Communication Theory II

Lecture 10: Convolution, Trellis Codes

Convolutional coding

- Convolutional coding is an error-correcting technique used in digital communication systems.
- It adds redundancy to data to enhance error detection and correction during transmission.
- Widely applied in wireless communication, satellite links, and digital broadcasting.
- Convolutional coding is not based on blocks of bits but rather the output code bits are determined by logic operations on the present bit in a stream and a small number of previous bits

Convolutional coding

- Shift Registers: Convolutional encoders use shift registers to store recent input data bits.

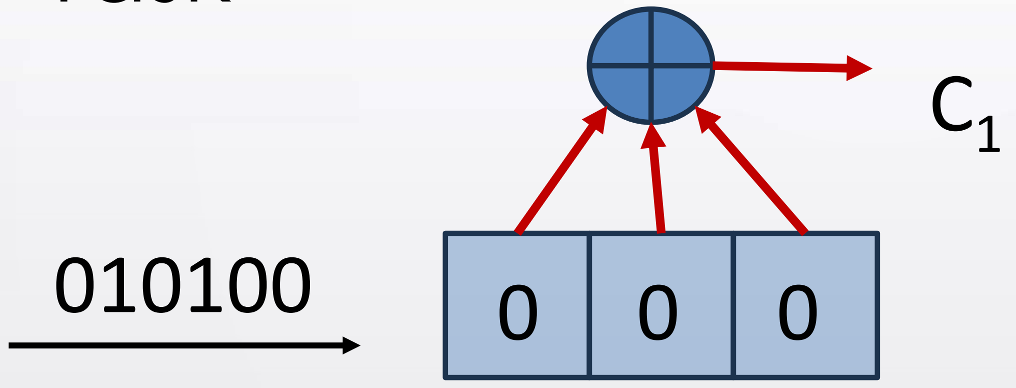
- Encoding Process: Binary operations like XOR are applied to shift register contents to generate new encoded bits.

- Code Rate: Code rate defines the ratio of output bits to input bits, e.g., 1/2, 2/3.

Convolutional coding

- Higher code rate implies more redundancy in the encoded data.
- Redundancy aids in detecting and correcting errors at the receiver.
- Convolutional coding enables error detection and correction.
- Short Constraint Length: Less error correction, lower complexity.
- Long Constraint Length: Stronger error correction, higher complexity.

Task



Each clock pulse signal shift to right

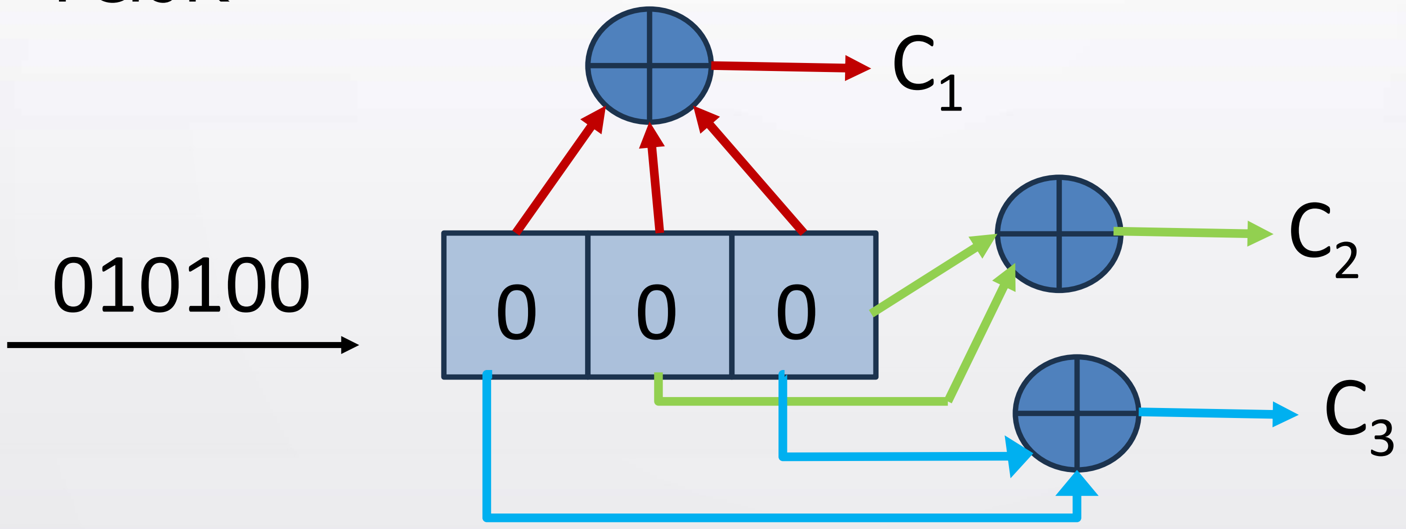
M2 addition

C_1 101100

Clock 1: 000	0
Clock 2: 000	0
Clock 3: 100	1
Clock 4: 010	1
Clock 5: 101	0
Clock 6: 010	1

Task

What is a Convolutional Code? - YouTube



C_1	001101
C_2	000111
C_3	001000

1 2 3	C_1	C_2	C_3
Clock 1: 000	0	0	0
Clock 2: 000	0	0	0
Clock 3: 100	1	0	1
Clock 4: 010	1	1	0
Clock 5: 101	0	1	0
Clock 6: 010	1	1	0

One input bit 3 coded bit

$C_3C_2C_1$

Code: 011 010 011 101 000 000

Input 010100

Constrained length convolutional code

- **Constraint Length (K):** The number of shift registers in the convolutional encoder.

A higher constraint length generally provides better error correction capabilities but increases complexity.

- **Code Rate (k/n):** ratio of output bits (k) to input bits (n).

Common code rates include 1/2, 2/3, 3/4

- **Design the Encoder:**

Encoder structure is determined by the constraint length and the code rate.

The encoder uses shift registers and feedback connections to generate the encoded output.

Constrained length convolutional code

- **State Transition Diagram:** State transition diagram that illustrates how the shift register contents change over time based on the input data.

