

CONVOLUTION CODES



INTRODUCTION TO COMMUNICATION SYSTEMS CT-216

Prof. YASH VASAVADA

GROUP - 2, SUBGROUP - 2

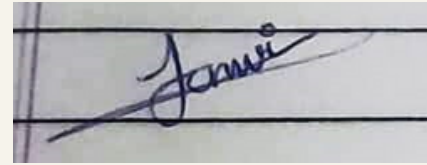
HONOR CODE

We declare that :

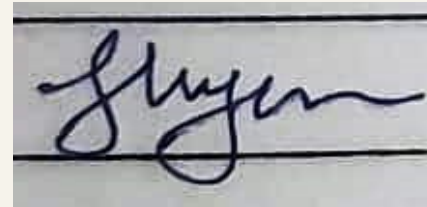
- The work that we are presenting is our own work.
- We have not copied the work (the code,the results,etc.) that someone else has done.
- Concepts,understanding and insights we will be describing are our own.
- We make this pledge truthfully. We know that violation of this solemn pledge can carry grave consequences.

GROUP MEMBER

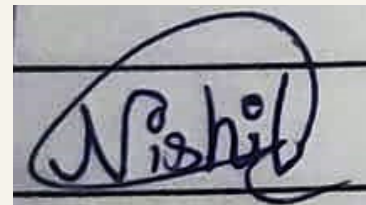
1) Janvi Ramani - 202201158



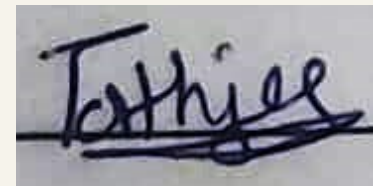
2) Shyam Ghetiya - 202201161



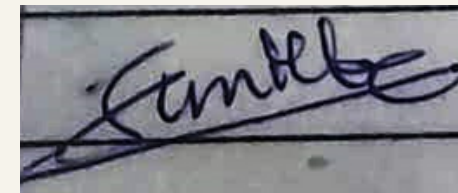
3) Nishil Patel - 202201166



4) Tathya Prajapati - 202201170



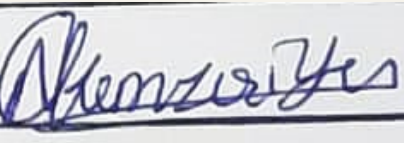
5) Sunil Rathva - 202201177



6) Vedant Savani - 202201178



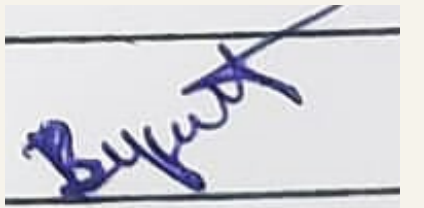
7) Nitin Kanzariya - 202201181



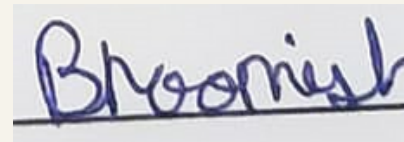
8) Akshat Joshi - 202201185



9) Bansi Patel - 202201190



10) Bhoomish Patel - 202201414



CONTENT

- 01** INTRODUCTION : WHAT IS CONVOLUTIONAL CODING ?
- 02** CONVOULTIONAL ENCODING
- 03** REPRESENTATION
- 04** TRANSFER FUNCTION
- 05** BPSK AND AWGN CHANNEL
- 06** HARD DECISION DECODING

- 07** SOFT DECISION DECODING
- 08** PSUEDO CODE
- 09** GRAPH , RESULTS , EXPLANATION
- 10** ADVANTAGES OF CONVOLUTIONAL CODING
- 11** REFERENCES
- 12** GROUP MEMBERS

INTRODUCTION

- Convolutional codes are introduced by Peter Elias in 1955 .
- There were two main categories of Error Correction (ECC) codes one is Block codes and other is Convolution codes.
- Convolutional codes doesn't have a finite block like Block codes because here we give input as stream not as a block.
- Convolution coding is a popular error-correcting coding method used to improve the reliability of communication system.

INTRODUCTION

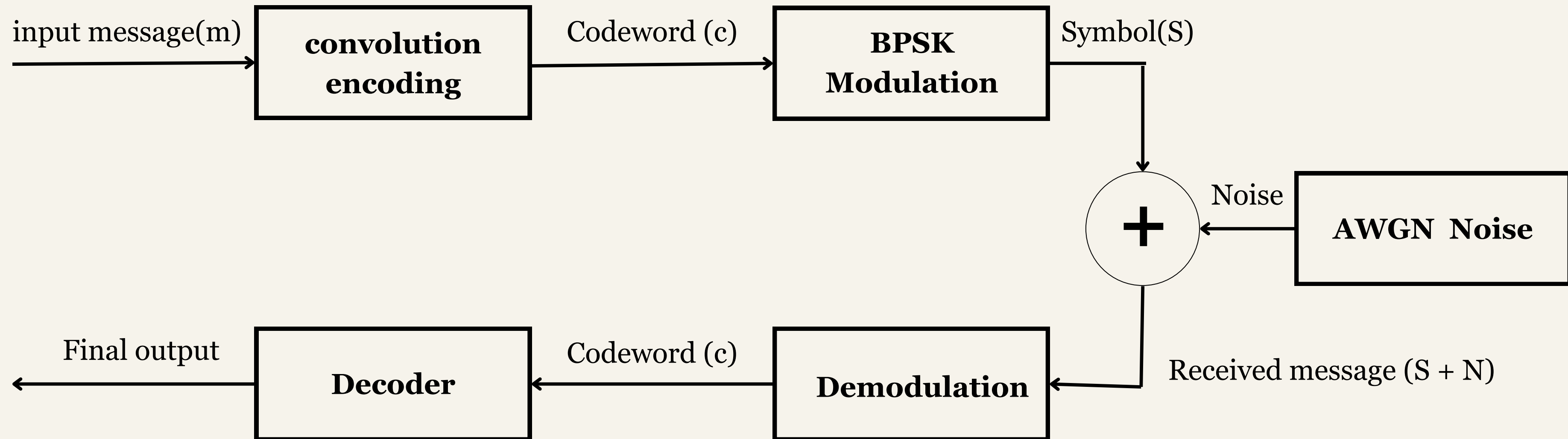
Convolution codes

- In Convolutional codes, information bits aren't followed by parity bit but are spread across the sequence.
- Convolutional codes use a few input bits to generate a few output bits for each interval (when $K=1$).
- The encoding of the current state is linked to the previous state and past elements, so it includes a memory element to store previous state information.

Linear Block codes

- In Linear Block codes, info bits are right away followed by the parity bits.
- Block codes transform a bunch of input bits into a bunch of output bits where both sets can be quite large, with k being the number of message bits.
- The encoding of the current state doesn't rely on the previous state, so it lacks any memory element. It only hinges on the current message bit.

FLOW OF BPSK & AWGN CHANNEL WITH CONVOLUTION ENCODING



ENCODING

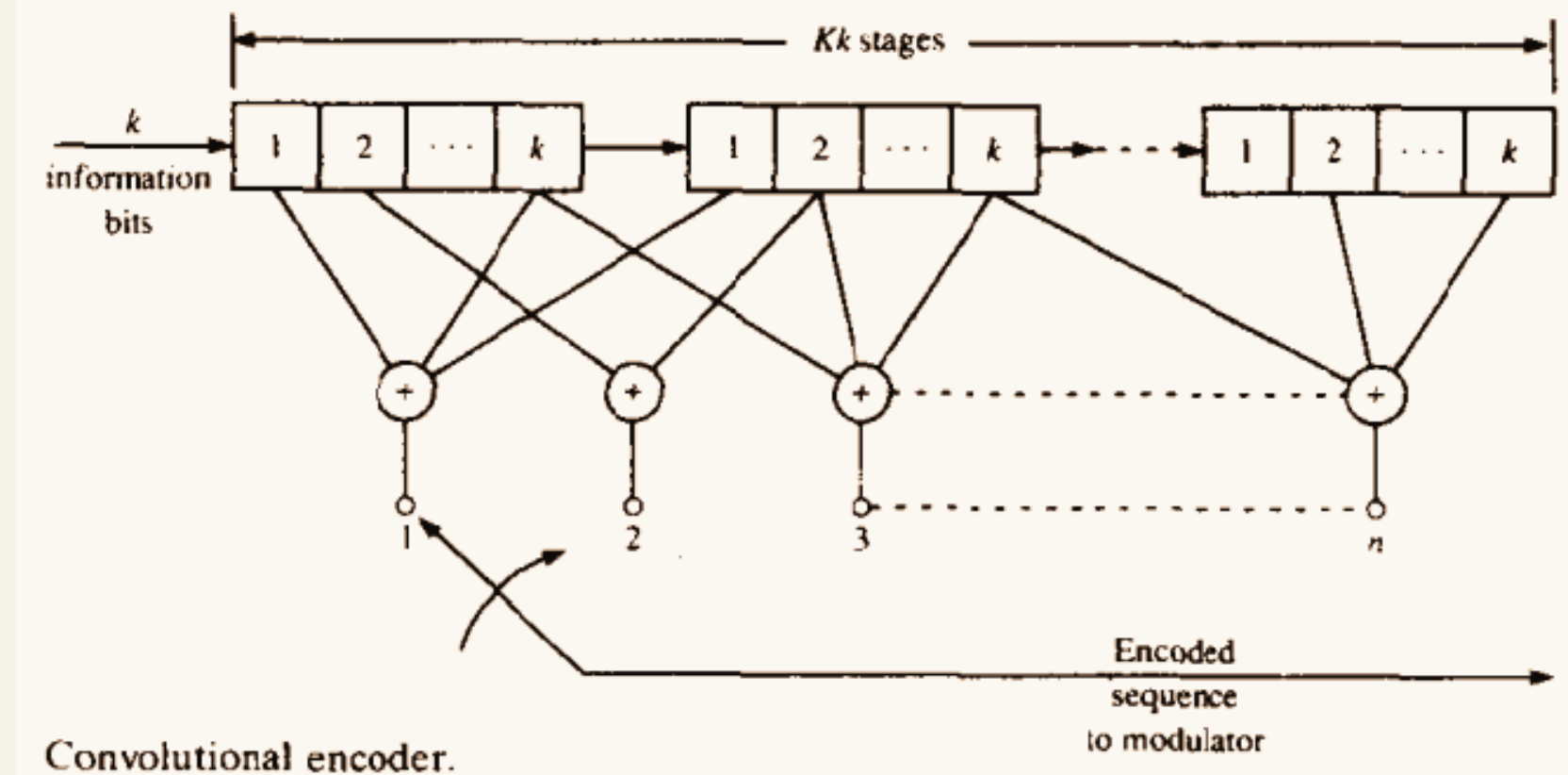
- Convolution codes are characterized by three parameters (n,k,m)

n = number of output bits produced for each k-bit sequence

k = number of input bits

m = number of memory registers

- Code Rate (R) = k/n
- Constraint length $K_c = k(m-1)$



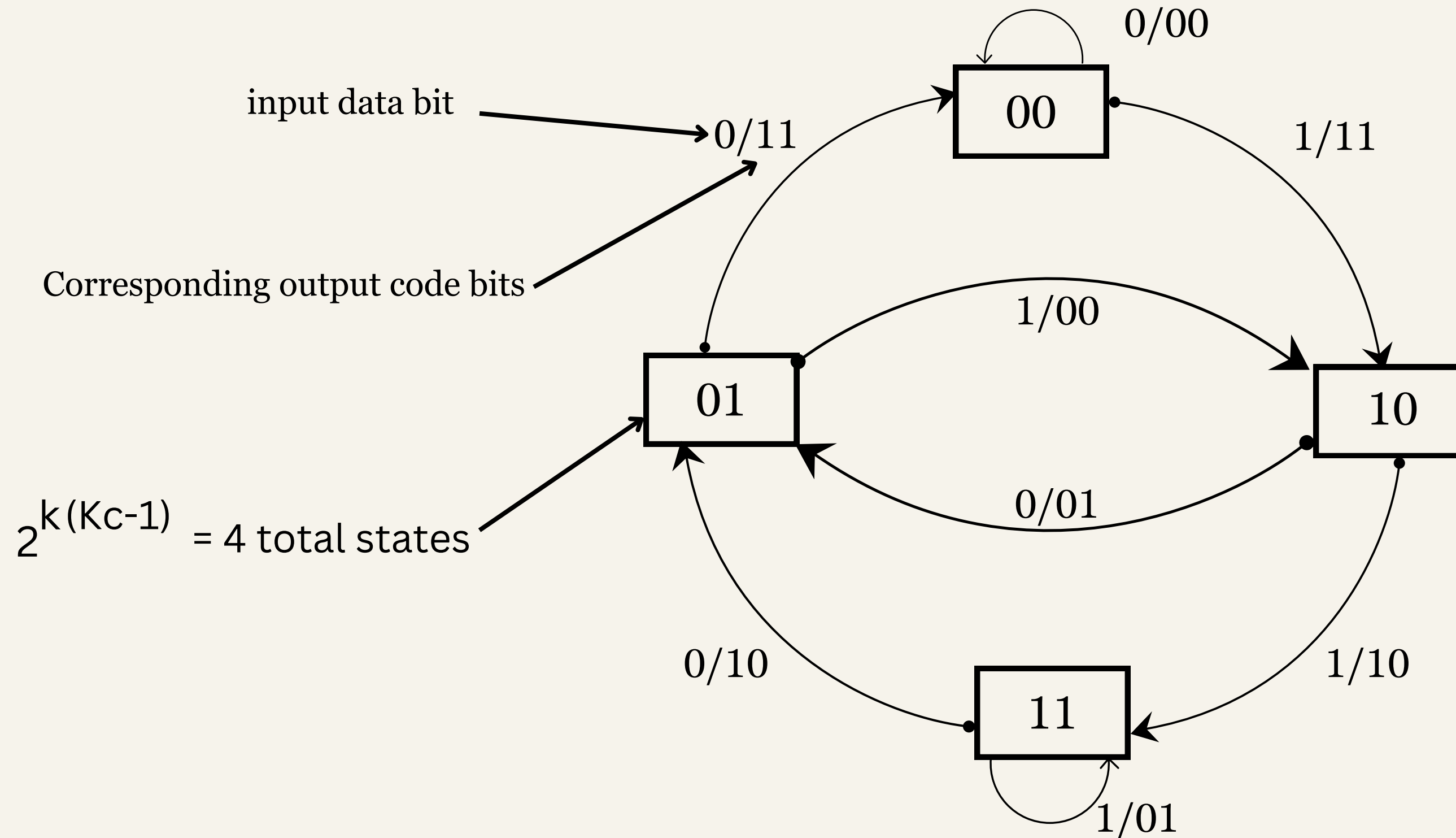
ENCODING REPRESENTATION

- ◆ There are some types of convolution encoding representation
 - State Diagram representation
 - Tree Diagram representation
 - Trellis representation

