

Experiment No: 6	
Name	Vaibhav Sharma
PRN	22070126125
Date of Performance	9/10/2024
Title	Write a program to generate CRC code for checking error.
Theory (short)	Cyclic Redundancy Check (CRC) is an error-detection technique used in digital networks and storage devices to detect accidental changes in raw data. It works by appending a sequence of redundant bits, known as the CRC code, to the data. These bits are derived from polynomial division of the data bits by a predefined key. At the receivers end, the data is checked by redoing the division. If the remainder is zero, the data is error-free; otherwise, errors are detected.
Procedure	<pre> import numpy as np def xor(a, b): return [i ^ j for i, j in zip(a, b)] def encode(data_input , key): data = data_input[:] temp = data + [0, 0, 0] if data[0] == 1: m = [1] temp = xor(temp, key) else: m = [0] temp = xor(temp, [0, 0, 0, 0]) for _ in range(3): temp = temp[1:] + [0] if temp[0] == 1: m.append(1) temp = xor(temp, key) else: m.append(0) temp = xor(temp, [0, 0, 0, 0]) print("Remainder: ", temp) </pre>

```

data_crc = data + temp[1:]
print("encoded Data with CRC: ", data_crc)
return data_crc

def decode(data_input , key):
    data = data_input[:]
    temp = data

    if data[0] == 1:
        m = [1]
        temp = xor(temp, key)
    else:
        m = [0]
        temp = xor(temp, [0, 0, 0, 0])

    for _ in range(3):
        temp = temp[1:] + [0]
        if temp[0] == 1:
            m.append(1)
            temp = xor(temp, key)
        else:
            m.append(0)
            temp = xor(temp, [0, 0, 0, 0])

    print("Remainder: ", temp)
    return temp[1:]

data_input = [int(x) for x in input("Enter data bits
separated by space: ").split()]
key = [int(x) for x in input("Enter the key separated by
space: ").split()]
encode(data_input, key)

encode_data_input = [int(x) for x in input("Enter encoded
data bits separated by space: ").split()]
if decode(encode_data_input, key) == [0, 0, 0]:
    print("no error")
else:
    print("error")

```

1. xor(a, b):

- This function performs a bitwise XOR operation between two lists a and b. Each pair of bits from both lists is XORed and stored in a new list, which is returned.

- Example: `xor([1, 0, 1], [0, 1, 1])` results in `[1, 1, 0]`.

2. `encode(data_input, key):`

- Inputs:

- `data_input`: A list representing the data bits to be transmitted.

- `key`: A list representing the generator polynomial (CRC key).

- Operation:

1. A copy of `data_input` is created and padded with three 0s (since the CRC key is of size 4).

2. For each bit of the data, it checks if the current bit is 1 and performs XOR with the key. If the bit is 0, it performs XOR with `[0, 0, 0, 0]`.

3. The process continues for the remaining bits of the padded data.

- Output:

- The program prints the remainder from the division process.

- The CRC is appended to the original data, and the program prints the encoded data.

- The encoded data is returned, consisting of the original data plus the calculated CRC.

3. `decode(data_input, key):`

- Inputs:

- `data_input`: The encoded data (original data plus CRC) received after transmission.

- `key`: The same generator polynomial used in encoding.

- Operation:

- Similar to the encoding process, this function checks if the remainder of the received data, after dividing by the key using XOR, is `[0, 0, 0]` (indicating no error).

- Output:

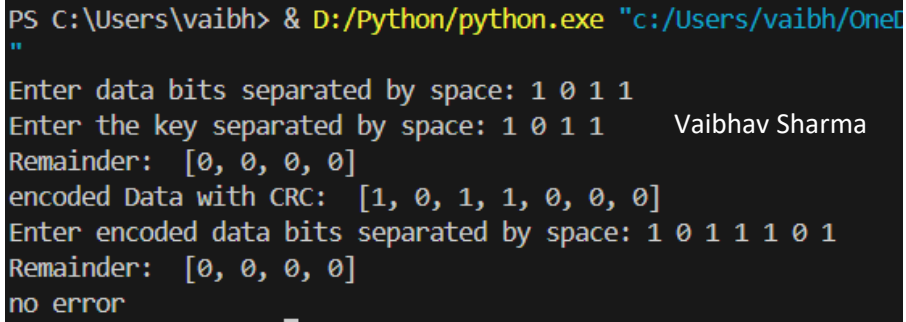
- If the remainder is `[0, 0, 0]`, it prints "no error"; otherwise, it prints "error" indicating that the data has been corrupted.

1. Encoding:

- The user is prompted to input the data bits and the key (CRC polynomial).

- The encode function is called, which generates the CRC and appends it to the data.

- The encoded data (original data + CRC) is printed.

	<p>2. Decoding:</p> <ul style="list-style-type: none"> - After transmission, the user is prompted to input the received (encoded) data. - The decode function is called to check if the received data has any errors. - If the remainder is [0, 0, 0], the program prints "no error", otherwise it detects an error. <p>- The process involves dividing the data by a key using XOR operation, where the remainder is used as the CRC (Cyclic Redundancy Check) bits.</p> <p>- If the remainder of the division at the receiver is zero, the transmission is error-free. If not, an error is detected.</p>
Output Screenshots	 <pre> PS C:\Users\vaibh> & D:/Python/python.exe "c:/Users/vaibh/OneD " Enter data bits separated by space: 1 0 1 1 Enter the key separated by space: 1 0 1 1 Vaibhav Sharma Remainder: [0, 0, 0, 0] encoded Data with CRC: [1, 0, 1, 1, 0, 0, 0] Enter encoded data bits separated by space: 1 0 1 1 1 0 1 Remainder: [0, 0, 0, 0] no error </pre>

<p>Self-assessment Q&A</p>	<p>Q: What is the purpose of CRC in data transmission?</p> <p>Ans: CRC is used to detect accidental changes or errors in data transmission by appending a redundant code derived from polynomial division.</p> <p>Q: How is the CRC code generated?</p> <p>Ans: The CRC code is generated by dividing the data bits by a predefined polynomial key and appending the remainder as the CRC code to the original data.</p> <p>Q: How does the receiver verify the integrity of the data using CRC?</p> <p>Ans: The receiver performs the same polynomial division on the received data, and if the remainder is zero, it indicates that the data is error-free; otherwise, errors are detected.</p>
<p>Conclusion</p>	<p>The experiment successfully demonstrates the generation and verification of CRC (Cyclic Redundancy Check) codes for error detection in data transmission. By applying the XOR operation using a given generator polynomial (key), the program appends CRC bits to the data. During transmission, the receiver decodes the data and checks for errors by recalculating the remainder. If the remainder is zero, the data is considered error-free. This method provides an efficient and reliable way to detect errors in digital communication systems.</p>