Chapter 8: Convolutional Codes

Convolutional codes were introduced in 1955 by Elias as an alternative to Block codes. Convolutional codes are different from block codes by the existence of memory in the encoding scheme. Though convolutional codes can detect errors, they are good for error correction. These codes can be used for correcting random errors, burst errors or both. Convolutional codes are also known as recurrent codes.

The fundamental hardware unit for convolutional encoder is a tapped shift register with (L+1) stages as shown below. L is the constraint length of the convolutional encoder and will be discussed later.

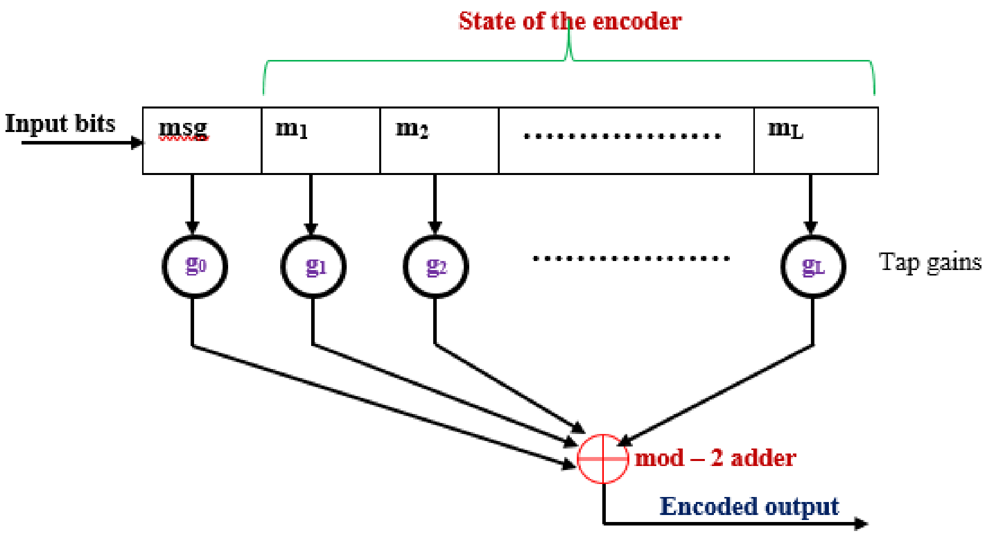
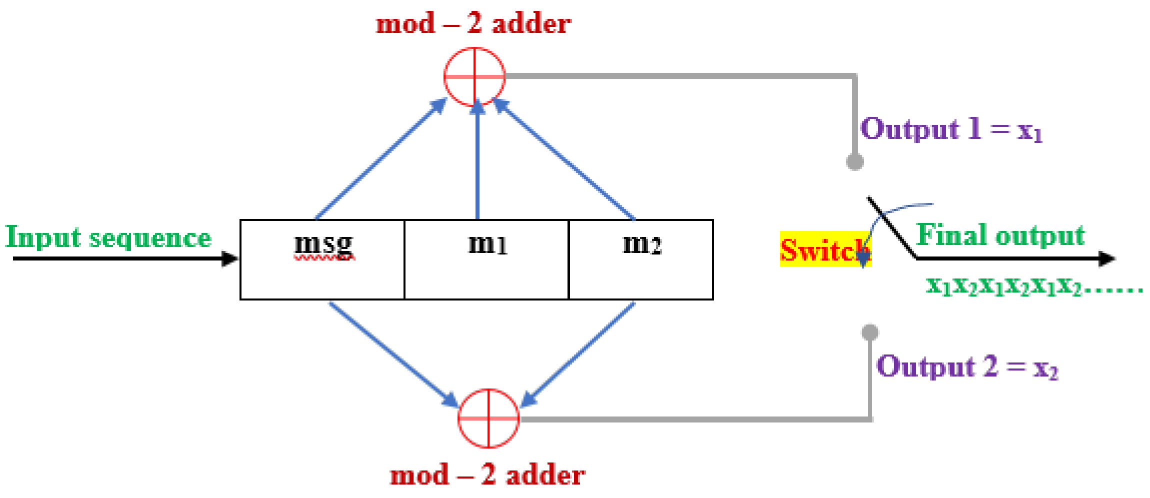


Fig: Block diagram of a general convolutional encoder

Here g_0, g_1, g_2,…… etc are tap gains which are nothing but binary digits 0 or 1. 0 represents no connection and 1 represents connection. So, each tap gain is a binary digit representing a short circuit or open circuit connection. The message bits enter one by one into the tapped shift register, which are then combined by mod-2 summers to generate the encoded output.

Consider below (2, 1, 2) convolutional encoder. msg box represents place for current input bit. Combination of m_1m_2 represents state of the encoder.



The above encoder can be described by two generator sequences:

$g^{(1)} = \left(g_0^{(1)}, g_1^{(1)}, g_2^{(1)}\right) = (1,1,1)$ $\longrightarrow$ represents 1st output x_1. Here tap gains $g_0^{(1)}, g_1^{(1)}, g_2^{(1)}$ are 1 1 1 because all stages are connected to mod-2 adder. 1 represents there is a connection to mod-2 adder.

$g^{(2)} = \left(g_0^{(2)}, g_1^{(2)}, g_2^{(2)}\right) = (1,0,1)$ $\longrightarrow$ rep resents 2nd output x_2. Here tap gains $g_0^{(2)}, g_1^{(2)}, g_2^{(2)}$ are 1 0 1 because m1 stage is not connected to mod-2 adder. 0 (zero) represents there is no connection to mod-2 adder.

Note that $g^{(1)}$ and $g^{(2)}$ are called generator sequences of the encoder. Generator sequences are nothing but impulse response of the encoder. The encoder output is obtained by the convolution of the input sequence with the impulse response of the encoder, hence the name convolutional code. Impulse response of the encoder is the response of the encoder to a single "1" bit that moves through it.

Numerous other convolutional codes are obtained by modifying the encoder shown in figure. If we just change the connections to the mod-2 summers, then the encoded output will change.

The message bits in the register are combined by mod-2 addition to form the encoded output. The input data to the encoder, which is assumed to be binary, is shifted into and along the shift register k-bits at a time. The number of output bits for each k-bit input sequence is n bits. The switch samples the all mod-2 adders in sequence, once during each bit interval.

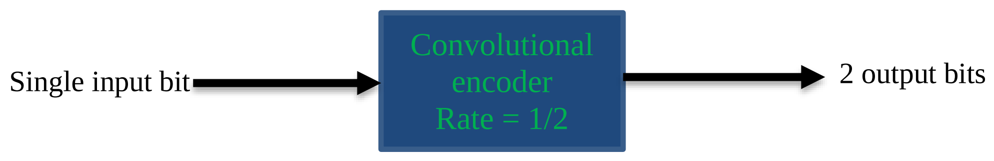
This encoder generates non-systematic codes. Unlike block codes, the use of non-systematic codes is usually preferred over systematic codes in convolutional coding. In systematic codes, message information can be seen and directly extracted from the encoded information.

Some important terms of convolutional codes

Code rate R

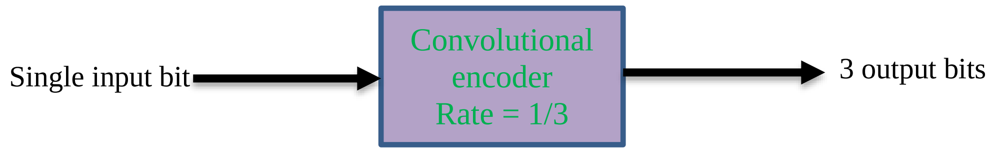
$$code\ rate\ R = \frac{k}{n} = \frac{number\ of\ message\ bits\ processed\ at\ a\ time}{number\ of\ encoded\ output\ bits}$$

Rate $\frac{1}{2}$ convolutional encoder



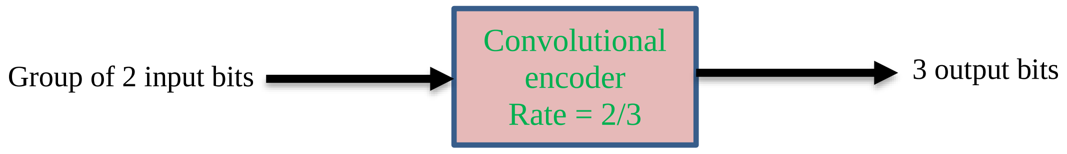
- Number of message bits k = 1
- Number of encoded bits n = 2
- Rate ½ means for each one-input bit, encoder provides 2 output bits. Encoder operates on one-bit at a time.
- Suppose input sequence = 10110, then total number of encoded output bits = 5 x 2 = 10

Rate $\frac{1}{3}$ convolutional encoder



- Number of message bits k = 1
- Number of encoded bits n = 3
- Rate $\frac{1}{3}$ means for each one-input bit, encoder provides 3 output bits. Encoder operates on one-bit at a time.
- Suppose input sequence = 11110, then total number of encoded output bits = 5 x 3 = 15

Rate $\frac{2}{3}$ convolutional encoder



- Number of message bits k = 2 (two input bits are processed at a time)
- Number of encoded bits n = 3 (each group of 2 inputs are transformed into 3 bits)
- Rate $\frac{2}{3}$ means for each group of 2 input bits, encoder provides 3 output bits. Encoder operates on 2 bits at a time.
- Suppose input sequence = 1110, then total number of encoded output bits = 3 + 3 = 6

For convolutional codes k and n are usually very small integers

Convolutional codes employed in FEC systems usually have small values of n and k, while constraint lengths typically fall in the range of 10 to 30. All convolutional encoders require a commutator switch at the output.