STATE DIAGRAM

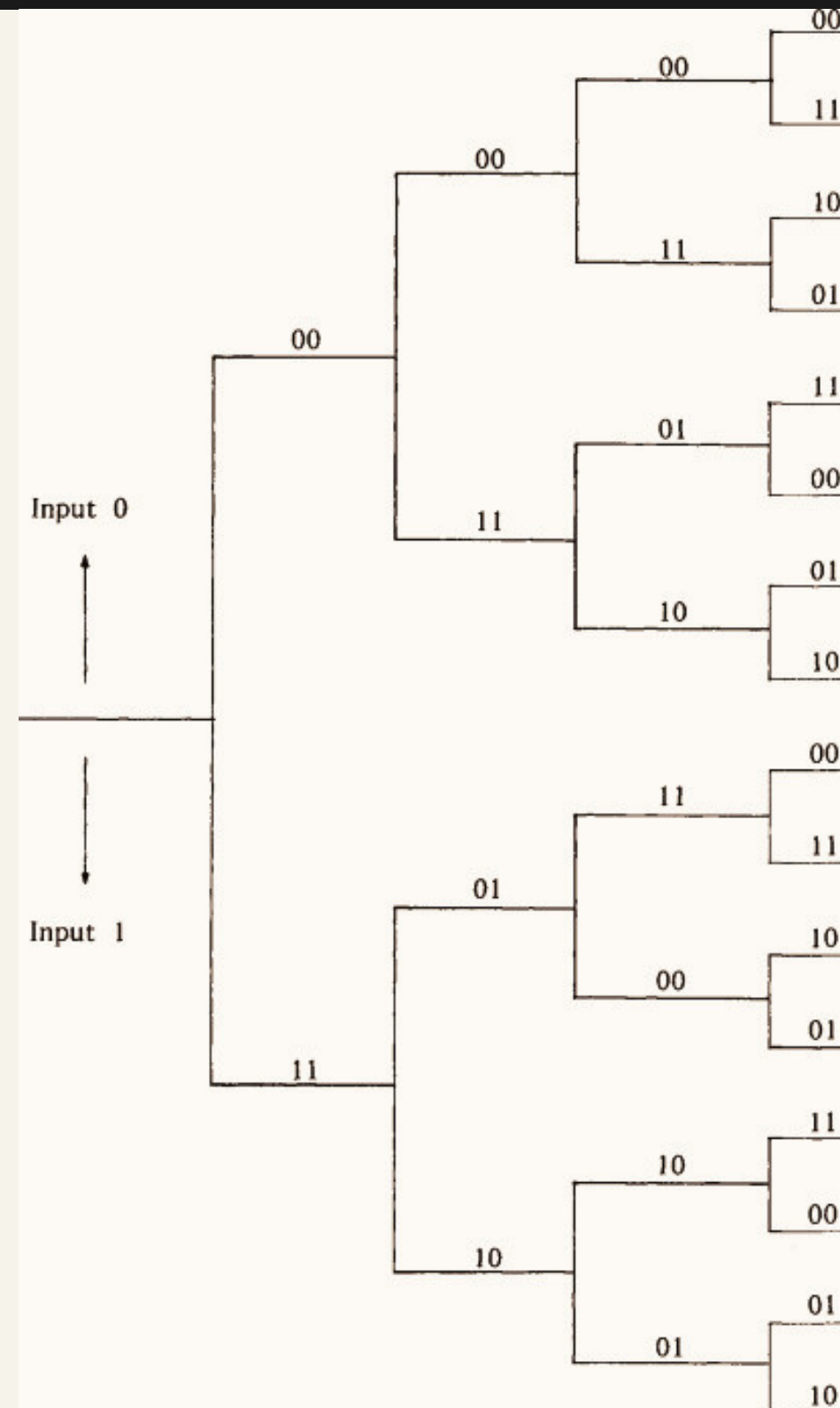
9

$$R = 1/2, K_c = 3, g_1 = [1 \ 0 \ 1], g_2 = [1 \ 1 \ 1]$$



TREE DIAGRAM

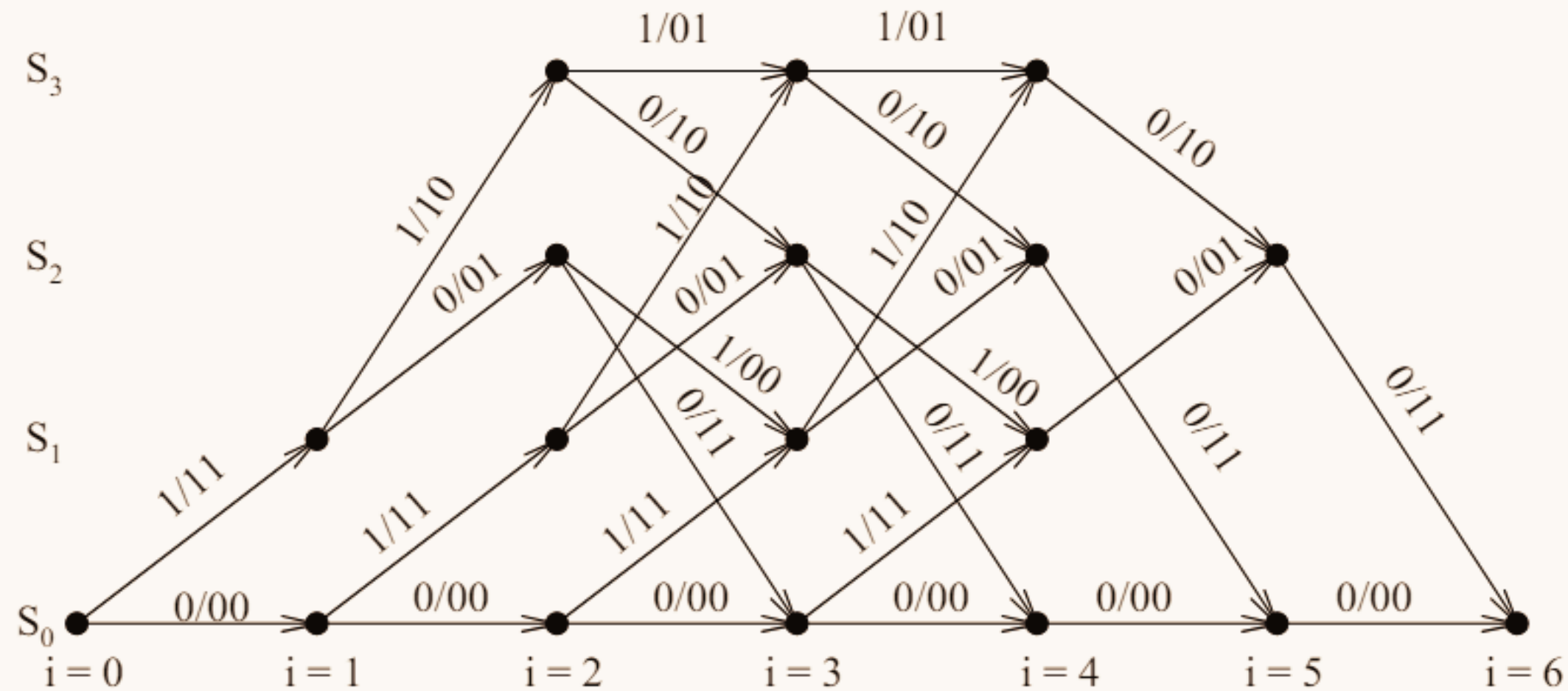
$$R = 1/2, K_c = 3, g_1 = [1 \ 0 \ 1], g_2 = [1 \ 1 \ 1]$$



Trellis Representation

11

$$R = 1/2, K_c = 3, g_1 = [1 \ 0 \ 1], g_2 = [1 \ 1 \ 1]$$



Transition Table

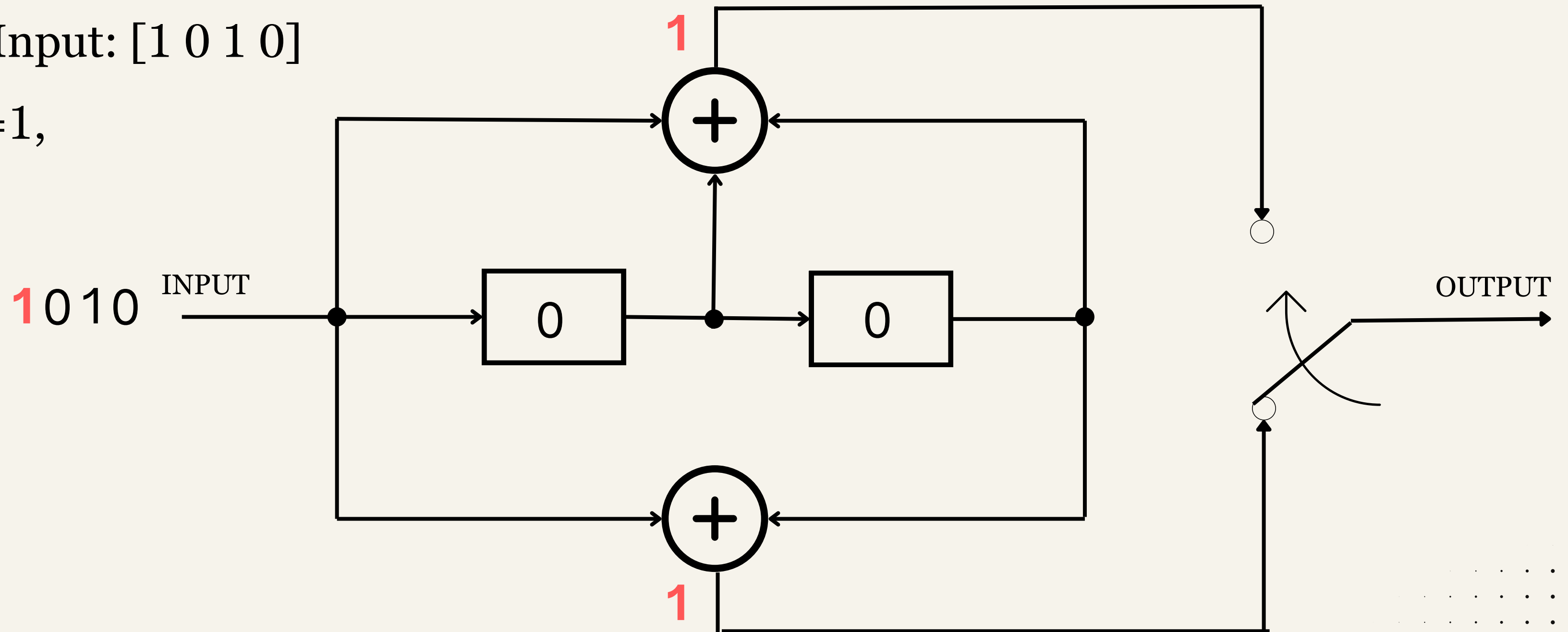
12

$R = 1/2$, $K_c = 3$, $g1 = [1\ 0\ 1]$, $g2 = [1\ 1\ 1]$

Input	Current State	Next State	Register Content	o1	o2
0	00	00	000	0	0
1	00	10	100	1	1
0	01	00	001	1	1
1	01	10	101	0	0
0	10	01	010	0	1
1	10	11	110	1	0
0	11	01	011	1	0
1	11	11	111	0	1

