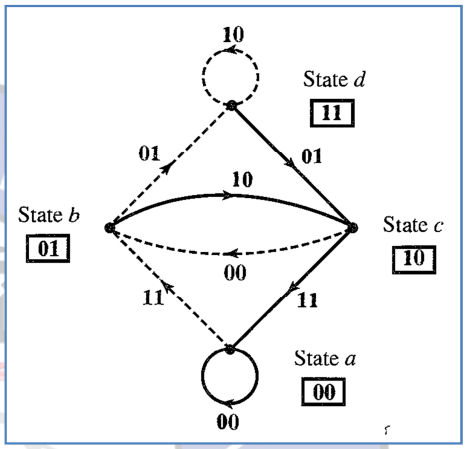
A data bit 0 or 1 generates four different outputs, depending on the encoder state. If the data bit is 0, the encoder output is: 00,10,11, 01, depending on whether the encoder state is.

if the data bit is 1, the encoder output is: 11,01, 00, or 10.



State	*Binary Description*
a	00
b	10
c	01
d	11

The recent scenario can be represented also by what is called state diagram:



We use solid lines when the input bit is 0, and dashed lines when the input bit is 1. For instance, when the encoder is in state a, and we input 1, the encoder output is 11 (dashed line). The encoder now goes to state *b* for the next data bit because at this point the previous two bits become $s_3 = 0$ and $s_2 = 1$. Similarly, when the encoder is in state *a* and the input is 0, the output is 00 (solid line), and the encoder remains in state *a*.

Another useful way of representing the code tree is the trellis diagram in Fig. B. The diagram starts from scratch (all 0's in the shift register, that is, state *a*) and makes transitions corresponding to each input data digit. These transitions are denoted by a solid line for the next data digit 0 and by a dashed line for the next data digit 1. Thus, when the first input digit is 0, the encoder output is 00 (solid line), and when the input digit is 1, the encoder output is 11 (dashed line). This is readily seen from the state diagram shown recently. We continue this way with the second input digit. After the first two input digits, the encoder is in one of the four states a, c, or d, as shown in Fig. B. If the encoder is in state *a* (previous two data digits 00), it goes to state *b* if the next input bit is 1 or remains in state *a* if the next input bit is 0. In so doing, the encoder output is 11 (a to b) or 00 (a to a). Note that the structure of the trellis diagram is completely repetitive, as expected, and can be readily drawn using the state diagram.

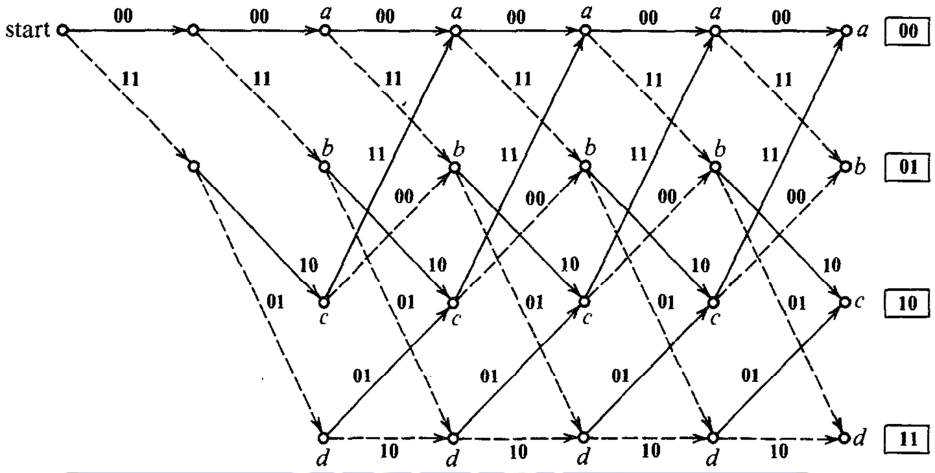


Fig. B, the Trellis diagram