- **Generator Polynomials:**

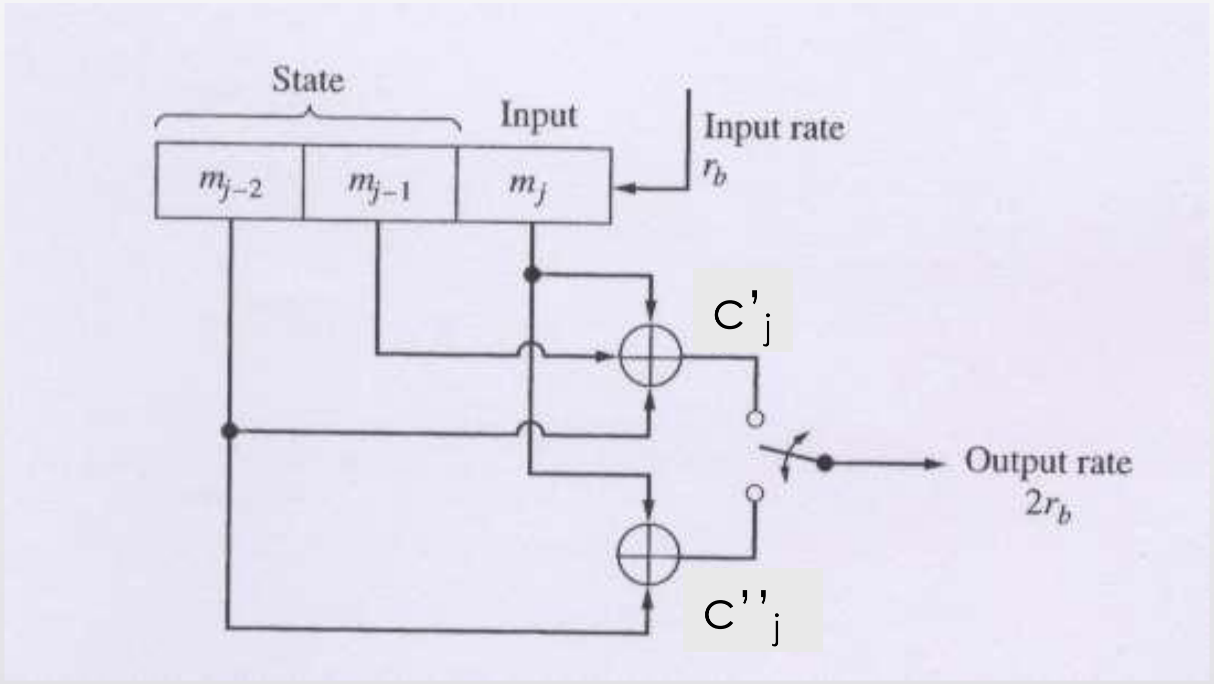
Generator polynomials determine the feedback connections and binary operations (usually XOR) used in the encoder.

The generator polynomials are usually expressed in terms of octal or hexadecimal numbers.

These polynomials are derived from the state transition diagram

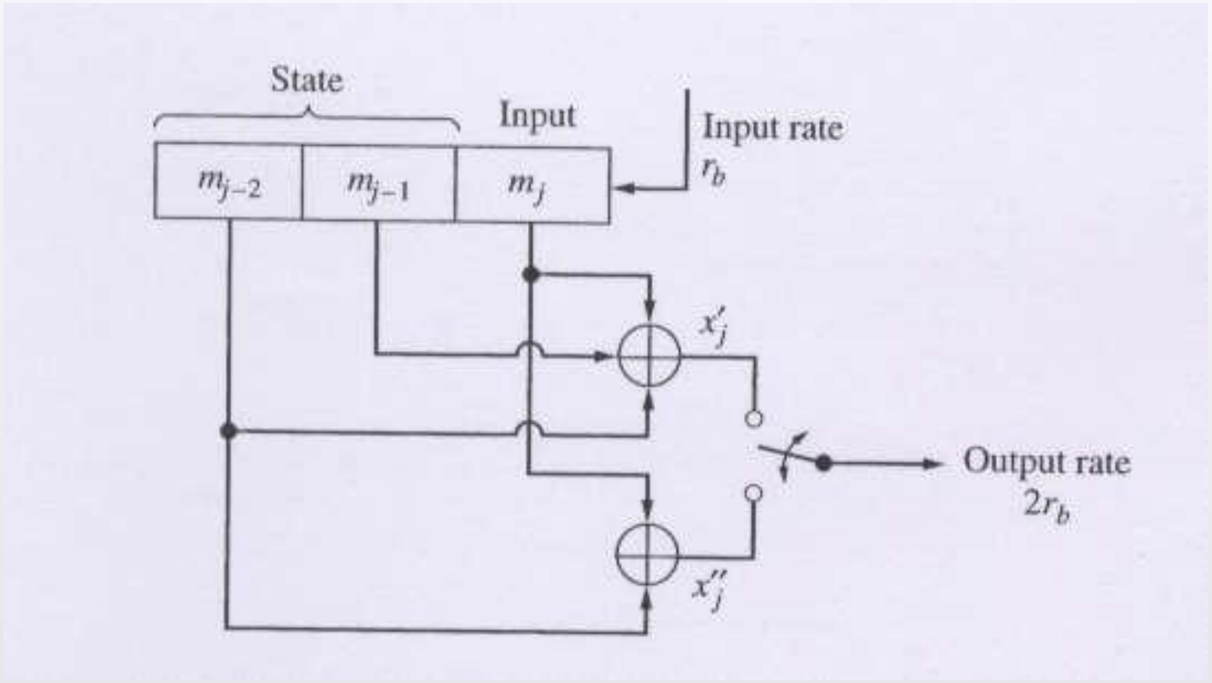
Convolutional Codes

To provide the extra check bits for error control, the encoded bits from multiple streams are interleaved.
For example, a convolutional encoder with n=2 (ie two streams of encoded bits)



Encoded bits from stream 1, $c'_j = m_{j-2} + m_{j-1} + m_j$
Encoded bits from stream 2, $c''_j = m_{j-2} + m_j$

Convolutional code: example



Input: 10100
Output?

Convolutional code

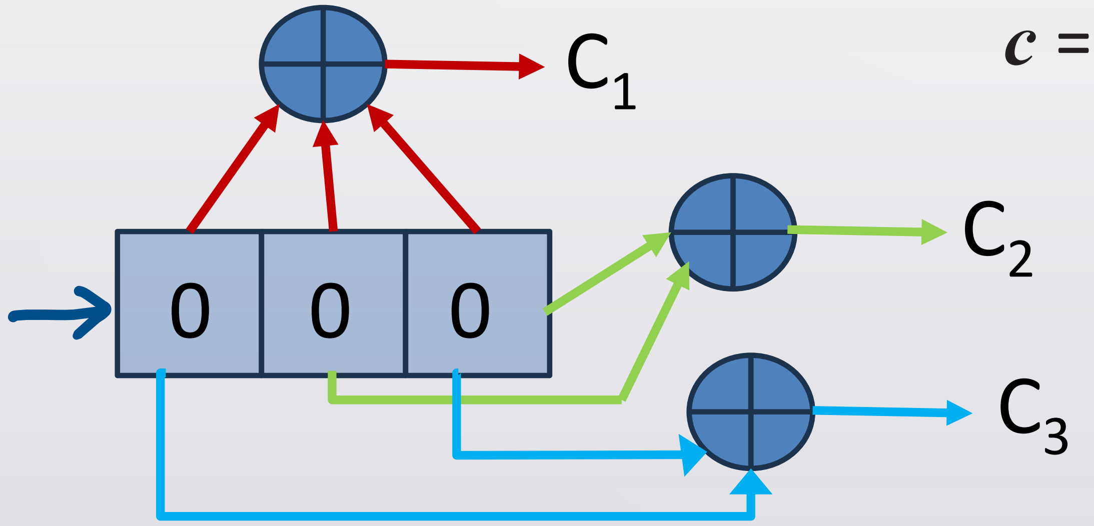
$g_1 = [111]$
$g_2 = [011]$
$g_3 = [101]$

impulse response

$c^{(1)} = u * g_1$
$c^{(2)} = u * g_2$
$c^{(3)} = u * g_3$

Code sequence is given by

$$c = c_1^{(1)}, c_1^{(2)}, c_1^{(3)}, c_2^{(1)}, c_2^{(2)}, c_2^{(3)}, \dots$$



convolutional operation is equivalent to multiplication in the transform domain

$g_1(D) = 1 + D + D^2$
$g_2(D) = D + D^2$
$g_3(D) = 1 + D^2$

$$u(D) = \sum_{i=0}^{\infty} u_i D^i$$

$$u(D) = u_0 + u_1 D + u_2 D^2 + \dots$$

$c^{(1)} = u(D)\, g_1(D)$
$c^{(2)} = u(D)\, g_2(D)$
$c^{(3)} = u(D)\, g_3(D)$

* Convolution operator

transform of the encoder output ***c*** is given by $c(D) = c^{(1)}(D^3) + Dc^{(2)}(D^3) + D^2c^{(3)}(D^3)$

Convolutional code

Generator Polynomials:

- In convolutional coding, each path that connects the output to the input of a convolutional encoder can be characterized in terms of its **impulse response**.