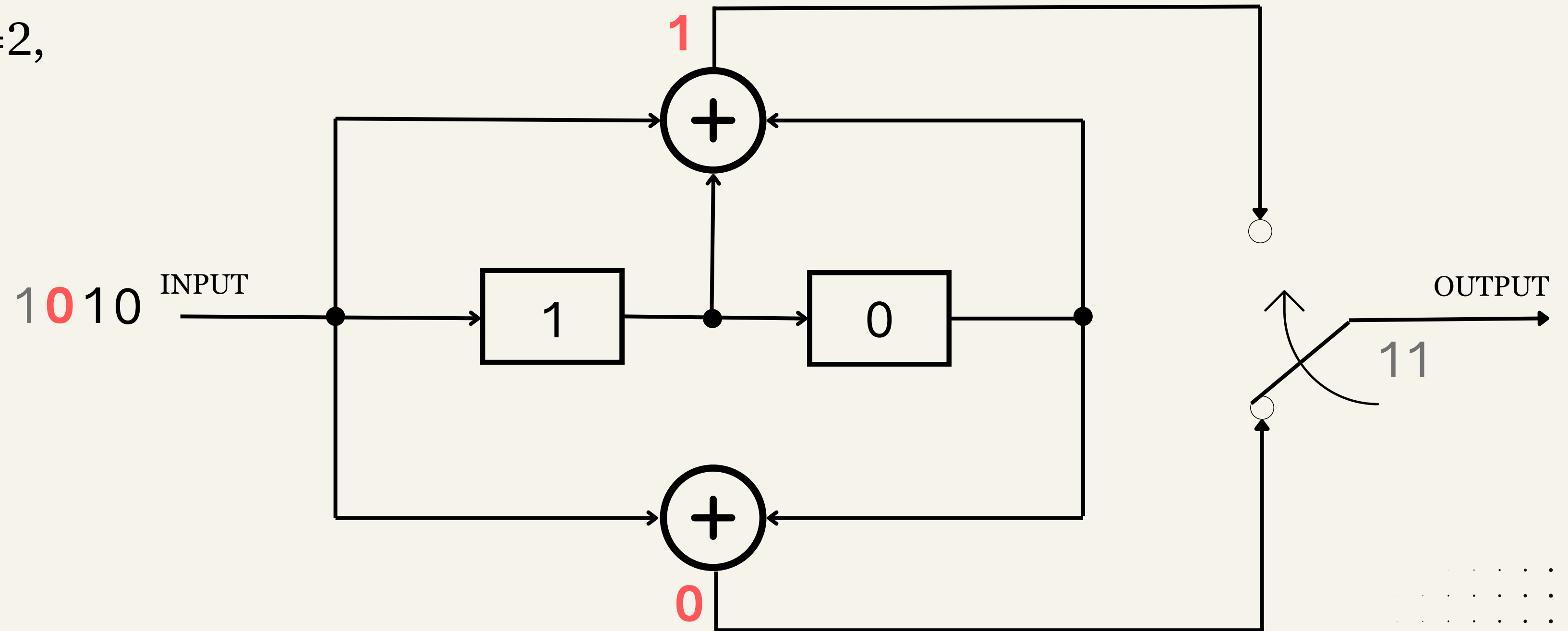
ENCODING

- For Input: [1 0 1 0]
- At $t=1$,



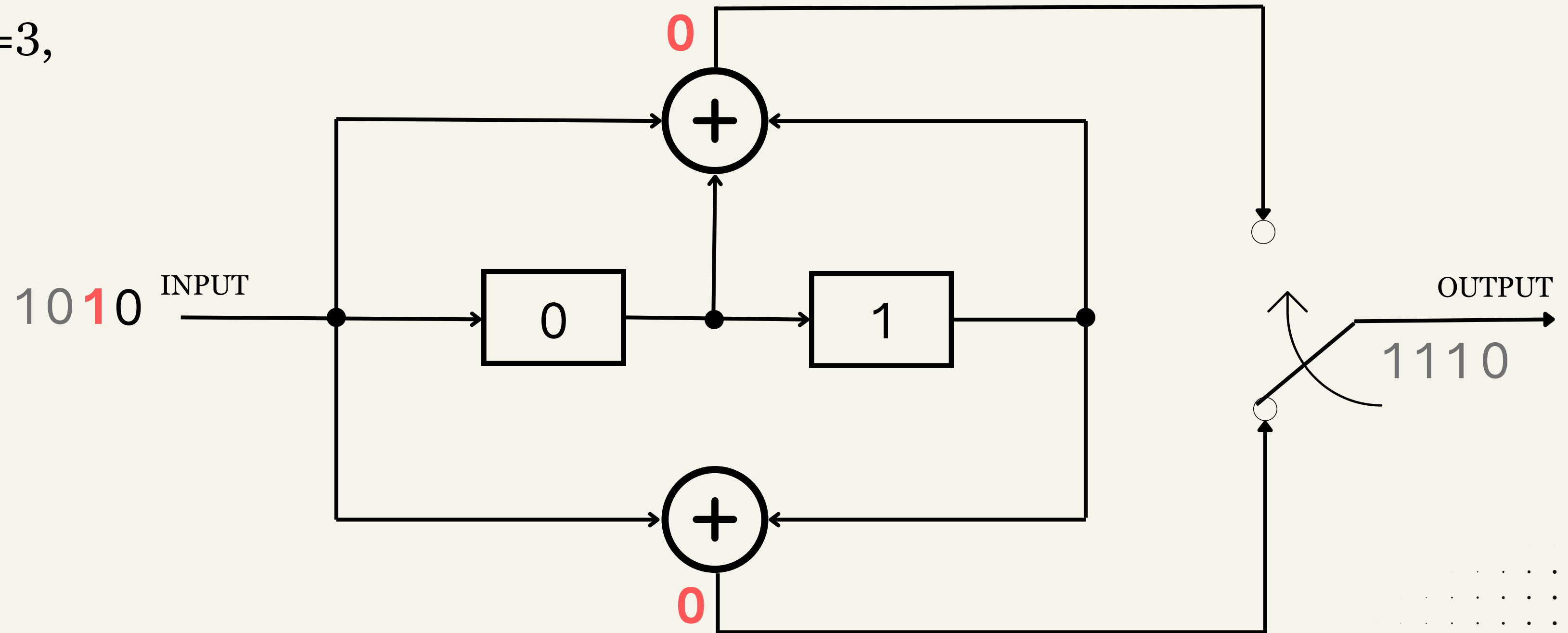
ENCODING

- At $t=2$,



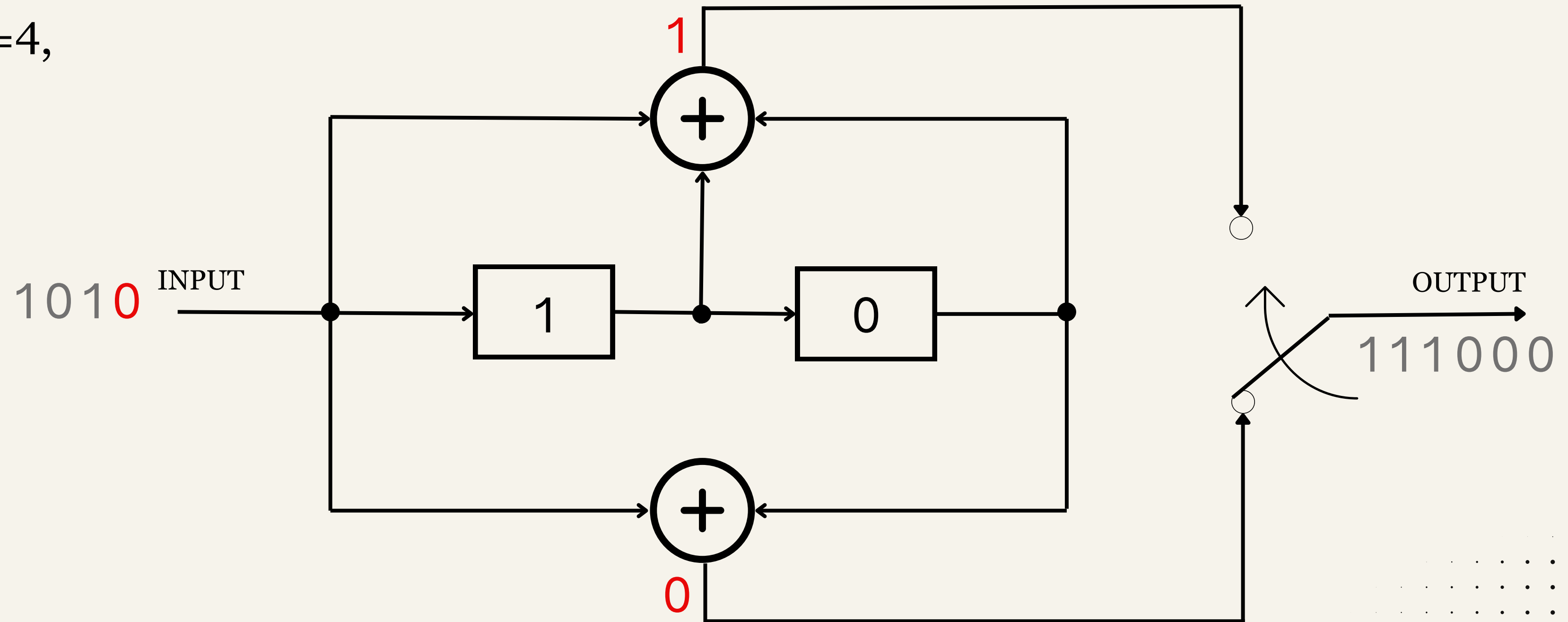
ENCODING

- At $t=3$,

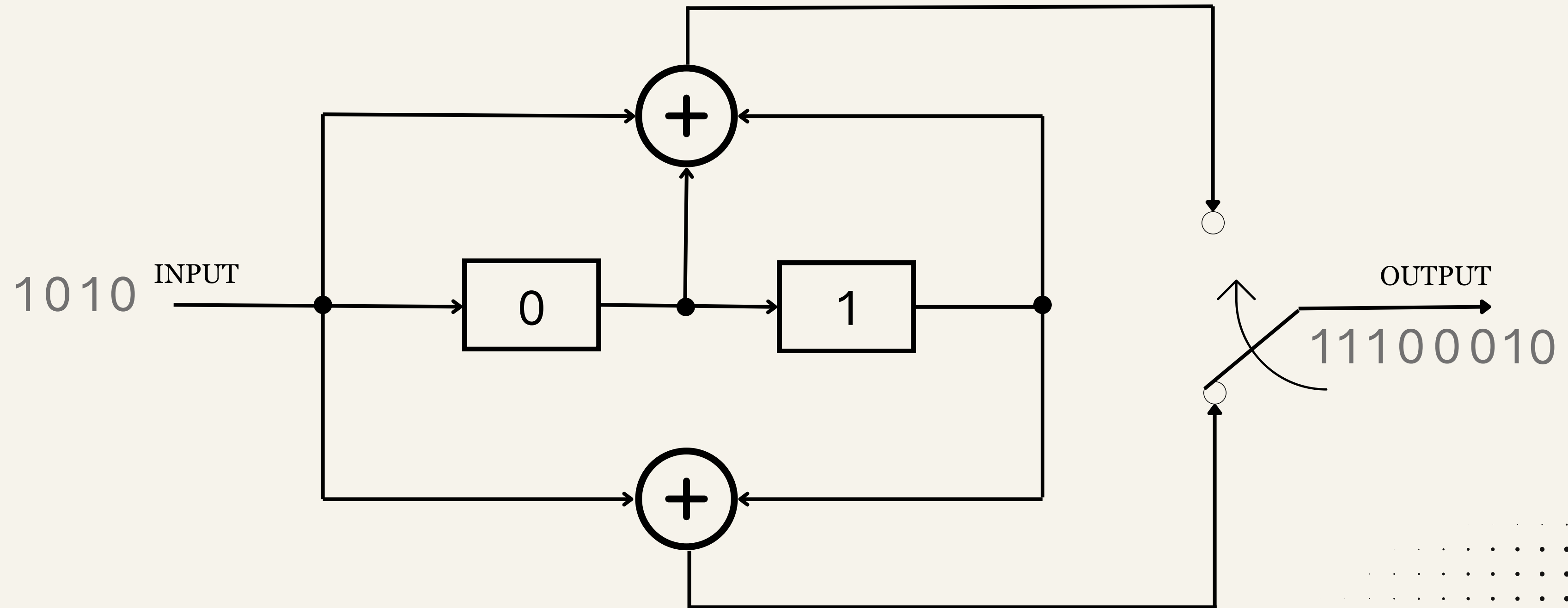


ENCODING

- At $t=4$,

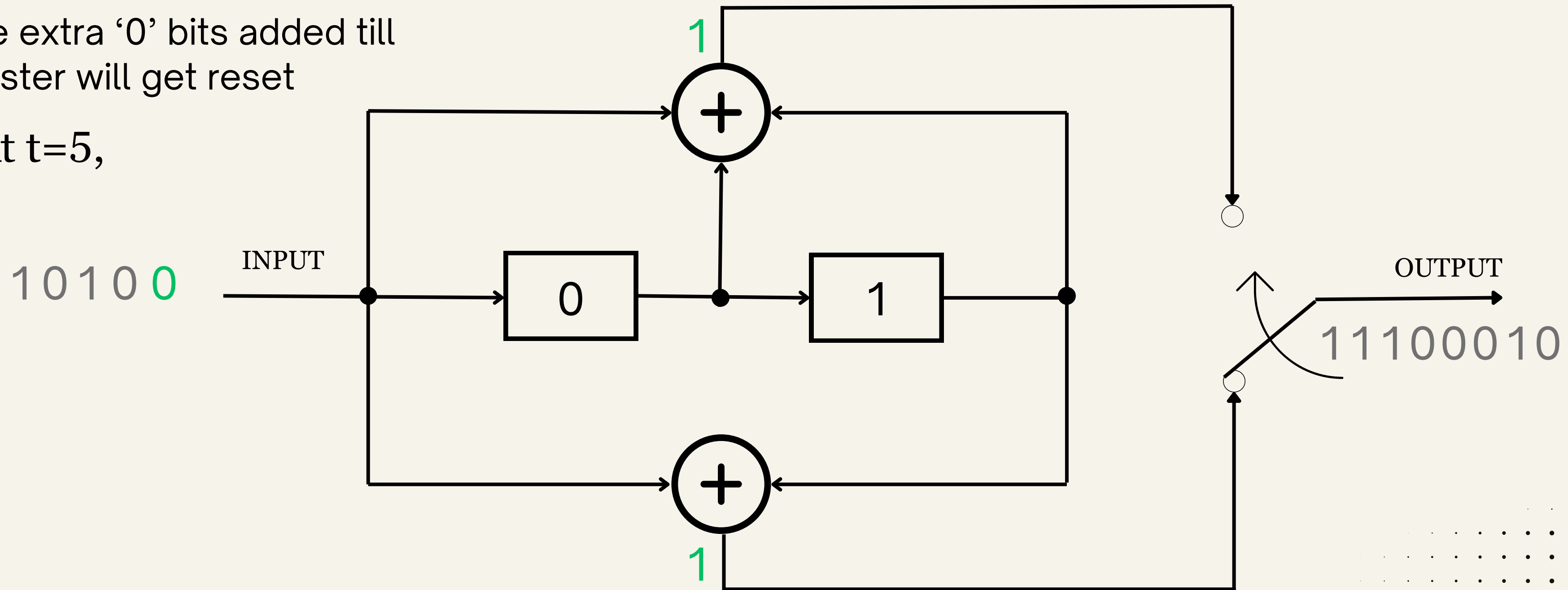


ENCODING



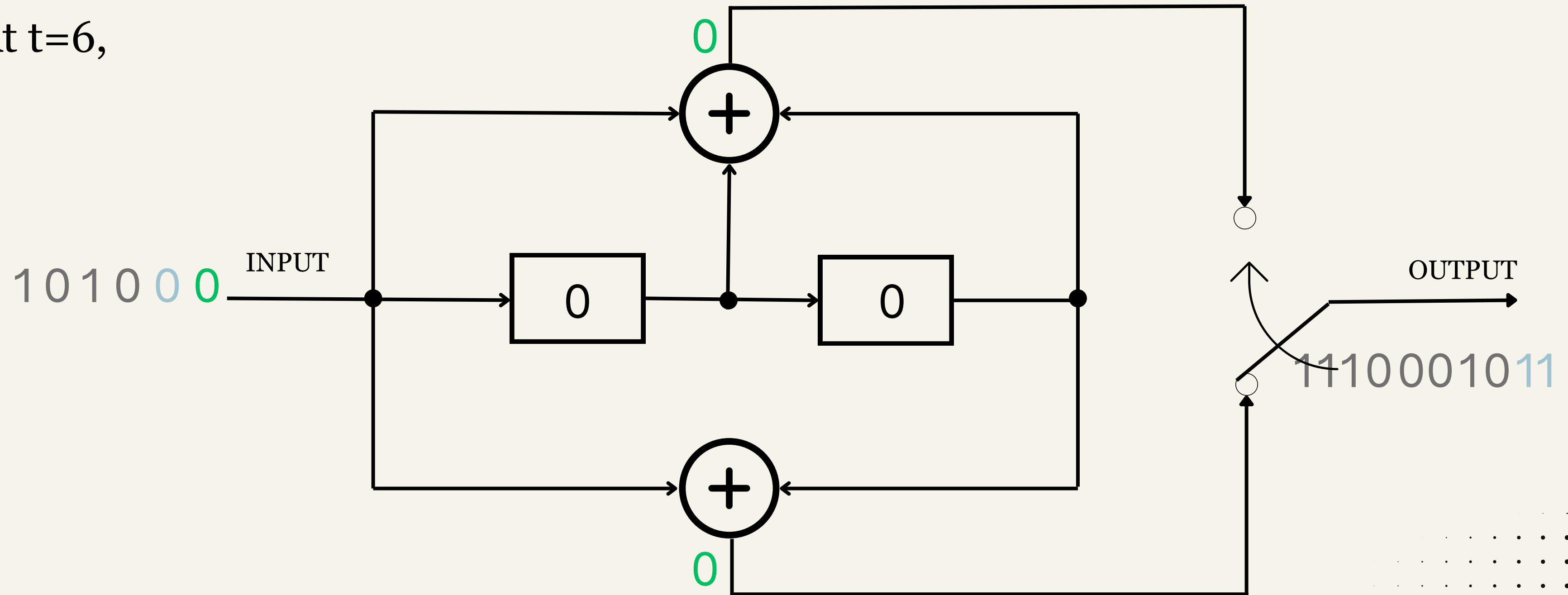
ENCODING

- The extra '0' bits added till register will get reset
- At t=5,

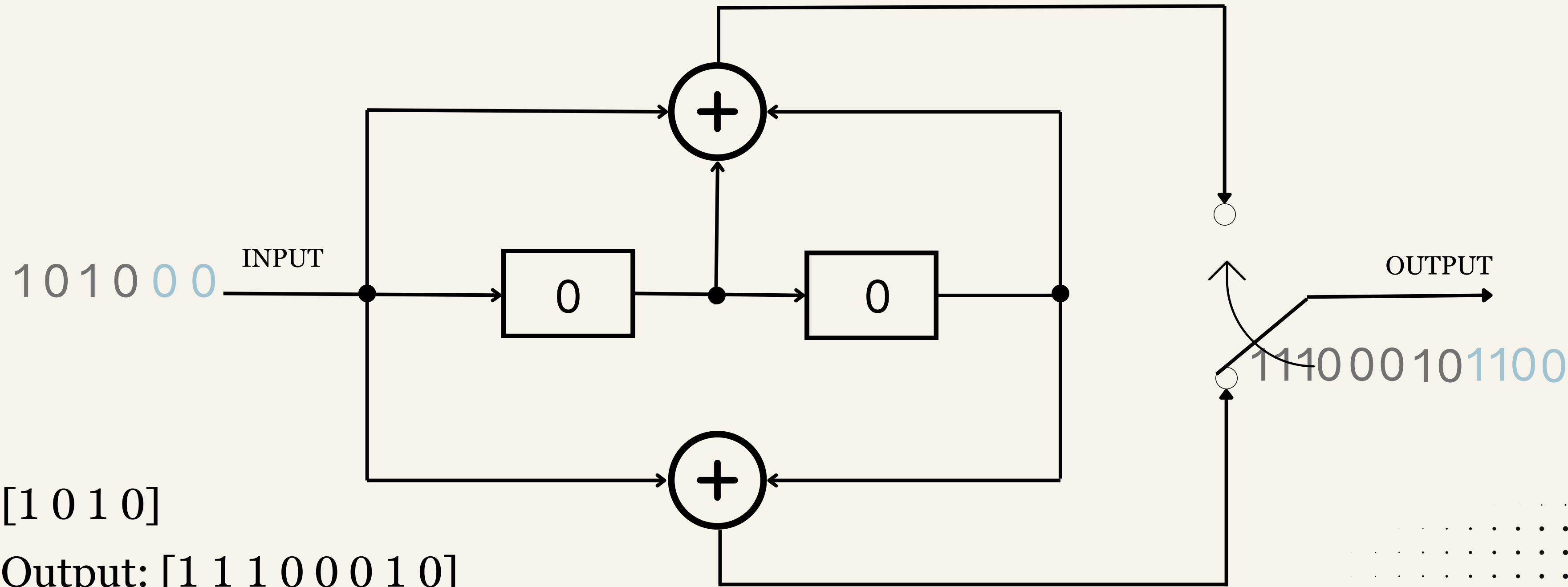


ENCODING

- At $t=6$,



ENCODING



Input: [1 0 1 0]

Actual Output: [1 1 1 0 0 0 1 0]

Output: [1 1 1 0 0 0 1 0 1 1 0 0]

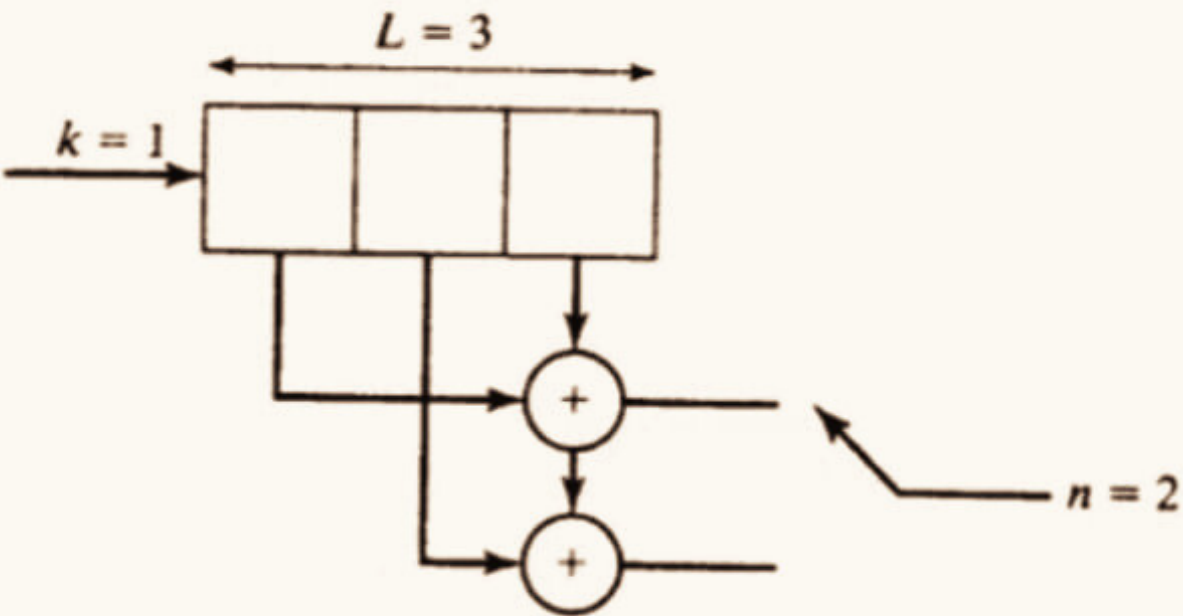
TRANSFER FUNCTION

- In convolutional coding, a transfer function describes how the input and output of a convolutional encoder are related. It's a useful tool for analyzing the performance of convolutional codes.

$$T(D, N, J) = \sum_{d=d_{free}}^{\infty} a_d D^d N^{f(d)} J^{g(d)}$$

- Exponent of J - g(d) - The length of the path that merges with the all zero path for the first time.
- Exponent of N - f(d) - The number of ones in the input sequence of K- bits.
- Exponent of D - d - The number of ones in the output sequence

For $R=1/2$, $k=1$, $n=2$

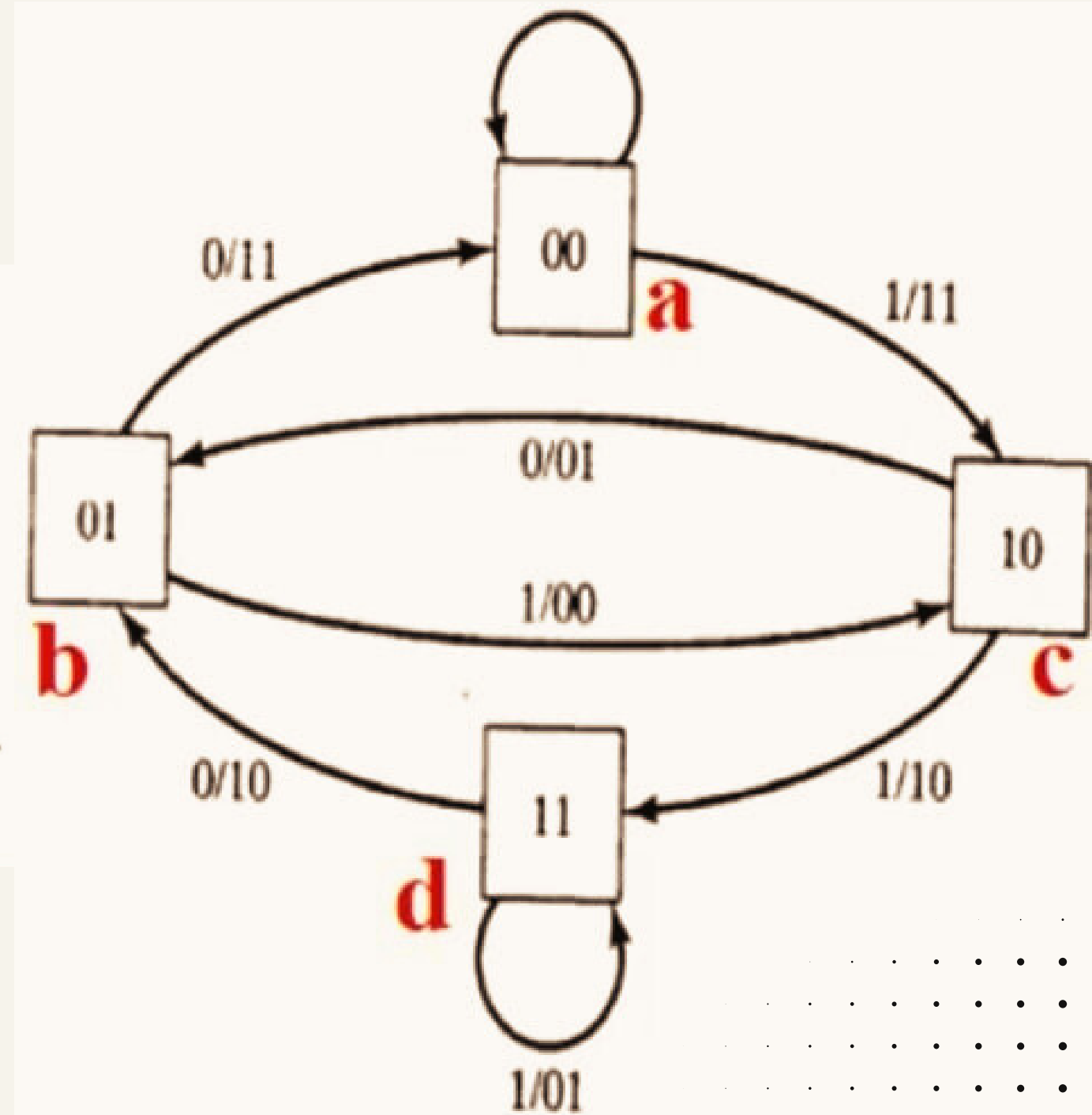
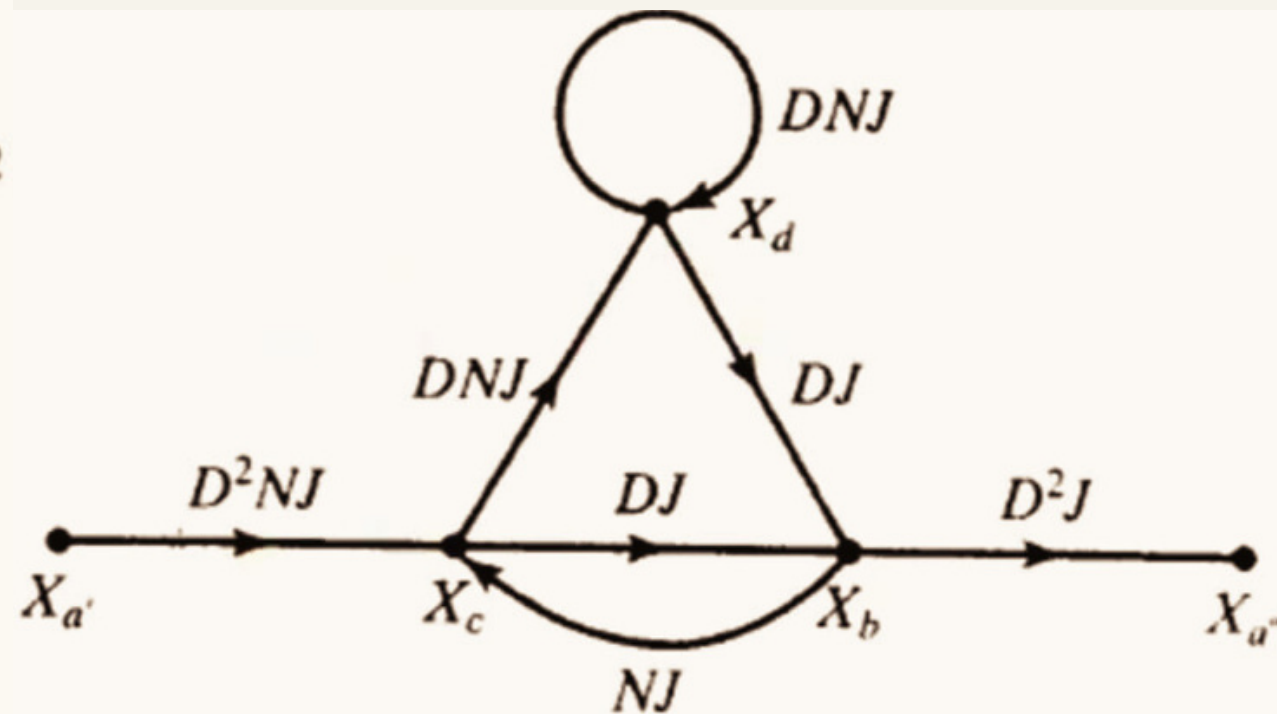


$$X_c = D^2NJ X_{a'} + NJX_b \rightarrow \textcircled{1}$$

$$X_b = DJ X_d + DJ X_c \rightarrow \textcircled{2}$$

$$X_d = DNJ X_c + DNJ X_d \rightarrow \textcircled{3}$$

$$X_{a''} = D^2J X_b \rightarrow \textcircled{4}$$



For $R=1/2$, $k=1$, $n=2$