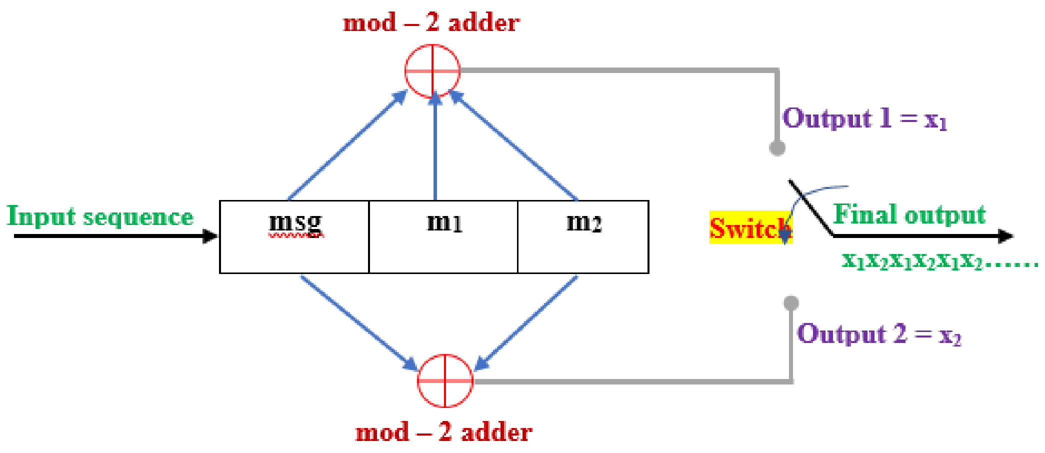
Constraint length (L)

A convolutional code is described by 3 integers: n, k, L

L = constraint length of encoder. Some authors represent constraint length with letter **K**

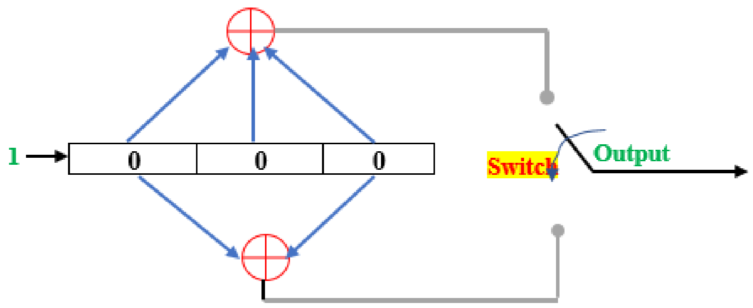
It is defined as the number of shifts over which a single message bit can influence the encoder output bit.

Consider below (2, 1, 2) convolutional encoder. msg box represents place for current input bit.

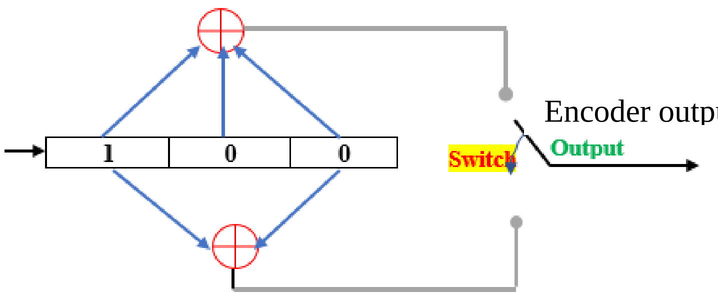


Let us assume input sequence = 1

Initial condition

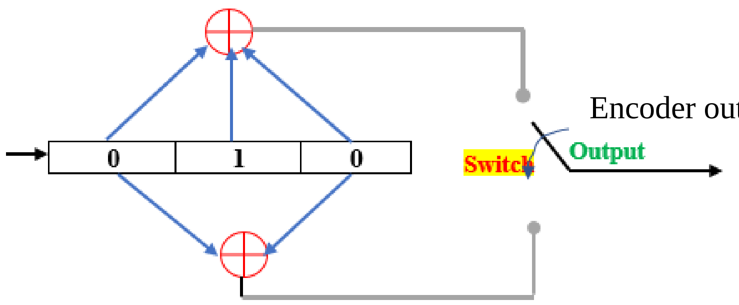


Place input bit 1 into msg box



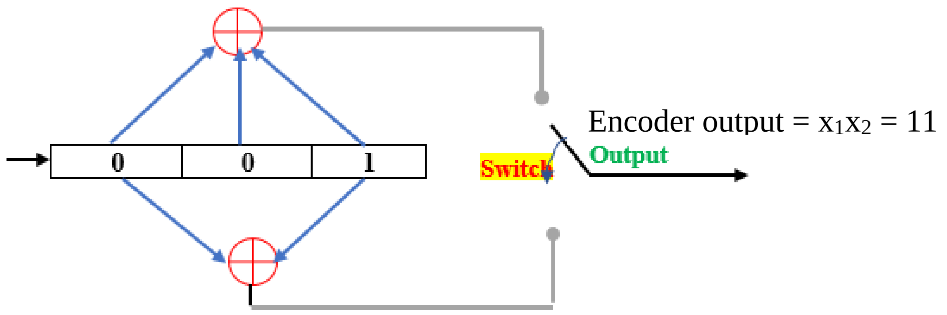
Encoder output = x_1x_2 = 11

After applying 1st clock pulse (1st shift)



Encoder output = x_1x_2 = 10

After applying 2nd clock pulse (2nd shift)



After 3rd clock pulse, the input bit "1" is discarded. So, the input bit 1 influences the output of the encoder for 2 shifts. Hence L (=2) is the number of shifts over which a single message bit can influence the encoder output bit.

Constraint length in terms of encoded output bits:

n(L+1) = 2(2+1) = 2(3) = 6 bits

So, each message bit influences a span of n(L+1) successive output bits. The quantity n(L+1) is called the constraint length measured in terms of encoded output bits. Here L is the encoder's memory or number of bits used to represent state of the encoder.

L = 3 means single message bit influences encoder output for 3 successive shifts.

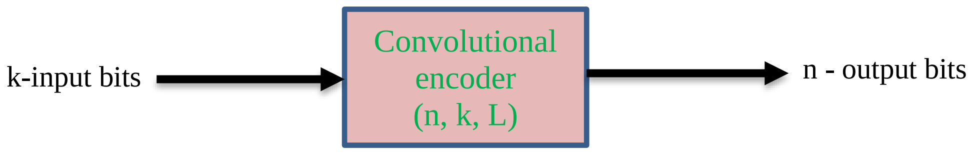
Code dimension

A convolutional code is described by 3 integers: n, k, L

$k = number\ of\ message\ bits\ processed\ at\ a\ time$

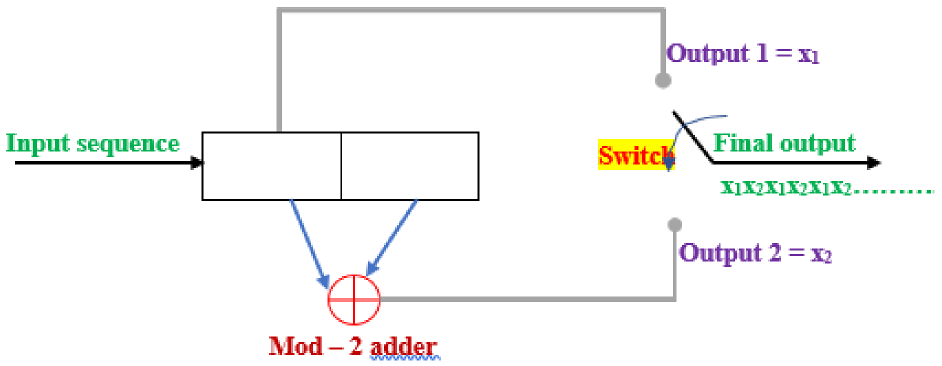
$n = number\ of\ encoded\ output\ bits$

L = constraint length of encoder. Some authors represent constraint length with letter **K**



Examples: (2, 1, 3) code, (3, 2, 4) code, (3, 2, 1) code etc.
In practice, n and k are small integers and L is varied to control the capability and complexity of the code.

Q. Below figure depicts a rate ½, constraint length L = 1, convolutional encoder. Sketch the tree diagram, the trellis diagram and the state diagram.



