Computer Networks Lab Report – Assignment 1

Name: Tanmoy Sarkar

Roll No: 002010501020

Class: BCSE 5th semester 3rd Year

Group: A1

Problem statement: Design and implement an error detection module

Design and implement an error detection module that has four schemes namely LRC, VRC, Checksum, and CRC. Please note that you may need to use these schemes separately for other applications (assignments). You can write the program in any language. The Sender program should accept the name of a test file (contains a sequence of 0,1) from the command line. Then it will prepare the data frame (decide the size of the frame) from the input. Based on the schemes, codewords will be prepared. The sender will send the codeword to the Receiver. The receiver will extract the data word from the codeword and show if there is any error detected. Test the same program to produce a PASS/FAIL result for the following cases (not limited to).

A. Error is detected by all four schemes. Use a suitable CRC polynomial (the list is given on the next page).
B. Error is detected by checksum but not by CRC.
C. Error is detected by VRC but not by CRC.

[Note: Inject error in random positions in the input data frame. Write a separate method for that.]

Submission Date: 07-08-2022

Deadline Date: 07-08-2022

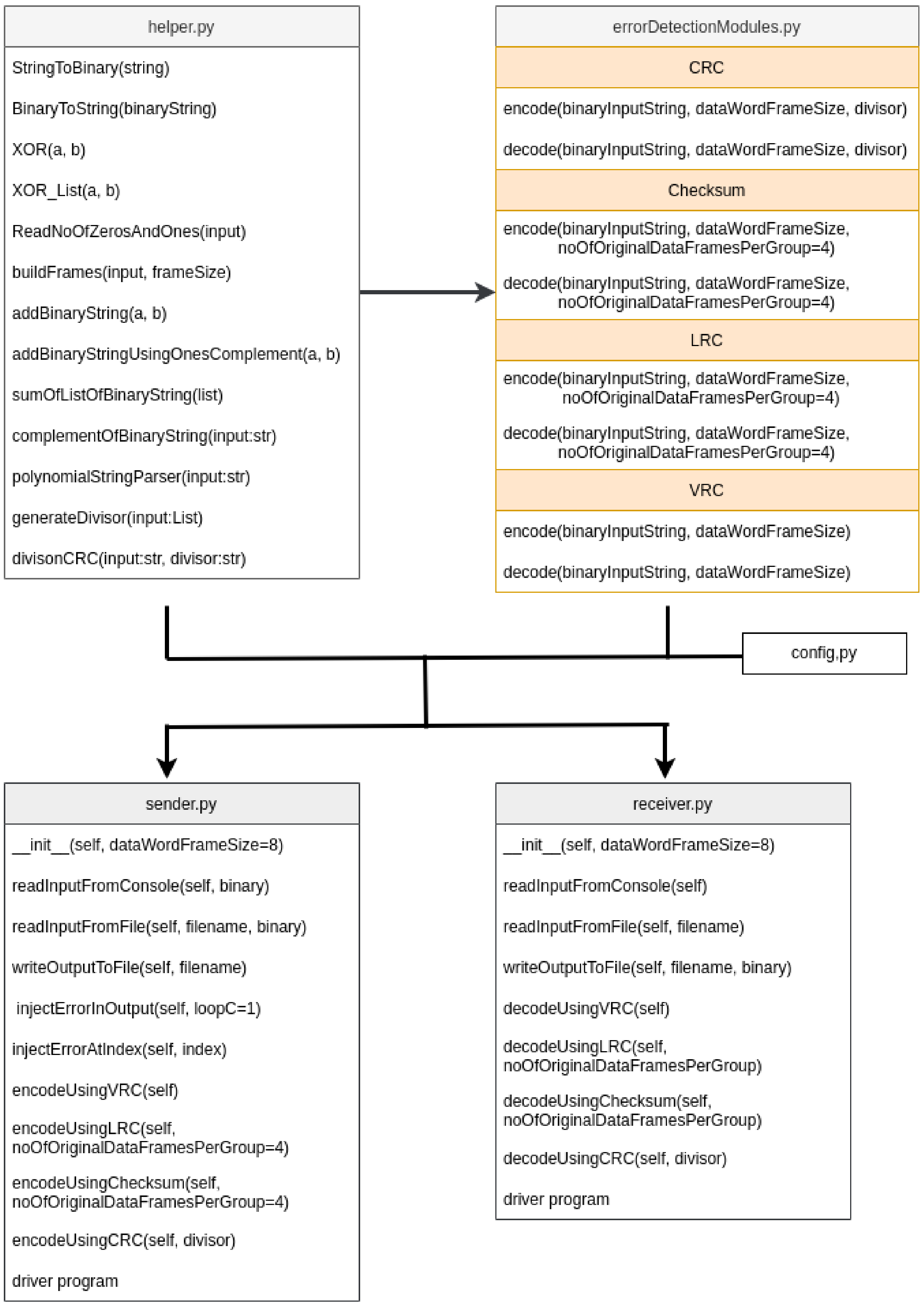
DESIGN

Purpose: Data is usually constructed by long strings of ‘1’s and ‘0’s. While traveling through a communication medium, due to electromagnetic interference, noise is introduced in data and the bits in data changed from ‘1’ to ‘0’ or vice-versa. So, on the receiver side, during the decoding of signals, the affected bits should be detected and corrected [if possible].

In this assignment, we will discuss the following error detection techniques.

1. VLC (Vertical Redundancy check)
2. LRC (Longitudinal Redundancy check)
3. Checksum (Checksum)
4. CRC (Cyclic Redundancy check)

Structure Diagram to Reflect The Procedural Organization of The Program



helper.py
StringToBinary(string)
BinaryToString(binaryString)
XOR(a, b)
XOR_List(a, b)
ReadNoOfZerosAndOnes(input)
buildFrames(input, frameSize)
addBinaryString(a, b)
addBinaryStringUsingOnesComplement(a, b)
sumOfListOfBinaryString(list)
complementOfBinaryString(input:str)
polynomialStringParser(input:str)
generateDivisor(input:List)
divisonCRC(input:str, divisor:str)
errorDetectionModules.py
CRC
encode(binaryInputString, dataWordFrameSize, divisor)
decode(binaryInputString, dataWordFrameSize, divisor)
Checksum
encode(binaryInputString, dataWordFrameSize, noOfOriginalDataFramesPerGroup=4)
decode(binaryInputString, dataWordFrameSize, noOfOriginalDataFramesPerGroup=4)
LRC
encode(binaryInputString, dataWordFrameSize, noOfOriginalDataFramesPerGroup=4)
decode(binaryInputString, dataWordFrameSize, noOfOriginalDataFramesPerGroup=4)
VRC
encode(binaryInputString, dataWordFrameSize)
decode(binaryInputString, dataWordFrameSize)
config.py
sender.py
__init__(self, dataWordFrameSize=8)
readInputFromConsole(self, binary)
readInputFromFile(self, filename, binary)
writeOutputToFile(self, filename)
injectErrorInOutput(self, loopC=1)
injectErrorAtIndex(self, index)
encodeUsingVRC(self)
encodeUsingLRC(self, noOfOriginalDataFramesPerGroup=4)
encodeUsingChecksum(self, noOfOriginalDataFramesPerGroup=4)
encodeUsingCRC(self, divisor)
driver program
receiver.py
__init__(self, dataWordFrameSize=8)
readInputFromConsole(self)
readInputFromFile(self, filename)
writeOutputToFile(self, filename, binary)
decodeUsingVRC(self)
decodeUsingLRC(self, noOfOriginalDataFramesPerGroup)
decodeUsingChecksum(self, noOfOriginalDataFramesPerGroup)
decodeUsingCRC(self, divisor)
driver program

The error detection module is split into 5 files.

- helper.py (It has various methods, which is used by other programs at a different position)
- errorDetectionModules.py (It consists of decode and encode methods of VRC, LRC, Checksum, CRC)
- config.py (Dictionary of CRC Code and Polynomials)
- sender.py (Sender program, including a driver program to act as an interface to sender and receiver)
- receiver.py (Receiver program, included a driver program to act as an interface to sender and receiver)

Input and Output Format

1. By running the *sender.py* and following the program instructions on the screen, we can take input from the console or from the file and then select any encoding method out of VRC, CRC, LRC, and Checksum. We can opt for injecting errors at random indexes or selected indexes [manually]. In the end, the output will be stored in a file and the same will be displayed in the console.
2. By running the *receiver.py* and following the program instructions on the screen, we can take input from the console or read from the file and select the preferred decoding technique out of VRC, CRC, LRC, and Checksum. The program will decode the data. If any error occurred the frame will not be added to the output. The final data will be stored in a file and also shown to the console. We can see the error status on the console. If no error is found, we will get "*PASS*" and if the error is found, we will get "FAILED".