- The impulse response of a path refers to how that particular path responds when a symbol 1 is applied to its input, assuming that each flip-flop in the encoder starts in the zero state.

- These paths can also be characterized using **generator polynomials**. A generator polynomial for a specific path is a polynomial representation that encapsulates the transformation carried out by that path on the input data. Specifically, it is the unit-delay transform of the impulse response for that path.

  The generator polynomial for the i-th path is given by:

$$g^{(i)}(D) = g_0^{(i)} + g_1^{(i)} D + g_2^{(i)} D^2 + \ldots + g_{L-1}^{(i)} D^{L-1}$$

D represents the unit delay operator

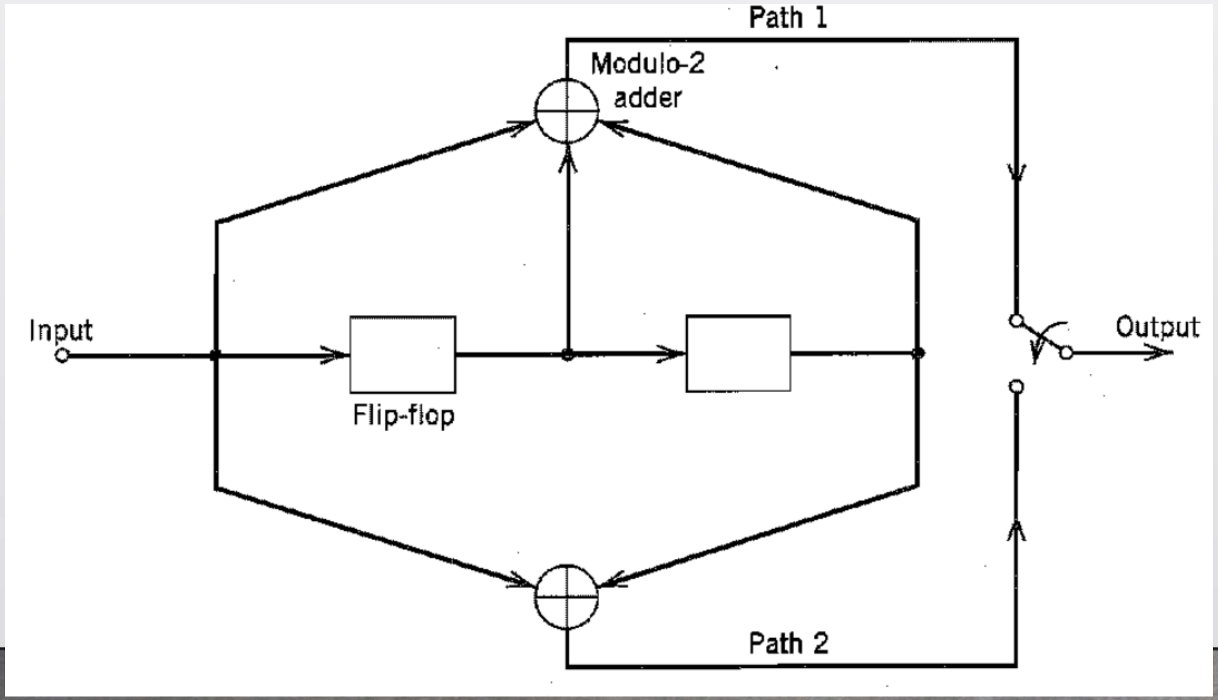
$g_0^{(i)}, \; g_1^{(i)}, \ldots, \; g_{L-1}^{(i)}$ *coefficients that can take binary values of 0 or 1.*

Convolutional code

Generator Polynomials:

The generator polynomial for the i-th path is given by:

$$g^{(i)}(D) = g_0^{(i)} + g_1^{(i)} D + g_2^{(i)} D^2 + \ldots + g_{L-1}^{(i)} D^{L-1}$$



Path 1: $1 + D + D^2$

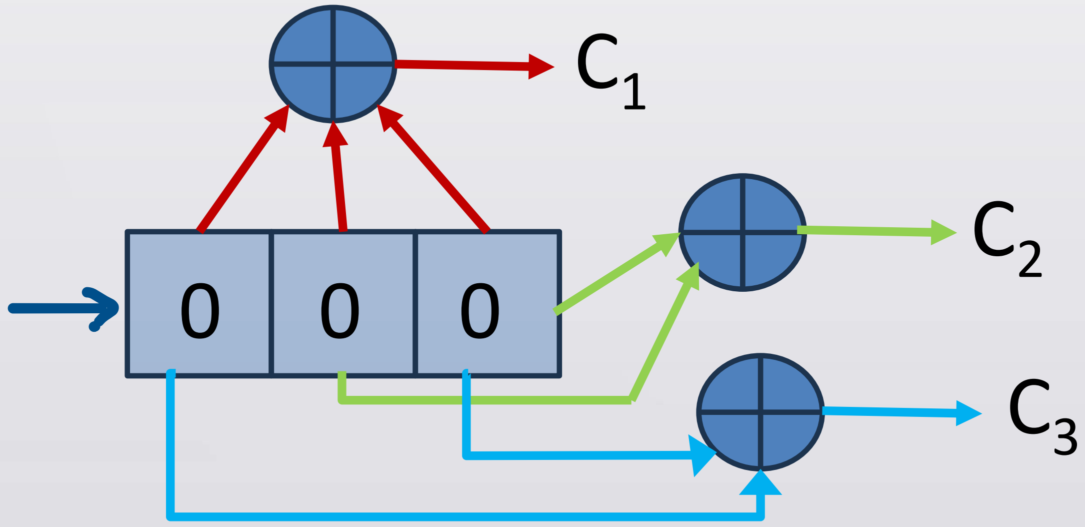
Path 2 : $1 + D^2$

Convolutional code

Generator Polynomials:

The generator polynomial for the i-th path is given by:

$$g^{(i)}(D) = g_0^{(i)} + g_1^{(i)} D + g_2^{(i)} D^2 + \ldots + g_{L-1}^{(i)} D^{L-1}$$