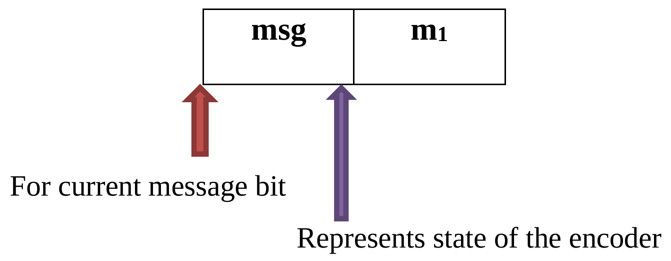
Sol:

$$Rate, r = \frac{1}{2}\ means\ 1\ input\ bit\ produces\ 2\ encoded\ output\ bits$$

$$Rate, r = \frac{1}{3}\ means\ 1\ input\ bit\ produces\ 3\ encoded\ output\ bits$$

$$Rate, r = \frac{2}{3}\ means\ group\ of\ 2\ input\ bits\ produces\ 3\ output\ bits$$

$$in\ general, code\ rate\ r = \frac{k}{n}\ means\ set\ of\ k\ input\ bits\ produces\ n\ output\ bits$$



Here, state of the encoder is represented by 1 – bit only.

$$\therefore\ 2^1 = 2\ states\ are\ possible$$

If 2 bits are used for state, the $2^2 = 4$ states are possible
If 3 bits are used for state, the $2^3 = 8$ states are possible
If 4 bits are used for state, the $2^4 = 16$ states are possible

State table

Incoming message bit **msg**	State of encoder **m_1**	Current state	Next state	Encoded output x_1x_2
------	Initial state = 0	Initial all – zero state		
0	0	a		
1	0	a		
0	1	b		
1	1	b		

Let us use ‘**a**’ to represent state **0** and **b** to state **1** as shown below.

m_1	State
0	**a**
1	**b**

m_1m_2	State
00	a
01	b
10	c
11	d

State table

Incoming message bit **msg**	State of encoder **m_1**	Current state	Next state	Encoded output x_1x_2
------	Initial state = 0	Initial all – zero state		
0	0	a		00
1	0	a		11
0	1	b		01
1	1	b		10

Output of encoder is calculated using equations

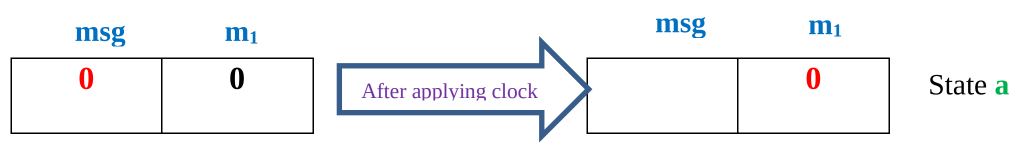
$$x_1 = msg$$
$$x_2 = msg \oplus m_1$$

$\oplus$ *represents Ex – OR logic or mod – 2 addition*

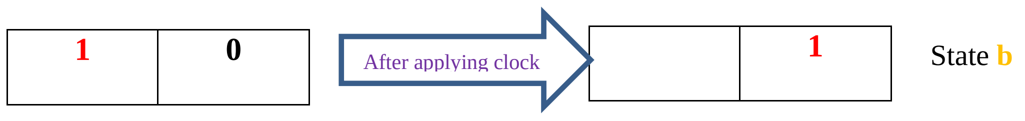
Let us calculate next states of given convolutional encoder

Incoming message bit msg	State of encoder m_1	Current state	Next state	Encoded output x_1x_2
------	Initial state = 0	Initial all – zero state		
0	0	a	a	00
1	0	a	b	11
0	1	b	a	01
1	1	b	b	10

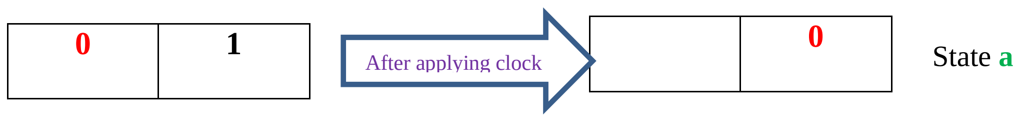
Step 1:



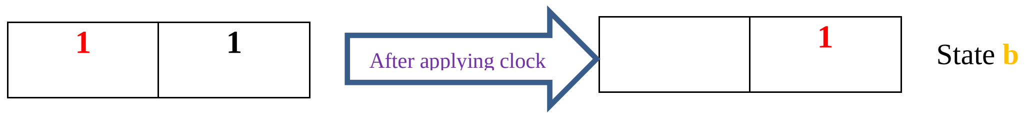
Step 2:



Step 3:

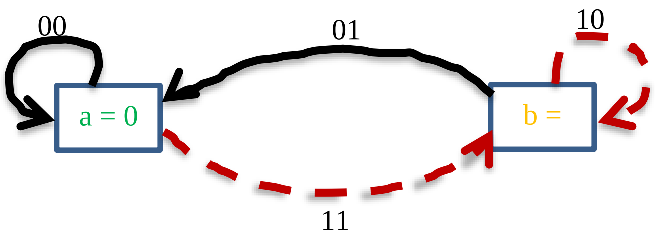


Step 4:



State table is very useful to draw STATE diagram, code TRELLIS and code TREE diagrams. These diagrams are graphical representation of convolutional codes, from which we can calculate the output of convolutional encoder for any given input.

State diagram

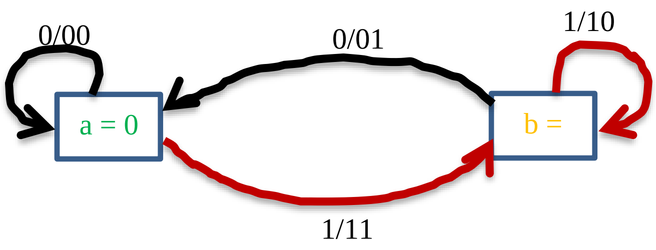


Conventions used:

Solid line ⟹ for bit 0
Dotted line ⟹ for bit 1

Other way of drawing state diagram:

0/01 means **input/output** ⟹ for current input bit 0, the resulting output is 01. Avoids use of dashed line notation.



Finding encoded output from STATE diagram

Let's us assume input sequence m = 11101

Initially start from all-zero state i.e., a = 0

Step 1: Initially, we are at state **a**. Check that 1st message bit = 1. See from node **a**. For bit 1 state changes from node **a** to **b** (Red line) and output = 11.

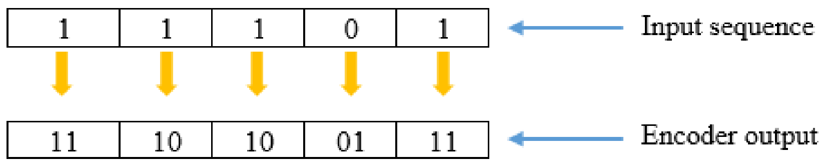
Step 2: Now we are at node **b**. Check that 2nd message bit = 1. See from node **b**. For bit 1 state changes from **b** to **b** (Red line) and output = 10.

Step 3: Now we are at node **b**. Check that 3rd message bit = 1. See from node **b**. For bit 1 state changes from **b** to **b** (Red line) and output = 10.

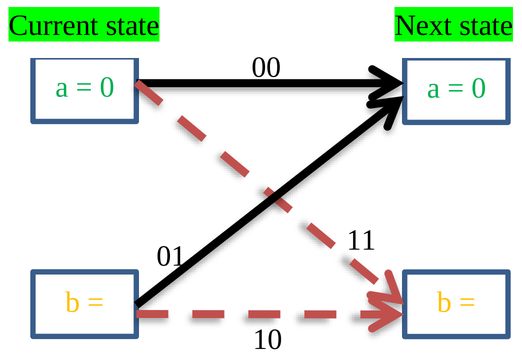
Step 4: Now we are at node **b**. Check that 4th message bit = 0. See from node **b**. For bit 0 state changes from **b** to **a** (Black line) and output = 01.

Step 5: Now we are at node **a**. Check that 5th message bit = 1. See from node **a**. For bit 1 state changes from **a** to **b** (Red line) and output = 11.

$\therefore$ *Encoded output or codeword is as follws*:



Code TRELLIS

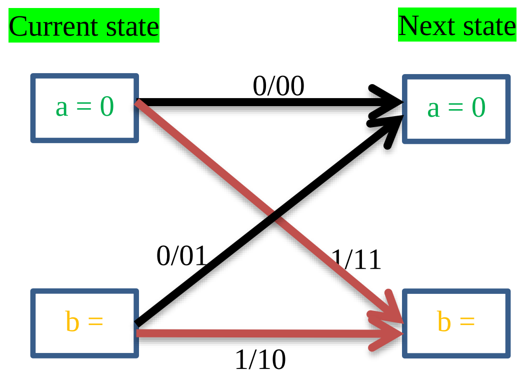


Conventions used

Solid line	for bit 0
Dotted line	for bit 1

Other way of drawing TRELLIS diagram

0/01 means **input/output** for current input bit 0, the resulting output is 01. Avoids use of dashed line notation.



Finding encoded output from TRELLIS diagram

Let's us assume input sequence m = 11101

Initially start from all-zero state i.e., a = 0

Step 1: Initially, we are at node **a**. Check that 1st message bit = 1. See from node **a**. If the incoming bit is 1 state changes from **a** to **b** (Red line) and output = 11.

Step 2: Now we are at node **b**. Check that 2nd message bit = 1. See from node **b**. If the incoming bit is 1 state changes from **b** to **b** (Red line) and output = 10.

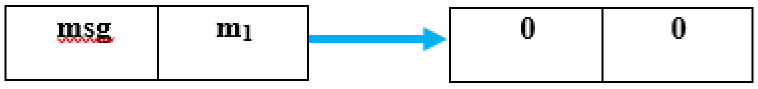
Step 3: Now we are at node **b**. Check that 3rd message bit = 1. See from node **b**. If the incoming bit is 1 state changes from **b** to **b** (Red line) and output = 10.

Step 4: Now we are at node **b**. Check that 4th message bit = 0. See from node **b**. If the incoming bit is 0 state changes from **b** to **a** (Black line) and output = 01.

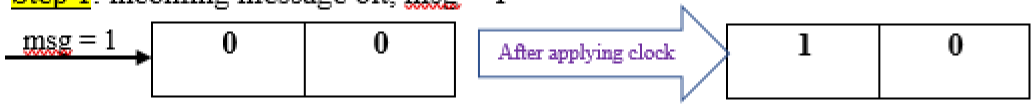
Step 5: Now we are at node **a**. Check that 5th message bit = 1. See from node **a**. If the incoming bit is 1 state changes from node **a** to **b** (Red line) and output = 11.