$$T(D, N, J) = D^5 N J^3 + D^6 N^2 J^4 + D^6 N^2 J^5 + D^7 N^3 J^5 + \dots$$

$$\frac{X_a''}{X_a'} = \frac{\frac{D^2 J X_b}{1 (X_c - N J X_b)}}{D^2 N J} = \frac{D^4 N J^2 X_b}{(X_c - N J X_b)}$$

$$\frac{X_a''}{X_a'} = \frac{D^5 N J^3}{(1 - D N J - D N J^2)}$$

- Properties of transfer function :-**

Provides the properties of all the paths

Minimum non-zero exponent of D gives the minimum hamming distance

BPSK (Binary Phase Shift Keying)

- BPSK modulation efficiently transmits digital data by altering the phase of a carrier signal.
- A binary 0 is represented by a phase shift of 180 degrees, while a binary 1 is represented by no phase shift in the carrier signal.
- This straightforward technique enables efficient transmission of digital data.
- It is also utilized in different wireless communication systems like Wi-Fi, Bluetooth, and satellite communication.

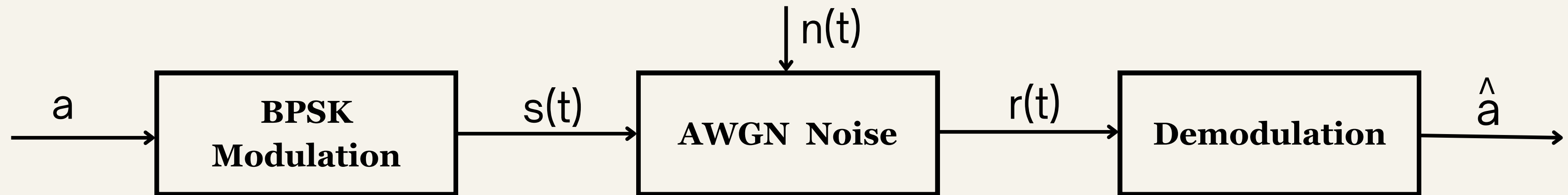
BPSK	
1	-1
0	1

AWGN(Additive white Gaussian Noise)

Each letter in the AWGN helps explain what the term means :

- **Additive** : AWGN channel, noise gets mixed in the original signal as it travels through the communication channel.
- **White** : The term “white” is in reference to the noise having the same power distribution across every frequency.
- **Gaussian** : Gaussian noise follows a Gaussian distribution, also known as a normal distribution. This means that the noise values are randomly distributed around a mean value with a certain standard deviation, following the bell-shaped curve characteristic of a Gaussian distribution.