Path 1: $1 + D + D^2$

Path 2 : $D + D^2$

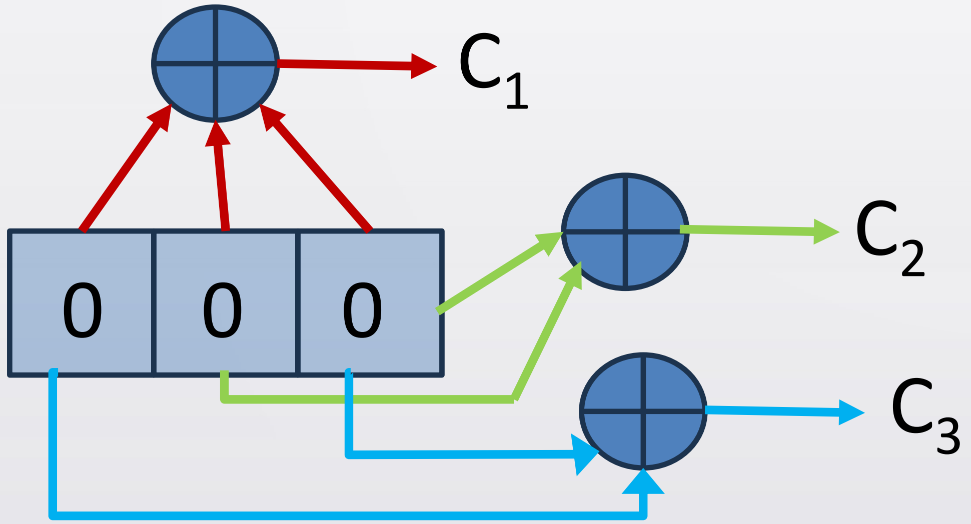
Path 3 : $1 + D^2$

$g_1 = [111]$
$g_2 = [011]$
$g_3 = [101]$

impulse response

Convolutional code

$\boldsymbol{u}$ = (111001)



$$u(D) = u_0 + u_1 D + u_2 D^2 + \dots$$

$$u(D) = 1 + D^3 + D^4 + D^5$$

$g_1(D)$, $g_2(D)$,$g_3(D)$?

$g_1(D)=1 + D + D^2$, $g_2(D)=D + D^2$, $g_3(D)= 1 + D^2$

$$c^{(1)}(D) = u(D)g_1(D)$$
$$c^{(2)}(D) = u(D)g_2(D)$$
$$c^{(3)}(D) = u(D)g_3(D)$$

$C^{(1)}(D)=(1+ D^{3+} D^4 + D^5) (1 + D + D^2)$

$= 1+ D + D^2 + D^3 + D^4 + D^5 + D^6 + D^7$

$C^{(2)}(D)= D+ D^2 + D^4 + D^5 + D^6 + D^7$

$C^{(3)}(D)= 1+ D^2 +D^3 + D^4 + D^5 + D^6 + D^7$

$$c(D) = c^{(1)}(D^3) + Dc^{(2)}(D^3) + D^2 c^{(3)}(D^3)$$

Convolutional code

$\boldsymbol{u} = (111001)$

$C^{(1)}(D) = 1 + D + D^2 + D^3 + D^4 + D^5 + D^6 + D^7$

$C^{(2)}(D) = D + D^2 + D^4 + D^5 + D^6 + D^7$

$C^{(3)}(D) = 1 + D^2 + D^3 + D^4 + D^5 + D^6 + D^7$

$$c(D) = c^{(1)}(D^3) + Dc^{(2)}(D^3) + D^2c^{(3)}(D^3)$$

$C^{(1)}(D^3) = 1 + D^3 + (D^3)^2 + (D^3)^3 + (D^3)^4 + (D^3)^5 + (D^3)^6 + (D^3)^7$

$C^{(1)}(D^3) = 1 + D^3 + D^6 + D^9 + D^{12} + D^{15} + D^{18} + D^{21}$

$C^{(2)}(D^3) = D^3 + D^6 + D^{12} + D^{15} + D^{18} + D^{21}$

$C^{(3)}(D^3) = 1 + D^6 + D^9 + D^{12} + D^{15} + D^{18} + D^{21}$

$C^{(1)}(D^3) = 1 + D^3 + D^6 + D^9 + D^{12} + D^{15} + D^{18} + D^{21}$

$DC^{(2)}(D^3) = D^4 + D^7 + D^{13} + D^{16} + D^{19} + D^{22}$

$D^2\,C^{(3)}(D^3) = D^2 + D^8 + D^{11} + D^{13} + D^{17} + D^{20} + D^{23}$

$$c(D) = c^{(1)}(D^3) + Dc^{(2)}(D^3) + D^2c^{(3)}(D^3)$$

$C(D) = 1 + D^3 + D^6 + D^9 + D^{12} + D^{15} + D^{18} + D^{21} + D^4 + D^7 + D^{13} + D^{16} + D^{19} + D^{22} + D^2 + D^8 + D^{11} + D^{13} + D^{17} + D^{20} + D^{23}$

$C(D) = 1 + D^2 + D^3 + D^{4\,+} D^6 + D^7 + D^8 + D^9 + D^{11} + D^{12} + D^{13} + D^{15} + D^{16} + D^{17} + D^{18} + D^{19}\, D^{20} + D^{21} + D^{22} + D^{23}$

Example

Consider the rate $r = 1/2$, constraint length $K = 2$ convolutional encoder of Fig. P10.16. The code is systematic. Find the encoder output produced by the message sequence 10111. . . .

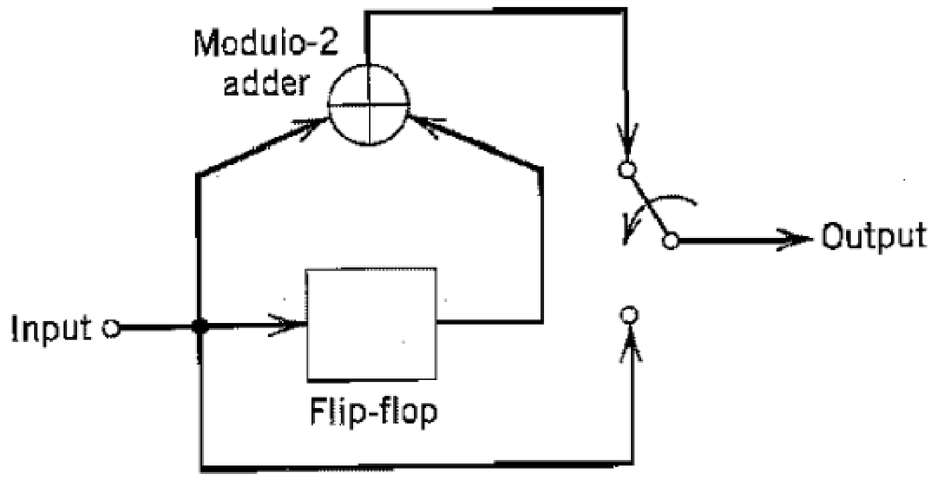


FIGURE P10.16

Message	1	0	1	1	1	1	...
Output	11	10	11	01	01	01	...

Example

Figure P10.17 shows the encoder for a rate $r = 1/2$, constraint length $K = 4$ convolutional code. Determine the encoder output produced by the message sequence 10111. . . .