Time-domain approach

Initially, shift register is at all-zero state. That the contents of 2 flip-flops are 0.

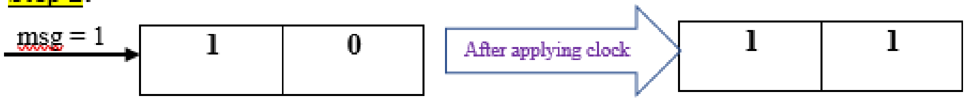


Step 1: incoming message bit, msg = 1



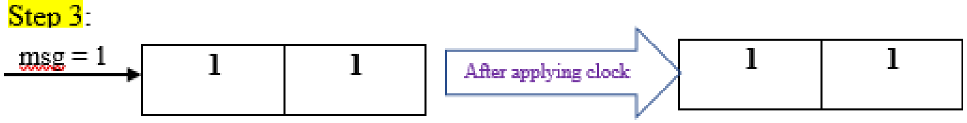
$$x_1 = msg = 1$$
$$x_2 = msg \oplus m_1$$
$$= 1 \oplus 0$$
$$= 1$$
$$\therefore\ x_1\, x_2 = 11$$

Step 2:



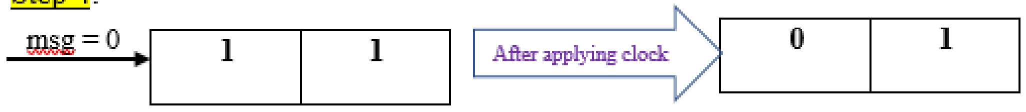
$$x_1 = msg = 1$$
$$x_2 = msg \oplus m_1$$
$$= 1 \oplus 1$$
$$= 0$$
$$\therefore\ x_1\, x_2 = 10$$

Step 3:



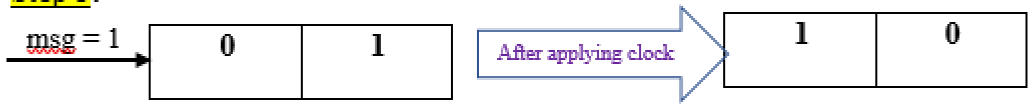
$$x_1 = msg = 1$$
$$x_2 = msg \oplus m_1$$
$$= 1 \oplus 1$$
$$= 0$$
$$\therefore\ x_1\, x_2 = 10$$

Step 4:



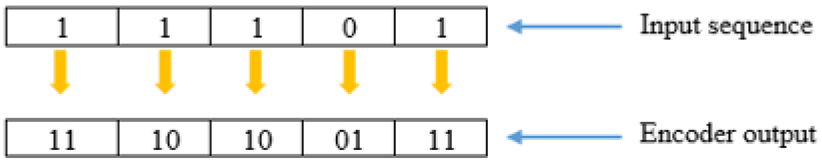
$$\begin{aligned} x_1 &= msg = 0 \\ x_2 &= msg \oplus m_1 \\ &= 0 \oplus 1 \\ &= 1 \\ \therefore\ x_1\,x_2 &= 01 \end{aligned}$$

Step 5:



$$\begin{aligned} x_1 &= msg = 1 \\ x_2 &= msg \oplus m_1 \\ &= 1 \oplus 0 \\ &= 1 \\ \therefore\ x_1\,x_2 &= 11 \end{aligned}$$

$$\therefore\ Encoded\ output\ or\ codeword\ is\ as\ follws:$$



Transform Domain Approach

$$m(D) = 1.D^0 + 1.D^1 + 1.D^2 + 0.D^3 + 1.D^4$$

$$= 1 + D + D^2 + D^4$$

For output 1: Let generator sequence $g^{(1)} = \left(g_0^{(1)}, g_1^{(1)}, g_2^{(1)}, \dots \dots \dots .. g_M^{(1)}\right)$

Where M = state of the shift register

and $g_0^{(1)}, g_1^{(1)}, g_2^{(1)} \dots \dots .\ are\ coefficients\ of\ polynomial\ g^{(1)}, which\ may\ be\ 1\ or\ 0$

$$g^{(1)} = \left(g_0^{(1)}, g_1^{(1)}\right) = (1, 0)$$

$$g^{(2)} = \left(g_0^{(2)}, g_1^{(2)}\right) = (1, 1)$$

Express generator sequence in polynomial form:

$$g^{(1)}(D) = g_0^{(1)}.D^0 + g_1^{(1)}.D^1 + g_2^{(1)}.D^2 + \cdots \dots .. + g_M^{(1)} D^M$$

$$\therefore\ g^{(1)}(D) = 1.D^0 + 0.D^1 = 1$$

For output 2: Let generator sequence $g^{(2)} = \left(g_0^{(2)}, g_1^{(2)}, g_2^{(2)}, \ldots\ldots\ldots\ldots g_M^{(2)}\right)$

Express generator sequence in polynomial form:

$$g^{(2)}(D) = g_0^{(2)}.D^0 + g_1^{(2)}.D^1 + g_2^{(2)}.D^2 + \cdots \ldots\ldots + g_M^{(2)} D^M$$
$$g_0^{(1)}, g_1^{(1)}, g_2^{(1)} \ldots\ldots.\, are\ coefficients\ of\ polynomial\ g^{(2)}, which\ may\ be\ 1\ or\ 0$$

$\therefore\ g^{(2)}(D) = 1.D^0 + 1.D^1 = 1 + D$

Note: the complete convolutional encoder is described by the set of generator polynomials

$$\left(g^{(1)}(D),\ g^{(2)}(D),\ g^{(3)}(D), \ldots\ldots\ldots.., g^{(M)}(D)\right)$$

Where D = unit delay variable

Note: the variable D is commonly used for convolutional codes and X for cyclic codes (traditional convention)

$x^1(D) = m(D).g^{(1)}(D)$

$$= (1 + D + D^2 + D^4)(1)$$
$$= 1 + D + D^2 + D^4$$
$$= 1.D^0 + 1.D^1 + 1.D^2 + 0.D^3 + 1.D^4$$

$\Longrightarrow$ (1 1 1 0 1)

$x^2(D) = m(D).g^{(2)}(D)$

$$= (1 + D + D^2 + D^4)(1 + D)$$
$$= 1 + D + D^2 + D^4 + D + D^2 + D^3 + D^5$$
$$= 1 + D(1 + 1) + D^2(1 + 1) + D^3 + D^5$$
$$= 1 + D(0) + D^2(0) + D^3 + D^5$$
$$= 1 + D^3 + D^5$$
$$= 1.D^0 + 0.D^1 + 0.D^2 + 1.D^3 + 1.D^4 + 0.D^5$$

$\Longrightarrow$ (1 0 0 1 1 1)

Finally multiplexing the 2 output sequences, we get the encoded sequence.

$$\therefore Encoded\ output\ or\ codeword\ is\ as\ follws:$$

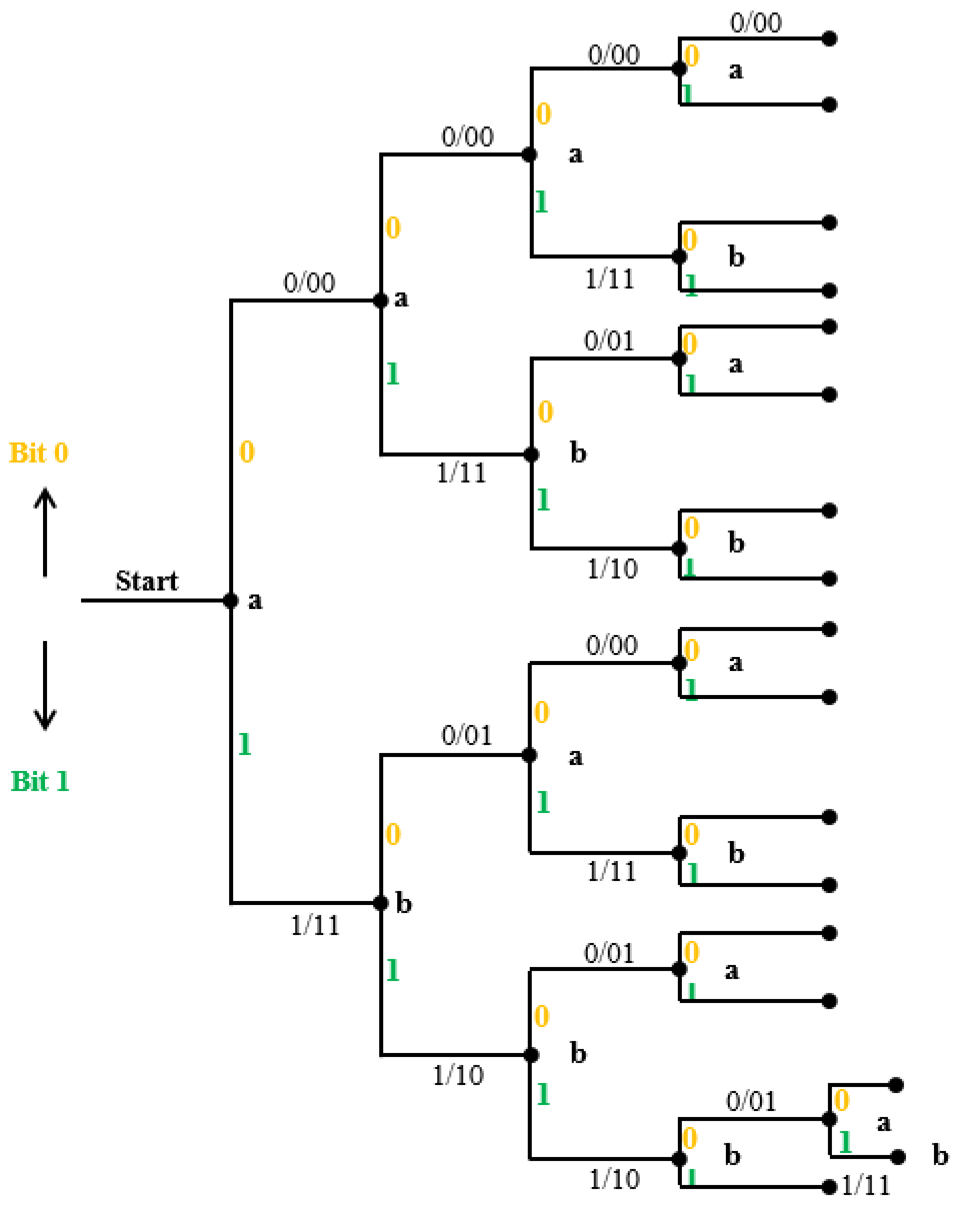
1	1	1	0	1	$\longleftarrow$ x^1
1	0	0	1	1	$\longleftarrow$ x^2
⇓	⇓	⇓	⇓	⇓	
11	10	10	01	11	$\longleftarrow$ **Encoder output**