MODULATION OVER AWGN CHANNEL



a = Encoded transmitted bits

\hat{a} = Receiver's estimated value of this transmitted bit

$s(t)$ = Waveform transmitted into the channel

$n(t)$ = White Gaussian Noise

$r(t) = s(t) + n(t)$: Waveform of received after addition of noise

DECODING

Decoding in convolutional code :-

There are two types of decoding techniques in convolutional coding :

- **Hard Decision Decoding :-** Hard Decision decoding works on principle of Minimum Hamming Distance.
- **Soft Decision Decoding :-** Soft Decision Decoding works on the principle of Minimum Euclidean Distance.

VITERBI DECODING(HDD)

Define States:

- Define states based on the constraint length K of the convolutional code, resulting $2^{(K-1)}$ possible states.

Initialize Trellis:

- Create an empty trellis with rows for each state and columns for encoded bits plus one for the initial state.

Start at Initial State:

- Place initial states in the leftmost column and assign a metric (typically distance) to each.

Transitioning:

- For each encoded bit, calculate the Hamming distance between current and possible next states.
- Choose transitions resulting in the minimum distance and update metrics accordingly.

Backtracking:

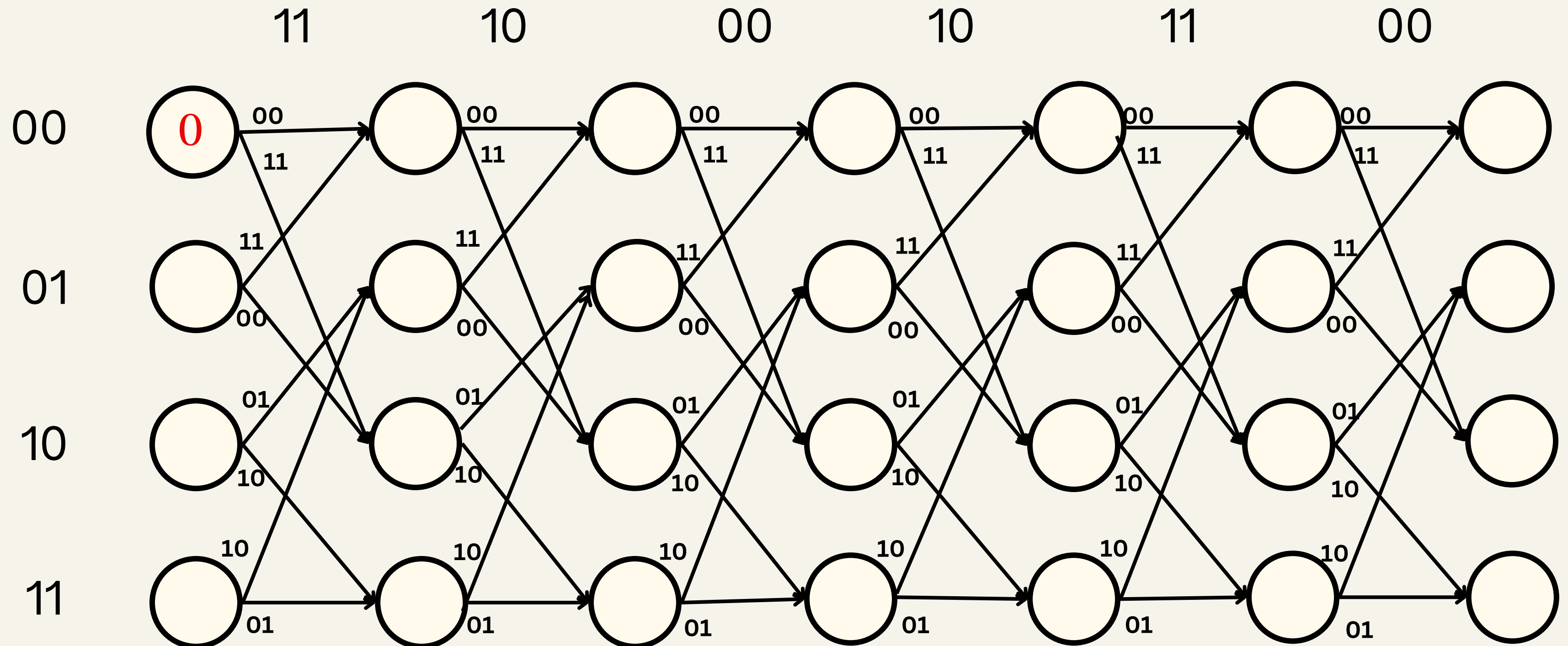
- At the end of the sequence, identify the path with the lowest cumulative metric.

Final Decoded Sequence:

- Backtrack through the trellis along the minimum distance path to obtain the decoded sequence.

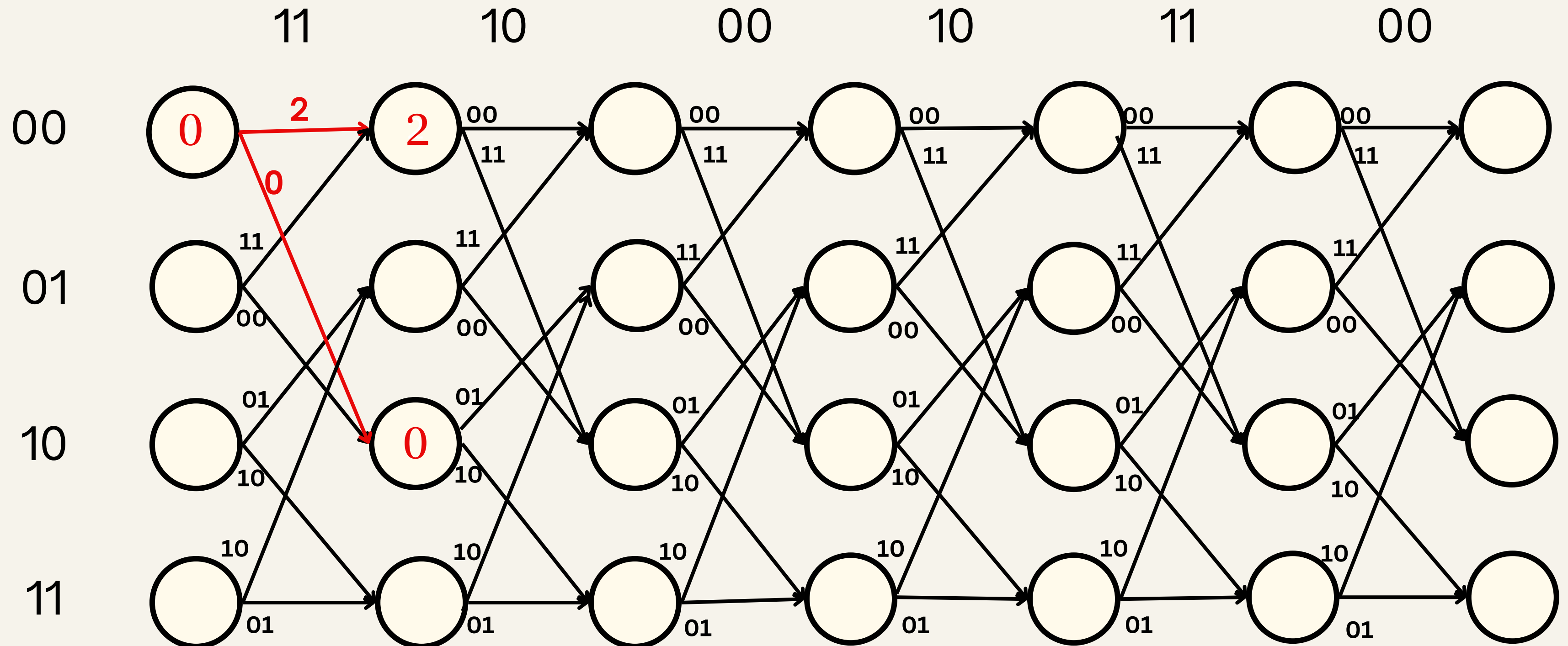
VITERBI DECODING (HDD)

For input = [1 0 1 0], g1=[1 0 1], g2=[1 1 1]



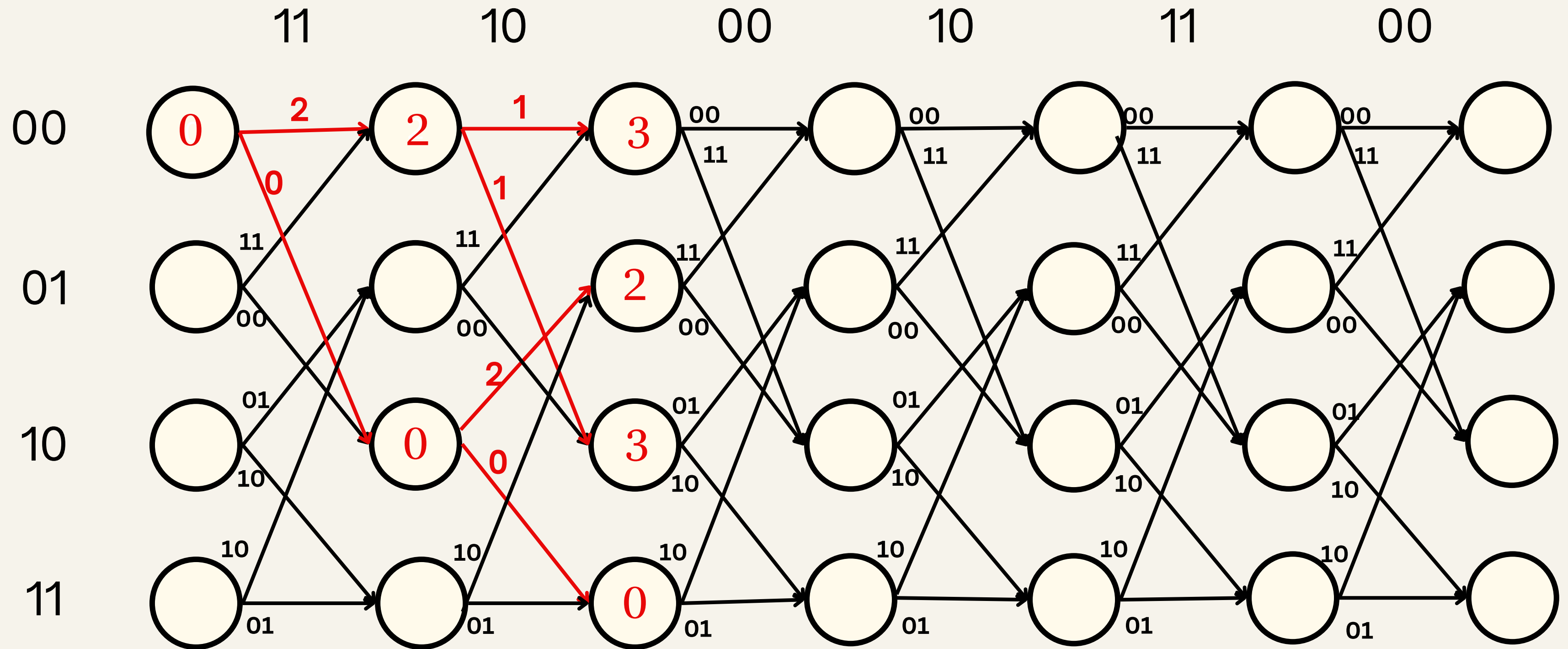
VITERBI DECODING (HDD)

For input = [1 0 1 0], g1=[1 0 1], g2=[1 1 1]



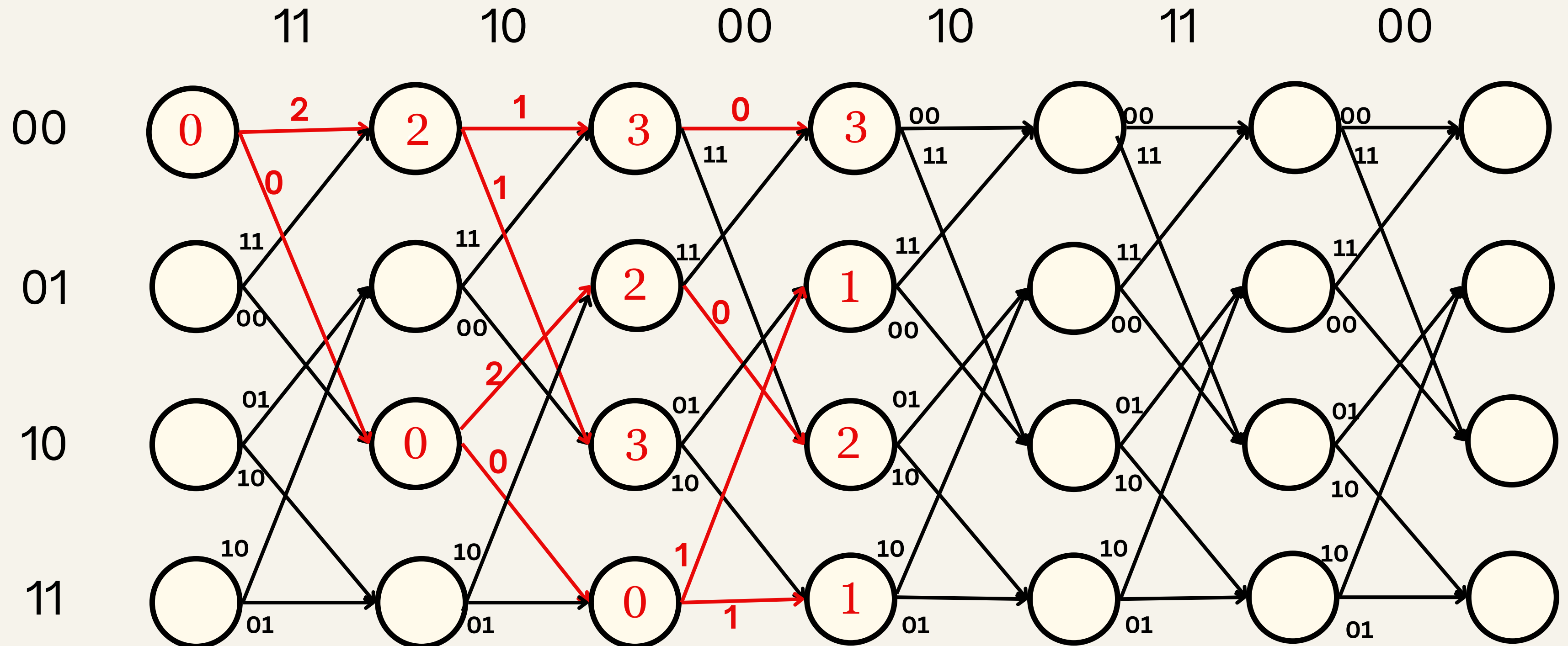
VITERBI DECODING (HDD)