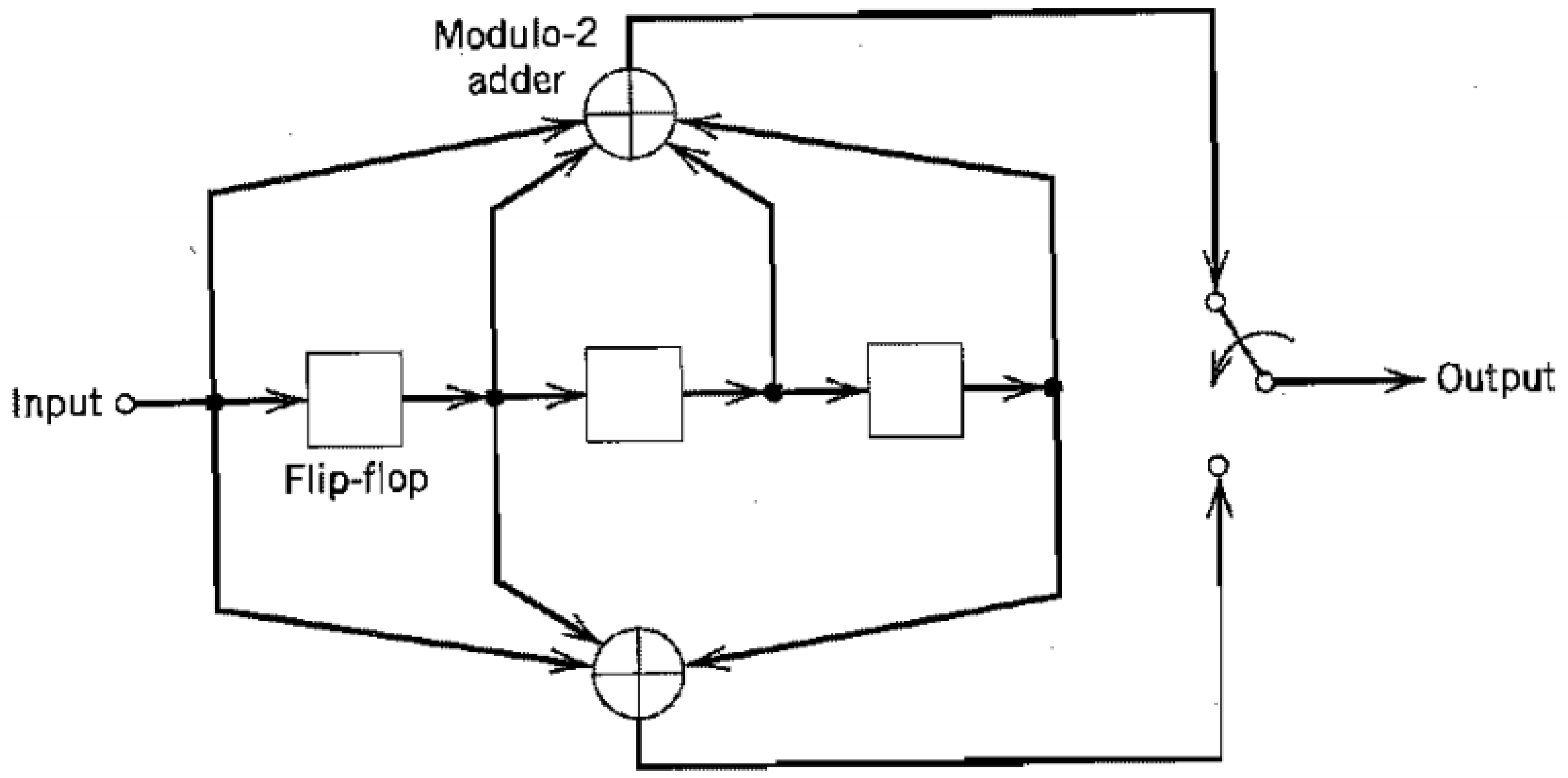


FIGURE P10.17

$$g^{(1)}(X) = 1 + X + X^2 + X^3$$

$$g^{(2)}(X) = 1 + X + \quad X^3$$

The message polynomial is

$$m(X) = 1 + X^2 + X^3 + X^4 + \ldots$$

$$c^{(1)}(X) = g^{(1)}(X)\, m(X)$$

$$= 1 + X + X^3 + X^4 + X^5 + \ldots.$$

$$c^{(2)}(X) = g^{(2)}(X)\, m(X)$$

$$= 1 + X + X^2 + X^3 + \quad X^6 + X^7 + \ldots$$

$$\{c^{(1)}\} = 1,1,0,1,1,1,\ldots$$

$$\{c^{(2)}\} = 1,1,1,1,0,0,\ldots$$

The encoder output is therefore 11, 11, 01, 11, 10, 10.

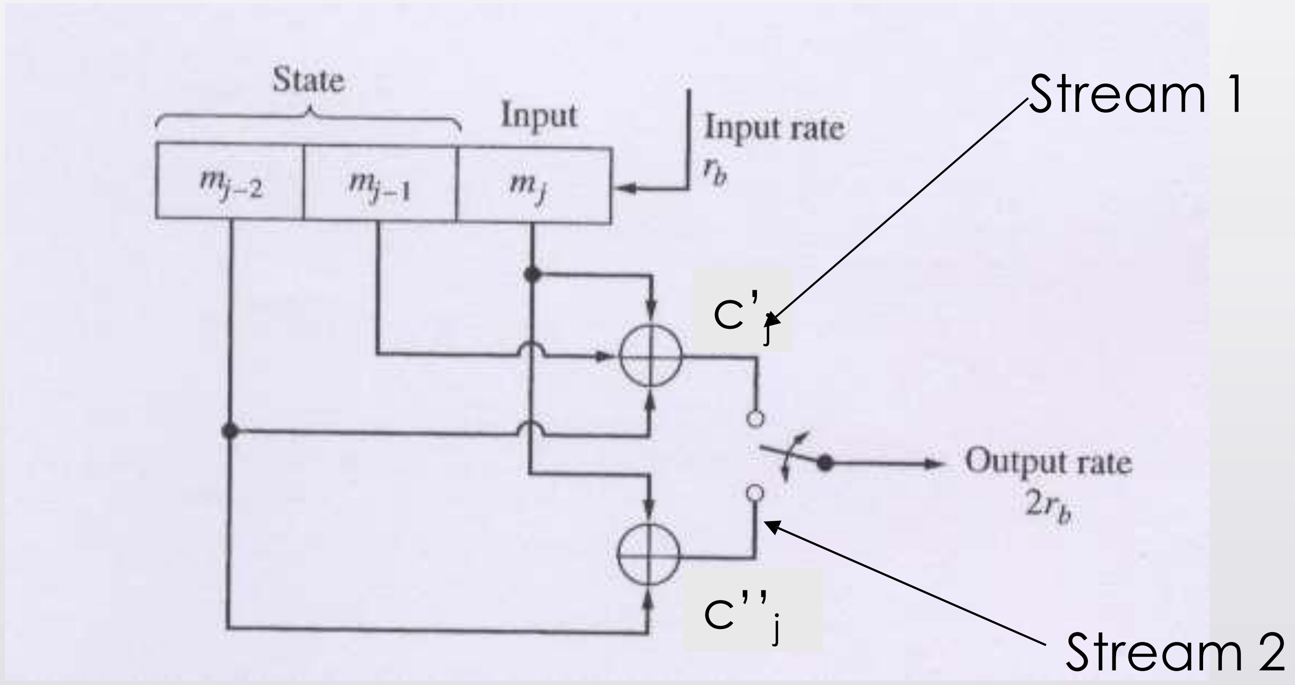
Tree diagram, the trellis diagram, and the state diagram

Three common approaches to explaining a convolutional code are using a tree diagram, a trellis diagram, and a state diagram

- **Tree Diagram**: Shows encoding paths in convolutional codes.
- **Trellis Diagram**: Displays code states for decoding decisions.
- **State Diagram**: Illustrates code encoder/decoder transitions.