IMPLEMENTATION

Code snippets of *helper.py*

```
from functools import cmp_to_key
from typing import List, Tuple

def StringToBinary(string):
    return ''.join(format(ord(x), '08b') for x in string)

def BinaryToString(binary):
    return ''.join(chr(int(binary[i:i+8], 2)) for i in range(0,
len(binary), 8))

def XOR(a, b):
    if a == b:
        return '0'
    return '1'

def XOR_List(a, b):
    return str(int(''.join([XOR(a[i], b[i]) for i in
range(len(a))])))

def ReadNoOfZerosAndOnes(input):
    noOfZeros = 0
    noOfOnes = 0

    for i in input:
        if i == '0':
            noOfZeros += 1
        elif i == '1':
            noOfOnes += 1

    return noOfZeros, noOfOnes

def buildFrames(input, frameSize):
    output = []
    for i in range(0, len(input), frameSize):
        output.append(input[i:i+frameSize])
    return output
```

```python
def addBinaryString(a, b):
    max_len = max(len(a), len(b))
    a = a.zfill(max_len)
    b = b.zfill(max_len)
    result = ''
    carry = 0

    # Traverse the string
    for i in range(max_len - 1, -1, -1):
        r = carry
        r += 1 if a[i] == '1' else 0
        r += 1 if b[i] == '1' else 0
        result = ('1' if r % 2 == 1 else '0') + result

        # Compute the carry.
        carry = 0 if r < 2 else 1

    if carry != 0:
        result = '1' + result

    return result.zfill(max_len)

def addBinaryStringUsingOnesComplement(a, b):
    max_len = max(len(a), len(b))
    a = a.zfill(max_len)
    b = b.zfill(max_len)
    result = ''
    carry = 0

    # Traverse the string
    for i in range(max_len - 1, -1, -1):
        r = carry
        r += 1 if a[i] == '1' else 0
        r += 1 if b[i] == '1' else 0
        result = ('1' if r % 2 == 1 else '0') + result

        # Compute the carry.
        carry = 0 if r < 2 else 1

    if carry != 0:
        result = addBinaryStringUsingOnesComplement(result, '1')
    return result.zfill(max_len)

def sumOfListOfBinaryString(l):
    result = l[0]
    for i in range(1, len(l)):
        result = addBinaryString(result, l[i])
    return result

def complementOfBinaryString(input:str):
    input = list(input)
    for i in range(len(input)):
        if input[i] == '0':
            input[i] = '1'
        else:
            input[i] = '0'
    return ''.join(input)

# Return sorted list of (power, coefficient)
# Valid polynomial format
# x^4 + x^2 + x^1 + 1
# return polynomials in form of list of (power, coefficient) in
descending order of power
def polynomialStringParser(input:str):
    # Raise exception for invalid input
    if input == '' :
        raise Exception("Invalid input")
    if input.find('x') != -1 and input.find('x^') == -1:
        raise Exception("Invalid input")

    # Main logic
    data = []

    # Iterate over list of (Splitted at the '+').removeSpaces
    for d in [j.strip() for j in input.split('+')]:
        if len(d) == 1:
            data.append((0, int(d)))
        else:
            tmp = d.split('x^')
            if len(tmp) == 1:
                data.append((int(tmp[0]), 1))
            elif len(tmp) == 2:
                if tmp[0] == '':
                    data.append((int(tmp[1]), 1))
                else:
                    data.append((int(tmp[1]), int(tmp[0])))

    # Sort the list by degree
    data = sorted(data, key=cmp_to_key(lambda item1, item2:
item2[0] - item1[0]))

    # Fill blank degress with 0 coefficients
    final_data = []

    i=0
    while i < len(data)-1:
        final_data.append(data[i])
        a = data[i][0]
        b = data[i+1][0]

        if a-b > 1:
            for j in range(b+1, a)[::-1]:
                final_data.append((j, 0))
        i+=1
    final_data.append(data[i])
    return final_data

# Accept polynomials in form of list of (power, coefficient) in
descending order of power
def generateDivisor(input:List):
    divisor = ''
    for i in input:
        divisor += ("1" if i[1] != 0 else "0")
    return divisor

# XOR List
def xor_list(a, b):

   # initialize result
   result = []

   # Traverse all bits, if bits are
   # same, then XOR is 0, else 1
   for i in range(1, len(a)):
       if a[i] == b[i]:
           result.append('0')
       else:
           result.append('1')

   return ''.join(result)

# Divide function for CRC
def divisonCRC(input:str, divisor:str):
   input = str(int(input))
   divisor = str(int(divisor))

   # If input is longer than divisor, return input
   if len(input) < len(divisor):
       return input

   pick = len(divisor)
   tmp = input[0 : pick]

   while pick < len(input):
       if tmp[0] == '1':
           tmp = xor_list(tmp, divisor) + input[pick]
       else:
           tmp = xor_list(tmp, '0'*pick) + input[pick]

       pick += 1

   if tmp[0] == '1':
       tmp = xor_list(tmp, divisor)
   else:
       tmp = xor_list(tmp, '0'*pick)

   return str(int(tmp))
```

Method descriptions of helper.py:

- StringToBinary(string) : This method is used to convert string to binary by traversing character-by-character using the built-in ord() and format() function of python.
- BinaryToString(binaryString) : This method is used to convert long binary string [e.g 11001001…..] to readable string format [e.g. Hi …]

- XOR(a,b): This method is used to XOR two bits
- XOR_List(a,b): This method is used to XOR two strings of bits by iterating over the strings and applying XOR(a,b) function. Example: XOR_List('110010', '001011') will return '111001'
- ReadNoOfZerosAndOnes(input): This method will count no of zeros and no of ones in binary string and will return a tuple of (no of zeros, no of ones). Example : ReadNoOfZerosAndOnes('110010') will return (3,3)
- buildFrames(input, frameSize): This method accept a string of '1' and '0' and frameSize. The method will split the string of '1' and '0' in frames of the length of `frameSize` & will return a list of frames. It will also add padding of '0's if there is not enough length to generate the frames. Example: buildFrames('10111110',4) will return ["1011", "1110"]
- addBinaryString(a,b): This method is used to add two binary strings. Example: addBinaryString("10101010", "11001100") will return "101110110"
- addBinaryStringUsingOnesComplement(a, b): This method is used to add two binary strings using 1s complement.
- sumOfListOfBinaryString(list): This method accepts a list of binary strings and returns the sum of binary strings
- complementOfBinaryString(input:str): This method takes a binary string and returns the complement of binary. complementOfBinaryString("110010") will return "1101"
- polynomialStringParser(input:str): This method accepts an polynomial string [like x^4 + x^2 + x^1 + 2] and return a list of tuples of (power, coefficient) in an sorted order . [(4,1),(3,0),(2,1),(1,1),(0,2)]
- generateDivisor(input:List): This method accepts list of tuples of (power, coefficient) of a polynomial string and generate the divisor. For an example, generateDivisor([(4,1),(3,0),(2,1),(1,1),(0,2)]) will return 10111
- divisonCRC(input:str, divisor:str): This method takes two arguments input and divisor. After dividing, the input [dividend] by divisor by

XORing, it will return the remainder. It is a helper function for CRC encoding and decoding.

Code Snippets of errorDetectionModules.py

```
from math import remainder
from helper import ReadNoOfZerosAndOnes,
addBinaryStringUsingOnesComplement, buildFrames,
complementOfBinaryString, divisonCRC


class VRC:
    @staticmethod
    def encode(binaryInputString:str, dataWordFrameSize:int):
        # ? Even Parity VRC
        output = ""
        for i in buildFrames(binaryInputString, dataWordFrameSize):
            _ , noOfOnes = ReadNoOfZerosAndOnes(i)
            output += i + ("1" if noOfOnes % 2 == 1 else "0")
        return output

    @staticmethod
    def decode(binaryInputString:str, dataWordFrameSize):
        frames = buildFrames(binaryInputString, dataWordFrameSize+1)
        errorFound = False
        output = ""
        for i in frames:
            # ? Even Parity VRC
            _ , noOfOnes = ReadNoOfZerosAndOnes(i)
            if noOfOnes % 2 == 0:
                i = i[:-1]
                output += i
            else:
                errorFound = True

        return output, errorFound

class LRC:
    @staticmethod
    def encode(binaryInputString:str, dataWordFrameSize:int,
noOfOriginalDataFramesPerGroup:int=4):
        output = ""
        frames = buildFrames(binaryInputString, dataWordFrameSize)
        # Check whether we can split in to equal length of frames
        noOfFramesRequiredToBeAdded = noOfOriginalDataFramesPerGroup -
```

```
len(frames)%noOfOriginalDataFramesPerGroup
        if noOfFramesRequiredToBeAdded > 0 and
noOfFramesRequiredToBeAdded != noOfOriginalDataFramesPerGroup:
            for _ in range(noOfFramesRequiredToBeAdded):
                frames.append("0"*dataWordFrameSize)

        # Iterate over the frames to calculate the parity
        for i in range(0, len(frames), noOfOriginalDataFramesPerGroup):
            tmpFrames = frames[i:i+noOfOriginalDataFramesPerGroup]
            parity = ""
            for index in range(dataWordFrameSize)[::-1]:
                noOfOnes = 0
                # Count no of ones in frame
                for frame in tmpFrames:
                    if frame[index] == "1":
                        noOfOnes += 1
                # Add parity bit
                parity = ("1" if noOfOnes % 2 != 0 else "0") + parity


            # Append to output
            output = output + ''.join(tmpFrames) + parity

        return output

    @staticmethod
    def decode(binaryInputString:str, dataWordFrameSize:int,
noOfOriginalDataFramesPerGroup:int=4):
        # Split frames from data
        frames = buildFrames(binaryInputString, dataWordFrameSize)
        errorFound = False
        output = ""
        # Iterate over the frames
        for i in range(0, len(frames),
noOfOriginalDataFramesPerGroup+1):
            tmp_frames = frames[i:i+noOfOriginalDataFramesPerGroup+1]
            parity_frame = tmp_frames[-1]
            frameErrorFound = False
            # Verify parity
            for index in range(dataWordFrameSize)[::-1]:
                noOfOnes = 0
                for frame in tmp_frames[:-1]:
                    noOfOnes = noOfOnes + (1 if frame[index] == '1' else
0)
                noOfOnes = noOfOnes + (1 if parity_frame[index] == '1'
else 0)
```

```
                if noOfOnes % 2 != 0:
                    frameErrorFound = True
                    break
            if frameErrorFound:
                errorFound = True
            else:
                output += ''.join(tmp_frames[:-1])

        return output, errorFound

class CheckSum:
    @staticmethod
    def encode(binaryInputString:str, dataWordFrameSize:int,
noOfOriginalDataFramesPerGroup:int=4):
        output = ""
        frames = buildFrames(binaryInputString, dataWordFrameSize)
        # Check whether we can split in to equal length of frames
        noOfFramesRequiredToBeAdded = noOfOriginalDataFramesPerGroup -
len(frames)%noOfOriginalDataFramesPerGroup
        if noOfFramesRequiredToBeAdded > 0 and
noOfFramesRequiredToBeAdded != noOfOriginalDataFramesPerGroup:
            for _ in range(noOfFramesRequiredToBeAdded):
                frames.append("0"*dataWordFrameSize)

        # Iterate over the frames to calculate the checksum
        for i in range(0, len(frames), noOfOriginalDataFramesPerGroup):
            tmpFrames = frames[i:i+noOfOriginalDataFramesPerGroup]
            tmp = tmpFrames[0]
            frameString = ""
            for frame in tmpFrames[1:]:
                tmp = addBinaryStringUsingOnesComplement(tmp, frame)
                frameString += frame

            # Complement and get checksum
            checksum = complementOfBinaryString(tmp)
            output = output + tmpFrames[0] + frameString + checksum

        return output

    @staticmethod
    def decode(binaryInputString:str, dataWordFrameSize:int,
noOfOriginalDataFramesPerGroup:int=4):
        output = ""
        frames = buildFrames(binaryInputString, dataWordFrameSize)
        errorFound = False
```

```python
        # Iterate over the frames to calculate the checksum and verify
        for i in range(0, len(frames), 
noOfOriginalDataFramesPerGroup+1):
            tmpFrames = frames[i:i+noOfOriginalDataFramesPerGroup+1]
            tmp = tmpFrames[0]
            frameString = ""
            for frame in tmpFrames[1:]:
                tmp = addBinaryStringUsingOnesComplement(tmp, frame)
                frameString += frame

            # Complement and get checksum
            checksum = complementOfBinaryString(tmp)
            if checksum == "0"*dataWordFrameSize:
                output += (tmpFrames[0] + 
frameString[:-dataWordFrameSize])
            else:
                errorFound = True

        return output, errorFound

class CRC:
    @staticmethod
    def encode(binaryInputString:str, dataWordFrameSize:int, 
divisor:str):
        output = ""
        crcSize = len(divisor)-1
        for i in buildFrames(binaryInputString, dataWordFrameSize):
            tmp = i+crcSize*"0" # [max degree of polynomial] times 0
            crc = divisonCRC(tmp, divisor)[:crcSize]
            output += i+crc.zfill(crcSize)
        return output

    @staticmethod
    def decode(binaryInputString:str, dataWordFrameSize:int, 
divisor:str):
        output = ""
        errorFound = False
        crcSize = len(divisor)-1
        for i in buildFrames(binaryInputString, 
dataWordFrameSize+crcSize):
            remainder = divisonCRC(i, divisor)
            if remainder == 0 or remainder == '0':
                messageData = i[:-crcSize]
                # print(messageData)
                output += messageData
            else:
                errorFound = True

        return output, errorFound
```