For input = [1 0 1 0], g1=[1 0 1], g2=[1 1 1]



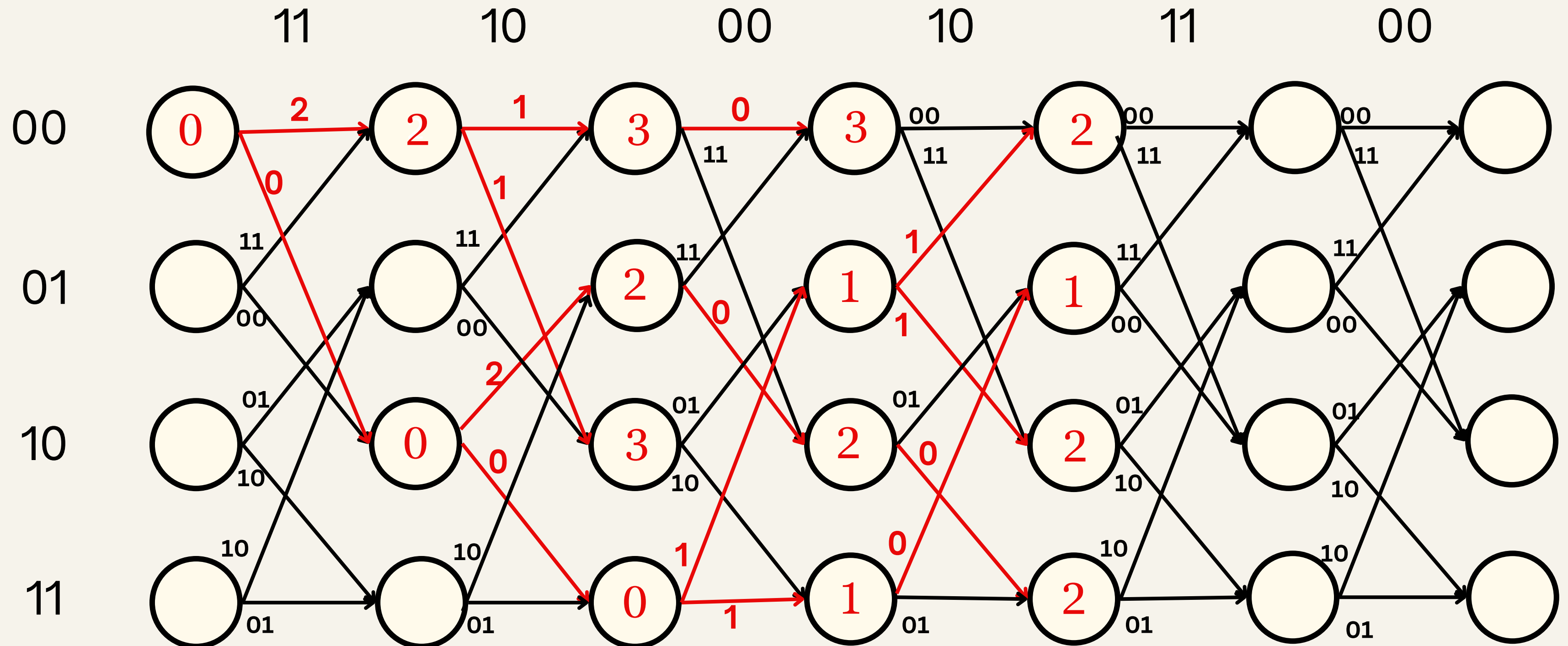
VITERBI DECODING (HDD)

For input = [1 0 1 0], g1=[1 0 1], g2=[1 1 1]



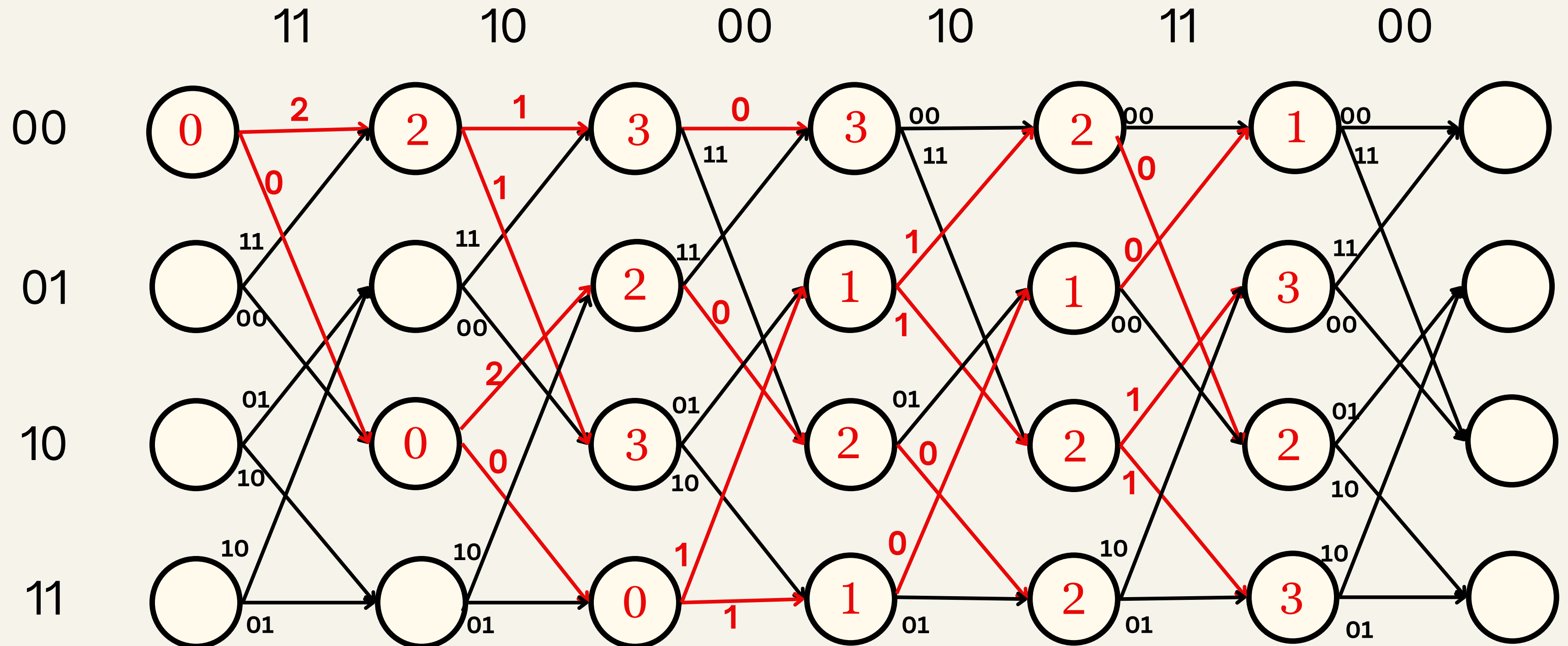
VITERBI DECODING (HDD)

For input = [1 0 1 0], g1=[1 0 1], g2=[1 1 1]



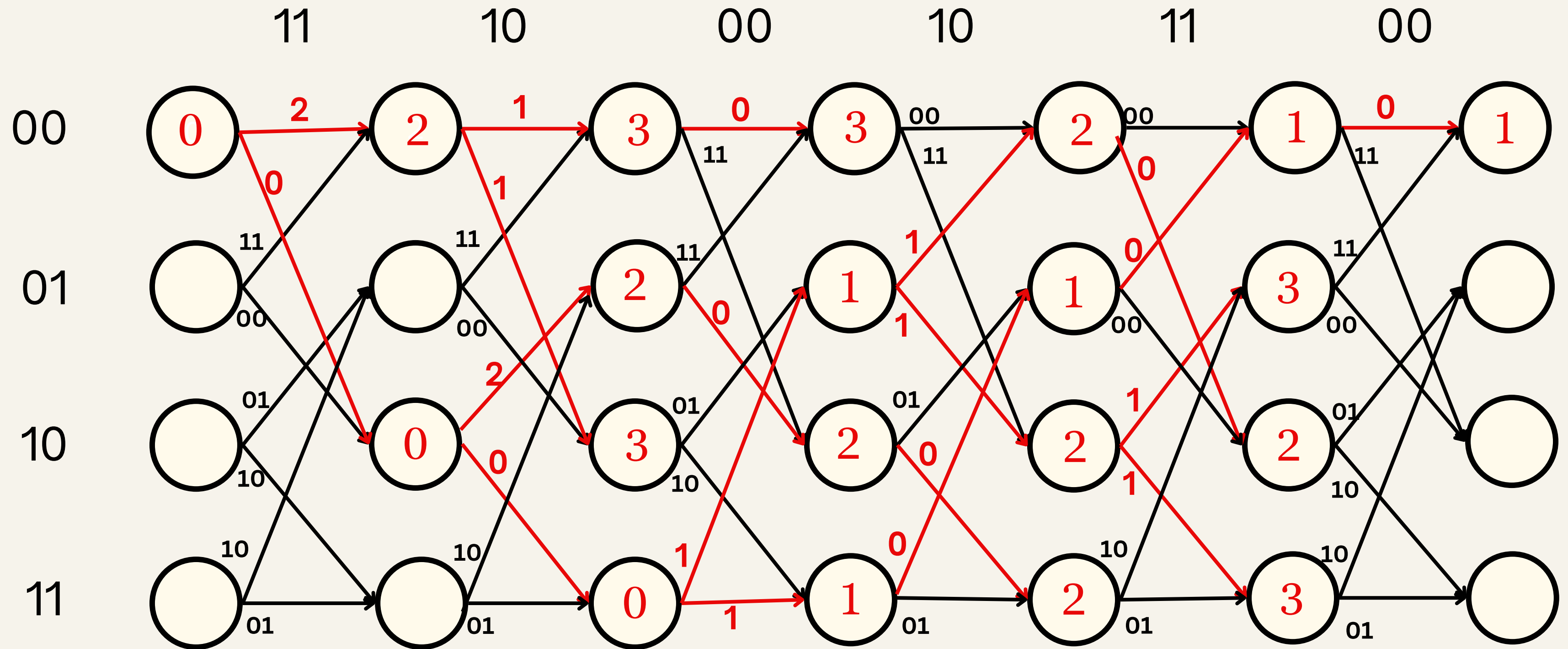
VITERBI DECODING (HDD)

For input = [1 0 1 0], g1=[1 0 1], g2=[1 1 1]



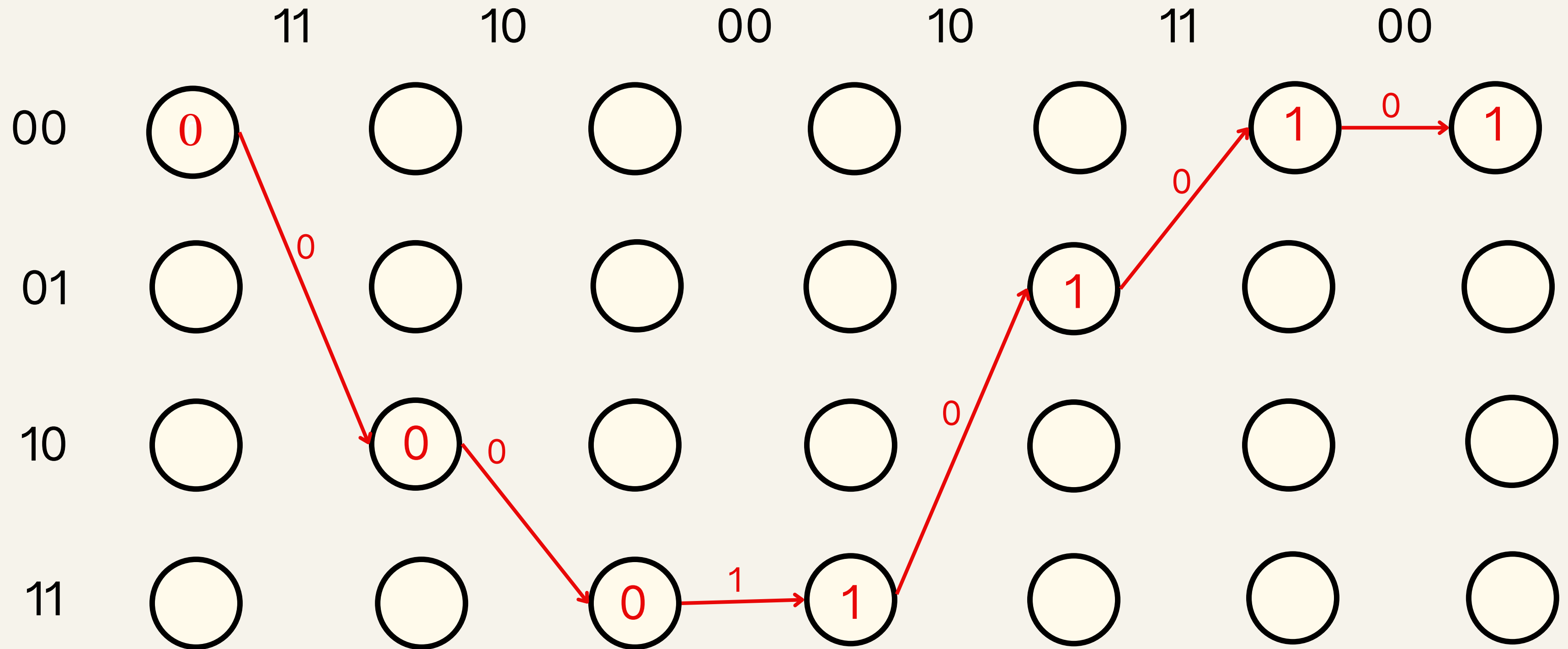
VITERBI DECODING (HDD)

For input = [1 0 1 0], g1=[1 0 1], g2=[1 1 1]



VITERBI DECODING (HDD)

For input = [1 0 1 0], g1=[1 0 1], g2=[1 1 1]



VITERBI DECODING(SDD)

Construct the trellis diagram.

Calculate Euclidean Distances:

- Extract blocks of size n from the received message and determine the Euclidean distances with the state outputs.
- For distance computation, modulate the state outputs, square the differences between corresponding bits, and sum these values.

Evaluate Branch Metrics:

- Nodes with multiple incoming branches require computation of individual branch metrics.
- Assess and compare these metrics to identify the minimum metric for each node.

Update Node Metrics:

- Revise the node metrics using the minimum branch metrics obtained.

Perform Backtracking:

- Upon reaching the trellis's end, backtrack along the path with the minimum path metric or Euclidean distance to ascertain the decoded output.