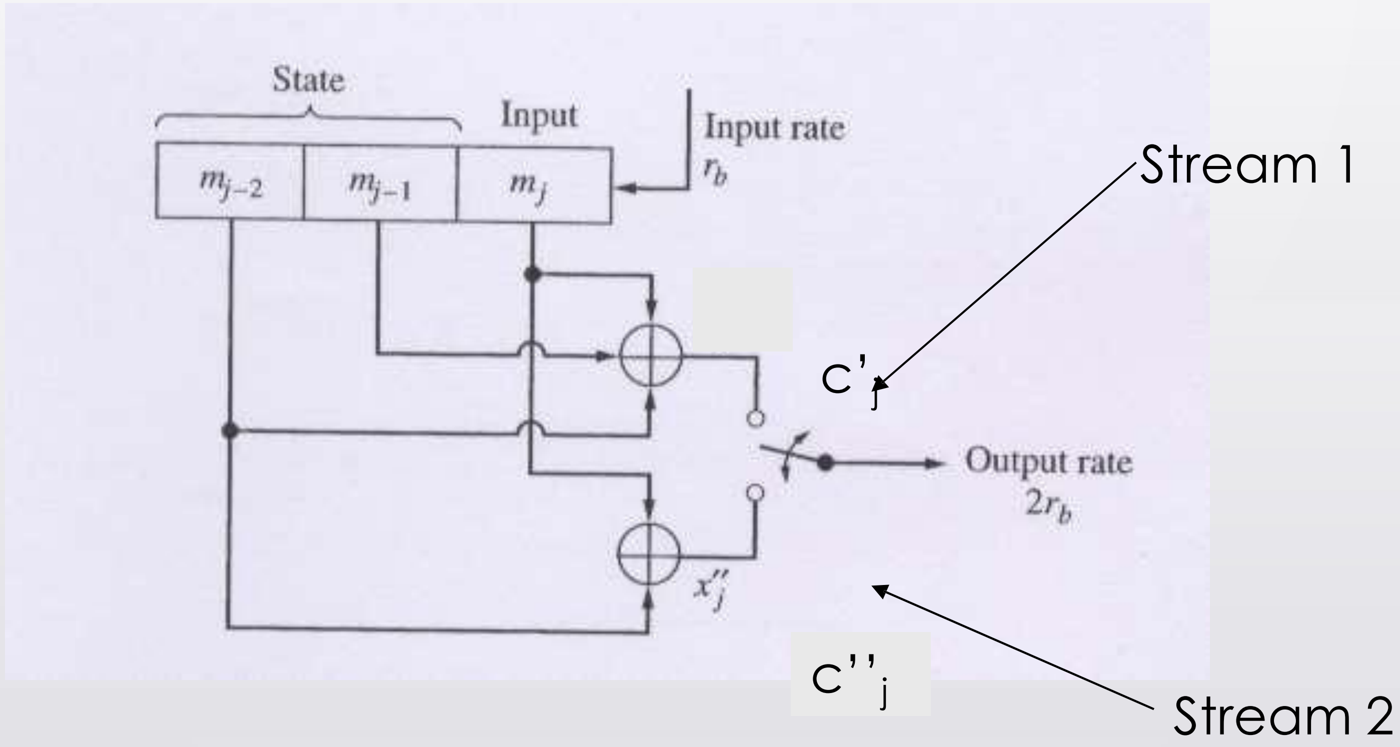
Convolutional Codes – Code Tree, Trellis and State Diagram

Draw a code tree for the convolutional encoder below.



The 3-bit output sequence for every input bit can be determined by both the input bit and the four potential states of the shift register, labeled as a = 00, b = 01, c = 10, and d = 11.

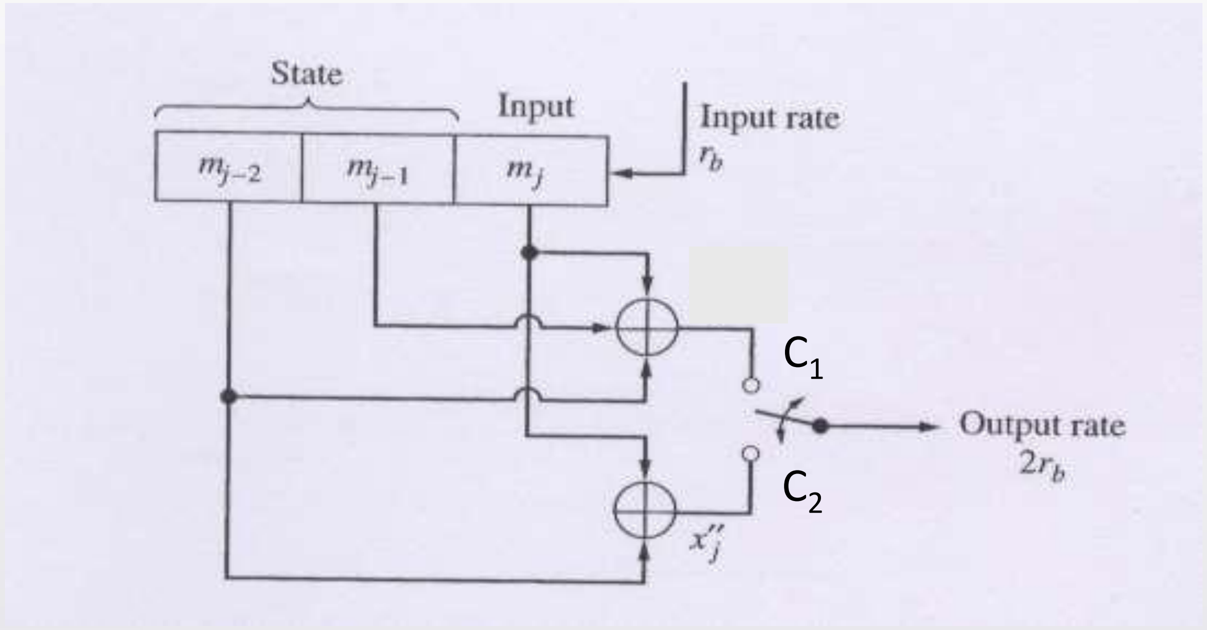
Convolutional Codes – Code Tree, Trellis and State Diagram