Classes of “errorDetectionModules.py”

1. VRC

   Methods Description

   a. encode(binaryInputString:str, dataWordFrameSize: int): This method first builds data words from binaryInputString and then calculates the parity bits, and generates the codeword.
   b. decode(binaryInputString:str, dataWordFrameSize): This method splits the binaryInputString in codewords and checks the parity bits of each codeword and checks for any error.
2. LRC

   Methods Description

   a. encode(binaryInputString:str, dataWordFrameSize:int, noOfOriginalDataFramesPerGroup:int=4): This method first builds data words from binaryInputString and then calculates the parity bits and generates the frame with parity bits. Then construct the output to be sent to the receiver.
   b. decode(binaryInputString:str, dataWordFrameSize:int, noOfOriginalDataFramesPerGroup:int=4): This method split binaryInputString into frames and checks the frame with parity bits to check for any error and then deletes the frame with parity bits, constructs the output and return.
3. Checksum

   Methods Description

   a. encode(binaryInputString:str, dataWordFrameSize:int, noOfOriginalDataFramesPerGroup:int=4): This method split binaryInputString in frames. Then Iterate over the frames to calculate the sum of frames by one’s complement method. After that, the sum is updated with its complement and constructs the final output.

b. decode(binaryInputString:str, dataWordFrameSize:int, noOfOriginalDataFramesPerGroup:int=4): This method split binaryInputString in frames.Then Iterate over the frames to calculate the sum of frames by one's complement method. After that, the sum is updated with its complement. If the result is 0, then the data has no error else there may be some sort of error.

4. CRC

Methods Description

a. encode(binaryInputString:str, dataWordFrameSize:int, divisor:str) : This function split binaryInputString in datawords. Then append no of zeros based on the CRC polynomial at the end of dataword. The using `divisonCRC` method calculates the CRC and builds the codeword.

b. decode(binaryInputString:str, dataWordFrameSize:int, divisor:str): First it splits binaryInputString in codewords. For each codeword, it divide the codeword with the divider using `divisonCRC` method, if the remainder comes 0 , then the codeword has no error else it has some sort of error.

Code Snippets of "config.py"

```
availableCRCPolynomials = {
    "CRC_1": "x+1",
    "CRC_4_ITU"   : "x^4 + x^1 + 1",
    "CRC_5_ITU"   : "x^5 + x^4 + x^2 + 1",
    "CRC_5_USB"   : "x^5 + x^2 + 1",
    "CRC_6_ITU"   : "x^6 + x^1 + 1",
    "CRC_7"       : "x^7 + x^3 + 1",
    "CRC_8_ATM"   : "x^8 + x^2 + x^1 + 1",
    "CRC_8_CCITT" : "x^8 + x^7 + x^3 + x^2 + 1",
    "CRC_8_MAXIM" : "x^8 + x^5 + x^4 + 1",
    "CRC_8"        : "x^8 + x^7 + x^6 + x^4 + X^2 +1",
    "CRC_8_SAE"    : "x^8 + x^4 + x^3 + X^2 +1",
    "CRC_10"       : "x^10 + x^9 + x^5 + x^4 + x^1 + 1",
    "CRC_12"       : "x^12 + x^11 + x^3 + x^2 + x + 1"
}
```

config.py has not any method. It has a dictionary with the CRC code and its polynomials.

Code Snippets of “sender.py”

```
import random

from helper import StringToBinary, generateDivisor,
polynomialStringParser
from errorDetectionModules import CRC, VRC, LRC, CheckSum
from config import availableCRCPolynomials

# ? Sender Class
class Sender:
    dataWordFrameSize = 0
    input = ""
    rawInput = "" # Holds input whether it was binary or string
    output = ""
    outputWithoutAnyError = ""

    def __init__(self, dataWordFrameSize=8):
        self.dataWordFrameSize = dataWordFrameSize

    # IO Related functions
    def readInputFromConsole(self, binary):
        tmp = input("Enter the input: ")
        self.rawInput = tmp
        self.input = StringToBinary(tmp) if not binary else tmp

    def readInputFromFile(self, filename, binary):
        t = ""
        with open(filename, 'r') as f:
            for i in f.readlines():
                t += i
        self.rawInput = t.replace('\n', '')
        self.input = StringToBinary(t.replace('\n', '')) if not binary
else t.replace('\n', '')
```

```
    def writeOutputToFile(self, filename):
        if self.output == "": raise Exception("Output is empty ! Nothing
to save")
        with open(filename, 'w') as f:
            f.write(self.output)
            f.close()

    # Error Injection Function
    def injectErrorInOutput(self, loopC=1):
        for i in range(loopC):
            random_bit_location = random.randint(0, len(self.output)-1)
            self.output = self.output[:random_bit_location] +  ('0' if
self.output[random_bit_location] == '1' else '1') +
self.output[random_bit_location+1:]

    # Inject error at specific index
    def injectErrorAtIndex(self, index):
        self.output = self.output[:index] + ('0' if self.output[index]
== '1' else '1') + self.output[index+1:]

    # Encode Related Wrapper unctions
    # VRC Encoding
    def encodeUsingVRC(self):
        self.output = VRC.encode(self.input, self.dataWordFrameSize)
        self.outputWithoutAnyError = self.output

    # LRC Encoding
    def encodeUsingLRC(self, noOfOriginalDataFramesPerGroup=4):
        self.output = LRC.encode(self.input, self.dataWordFrameSize,
noOfOriginalDataFramesPerGroup)
        self.outputWithoutAnyError = self.output

    # Checksum Encoding
    def encodeUsingChecksum(self, noOfOriginalDataFramesPerGroup=4):
        self.output = CheckSum.encode(self.input,
self.dataWordFrameSize, noOfOriginalDataFramesPerGroup)
        self.outputWithoutAnyError = self.output

    # CRC Encoding
    def encodeUsingCRC(self, divisor):
        self.output = CRC.encode(self.input, self.dataWordFrameSize,
divisor)
        self.outputWithoutAnyError = self.output
```

```python
# ? ################# DRIVER CODE ###########
if __name__ == "__main__":
    # ! Data word frame size
    dataWordFrameSize = input("Enter no of bits in each data frame of
dataword [default : 8]: ")
    dataWordFrameSize =  8 if dataWordFrameSize == '' else
int(dataWordFrameSize)

    # ! Sender object
    sender = Sender(dataWordFrameSize=dataWordFrameSize)

    # ! Take input
    inputSourceAsTerminal = input("Do you want to use terminal as input
source [y/n] : ")
    inputIsBinary = input("Do you want to input binary data [y/n] : ")
    if inputSourceAsTerminal.lower() == "y":
        sender.readInputFromConsole( binary= (inputIsBinary.lower() ==
"y"))
    else:
        filename = input("Enter the filename [default :
assets/sender_input.txt ]: ")
        filename = "assets/sender_input.txt" if filename == '' else
filename
        sender.readInputFromFile(filename, binary=
(inputIsBinary.lower() == "y"))

    # ! Choose encoding method
    encodingMethod = input("Enter the encoding method [VRC, LRC, CRC,
CHECKSUM] : ")
    if encodingMethod not in ["VRC", "LRC", "CRC", "CHECKSUM"] : raise
Exception("Invalid encoding method") # Check for invalid input
    selectedPolynomial = ""

    if encodingMethod == "VRC":
        sender.encodeUsingVRC()
    elif encodingMethod == "LRC":
        noOfOriginalDataFramesPerGroup = input("Enter the no of data
frames per group [default : 4]: ")
        noOfOriginalDataFramesPerGroup = 4 if
noOfOriginalDataFramesPerGroup == '' else
int(noOfOriginalDataFramesPerGroup)

sender.encodeUsingLRC(noOfOriginalDataFramesPerGroup=noOfOriginalDataFra
mesPerGroup)
    elif encodingMethod == "CHECKSUM":
        noOfOriginalDataFramesPerGroup = input("Enter the no of data
frames per group [default : 4]: ")
        noOfOriginalDataFramesPerGroup = 4 if
noOfOriginalDataFramesPerGroup == '' else
int(noOfOriginalDataFramesPerGroup)

sender.encodeUsingChecksum(noOfOriginalDataFramesPerGroup=noOfOriginalDa
taFramesPerGroup)
    elif encodingMethod == "CRC":
        print("=== Available polynomials ===")
        for i in availableCRCPolynomials:
            print(i, end=", ")
        print(end="\n")
        selectedPolynomial = input("Enter the polynomial [default :
CRC_4_ITU]: ")
        selectedPolynomial = "CRC_4_ITU" if selectedPolynomial == ''
else selectedPolynomial
        parsedPolynomial =
polynomialStringParser(input=availableCRCPolynomials[selectedPolynomial]
)
        divisor = generateDivisor(parsedPolynomial)
        sender.encodeUsingCRC(divisor=divisor)

    # ! Inject error
    if input("Do you want to inject error in output [y/n] : ").lower()
== "y":
        if input("Mnaully inject error [y/n] : ").lower() == "y":
            specificBitsToInjectError = input("Enter the specific bits
to inject error [seperate by commas] : ")
            specificBitsToInjectError = [int(i.strip()) for i in
specificBitsToInjectError.split(",")]
            for i in specificBitsToInjectError:
                sender.injectErrorAtIndex(i)
        else:
            loopC = input("Enter the no of times you want to inject
error [default : 1]: ")
            loopC = 1 if loopC == '' else int(loopC)
            sender.injectErrorInOutput(loopC=loopC)

    # ! File name to store output
    outputFileName = input("Enter the filename to store output [default
: assets/sender_output.txt ]: ")
    outputFileName = "assets/sender_output.txt" if outputFileName == ''
else outputFileName
    sender.writeOutputToFile(outputFileName)
    # ! Print data

print("==========================================================================
========================")
    print("Raw Input              : ", sender.rawInput)
    print("Final Input            : ", sender.input)
    print("Encoding technique     : ", encodingMethod+"
"+selectedPolynomial if encodingMethod == "CRC" else encodingMethod)
    print("Output [Without error] : ", sender.outputWithoutAnyError),
    print("Output [May have error]: ", sender.output)
    print("Written output in file : ", outputFileName)

print("==========================================================================
========================")
```