PSUEDO CODE : ENCODING

```

FUNCTION encoding(kc, generation_polynomials, inp)
    // Initialize variables
    n = LENGTH(generation_polynomials) // Number of generation
polynomials
    ans = [] // Initialize empty list for storing results
    regval = 0 // Initialize register value to zero

    // Loop through each input bit
    FOR i FROM 1 TO LENGTH(inp)
        curbit = inp[i] // Get current input bit
        regval = BITOR(BITSHIFT(regval, -1), BITSHIFT(curbit, (kc - 1))) //
Update register value

        // Loop through each generation polynomial
        FOR j FROM 1 TO n
            cur_poly = generation_polynomials[j] // Get current generation polynomial
            output = 0 // Initialize output to zero
            p = DEC2BIN(cur_poly, kc) // Convert current polynomial to binary

            p = FLIPLR(p) - '0' // Reverse the binary representation and
convert to integer array
            rv = DEC2BIN(regval, kc) - '0' // Convert register value to
binary
            output = 0 // Initialize output to zero

            // Compute output using bitwise AND and XOR operations
            FOR k FROM 1 TO kc
                output = XOR(output, (p[k] AND rv[k])) // Update
output using XOR and AND operations
            END FOR

            ans = CONCATENATE(ans, output) // Append output
to result list
        END FOR
    END FOR

    RETURN ans // Return the encoded output
END FUNCTION

```

PSUEDO CODE : HARD DECODING

FUNCTION decoding(kc, generation_polynomials, inp)

// Initialize variables

ns = BITSHIFT(1, (kc - 1)) **// Number of states**

n = LENGTH(generation_polynomials) **// Number of generation**

polynomials

mtr = ONES(FLOOR(LENGTH(inp) / n) + 1, ns) * 1e9 **// Initialize metric matrix with large values**

previous_states = CELL(FLOOR(LENGTH(inp) / n) + 1, 1) **// Initialize previous states cell array**

// Initialize previous_states with large values

FOR i FROM 1 TO LENGTH(previous_states)

previous_states[i] = CELL(ns, 1)

FOR j FROM 1 TO ns

previous_states[i][j] = {1e9, 1e9}

END FOR

END FOR

idx = 0

// Loop through time instances

FOR t FROM 1 TO (LENGTH(inp) / n)

// Loop through states

FOR st FROM 0 TO ns - 1

// Initialize metric for the first time instance and state 0

IF (t == 1) AND (st == 0)

mtr(t, st + 1) = 0

END IF

// Loop through input bits (0 and 1)

FOR input_bits FROM 0 TO 1

curstate = st

next_state = BITOR(BITSHIFT(curstate, -1), (input_bits * BITSHIFT(1, (kc - 2))))

hamming_distance = 0

// Compute output and update hamming distance

FOR i FROM 1 TO n

output = 0

poly = generation_polynomials[i]

regvalue = BITOR(curstate, BITSHIFT(input_bits, (kc - 1)))

PSUEDO CODE : HARD DECODING

```

FOR k FROM 0 TO (kc - 1)
    output = BITXOR(output, BITAND(BITSHIFT(poly, -k), 1)
AND BITAND(BITSHIFT(regvalue, -(kc - k - 1)), 1))
END FOR

// Update hamming distance
IF output != inp(idx + i)
    hamming_distance = hamming_distance + 1
END IF
END FOR

a = mtr(t + 1, next_state + 1)
b = mtr(t, curstate + 1)

// Update metric and previous states
IF a > b + hamming_distance
    mtr(t + 1, next_state + 1) = b + hamming_distance
    previous_states[t + 1][next_state + 1] = {curstate, input_bits}
END IF

```

```

END FOR
END FOR
idx = idx + n
END FOR

// Initialize ans list
ans = []

// Trace back to find the optimal path
temp = previous_states[(LENGTH(inp) / n) + 1][1]
ans = [temp[2] ans]
cur = LENGTH(inp) / n

WHILE cur > 1
    temp = previous_states[cur][temp[1] + 1]
    ans = [temp[2] ans]
    cur = cur - 1
END WHILE

RETURN ans // Return the decoded output
END FUNCTION

```

PSUEDO CODE : SOFT DECODING

```

FUNCTION decoding_soft(kc, generation_polynomials, inp)
    // Initialize variables
    ns = BITSHIFT(1, (kc - 1))           // Number of states
    n = LENGTH(generation_polynomials)   // Number of generation
    polynomials
    mtr = ONES(FLOOR(LENGTH(inp) / n) + 1, ns) * 1e9 // Initialize metric
    matrix with large values
    previous_states = CELL(FLOOR(LENGTH(inp) / n) + 1, 1) // Initialize
    previous states cell array

    // Initialize previous_states with large values
    FOR i FROM 1 TO LENGTH(previous_states)
        previous_states[i] = CELL(ns, 1)
        FOR j FROM 1 TO ns
            previous_states[i][j] = {1e9, 1e9}
        END FOR
    END FOR

    idx = 0

    // Loop through time instances
    FOR t FROM 1 TO (LENGTH(inp) / n)
        // Loop through states
        FOR st FROM 0 TO ns - 1
            // Initialize metric for the first time instance and state 0
            IF (t == 1) AND (st == 0)
                mtr(t, st + 1) = 0
            END IF

            // Loop through input bits (0 and 1)
            FOR input_bits FROM 0 TO 1
                curstate = st
                next_state = BITOR(BITSHIFT(curstate, -1), (input_bits *
                BITSHIFT(1, (kc - 2))))
                euclidean_distance = 0

                // Compute output and update euclidean distance
                FOR i FROM 1 TO n
                    output = 0
                    poly = generation_polynomials[i]
                    regvalue = BITOR(curstate, BITSHIFT(input_bits, (kc - 1)))

```

PSUEDO CODE : SOFT DECODING

```

// Compute output using bitwise operations
FOR k FROM 0 TO (kc - 1)
    output = BITXOR(output, BITAND(BITSHIFT(poly, -k), 1)
AND BITAND(BITSHIFT(regvalue, -(kc - k - 1)), 1))
END FOR

output = 1 - 2 * output
euclidean_distance = euclidean_distance + (ABS(output * inp(idx +
i)) * ABS(output - inp(idx + i)))
END FOR

euclidean = Sqrt(euclidean_distance)
a = mtr(t + 1, next_state + 1)
b = mtr(t, curstate + 1)

// Update metric and previous states
IF a > b + euclidean_distance
    mtr(t + 1, next_state + 1) = b + euclidean_distance
    previous_states[t + 1][next_state + 1] = {curstate, input_bits}
END IF

```

```

END FOR
END FOR
    idx = idx + n
END FOR

// Initialize ans list
ans = []

// Trace back to find the optimal path
temp = previous_states[(LENGTH(inp) / n) + 1][1]
ans = [temp[2] ans]
cur = LENGTH(inp) / n

WHILE cur > 1
    temp = previous_states[cur][temp[1] + 1]
    ans = [temp[2] ans]
    cur = cur - 1
END WHILE

RETURN ans // Return the decoded output
END FUNCTION

```

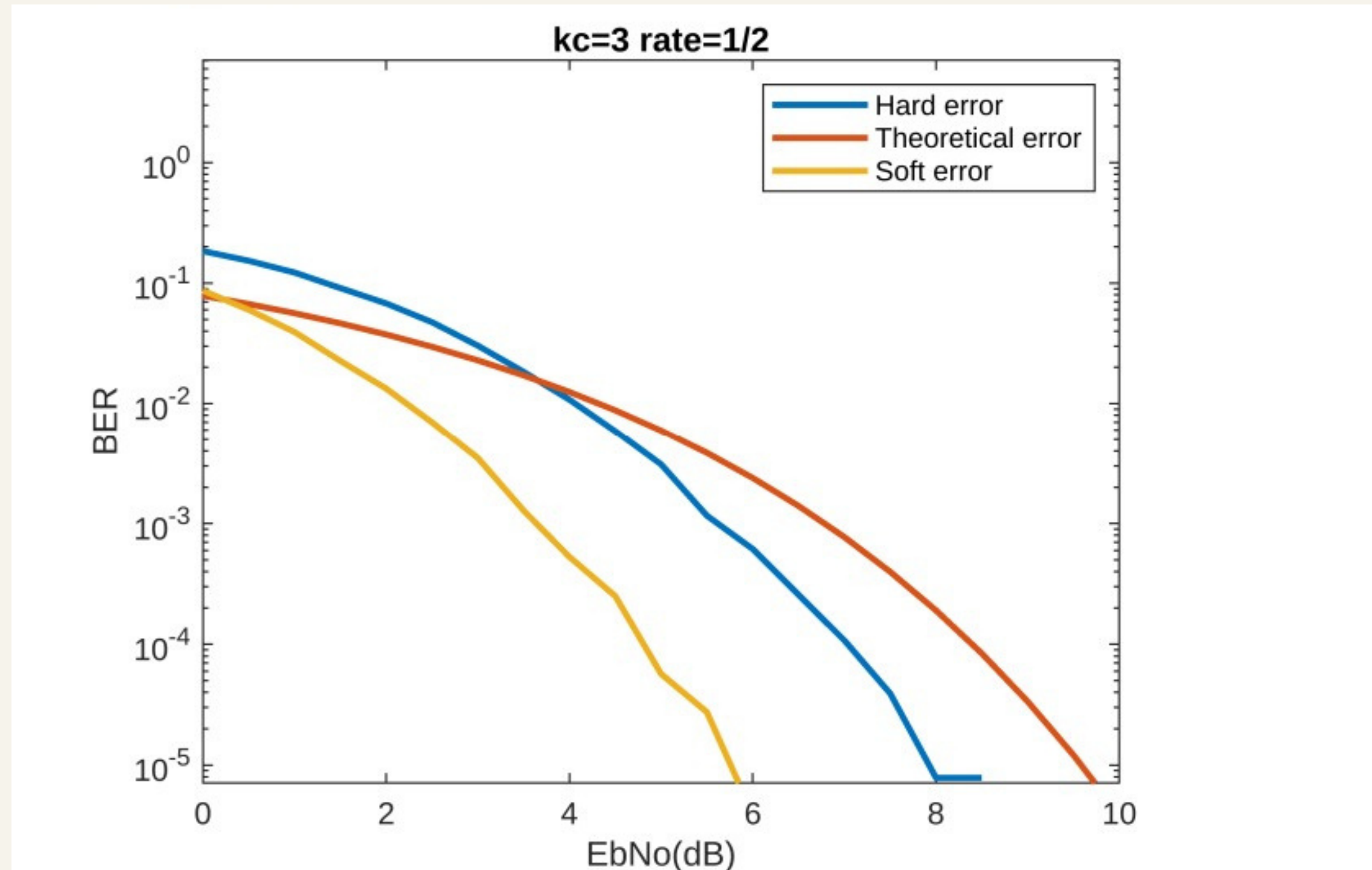
PSUEDO CODE : MODULATION

```
FUNCTION modulator(encoded_message, sigma)
    // BPSK modulation: Mapping binary values 1 to -1 and 0 to 1
    s = 1 - 2 * encoded_message

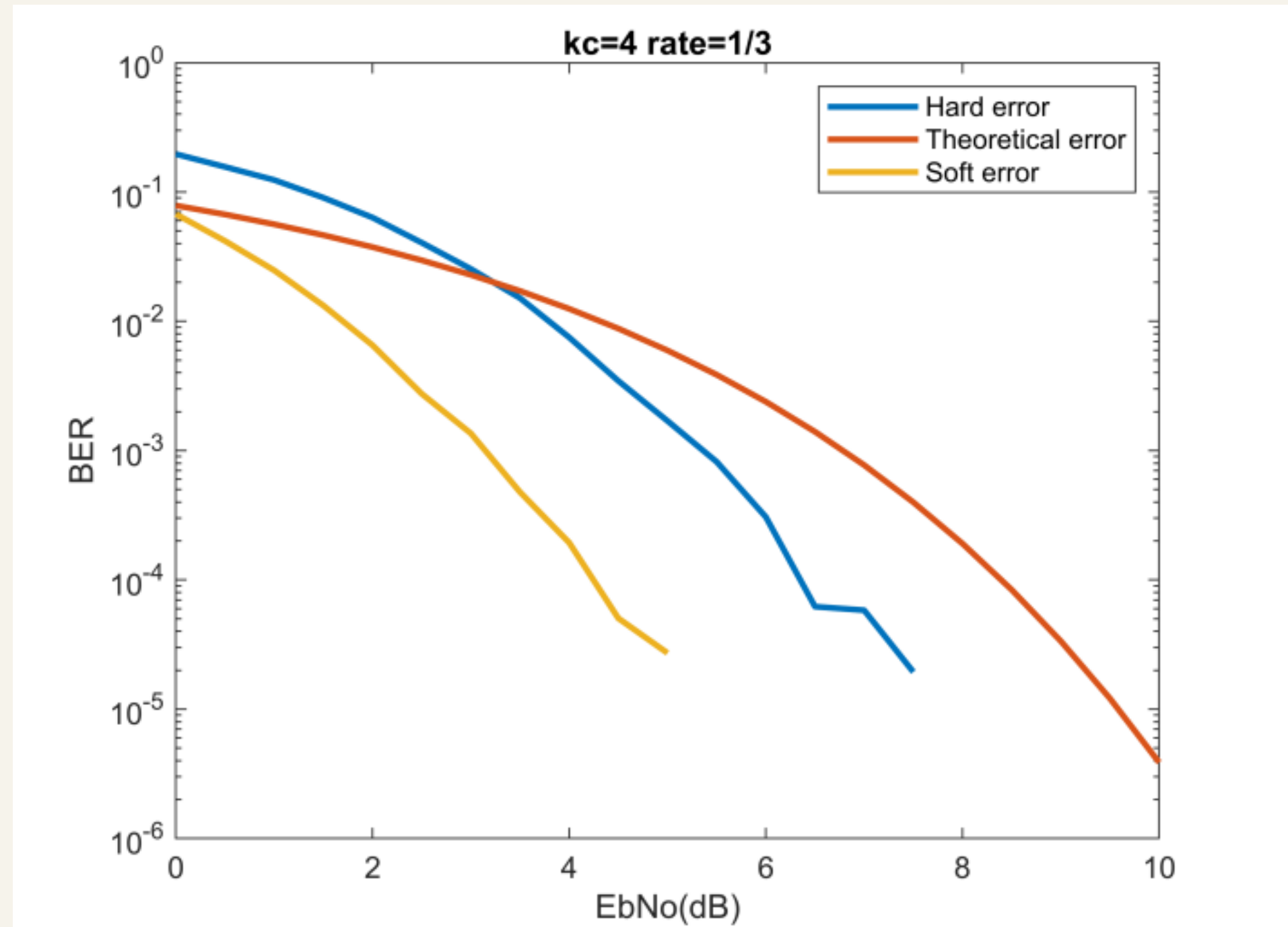
    // Add Gaussian noise to the modulated signal
    modulated_op = s + sigma * RANDN(1, LENGTH(encoded_message))

    RETURN modulated_op // Return the modulated signal with noise
END FUNCTION
```

