|  |  |  |  |
| --- | --- | --- | --- |
| **Course Name:** | **Information Theory and Coding Techniques** | **Semester:** | **V** |
| **Date of Performance:** | **24 / 09 / 2024** | **Batch No:** | **B – 1** |
| **Faculty Name:** | **Prof. Makrand Kulkarni** | **Roll No:** | **16014022050** |
| **Faculty Sign & Date:** |  | **Grade/Marks:** | **\_\_\_ / 25** |

**Experiment No: 7**

**Title:** To generate and study systematic and non-systematic cyclic codes using MATLAB

|  |
| --- |
| **Aim and Objective of the Experiment:** |
| * To understand the error detecting and correcting codes * To implement Cyclic code * To visualize the Cyclic code output to make appropriate conclusions |

|  |
| --- |
| **COs to be achieved:** |
| **CO3:** To apply different error correcting codes for digital transmission |

|  |
| --- |
| **Theory:** |
| In today's era of digital communication, various types of codes are used. The use of these codes has various advantages. Cyclic codesare [linear](http://en.wikipedia.org/wiki/Linear_code) [block](http://en.wikipedia.org/wiki/Block_code) error-correcting codes that have convenient algebraic structures for efficient error detection and correction. They can be efficiently implemented using simple shift register.  In systematic cyclic coding, the code is calculated as,  C(x) = x^(n-k)d(x)+ p(x)  Where, d(x) = data polynomial  p(x) = Check bit polynomial  p(x) = Remainder{x^(n-k)d(x) / g(x)}  g(x) = Generator polynomial  In Non-systematic cyclic coding, the code is calculated as,  C(x) = d(x) g(x)  Where, d(x) = data polynomial,  g(x) = Generator polynomial |

|  |
| --- |
| **Step-wise Procedure:** |
| **Steps of algorithm:**   1. Start the program 2. Enter total length of required code 3. Input the message bits as d(x) 4. Compute the code word C(x)=x^(n-k)d(x)+ p(x) 5. Display generator and output 6. Compute the generator matrix 7. Stop   **SIMULATION:**   1. Attach the MATLAB simulation results for the same. 2. Prove the results with theoretical solutions. |

|  |
| --- |
| **Observations:** |
| 1. **Code:**   import numpy as np  def polynomial\_division(dividend, divisor):      """Perform polynomial division and return the remainder"""      dividend = np.array(dividend, dtype=int)      divisor = np.array(divisor, dtype=int)        temp = dividend.copy()      n = len(dividend)      k = len(divisor)        for i in range(n - k + 1):          if temp[i] == 1:              for j in range(k):                  temp[i + j] ^= divisor[j]        # Return the remainder      return temp[-(k-1):] if k > 1 else []  def generate\_systematic\_cyclic\_code(message\_bits, generator\_poly):      """Generate systematic cyclic code"""      k = len(message\_bits)  # Message length      n = len(generator\_poly) + k - 1  # Total code length        # Step 1: Append (n-k) zeros to message bits      augmented\_message = np.append(message\_bits, np.zeros(n-k, dtype=int))        # Step 2: Find remainder      remainder = polynomial\_division(augmented\_message, generator\_poly)        # Step 3: Generate codeword = message bits + remainder      codeword = np.zeros(n, dtype=int)      codeword[:k] = message\_bits      if len(remainder) > 0:          codeword[k:] = remainder        return codeword  def print\_polynomial(coefficients, var='x'):      """Pretty print a polynomial"""      terms = []      for i, coef in enumerate(coefficients):          if coef == 1:              if i == 0:                  terms.append('1')              elif i == 1:                  terms.append(var)              else:                  terms.append(f'{var}^{i}')      if not terms:          return '0'      return ' + '.join(reversed(terms))  def generate\_generator\_matrix(generator\_poly, k, n):      """Generate the generator matrix G"""      G = np.zeros((k, n), dtype=int)        for i in range(k):          shift = np.zeros(n, dtype=int)          shift[i:i+len(generator\_poly)] = generator\_poly          G[i] = shift        return G  print("Input Parameters:")  print("-----------------------")  print("1. Generator polynomial g(x) = x^3 + x + 1")  print("2. Message bits = [1, 0, 1, 1]")  print("3. Message length k = 4")  print("\nOutput:")  print("--------------")  # Parameters  generator\_poly = [1, 0, 1, 1]  # x^3 + x + 1  message\_bits = [1, 0, 1, 1]  k = len(message\_bits)  # Message length  n = len(generator\_poly) + k - 1  # Total code length  # Generate systematic cyclic code  codeword = generate\_systematic\_cyclic\_code(message\_bits, generator\_poly)  print(f"\nMessage polynomial d(x) = {print\_polynomial(message\_bits)}")  print(f"Generator polynomial g(x) = {print\_polynomial(generator\_poly)}")  print(f"Generated codeword c(x) = {print\_polynomial(codeword)}")  print("\nIn binary form:")  print(f"Message bits: {message\_bits}")  print(f"Codeword: {list(map(int, codeword))}")  # Generate and display generator matrix  G = generate\_generator\_matrix(generator\_poly, k, n)  print("\nGenerator Matrix G:")  print(G)  print(f"\nCode parameters:")  print(f"Message length (k): {k}")  print(f"Code length (n): {n}")  print(f"Number of parity bits (n-k): {n-k}")   1. **Output of Code:** |

|  |
| --- |
| **Post Lab Subjective/Objective type Questions:** |
| 1. **For a (7, 4) cyclic code, the generator polynomial is g(x) = x^3+x^2+1. Sketch the shift register implementation for its encoder.**      1. **Compare systematic cyclic code with non-systematic cyclic code.** 2. Structure:    1. Systematic Cyclic Code: The original data (message bits) appears unchanged in the codeword, followed by the parity (check bits).    2. Non-Systematic Cyclic Code: The codeword is a transformed version of the message bits where the original data is not directly visible. 3. Encoding Complexity:    1. Systematic Cyclic Code: Easier encoding, as the message is included directly in the codeword, making error detection simpler.    2. Non-Systematic Cyclic Code: Encoding involves more complex transformations since the message is embedded in a different form. 4. Decoding:    1. Systematic Cyclic Code: Easier to decode because the message part is directly retrievable from the codeword.    2. Non-Systematic Cyclic Code: Requires additional steps to recover the original message due to the lack of a clear message-parity separation. 5. Usage:    1. Systematic Cyclic Code: Preferred when quick message extraction and error detection are needed.    2. Non-Systematic Cyclic Code: Used when the overall code efficiency or structure requires a fully transformed codeword. |

|  |
| --- |
| **Conclusion:** |
| We have successfully understood how to generate and study systematic and non-systematic cyclic codes. |

|  |
| --- |
| **Signature of faculty in-charge with Date:** |