Method Descriptions of “sender.py”

- readInputFromConsole(self, binary) : Read the input from user input. If binary is true, it will accept binary data [100110101….], else it will accept any string and later will convert it to binary.
- readInputFromFile(self, filename, binary) : Read the input from file. If binary is true, it will accept binary data [100110101….], else it will accept any string and later will convert it to binary.
- writeOutputToFile(self, filename): This method will write the output to a file.
- injectErrorInOutput(self, loopC=1) : This method will inject error at random position of data. The loopC parameter signifies no fo times this process will repeat.
- injectErrorAtIndex(self, index) : By this method , we can inject an error in the data at a selected index.
- encodeUsingVRC(self) : This is a wrapper method around the encode method of VRC.
- encodeUsingLRC(self, noOfOriginalDataFramesPerGroup=4) : This is a wrapper method around the encode method of LRC.

- encodeUsingChecksum(self, noOfOriginalDataFramesPerGroup=4) : This is a wrapper method around the encode method of Checksum.
- encodeUsingCRC(self, divisor) : This is a wrapper method around the encode method of CRC.
- Driver Program : This program is responsible to show the desired options , instructions and results to the user and to test the Sender class.

Code Snippets of “receiver.py”

```
from helper import BinaryToString, ReadNoOfZerosAndOnes, buildFrames,
generateDivisor, polynomialStringParser
from errorDetectionModules import CRC, VRC, LRC, CheckSum
from config import availableCRCPolynomials

class Receiver:
    dataWordFrameSize=0
    input = ""
    output = ""
    outputData = ""
    errorFound = False

    def __init__(self, dataWordFrameSize=8):
        self.dataWordFrameSize = dataWordFrameSize

    # IO Related functions
    def readInputFromConsole(self):
        self.input = input("Enter the input in binary format: ")

    def readInputFromFile(self, filename):
        t = ""
        with open(filename, 'r') as f:
            for i in f.readlines():
                t += i
        self.input = t.replace('\n', '')

    def writeOutputToFile(self, filename, binary):
        tmp = BinaryToString(self.output) if not binary else self.output
        with open(filename, 'w') as f:
            f.write(tmp)
            f.close()
    # Decode Wrapper Methods
    # VRC Decoding
    def decodeUsingVRC(self):
        output, errorFound = VRC.decode(self.input,
self.dataWordFrameSize)
        self.output = output
        self.outputData = BinaryToString(self.output)
        self.errorFound = errorFound

    # LRC Decoding
    def decodeUsingLRC(self, noOfOriginalDataFramesPerGroup):
        output, errorFound = LRC.decode(self.input,
self.dataWordFrameSize, noOfOriginalDataFramesPerGroup)
        self.output = output
        self.outputData = BinaryToString(self.output)
        self.errorFound = errorFound

    # Checksum Decoding
    def decodeUsingChecksum(self, noOfOriginalDataFramesPerGroup):
        output, errorFound = CheckSum.decode(self.input,
self.dataWordFrameSize, noOfOriginalDataFramesPerGroup)
        self.output = output
        self.outputData = BinaryToString(self.output)
        self.errorFound = errorFound

    # CRC Decoding
    def decodeUsingCRC(self, divisor):
        output, errorFound = CRC.decode(self.input,
self.dataWordFrameSize, divisor)
        self.output = output
        self.outputData = BinaryToString(self.output)
        self.errorFound = errorFound


if __name__ == "__main__":
    # ! Data word frame size
    dataWordFrameSize = input("Enter no of bits in each data frame of
dataword [default : 8]: ")
    dataWordFrameSize =  8 if dataWordFrameSize == '' else
int(dataWordFrameSize)

    # ! Receiver object
    receiver = Receiver(dataWordFrameSize=dataWordFrameSize)

    # ! Input for receiver
    if input("Do you want to read input from console? (y/n): ").lower()
```

```
== "y":
        receiver.readInputFromConsole()
    else:
        inputfilename = input("Enter the filename [default :
assets/sender_output.txt]: ")
        inputfilename = "assets/sender_output.txt" if inputfilename ==
'' else inputfilename
        receiver.readInputFromFile(inputfilename)


    # ! Choose decoding method
    decodingMethod = input("Enter the decoding method [VRC, LRC, CRC,
CHECKSUM] : ")
    if decodingMethod not in ["VRC", "LRC", "CRC", "CHECKSUM"] : raise
Exception("Invalid encoding method") # Check for invalid input
    selectedPolynomial = ""

    # ! Decode
    if decodingMethod == "VRC":
        receiver.decodeUsingVRC()
    elif decodingMethod == "LRC":
        noOfOriginalDataFramesPerGroup = input("Enter the no of data
frames per group [default : 4]: ")
        noOfOriginalDataFramesPerGroup = 4 if
noOfOriginalDataFramesPerGroup == '' else
int(noOfOriginalDataFramesPerGroup)

receiver.decodeUsingLRC(noOfOriginalDataFramesPerGroup=noOfOriginalDataF
ramesPerGroup)
    elif decodingMethod == "CHECKSUM":
        noOfOriginalDataFramesPerGroup = input("Enter the no of data
frames per group [default : 4]: ")
        noOfOriginalDataFramesPerGroup = 4 if
noOfOriginalDataFramesPerGroup == '' else
int(noOfOriginalDataFramesPerGroup)

receiver.decodeUsingChecksum(noOfOriginalDataFramesPerGroup=noOfOriginal
DataFramesPerGroup)
    elif decodingMethod == "CRC":
        print("=== Available polynomials ===")
        for i in availableCRCPolynomials:
            print(i, end=", ")
        print(end="\n")
        selectedPolynomial = input("Enter the polynomial [default :
CRC_4_ITU]: ")
        selectedPolynomial = "CRC_4_ITU" if selectedPolynomial == ''
```

```
else selectedPolynomial
        parsedPolynomial =
polynomialStringParser(input=availableCRCPolynomials[selectedPolynomial]
)
        divisor = generateDivisor(parsedPolynomial)
        receiver.decodeUsingCRC(divisor=divisor)

    # ! Write output to file
    outputFilename = input("Enter the filename [default :
assets/receiver_output.txt]: ")
    outputFilename = "assets/receiver_output.txt" if outputFilename ==
'' else outputFilename
    tmpFilename = outputFilename.split(".")[0]

    receiver.writeOutputToFile(filename=tmpFilename+"_binary.txt",
binary=True)
    receiver.writeOutputToFile(filename=tmpFilename+"_data.txt",
binary=False)

    # ! Print data

print("=================================================================
========================")
    print("Input                  : ", receiver.input)
    print("Decoding technique     : ", decodingMethod+"
"+selectedPolynomial if decodingMethod == "CRC" else decodingMethod)
    print("Output Binary          : ", receiver.output)
    print("Output Data            : ", receiver.outputData)
    print("Status                 : ", "FAILED" if receiver.errorFound
else "PASS")
    print("Written binary output in file : ", tmpFilename+"_binary.txt")
    print("Written binary output in file : ", tmpFilename+"_data.txt")

print("=================================================================
========================")
```

Method description of “receiver.py”

- readInputFromConsole(self) : Read the input from user input.
- readInputFromFile(self, filename) : Read the input from file.
- writeOutputToFile(self, filename, binary): This method will write the output to a file. If the binary is true , then it will store the binary data in the file, otherwise, it will convert the binary to string and then store in file

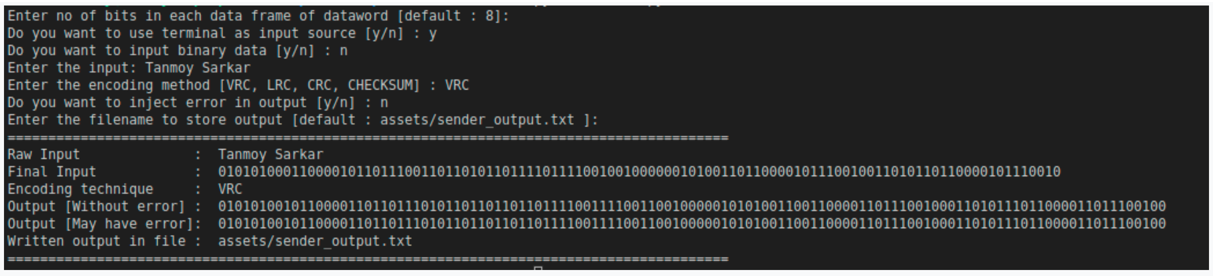
- decodeUsingVRC(self) : This is a wrapper method around the decode method of VRC.
- decodeUsingLRC(self, noOfOriginalDataFramesPerGroup) : This is a wrapper method around the decode method of LRC.
- decodeUsingChecksum(self, noOfOriginalDataFramesPerGroup) : This is a wrapper method around the decode method of Checksum.
- decodeUsingCRC(self, divisor) : This is a wrapper method around the decode method of CRC.
- Driver Program: This program is responsible to show the desired options, instructions and results to the user and testing the Receiver class.