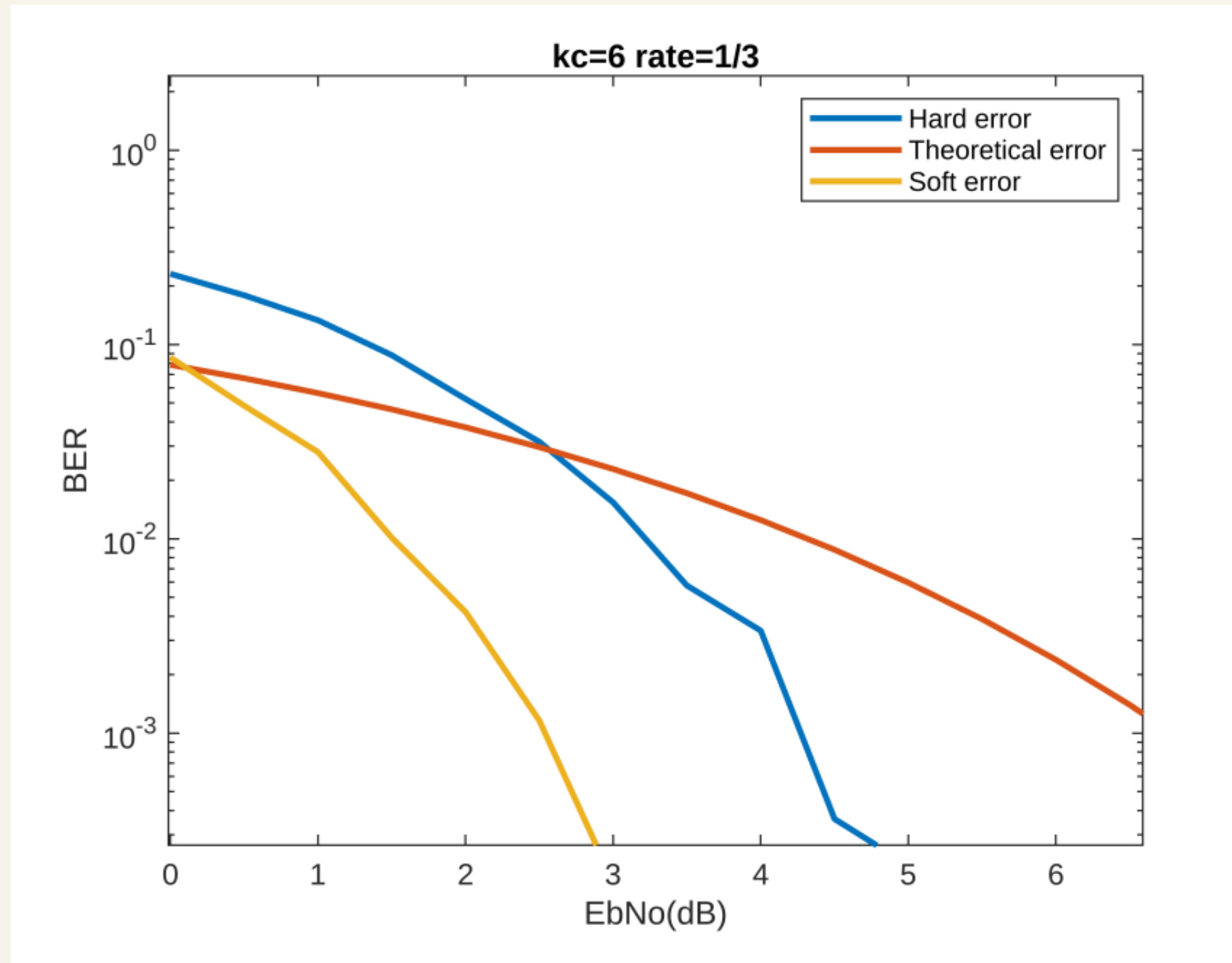
GRAPH : $R=1/2$ AND $KC=3$



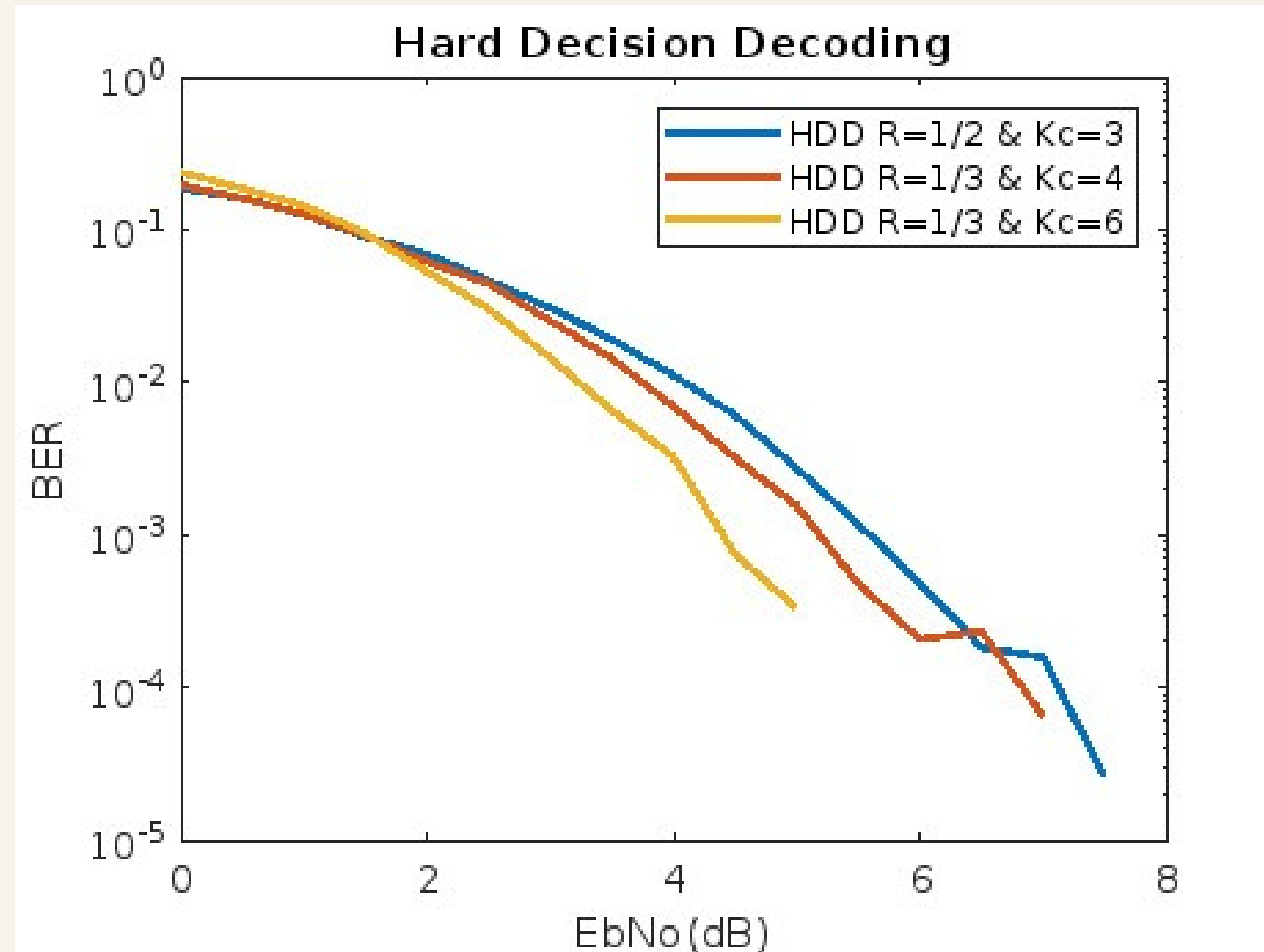
GRAPH : $R=1/3$ AND $KC=4$



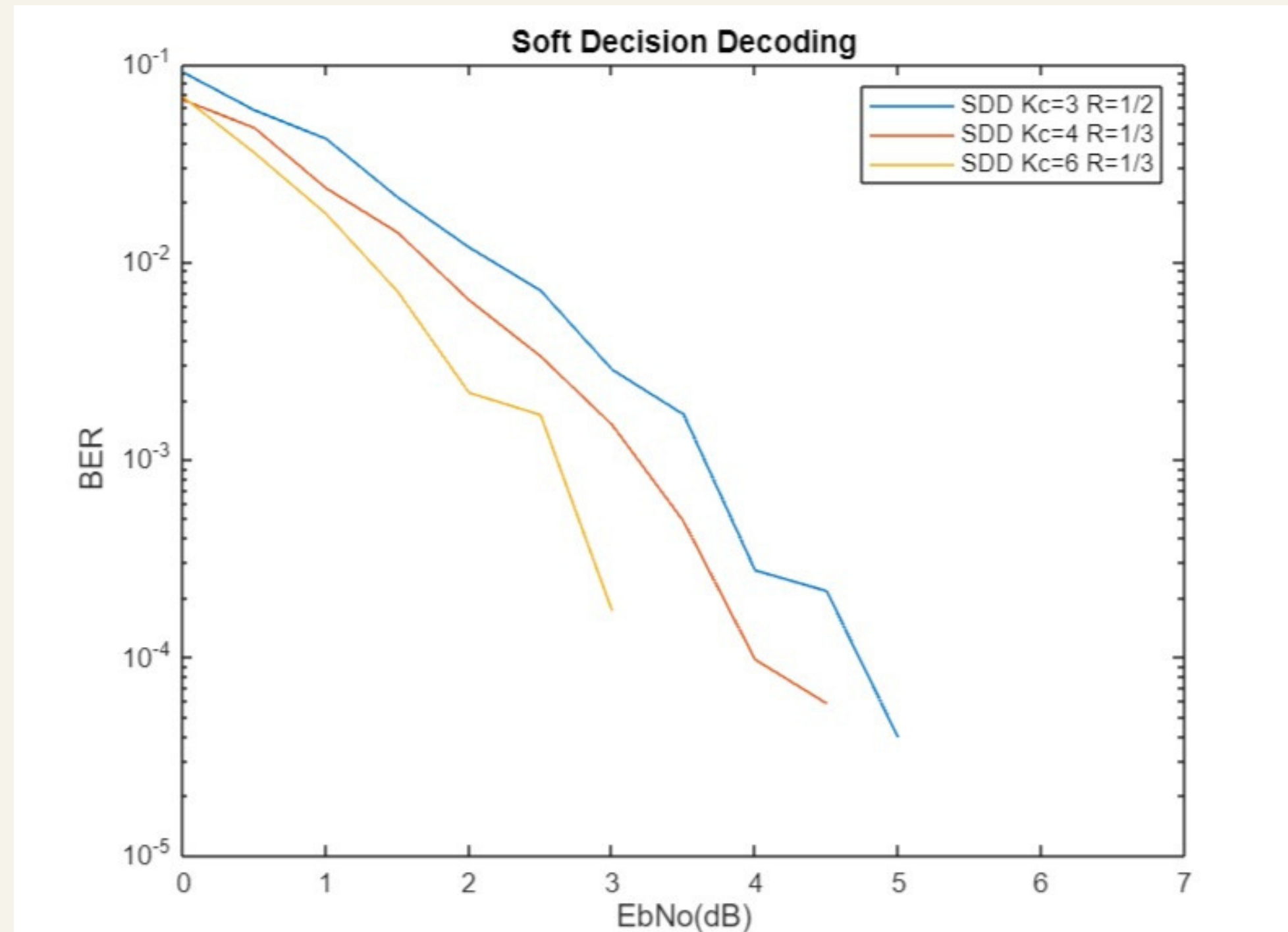
GRAPH : $R=1/3$ AND $KC=6$



GRAPH : FOR ALL RATES(HDD)



GRAPH : FOR ALL RATES(SDD)



RESULTS AND EXPLANATION

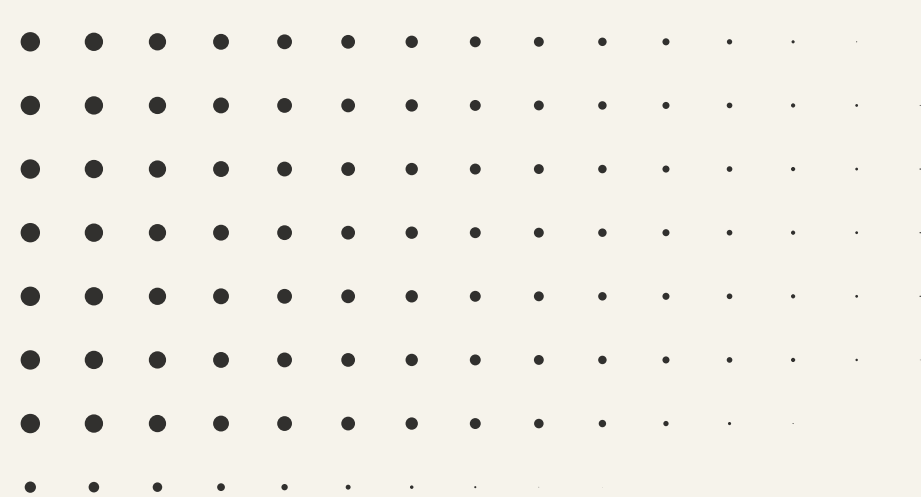
- > The performance of the Viterbi for each hard and soft increases with an increase in SNR.
- > Viterbi Soft decision decoding performs better than Viterbi Hard decision decoding.
- > Bit Error Rate (BER) and Probability of Detection Error (PDE) will be high for the Lower value of SNR and will be low for the high value of SNR.
- > Increasing the constraint length significantly improves error correction capabilities, but it comes at the cost of significantly higher computational complexity, leading to increased processing time.
- > What are the reasons that Soft decision decoding is better than hard decision decoding?
 - As in hard decision decoding, each received symbol is demodulated into bits based on a set threshold, which can introduce ambiguity as the decoder might select incorrect bits due to uncertainty.
 - Soft decision decoding utilizes the received symbols directly, rather than demodulating them into bits. Unlike Viterbi hard decision decoding, there is no ambiguity in soft decision decoding because it works with the continuous values of the received symbols.

ADVANTAGES OF CONVOLUTION CODE

- Convolution coding is a technique used in digital communications to correct errors in data mistakes.
- Convolutional coding performs better when you have higher probability rates and noisy channels.
- Using the concept of convolutional coding scheme we are able to remove the most of the errors from the message.
- Convolution coding scheme is more suitable for digital communication system because it works well in continuous transmission of data.

REFERENCES

- [1] Digital Communications, John G. Proakis, 4th Edition, 1995.
- [2] CT216. Introduction to communication system: Channel Coding for IEEE 802.16e Mobile WiMAX by prof. Matthew C. Valenti .
- [3] Lectures of MIT :
<https://web.mit.edu/6.02/www/f2010/handouts/lectures/L8.pdf>
- [4] Video lectures on Convolutional coding by Subhramanyam K N :
<https://shorturl.at/hZ136>



Thank You !