This answer is same as we got from time-domain approach as well as from tree diagram

The message sequence of length m =5 bits produce an encoded sequence of length n(m+L-1) = 2(5+3-1) = 14 bits.

For the shift register to be restored to its zero-initial state, a terminating sequence of L=1 = 3-1=2 zeros is appended to the last input bit of the message sequence. The terminating zeros is called the TAIL OF THE MESSAGE.

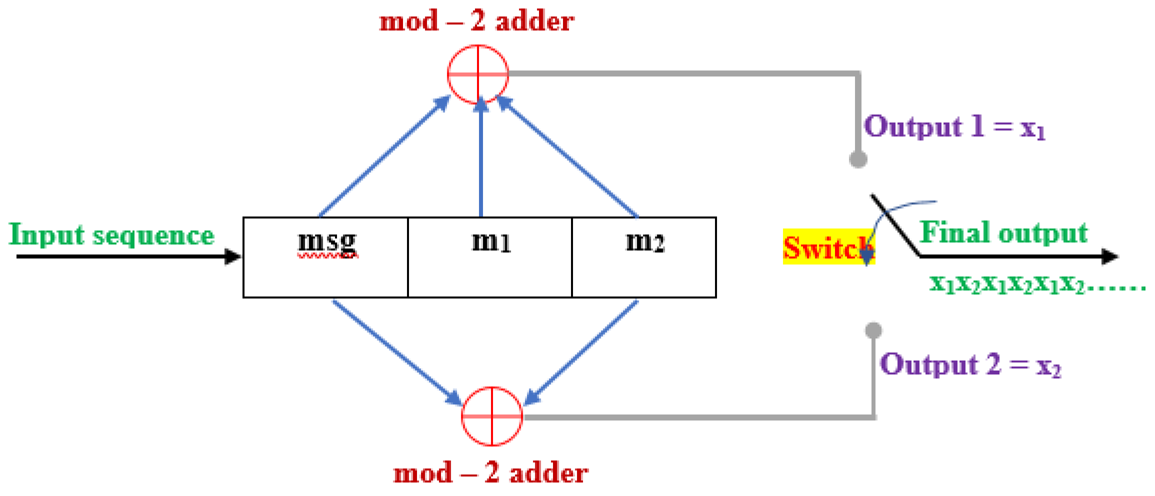
Code Tree diagram



State Table

Incoming message bit msg	State of encoder m_1	Current state	Next state	Encoded output x_1x_2
------	Initial state = 0	Initial all – zero state		
0	0	a	a	00
1	0	a	b	11
0	1	b	a	01
1	1	b	b	10

Q. Below figure depicts a rate ½, constraint length L = 2, convolutional encoder. Sketch the tree diagram, the trellis diagram and the state diagram.



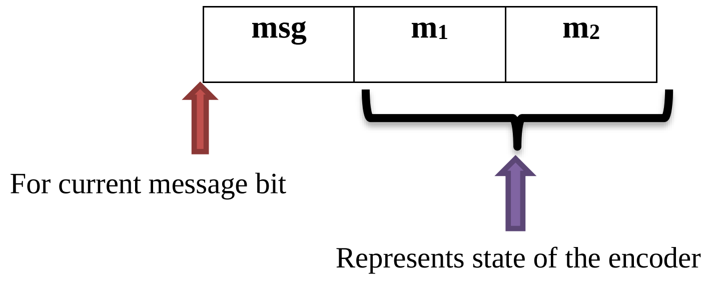
Sol:

$$Rate, r = \frac{1}{2} \; means \; 1 \; input \; bit \; produces \; 2 \; encoded \; output \; bits$$

$$Rate, r = \frac{1}{3} \; means \; 1 \; input \; bit \; produces \; 3 \; encoded \; output \; bits$$

$$Rate, r = \frac{2}{3} \; means \; group \; of \; 2 \; input \; bits \; produces \; 3 \; output \; bits$$

$$in \; general, code \; rate \; r = \frac{k}{n} \; means \; set \; of \; k \; input \; bits \; produces \; n \; output \; bits$$



Here, state of the encoder is represented by 2 – bit.

$$\therefore \; 2^2 = 4 \; states \; are \; possible$$

If 2 bits are used for state, the 2^2 = 4 states are possible
If 3 bits are used for state, the 2^3 = 8 states are possible
If 4 bits are used for state, the 2^4 = 16 states are possible

State table

Incoming message bit **msg**	State of encoder **m_1 m_2**	Current state	Next state	Encoded output x_1x_2
------	Initial state = 0	Initial all – zero state		
0	00	**a**		
1	00	**a**		
0	01	**b**		
1	01	**b**		
0	10	**c**		
1	10	**c**		
0	11	**d**		
1	11	**d**		

Let us use '**a**' to represent state **00**, '**b**' to state **01**, '**c**' to state **10** and 'd' to state **11** as shown below.

m_1m_2	State
00	**a**
01	**b**
10	**c**
11	**d**

State table

Incoming message bit **msg**	State of encoder **m_1 m_2**	Current state	Next state	Encoded output x_1x_2
------	Initial state = 0	Initial all – zero state		
0	00	**a**		**00**
1	00	**a**		**11**
0	01	**b**		**11**
1	01	**b**		**00**
0	10	**c**		**10**
1	10	**c**		**01**
0	11	**d**		**01**
1	11	**d**		**10**

Output of encoder is calculated using equations

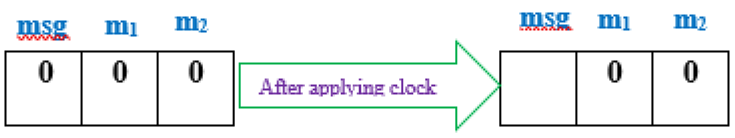
$$x_1 = msg \oplus m_1 \oplus m_2$$
$$x_2 = msg \oplus m_2$$

$\oplus$ *represents Ex − OR logic or mod − 2 addition*

Let us calculate next states of given convolutional encoder

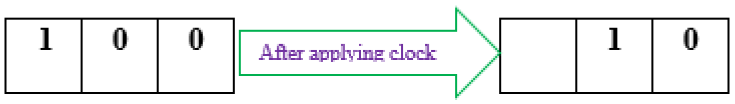
Incoming message bit msg	State of encoder m_1 m_2	Current state	Next state	Encoded output x_1x_2
------	Initial state = 0	Initial all – zero state		
0	00	**a**	**a**	**00**
1	00	**a**	**c**	**11**
0	01	**b**	**a**	**11**
1	01	**b**	**c**	**00**
0	10	**c**	**b**	**10**
1	10	**c**	**d**	**01**
0	11	**d**	**b**	**01**
1	11	**d**	**d**	**10**

Step 1:



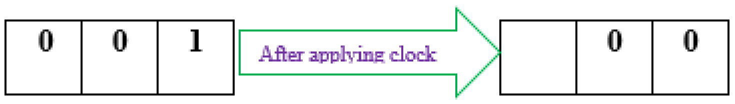
State **a**

Step 2:



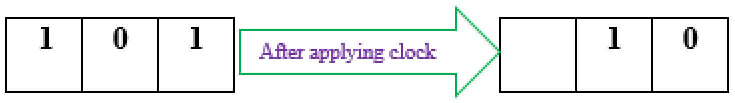
State **c**

Step 3:



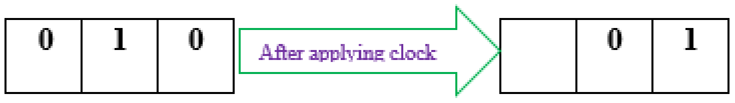
State **a**

Step 4:



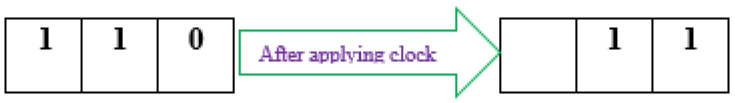
State **c**

Step 5:



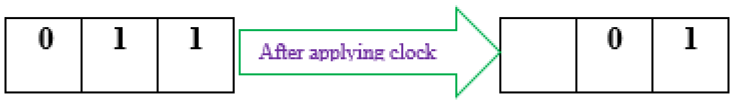
State **b**

Step 6:



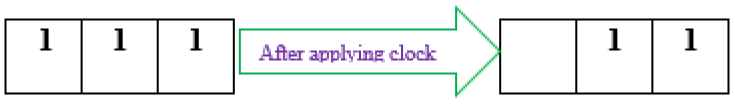
State **d**

Step 7:



State **b**

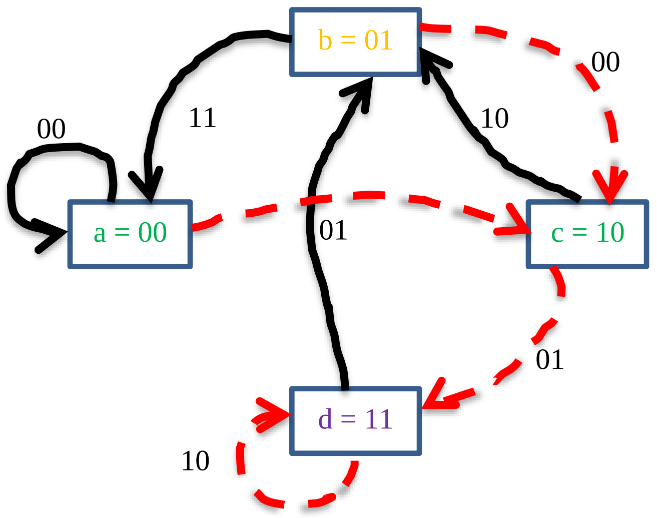
Step 8:



State **d**

State table is very useful to draw STATE diagram, code TRELLIS and code TREE diagrams. These diagrams are graphical representation of convolutional codes, from which we can calculate the output of convolutional encoder for any given input.

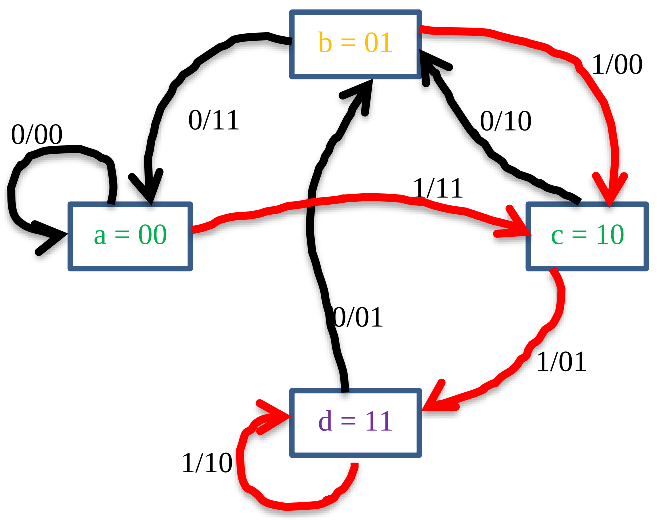
State diagram



Conventions used

Solid line → for bit 0
Dotted line → for bit 1

Another way of drawing state diagram



0/01 means **input/output** → for current input bit 0, the resulting output is 01. Avoids use of dashed line notation.

Finding encoded output from STATE diagram

Let's us assume input sequence m = 11101

Initially start from all-zero state i.e., a = 00

Step 1: Initially, we are at state **a**. Check incoming msg bit. msg = 1. See from node **a**. State changes from **a** to **c** (Red line) and output = 11.

Step 2: Now we are at node **c**. Check incoming msg bit. msg = 1 again. See from node **c**. State changes from **c** to **d** (Red line) and output = 01.

Step 3: Now we are at node **d**. Check incoming msg bit. msg = 1 again. See from node **d**. State changes from **d** to **d** (Red line) and output = 10.

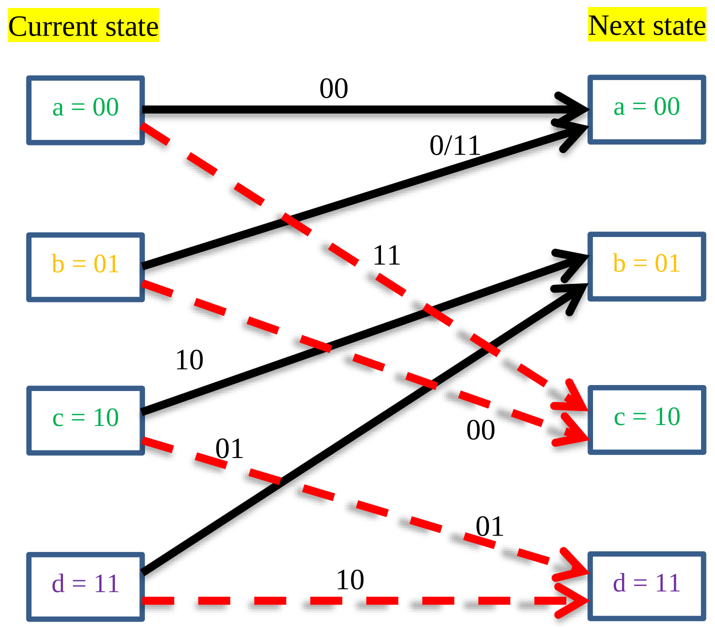
Step 4: Now we are at node **d**. Check incoming msg bit. msg = 0. See from node **d**. State changes from **d** to **b** (Black line) and output = 01.

Step 5: Now we are at node **b**. Check incoming msg bit. msg = 1. See from node **b**. State changes from **b** to **c** (Red line) and output = 01.

∴ *Encoded output or codeword is as follws:*

1	1	1	0	1	← Input sequence
⬇	⬇	⬇	⬇	⬇	
11	10	10	01	11	← Encoder output

Code TRELLIS

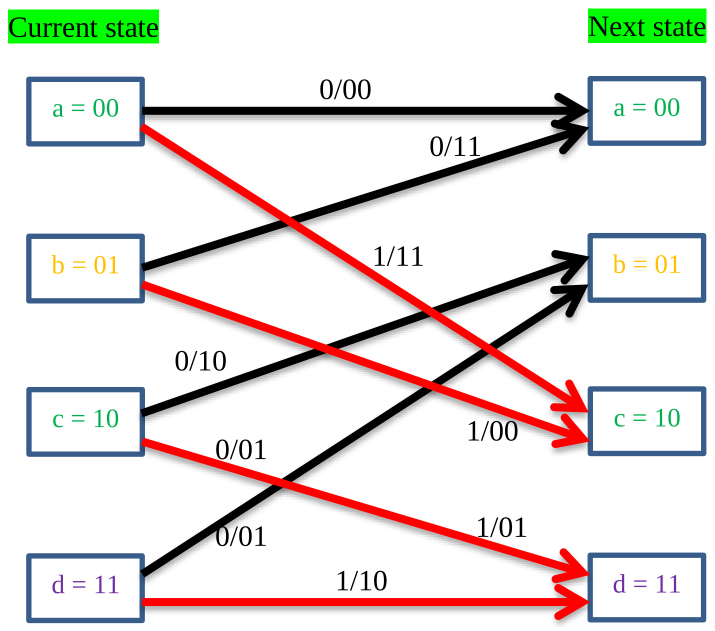


Conventions used:

Solid line	➡	for bit 0
Dotted line	➡	for bit 1

Other way of drawing TRELLIS diagram

0/01 means **input/output** ➡ for current input bit 0, the resulting output is 01. Avoids use of dashed line notation.



Finding encoded output from TRELLIS diagram

Let's us assume input sequence m = 11101

Initially start from all-zero state i.e., a = 0

Step 1: Initially, we are at node **a**. Check that 1st message bit = 1. See from node **a**. If the incoming bit is 1 state changes from **a** to **c** (Red line) and output = 11.

Step 2: Now we are at node **c**. Check that 2nd message bit = 1. See from node **c**. If the incoming bit is 1 state changes from **c** to **d** (Red line) and output = 01.

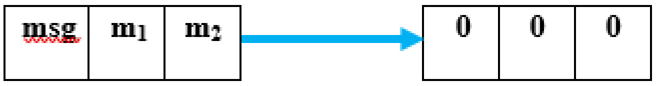
Step 3: Now we are at node **d**. Check that 3rd message bit = 1. See from node **d**. If the incoming bit is 1 state changes from **d** to **d** (Red line) and output = 10.

Step 4: Now we are at node **d**. Check that 4th message bit = 0. See from node **d**. If the incoming bit is 0 state changes from **d** to **b** (Black line) and output = 01.

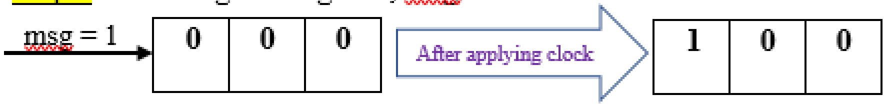
Step 5: Now we are at node **b**. Check that 5th message bit = 1. See from node **b**. If the incoming bit is 1 state changes from node **b** to **c** (Red line) and output = 00.

Time-domain approach

Initially, shift register is at all-zero state. That the contents of 2 flip-flops are 0.