RESULTS

=> Here are some screenshots of tests on VRC

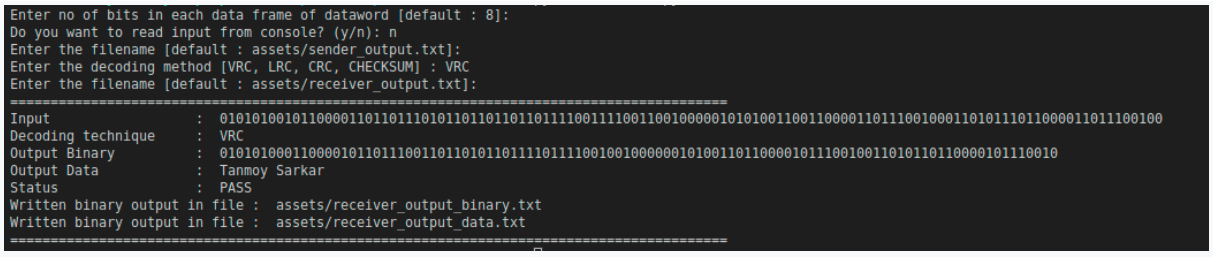
Testing with no error

Encode



```
Enter no of bits in each data frame of dataword [default : 8]:
Do you want to use terminal as input source [y/n] : y
Do you want to input binary data [y/n] : n
Enter the input: Tanmoy Sarkar
Enter the encoding method [VRC, LRC, CRC, CHECKSUM] : VRC
Do you want to inject error in output [y/n] : n
Enter the filename to store output [default : assets/sender_output.txt ]:
=========================================================================================
Raw Input             :  Tanmoy Sarkar
Final Input           :  01010100011000010110111001101101011011110111100100100000010100110110000101110010011010110110000101110010
Encoding technique    :  VRC
Output [Without error] :  010101001011000011011011101011011011011011110011110011001000001010100110011000011011100100011010111011000011011100100
Output [May have error]:  010101001011000011011011101011011011011011110011110011001000001010100110011000011011100100011010111011000011011100100
Written output in file :  assets/sender_output.txt
=========================================================================================
```

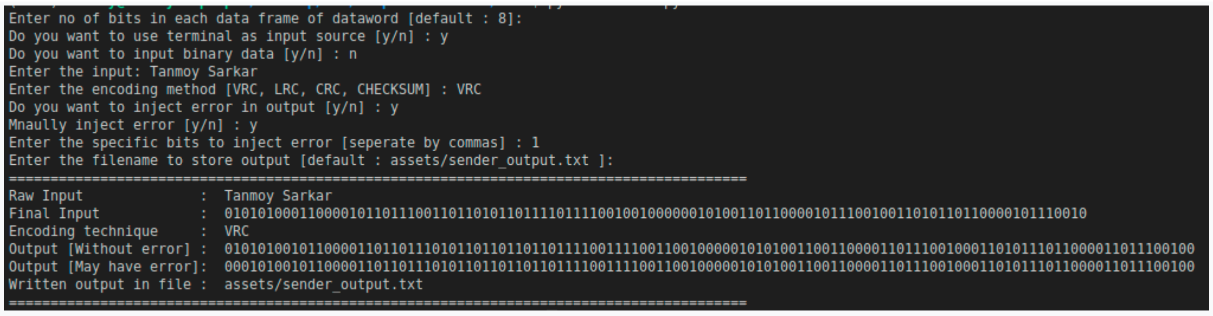
Decode



```
Enter no of bits in each data frame of dataword [default : 8]:
Do you want to read input from console? (y/n): n
Enter the filename [default : assets/sender_output.txt]:
Enter the decoding method [VRC, LRC, CRC, CHECKSUM] : VRC
Enter the filename [default : assets/receiver_output.txt]:
=========================================================================================
Input                 :  010101001011000011011011101011011011011011110011110011001000001010100110011000011011100100011010111011000011011100100
Decoding technique    :  VRC
Output Binary         :  01010100011000010110111001101101011011110111100100100000010100110110000101110010011010110110000101110010
Output Data           :  Tanmoy Sarkar
Status                :  PASS
Written binary output in file :  assets/receiver_output_binary.txt
Written binary output in file :  assets/receiver_output_data.txt
=========================================================================================
```

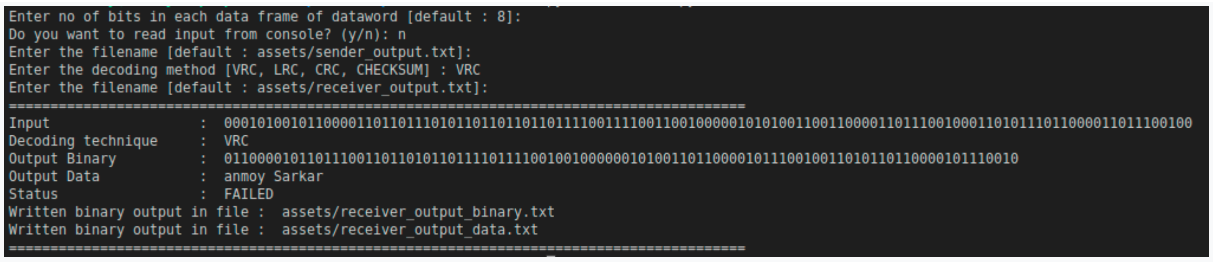
Testing with error [manually injected]

Encode



```
Enter no of bits in each data frame of dataword [default : 8]:
Do you want to use terminal as input source [y/n] : y
Do you want to input binary data [y/n] : n
Enter the input: Tanmoy Sarkar
Enter the encoding method [VRC, LRC, CRC, CHECKSUM] : VRC
Do you want to inject error in output [y/n] : y
Mnaully inject error [y/n] : y
Enter the specific bits to inject error [seperate by commas] : 1
Enter the filename to store output [default : assets/sender_output.txt ]:
=========================================================================================
Raw Input             :  Tanmoy Sarkar
Final Input           :  01010100011000010110111001101101011011110111100100100000010100110110000101110010011010110110000101110010
Encoding technique    :  VRC
Output [Without error] :  010101001011000011011011101011011011011011110011110011001000001010100110011000011011100100011010111011000011011100100
Output [May have error]:  000101001011000011011011101011011011011011110011110011001000001010100110011000011011100100011010111011000011011100100
Written output in file :  assets/sender_output.txt
=========================================================================================
```

Decode

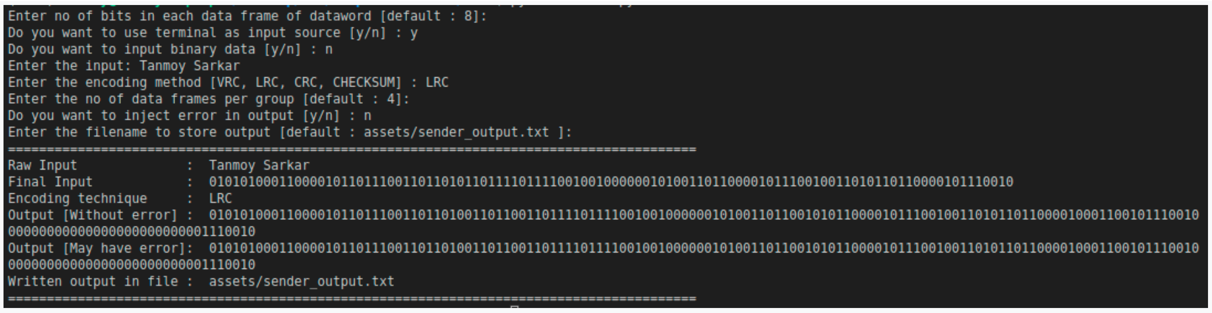


```
Enter no of bits in each data frame of dataword [default : 8]:
Do you want to read input from console? (y/n): n
Enter the filename [default : assets/sender_output.txt]:
Enter the decoding method [VRC, LRC, CRC, CHECKSUM] : VRC
Enter the filename [default : assets/receiver_output.txt]:
=========================================================================================
Input                 :  000101001011000011011011101011011011011011110011110011001000001010100110011000011011100100011010111011000011011100100
Decoding technique    :  VRC
Output Binary         :  011000010110111001101101011011110111100100100000010100110110000101110010011010110110000101110010
Output Data           :  anmoy Sarkar
Status                :  FAILED
Written binary output in file :  assets/receiver_output_binary.txt
Written binary output in file :  assets/receiver_output_data.txt
=========================================================================================
```