Write path 1 and path 2 output based on m_{j-2} , m_{j-1} and m_j

$c'_j = m_{j-2} + m_{j-1} + m_j$,
$c''_j = m_{j-2} + m_j$

Convolutional Codes – Code Tree, Trellis and State Diagram



Assuming flipflops are initialized as 000, What is the output for input bit 1

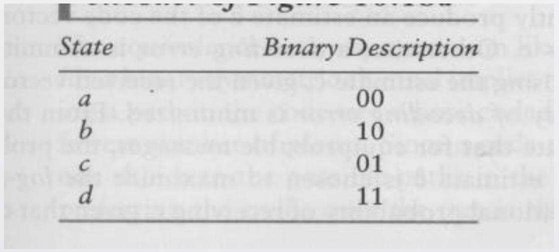
Assuming flipflops are initialized as 000, What is the output for input bit 0

$C_1 = m_{j-2} + m_{j-1} + m_j$,
$C_2 = m_{j-2} + m_j$

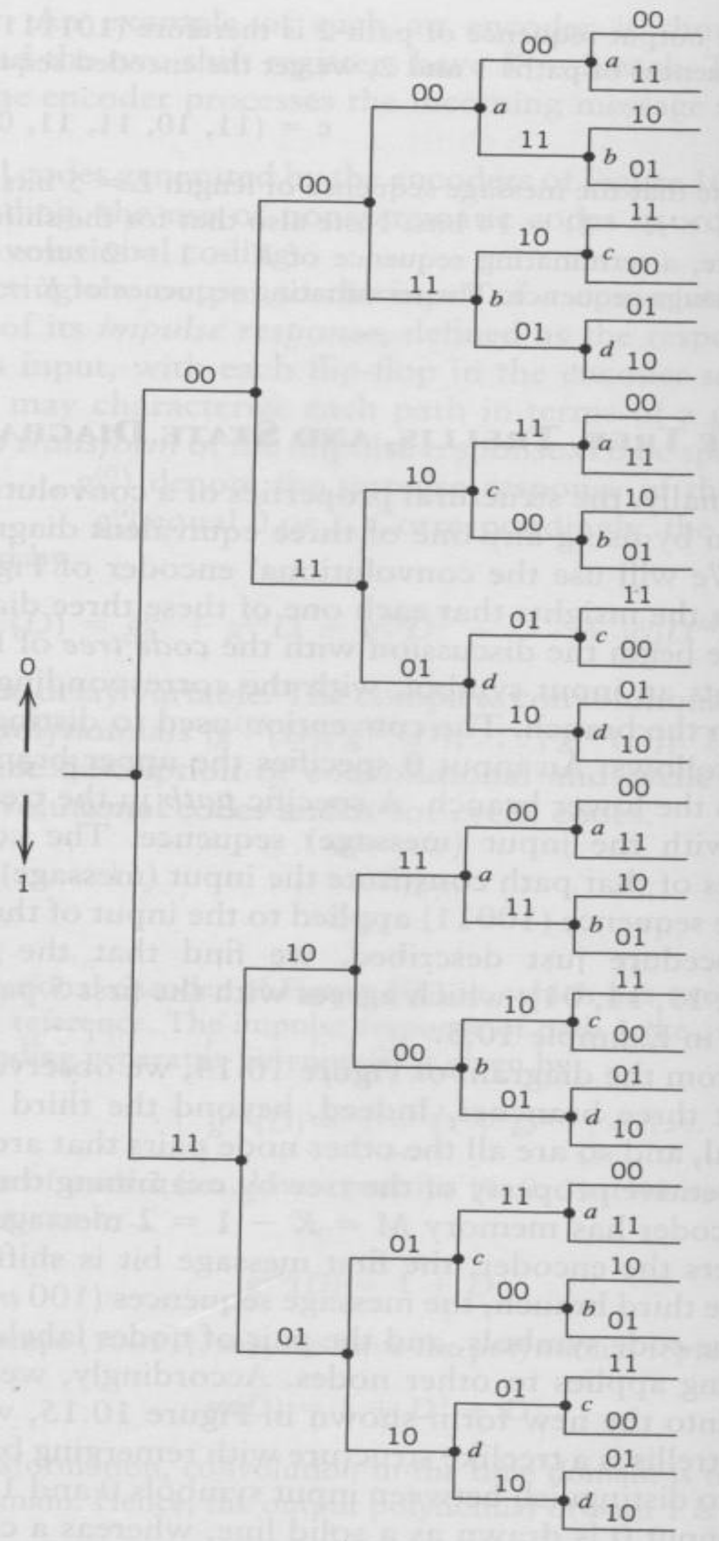
Assuming flipflops are initialized as 000, What is the output for input bit 0 after bit 1

Assuming flipflops are initialized as 000, What is the output for input bit 1 after bit 1