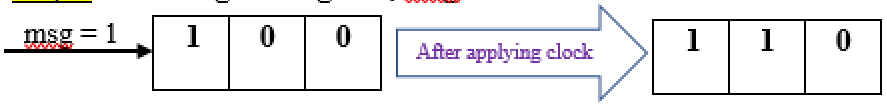


Step 1: incoming message bit, msg = 1



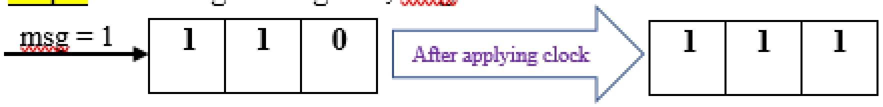
$$x_1 = msg \oplus m_1 \oplus m_2 = 1\oplus0\oplus0 = 1$$
$$x_2 = msg \oplus m_2 = 1\oplus0 = 1$$
$$\therefore\ x_1\, x_2 = 11$$

Step 2: incoming message bit, msg = 1



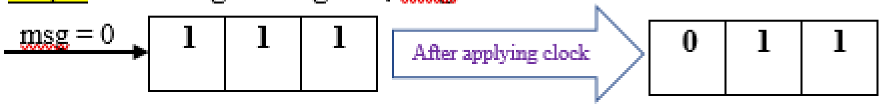
$$x_1 = msg \oplus m_1 \oplus m_2 = 1\oplus1\oplus0 = 0$$
$$x_2 = msg \oplus m_2 = 1\oplus0 = 1$$
$$\therefore\ x_1\, x_2 = 01$$

Step 3: incoming message bit, msg = 1



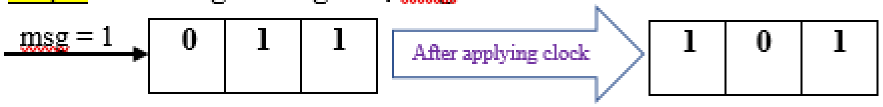
$$x_1 = msg \oplus m_1 \oplus m_2 = 1\oplus1\oplus1 = 1$$
$$x_2 = msg \oplus m_2 = 1\oplus1 = 0$$
$$\therefore\ x_1\, x_2 = 10$$

Step 4: incoming message bit, msg = 0



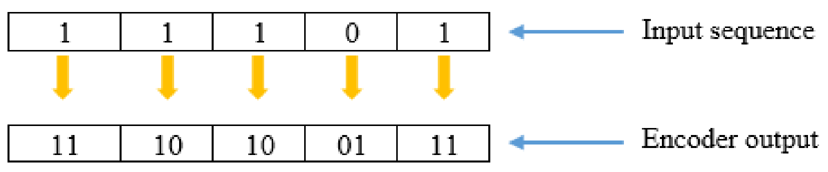
$$x_1 = msg \oplus m_1 \oplus m_2 = 0\oplus1\oplus1 = 0$$
$$x_2 = msg \oplus m_2 = 0\oplus1 = 1$$
$$\therefore\ x_1\, x_2 = 01$$

Step 5: incoming message bit, msg = 1



$$x_1 = msg \oplus m_1 \oplus m_2 = 1\oplus0\oplus1 = 0$$
$$x_2 = msg \oplus m_2 = 1\oplus1 = 0$$
$$\therefore\ x_1\, x_2 = 00$$

$$\therefore\ Encoded\ output\ or\ codeword\ is\ as\ follws:$$



Transform Domain Approach

$$m(D) = 1.D^0 + 1.D^1 + 1.D^2 + 0.D^3 + 1.D^4$$

$$= 1 + D + D^2 + D^4$$

For output 1: Let generator sequence $g^{(1)} = \left(g_0^{(1)}, g_1^{(1)}, g_2^{(1)}, \dots \dots \dots .. g_M^{(1)}\right)$

Where M = state of the shift register
and $g_0^{(1)}, g_1^{(1)}, g_2^{(1)} \dots \dots .\, are\ coefficients\ of\ polynomial\ g^{(1)}, which\ may\ be\ 1\ or\ 0$

$$g^{(1)} = \left(g_0^{(1)}, g_1^{(1)}, g_2^{(1)}\right) = (1, 1, 1)$$

$$g^{(2)} = \left(g_0^{(2)}, g_1^{(2)}, g_2^{(2)}\right) = (1, 0, 1)$$

Express generator sequence in polynomial form:

$$g^{(1)}(D) = g_0^{(1)}.D^0 + g_1^{(1)}.D^1 + g_2^{(1)}.D^2 + \cdots \dots .. + g_M^{(1)} D^M$$

$$\therefore g^{(1)}(D) = 1.D^0 + 1.D^1 + 1.D^2$$
$$= 1 + D + D^2$$

For output 2: Let generator sequence $g^{(2)} = \left(g_0^{(2)}, g_1^{(2)}, g_2^{(2)}, \dots \dots \dots .. g_M^{(2)}\right)$

Express generator sequence in polynomial form:

$$g^{(2)}(D) = g_0^{(2)}.D^0 + g_1^{(2)}.D^1 + g_2^{(2)}.D^2 + \cdots \dots .. + g_M^{(2)} D^M$$

$g_0^{(1)}, g_1^{(1)}, g_2^{(1)} \dots \dots .\, are\ coefficients\ of\ polynomial\ g^{(2)}, which\ may\ be\ 1\ or\ 0$

$$\therefore\ g^{(2)}(D) = 1.D^0 + 0.D^1 + 1.D^2$$
$$= 1 + D^2$$

Note: the complete convolutional encoder is described by the set of generator polynomials

$$\left(g^{(1)}(D),\ g^{(2)}(D),\ g^{(3)}(D), \dots \dots \dots .., g^{(M)}(D)\right)$$

Where D = unit delay variable

Note: the variable D is commonly used for convolutional codes and X for cyclic codes (traditional convention)

$x^1(D) = m(D).g^{(1)}(D)$

$$
\begin{aligned}
&= (1 + D + D^2 + D^4)(1 + D + D^2) \\
&= 1 + D + D^2 + D^4 + D + D^2 + D^3 + D^5 + D^2 + D^3 + D^4 + D^6 \\
&= 1 + D(1 + 1) + D^2(1 + 1 + 1) + D^3(1 + 1) + D^4(1 + 1) + D^5 + D^6 \\
&= 1 + D(0) + D^2 1 + D^3(0) + D^4(0) + D^5 + D^6 \\
&= 1 + D^2 + D^5 + D^6 \\
&= 1.D^0 + 0.D^1 + 1.D^2 + 0.D^3 + 0.D^4 + 1.D^5 + 1.D^6
\end{aligned}
$$

$\Longrightarrow$ (1 0 1 0 0 1 1)

$x^2(D) = m(D).g^{(2)}(D)$

$$
\begin{aligned}
&= (1 + D + D^2 + D^4)(1 + D^2) \\
&= 1 + D + D^2 + D^4 + D^2 + D^3 + D^4 + D^6 \\
&= 1 + D + D^2(1 + 1) + D^3 + D^4(1 + 1) + D^6 \\
&= 1 + D + D^2(0) + D^3 + D^4(0) + D^6 \\
&= 1 + D + D^3 + D^6 \\
&= 1.D^0 + 1.D^1 + 0.D^2 + 1.D^3 + 0.D^4 + 0.D^5 + 1.D^6
\end{aligned}
$$

$\Longrightarrow$ (1 1 0 1 0 0 1)

Finally multiplexing the 2 output sequences, we get the encoded sequence.

$$\therefore \textit{Encoded output or codeword is as follws}:$$

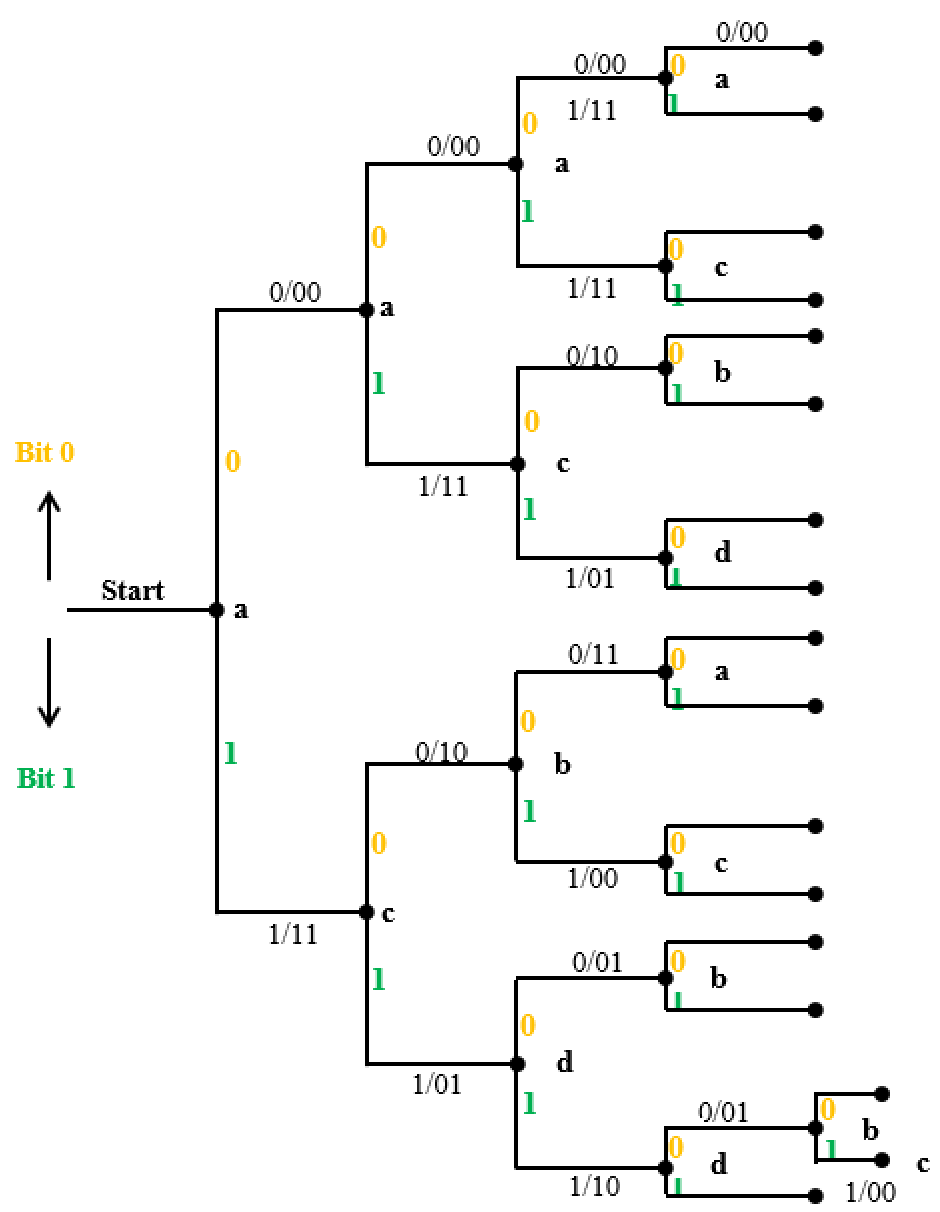
1	0	1	0	0	1	1	$\leftarrow$ x^1
1	1	0	1	0	0	1	$\leftarrow$ x^2
↓	↓	↓	↓	↓			
11	01	10	01	00	10	11	$\leftarrow$ **Encoder output**

This answer is same as we got from time-domain approach as well as from tree diagram

The message sequence of length m = 5 bits produce an encoded sequence of length n(m+L-1) = 2(5+3-1) = 14 bits.