=> Here are some screenshots of tests on LRC

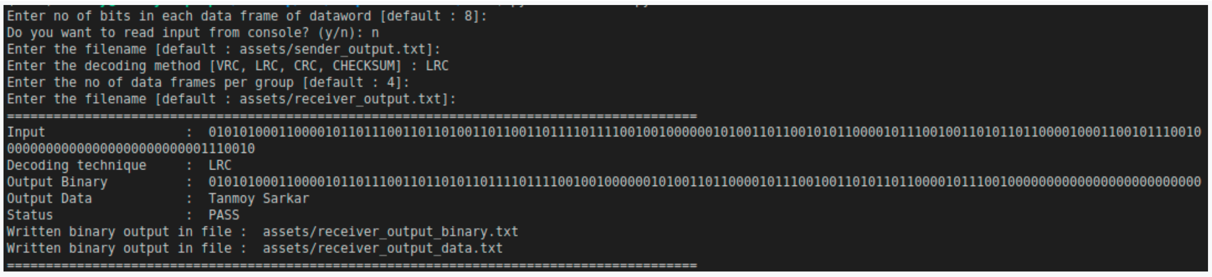
Testing with no error

Encode



```
Enter no of bits in each data frame of dataword [default : 8]:
Do you want to use terminal as input source [y/n] : y
Do you want to input binary data [y/n] : n
Enter the input: Tanmoy Sarkar
Enter the encoding method [VRC, LRC, CRC, CHECKSUM] : LRC
Enter the no of data frames per group [default : 4]:
Do you want to inject error in output [y/n] : n
Enter the filename to store output [default : assets/sender_output.txt ]:
=========================================================================================
Raw Input              :  Tanmoy Sarkar
Final Input            :  01010100011000010110111001101101011011110111100100100000010100110110000101110010011010110110000101110010
Encoding technique     :  LRC
Output [Without error] :  0101010001100001011011100110110100110110011011110111100100100000010100110110010101100001011100100110101101100001000110010111001 0
00000000000000000000000001110010
Output [May have error]:  0101010001100001011011100110110100110110011011110111100100100000010100110110010101100001011100100110101101100001000110010111001 0
00000000000000000000000001110010
Written output in file :  assets/sender_output.txt
=========================================================================================
```

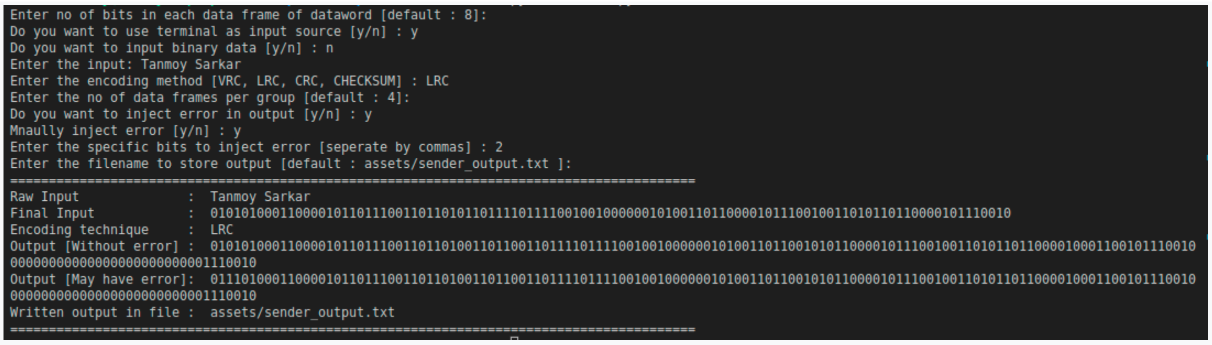
Decode



```
Enter no of bits in each data frame of dataword [default : 8]:
Do you want to read input from console? (y/n): n
Enter the filename [default : assets/sender_output.txt]:
Enter the decoding method [VRC, LRC, CRC, CHECKSUM] : LRC
Enter the no of data frames per group [default : 4]:
Enter the filename [default : assets/receiver_output.txt]:
=========================================================================================
Input                  :  0101010001100001011011100110110100110110011011110111100100100000010100110110010101100001011100100110101101100001000110010111001 0
00000000000000000000000001110010
Decoding technique     :  LRC
Output Binary          :  0101010001100001011011100110110101101111011110010010000001010011011000010111001001101011011000010111001000000000000000000000000000
Output Data            :  Tanmoy Sarkar
Status                 :  PASS
Written binary output in file :  assets/receiver_output_binary.txt
Written binary output in file :  assets/receiver_output_data.txt
=========================================================================================
```

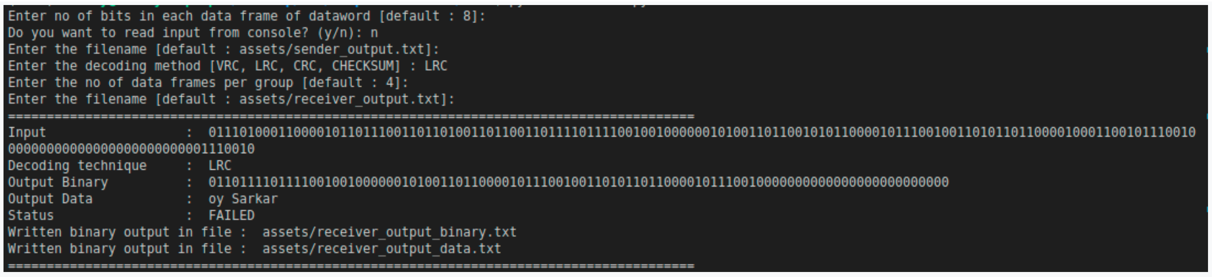
Testing with error [manually injected]

Encode



```
Enter no of bits in each data frame of dataword [default : 8]:
Do you want to use terminal as input source [y/n] : y
Do you want to input binary data [y/n] : n
Enter the input: Tanmoy Sarkar
Enter the encoding method [VRC, LRC, CRC, CHECKSUM] : LRC
Enter the no of data frames per group [default : 4]:
Do you want to inject error in output [y/n] : y
Mnaully inject error [y/n] : y
Enter the specific bits to inject error [seperate by commas] : 2
Enter the filename to store output [default : assets/sender_output.txt ]:
=========================================================================================
Raw Input              :  Tanmoy Sarkar
Final Input            :  01010100011000010110111001101101011011110111100100100000010100110110000101110010011010110110000101110010
Encoding technique     :  LRC
Output [Without error] :  0101010001100001011011100110110100110110011011110111100100100000010100110110010101100001011100100110101101100001000110010111001 0
00000000000000000000000001110010
Output [May have error]:  0111010001100001011011100110110100110110011011110111100100100000010100110110010101100001011100100110101101100001000110010111001 0
00000000000000000000000001110010
Written output in file :  assets/sender_output.txt
=========================================================================================
```

Decode

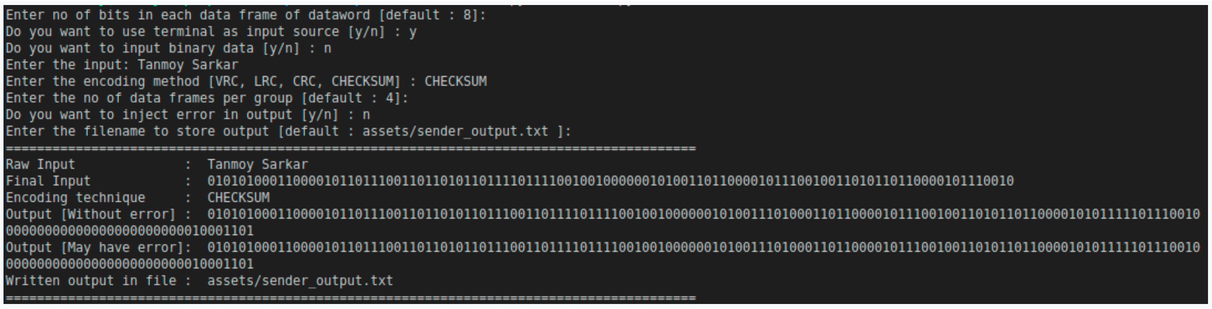


```
Enter no of bits in each data frame of dataword [default : 8]:
Do you want to read input from console? (y/n): n
Enter the filename [default : assets/sender_output.txt]:
Enter the decoding method [VRC, LRC, CRC, CHECKSUM] : LRC
Enter the no of data frames per group [default : 4]:
Enter the filename [default : assets/receiver_output.txt]:
=========================================================================================
Input                  :  0111010001100001011011100110110100110110011011110111100100100000010100110110010101100001011100100110101101100001000110010111001 0
00000000000000000000000001110010
Decoding technique     :  LRC
Output Binary          :  011011110111100100100000010100110110000101110010011010110110000101110010000000000000000000000000
Output Data            :  oy Sarkar
Status                 :  FAILED
Written binary output in file :  assets/receiver_output_binary.txt
Written binary output in file :  assets/receiver_output_data.txt
=========================================================================================
```

=> Here are some screenshots of tests on Checksum

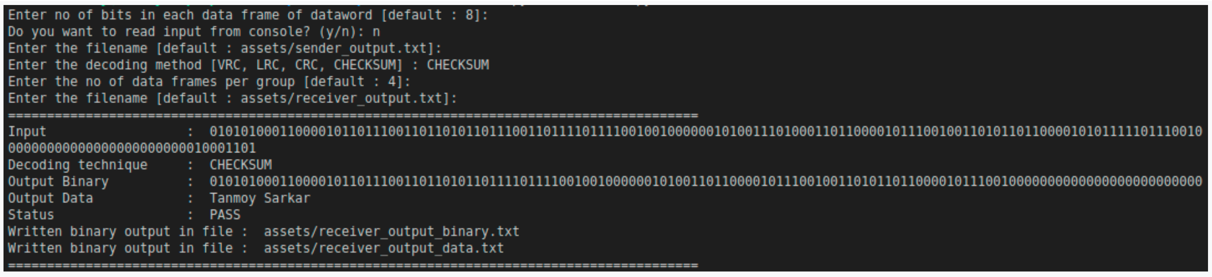
Testing with no error

Encode



```
Enter no of bits in each data frame of dataword [default : 8]:
Do you want to use terminal as input source [y/n] : y
Do you want to input binary data [y/n] : n
Enter the input: Tanmoy Sarkar
Enter the encoding method [VRC, LRC, CRC, CHECKSUM] : CHECKSUM
Enter the no of data frames per group [default : 4]:
Do you want to inject error in output [y/n] : n
Enter the filename to store output [default : assets/sender_output.txt ]:
==========================================================================================
Raw Input             :  Tanmoy Sarkar
Final Input           :  01010100011000010110111001101101011011110111100100100000010100110110000101110010011010110110000101110010
Encoding technique    :  CHECKSUM
Output [Without error] :  010101000110000101101110011011010110111001101111011110010010000001010011101000110110000101110010011010110110000101011111101110010
00000000000000000000000010001101
Output [May have error]:  010101000110000101101110011011010110111001101111011110010010000001010011101000110110000101110010011010110110000101011111101110010
00000000000000000000000010001101
Written output in file :  assets/sender_output.txt
==========================================================================================
```