Code Tree

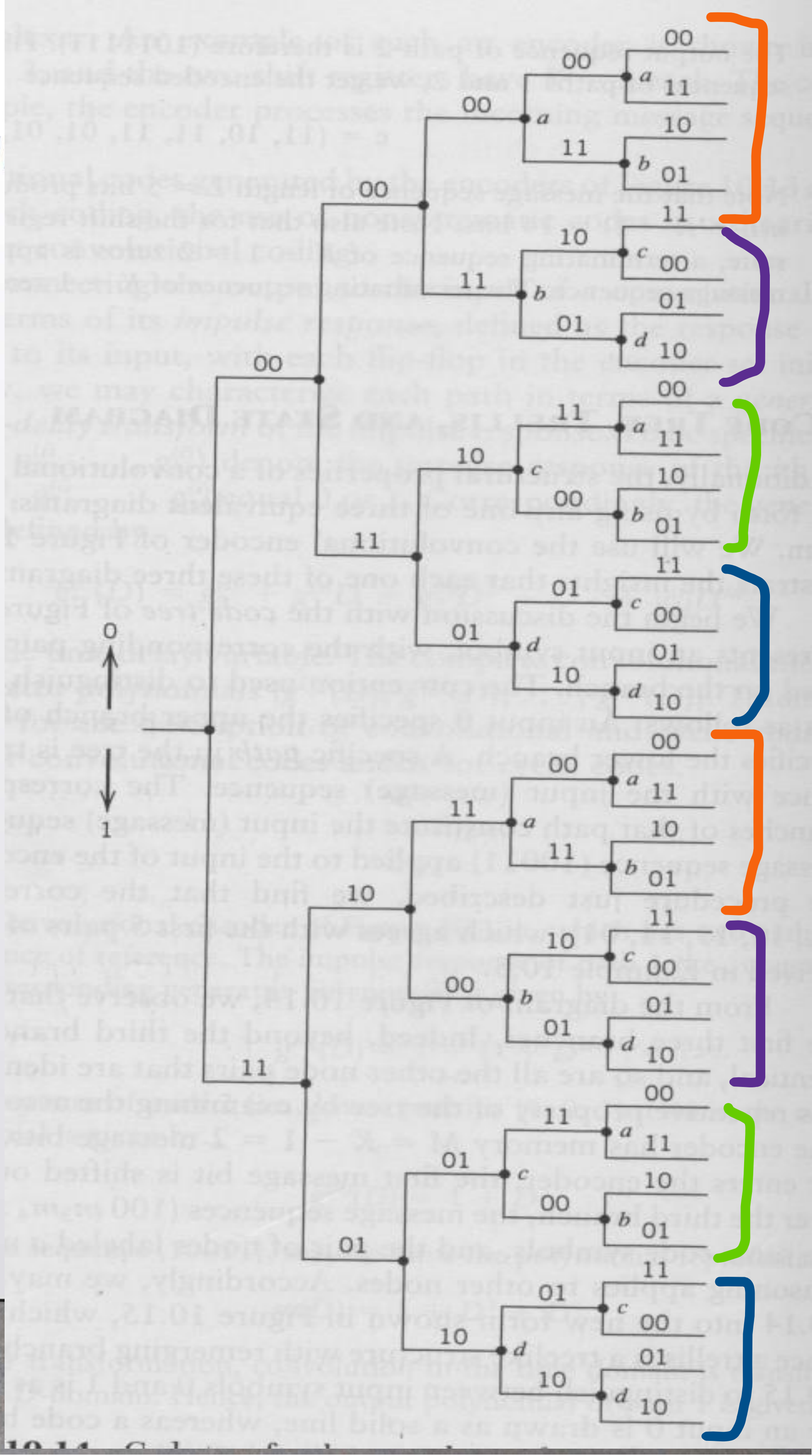


State	Binary Description
a	00
b	10
c	01
d	11

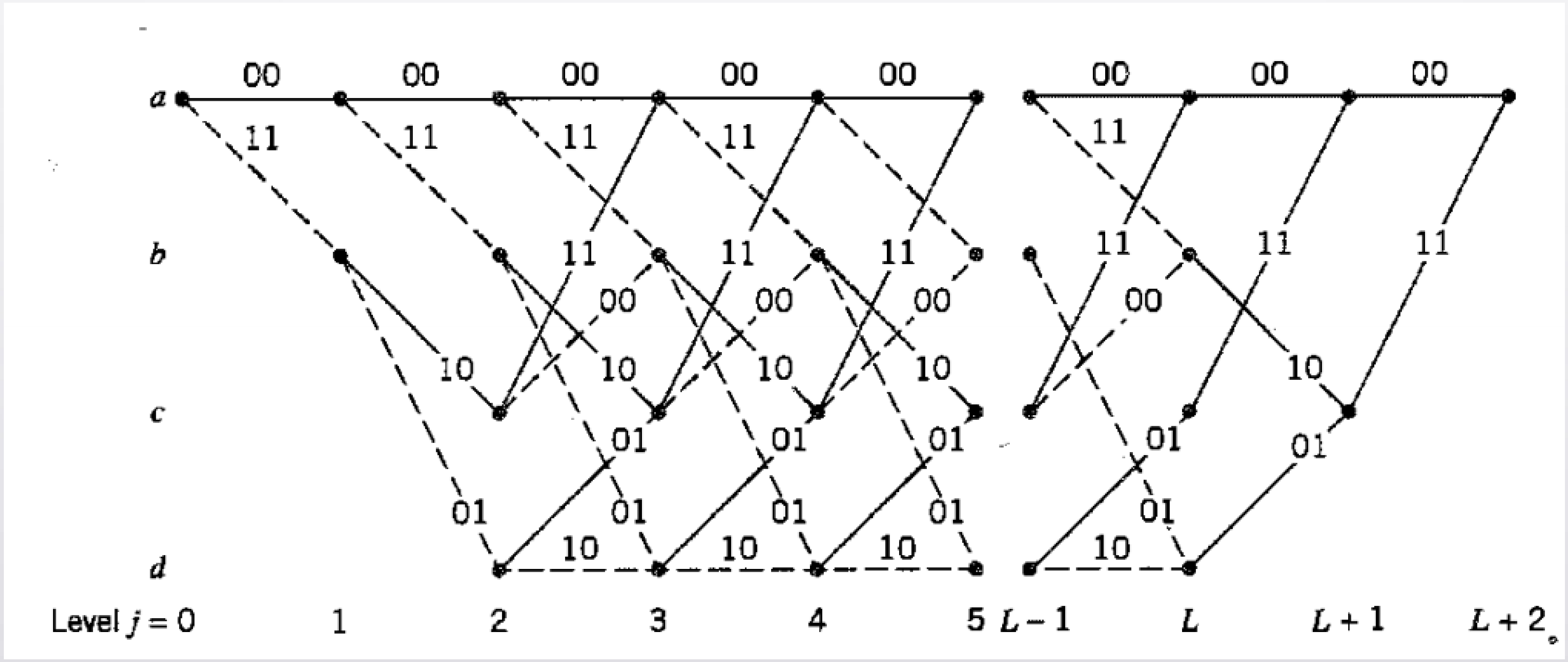


Code Tree

- Repetition in the tree occurs after the third branch.
- E.g., Nodes labeled "a" become identical beyond that point.
- After the third branch, (100...) and (000...) yield same codes. So thus, we can merge them.
- It is possible to collapse the tree.
- A trellis is tree-like with converging branches

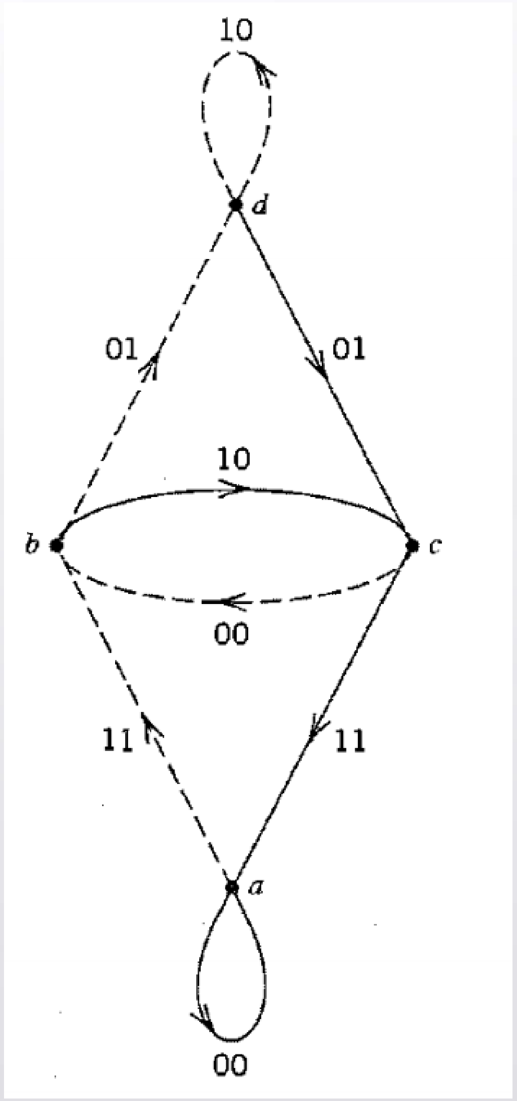


Trellis



State Diagram

- Solid branch for input 0, dashed for input 1.
- Binary label on a branch indicates encoder's output.
- State (01) → Node c, input 1 → (10), output (00).
- State diagram helps determine encoder output for any input.
- Begin at state a, follow solid for 0, dashed for 1.
- Output binary label on traversed branch.
- Message (10011) → Path abcabd → Output (11, 10, 11, 11, 01).


10
d
01
01
10
b
c
00
11
11
a
00