For the shift register to be restored to its zero-initial state, a terminating sequence of L=1 = 3-1=2 zeros is appended to the last input bit of the message sequence. The terminating zeros is called the TAIL OF THE MESSAGE.

Code Tree



State Table

Incoming message bit msg	State of encoder m_1 m_2	Current state	Next state	Encoded output x_1x_2
------	Initial state = 0	Initial all – zero state		
0	00	**a**	**a**	**00**
1	00	**a**	**c**	**11**
0	01	**b**	**a**	**11**
1	01	**b**	**c**	**00**
0	10	**c**	**b**	**10**
1	10	**c**	**d**	**01**
0	11	**d**	**b**	**01**
1	11	**d**	**d**	**10**

Practical applications of coding

- Coding for deep space communications
- Coding for telephone – line modems
- Coding for compact disks

Convolutional codes can either detect or correct errors. Convolutional codes are more suitable for error correction.

Convolutional codes are well suited to space and satellite communication systems that require simple encoders and achieve high performance by sophisticated decoding methods.

In many applications where noise is predominantly Gaussian, the best solution is obtained when Block and Convolutional codes are used in series.

Today channel coding has been widely used in home entertainment systems (e.g. audio & DVD), computer storage systems (e.g. CDROM, hard disk, floppy disk and magnetic tape), computer communication, wireless communication, and deep space communication.

Convolutional codes are widely used in practical applications of communication system design. Viterbi decoding is predominantly used for short constraint lengths ($L \leq 10$), while sequential decoding is used for long constraint length codes, where the complexity of the Viterbi decoding becomes prohibitive.

Block codes (vs) Convolutional codes

Block codes	Convolutional codes
The block of n – bits depends only on the block of k – message bits	The block of n – bits not only depends on the k – message bits but also on the preceding L blocks of the message bits
More suitable for error detection	More suitable for error correction
Syndrome decoding is used	Viterbi decoding or sequential decoding is used
Encoder consists of a shift register and modulo – 2 adders	Encoder consists of shift register, mod – 2 adders and commutator switch
Graphical representation is not possible	Graphical representations such as code tree, code trellis, state diagram are possible.

Q. Consider 5-tupples 10011 and 11000. Find hamming distance.

	✓	✗	✓	✗	✗
Codeword 1	1	0	0	1	1
Codeword 2	1	1	0	0	0

Hamming distance = 3 as the given two codewords differ in 3 places (✗)

Q. the code set is given by: {000, 011, 101, 110}. Find minimum Hamming distance d_{min}.

Code set	000	011	101	110
Weight of codes	0	2	2	2
$\therefore d_{min} = 2$				

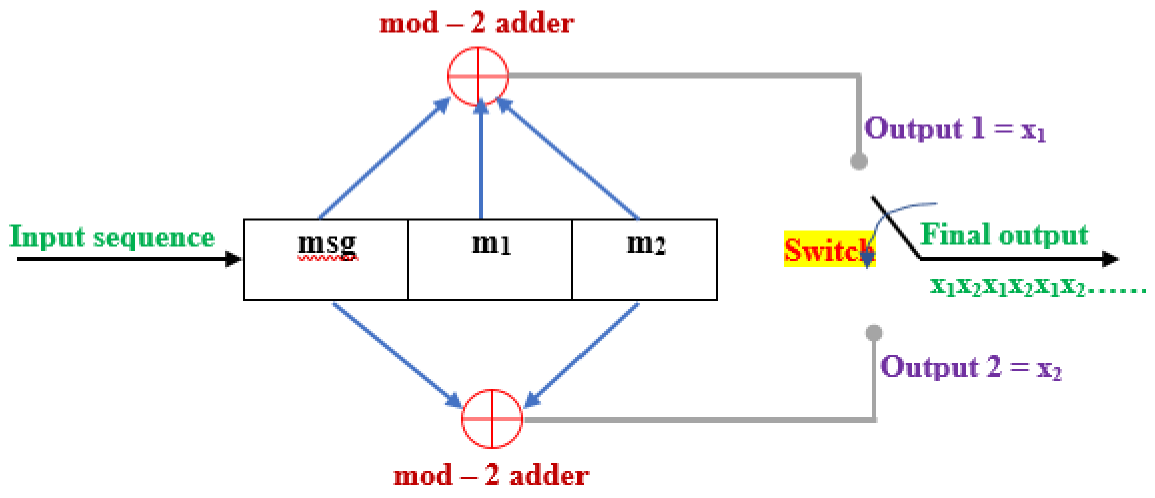
Transform domain approach

Convolution in time-domain = multiplication in spectral (frequency) domain. We can use this principle in the transform domain approach. The encoder output is obtained by the convolution of the input sequence with the impulse response of the encoder, hence the name convolutional code.

Two approaches can be used to find codewords in convolutional coding.

- Time domain approach
- D - Transform domain approach

Codewords obtained using the time domain approach and frequency domain approach produces same results.



Observe above convolutional code (2, 1, 3) diagram.

- 1st function generator is connected to stage 1, 2 & 3. Therefore $g^{(1)}$ = [111]
- 2nd generator is connected to stages 1 & 3. Therefore $g^{(2)}$ = [101]
- $g^{(1)}$ & $g^{(2)}$ are known as generator sequences of convolutional encoder. Note that generator sequences are also known as impulse responses of encoder.
- In $g^{(1)}$ = [101], 1 means there is a connection and 0 means corresponding flipflop is not connected.
- Impulse responses are also known as generator sequences of the convolutional code.
- Impulse responses $g^{(1)}$ and $g^{(2)}$ are called the generator sequences of the encoder.

Traditionally different variables are used for description of convolutional and cyclic codes, with D being commonly used for convolutional codes and X for cyclic codes.

Graphical representation of Convolutional codes

3 different but related graphical representations can be used to study of convolutional encoding.

- Code tree = Tree diagram
- Code trellis = Trellis diagram
- State diagram

Note that we can easily find output of the encoder from any of the above diagrams.
Given a sequence of message bits and the initial state, you can use any of following 3 diagrams to find the resulting output bits.

State diagram:

For convolutional encoders, it is sometimes useful to draw the state transition diagram. The nodes of the figure represent the 4 possible states of the encoder, with each node having 2 incoming branches and 2 outgoing branches.

- A transition from one state to another in response to input 0 is represented by a solid branch.
- A transition in response to 1 is represented by a dashed line.
- The binary label on each branch represents the encoder's output as it moves from one state to other.

With the help of state diagram, we can determine the output of the encoder for any incoming message sequence. We simply start at state 'a', the all zero initial state and walk through the state diagram in accordance with the message sequence. We follow a solid branch if the input is a zero and the dashed branch if it is a 1.

Because a convolutional encoder has finite memory, it can easily be represented by a state transition diagram. In this diagram, each state of the convolutional encoder is represented by a box and transitions between states are denoted by lines connecting these boxes. On each line both the input causing that transition and the corresponding output are specified.

The number of lines emerging from each state is equal to the number of possible inputs to the encoder at that state, which is equal to 2^k.

The number of lines merging at each state is equal to the number of states from which a transition is possible to this state. This again equal to 2^k.

Code Tree

The convention used to distinguish the input binary symbols is as follows:

Input 0 ---- specifies upper branch
Input 1 ---- specifies lower branch

Each branch of the tree represents an input symbol, with the corresponding pair of output binary symbols indicated on the branch.

It is often convenient to represent the codewords of a convolutional code as paths through a code tree. A convolutional code is sometimes called a (linear) tree code.

Code tree: the left most node is called the root. Since the encoder has 1 binary input, there are 2 branches stemming from each node. (starting at the root). The upper branch leaving each node corresponds to input 0 and the lower branch corresponds to the input digit 1. On each branch we have 2 binary code digits viz., the 2 outputs from the encoder.

Each branch of the tree represents an input symbol, with the corresponding pair of output binary symbols indicated on the branch.

Trellis diagram

A more popular and compact method to describe convolutional codes is to specify their trellis diagram. It is so called since trellis is a tree like structure with emerging branches.

Convention used:

- A code branch produced by an input 0 is drawn as a solid line
- A code branch produced by an input 1 is drawn as a dashed line