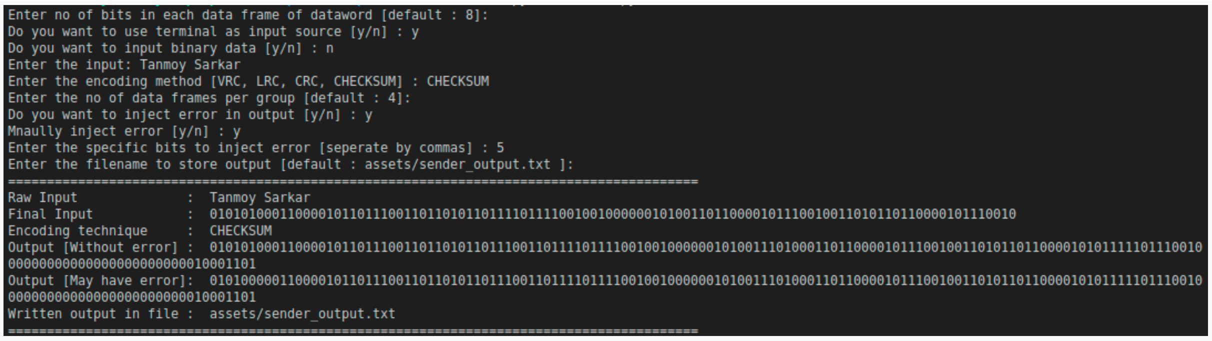
Decode



```
Enter no of bits in each data frame of dataword [default : 8]:
Do you want to read input from console? (y/n): n
Enter the filename [default : assets/sender_output.txt]:
Enter the decoding method [VRC, LRC, CRC, CHECKSUM] : CHECKSUM
Enter the no of data frames per group [default : 4]:
Enter the filename [default : assets/receiver_output.txt]:
==========================================================================================
Input                 :  010101000110000101101110011011010110111001101111011110010010000001010011101000110110000101110010011010110110000101011111101110010
00000000000000000000000010001101
Decoding technique    :  CHECKSUM
Output Binary         :  01010100011000010110111001101101011011110111100100100000010100110110000101110010011010110110000101110010000000000000000000000000000
Output Data           :  Tanmoy Sarkar
Status                :  PASS
Written binary output in file :  assets/receiver_output_binary.txt
Written binary output in file :  assets/receiver_output_data.txt
==========================================================================================
```

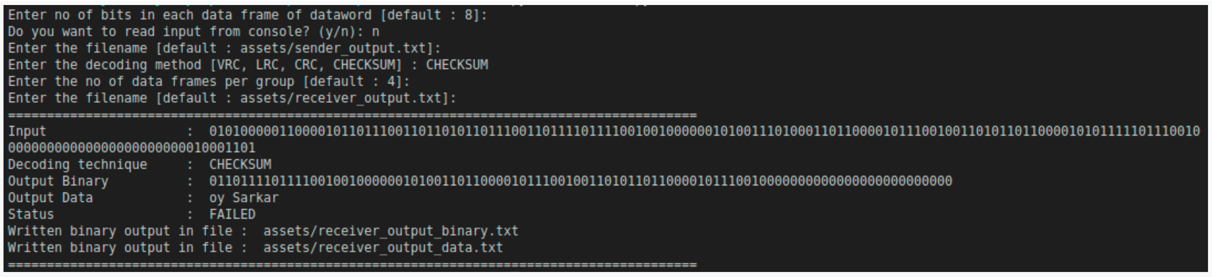
Testing with error [manually injected]

Encode



```
Enter no of bits in each data frame of dataword [default : 8]:
Do you want to use terminal as input source [y/n] : y
Do you want to input binary data [y/n] : n
Enter the input: Tanmoy Sarkar
Enter the encoding method [VRC, LRC, CRC, CHECKSUM] : CHECKSUM
Enter the no of data frames per group [default : 4]:
Do you want to inject error in output [y/n] : y
Mnaully inject error [y/n] : y
Enter the specific bits to inject error [seperate by commas] : 5
Enter the filename to store output [default : assets/sender_output.txt ]:
==========================================================================================
Raw Input             :  Tanmoy Sarkar
Final Input           :  01010100011000010110111001101101011011110111100100100000010100110110000101110010011010110110000101110010
Encoding technique    :  CHECKSUM
Output [Without error] :  010101000110000101101110011011010110111001101111011110010010000001010011101000110110000101110010011010110110000101011111101110010
00000000000000000000000010001101
Output [May have error]:  010100000110000101101110011011010110111001101111011110010010000001010011101000110110000101110010011010110110000101011111101110010
00000000000000000000000010001101
Written output in file :  assets/sender_output.txt
==========================================================================================
```

Decode

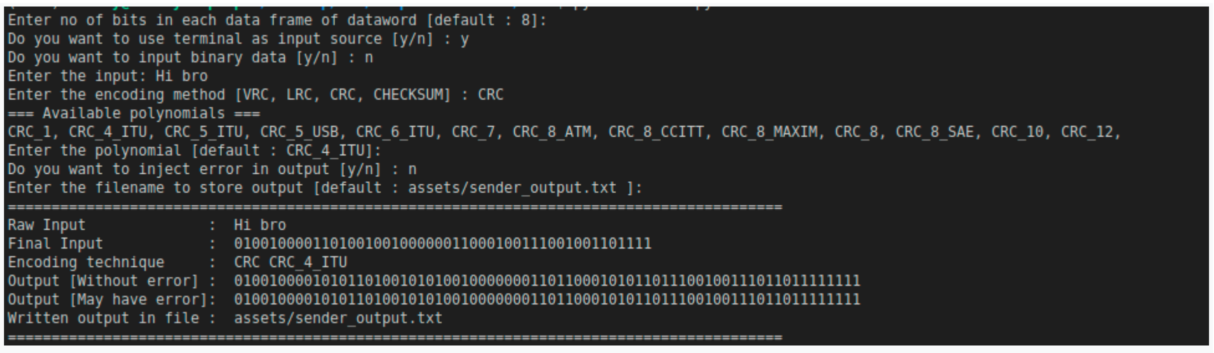


```
Enter no of bits in each data frame of dataword [default : 8]:
Do you want to read input from console? (y/n): n
Enter the filename [default : assets/sender_output.txt]:
Enter the decoding method [VRC, LRC, CRC, CHECKSUM] : CHECKSUM
Enter the no of data frames per group [default : 4]:
Enter the filename [default : assets/receiver_output.txt]:
==========================================================================================
Input                 :  010100000110000101101110011011010110111001101111011110010010000001010011101000110110000101110010011010110110000101011111101110010
00000000000000000000000010001101
Decoding technique    :  CHECKSUM
Output Binary         :  0110111101111001001000000101001101100001011100100110101101100001011100100000000000000000000000000
Output Data           :  oy Sarkar
Status                :  FAILED
Written binary output in file :  assets/receiver_output_binary.txt
Written binary output in file :  assets/receiver_output_data.txt
==========================================================================================
```

=> Here are some screenshots of tests on CRC

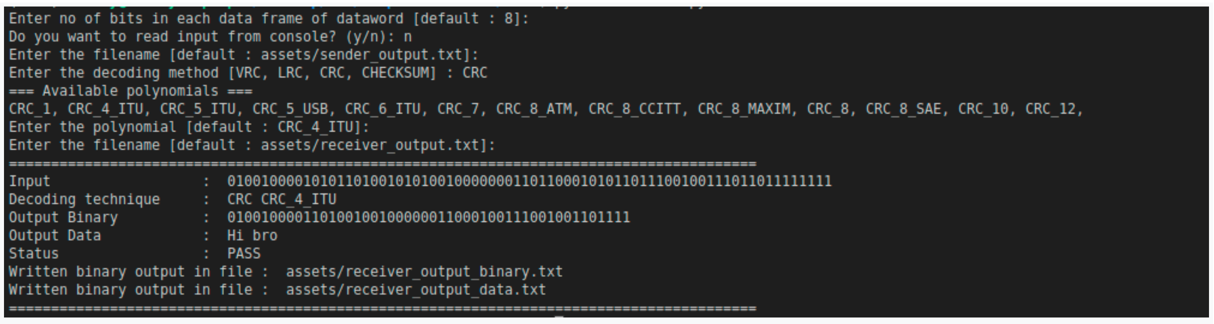
Testing with no error

Encode



```
Enter no of bits in each data frame of dataword [default : 8]:
Do you want to use terminal as input source [y/n] : y
Do you want to input binary data [y/n] : n
Enter the input: Hi bro
Enter the encoding method [VRC, LRC, CRC, CHECKSUM] : CRC
=== Available polynomials ===
CRC_1, CRC_4_ITU, CRC_5_ITU, CRC_5_USB, CRC_6_ITU, CRC_7, CRC_8_ATM, CRC_8_CCITT, CRC_8_MAXIM, CRC_8, CRC_8_SAE, CRC_10, CRC_12,
Enter the polynomial [default : CRC_4_ITU]:
Do you want to inject error in output [y/n] : n
Enter the filename to store output [default : assets/sender_output.txt ]:
===================================================================================
Raw Input              :  Hi bro
Final Input            :  010010000110100100100000011000100111001001101111
Encoding technique     :  CRC CRC_4_ITU
Output [Without error] :  010010000101011010010101001000000011011000101011011100100111011011111111
Output [May have error]:  010010000101011010010101001000000011011000101011011100100111011011111111
Written output in file :  assets/sender_output.txt
===================================================================================
```

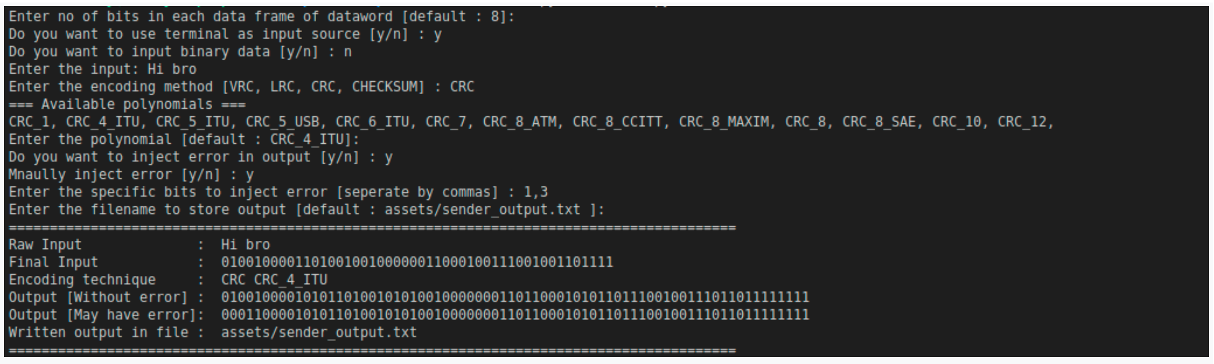
Decode



```
Enter no of bits in each data frame of dataword [default : 8]:
Do you want to read input from console? (y/n): n
Enter the filename [default : assets/sender_output.txt]:
Enter the decoding method [VRC, LRC, CRC, CHECKSUM] : CRC
=== Available polynomials ===
CRC_1, CRC_4_ITU, CRC_5_ITU, CRC_5_USB, CRC_6_ITU, CRC_7, CRC_8_ATM, CRC_8_CCITT, CRC_8_MAXIM, CRC_8, CRC_8_SAE, CRC_10, CRC_12,
Enter the polynomial [default : CRC_4_ITU]:
Enter the filename [default : assets/receiver_output.txt]:
===================================================================================
Input                  :  010010000101011010010101001000000011011000101011011100100111011011111111
Decoding technique     :  CRC CRC_4_ITU
Output Binary          :  010010000110100100100000011000100111001001101111
Output Data            :  Hi bro
Status                 :  PASS
Written binary output in file :  assets/receiver_output_binary.txt
Written binary output in file :  assets/receiver_output_data.txt
===================================================================================
```

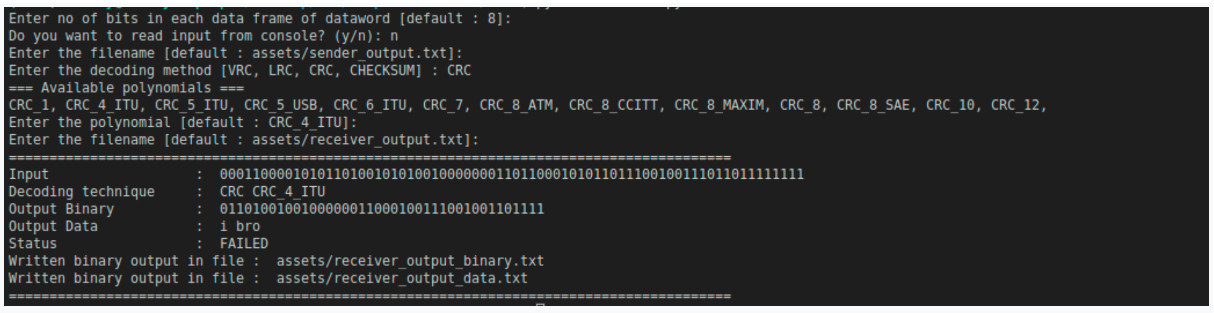
Testing with error [manually injected]

Encode



```
Enter no of bits in each data frame of dataword [default : 8]:
Do you want to use terminal as input source [y/n] : y
Do you want to input binary data [y/n] : n
Enter the input: Hi bro
Enter the encoding method [VRC, LRC, CRC, CHECKSUM] : CRC
=== Available polynomials ===
CRC_1, CRC_4_ITU, CRC_5_ITU, CRC_5_USB, CRC_6_ITU, CRC_7, CRC_8_ATM, CRC_8_CCITT, CRC_8_MAXIM, CRC_8, CRC_8_SAE, CRC_10, CRC_12,
Enter the polynomial [default : CRC_4_ITU]:
Do you want to inject error in output [y/n] : y
Mnaully inject error [y/n] : y
Enter the specific bits to inject error [seperate by commas] : 1,3
Enter the filename to store output [default : assets/sender_output.txt ]:
===================================================================================
Raw Input              :  Hi bro
Final Input            :  010010000110100100100000011000100111001001101111
Encoding technique     :  CRC CRC_4_ITU
Output [Without error] :  010010000101011010010101001000000011011000101011011100100111011011111111
Output [May have error]:  000110000101011010010101001000000011011000101011011100100111011011111111
Written output in file :  assets/sender_output.txt
===================================================================================
```

Decode



```
Enter no of bits in each data frame of dataword [default : 8]:
Do you want to read input from console? (y/n): n
Enter the filename [default : assets/sender_output.txt]:
Enter the decoding method [VRC, LRC, CRC, CHECKSUM] : CRC
=== Available polynomials ===
CRC_1, CRC_4_ITU, CRC_5_ITU, CRC_5_USB, CRC_6_ITU, CRC_7, CRC_8_ATM, CRC_8_CCITT, CRC_8_MAXIM, CRC_8, CRC_8_SAE, CRC_10, CRC_12,
Enter the polynomial [default : CRC_4_ITU]:
Enter the filename [default : assets/receiver_output.txt]:
===================================================================================
Input                  :  000110000101011010010101001000000011011000101011011100100111011011111111
Decoding technique     :  CRC CRC_4_ITU
Output Binary          :  0110100100100000011000100111001001101111
Output Data            :  i bro
Status                 :  FAILED
Written binary output in file :  assets/receiver_output_binary.txt
Written binary output in file :  assets/receiver_output_data.txt
===================================================================================
```

TEST CASES

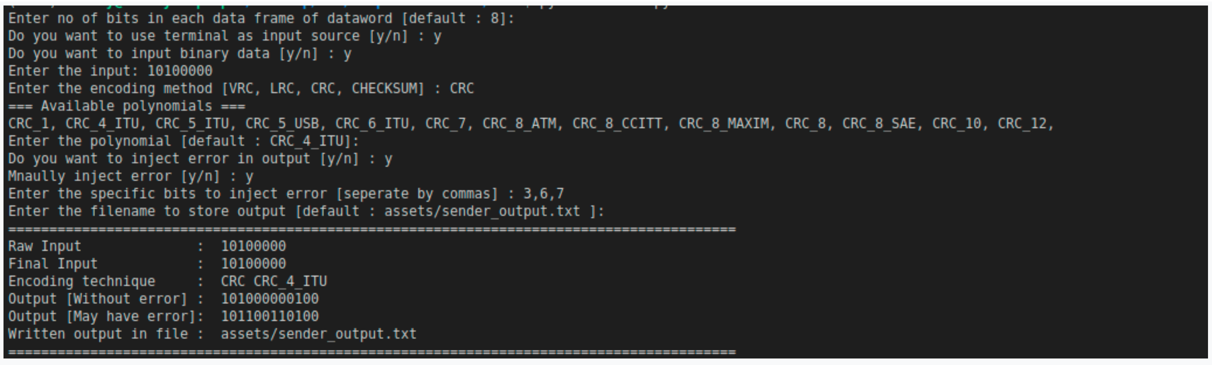
a) Error is detected by checksum but not by CRC

Dataword -> 10 10 00 00

Insert error at index -> 3, 6, 7

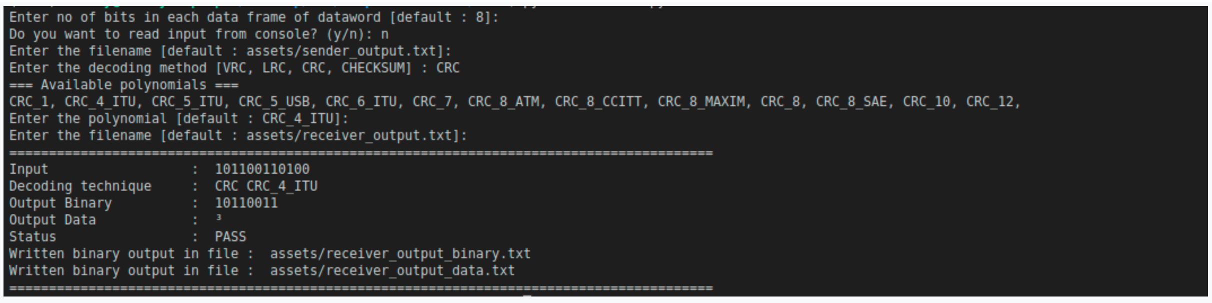
Module used: CRC 4 ITU

Sender Side



```
Enter no of bits in each data frame of dataword [default : 8]:
Do you want to use terminal as input source [y/n] : y
Do you want to input binary data [y/n] : y
Enter the input: 10100000
Enter the encoding method [VRC, LRC, CRC, CHECKSUM] : CRC
=== Available polynomials ===
CRC_1, CRC_4_ITU, CRC_5_ITU, CRC_5_USB, CRC_6_ITU, CRC_7, CRC_8_ATM, CRC_8_CCITT, CRC_8_MAXIM, CRC_8, CRC_8_SAE, CRC_10, CRC_12,
Enter the polynomial [default : CRC_4_ITU]:
Do you want to inject error in output [y/n] : y
Mnaully inject error [y/n] : y
Enter the specific bits to inject error [seperate by commas] : 3,6,7
Enter the filename to store output [default : assets/sender_output.txt ]:
=========================================================================================
Raw Input               :  10100000
Final Input             :  10100000
Encoding technique      :  CRC CRC_4_ITU
Output [Without error] :  101000000100
Output [May have error]:  101100110100
Written output in file :  assets/sender_output.txt
=========================================================================================
```

Receiver Side

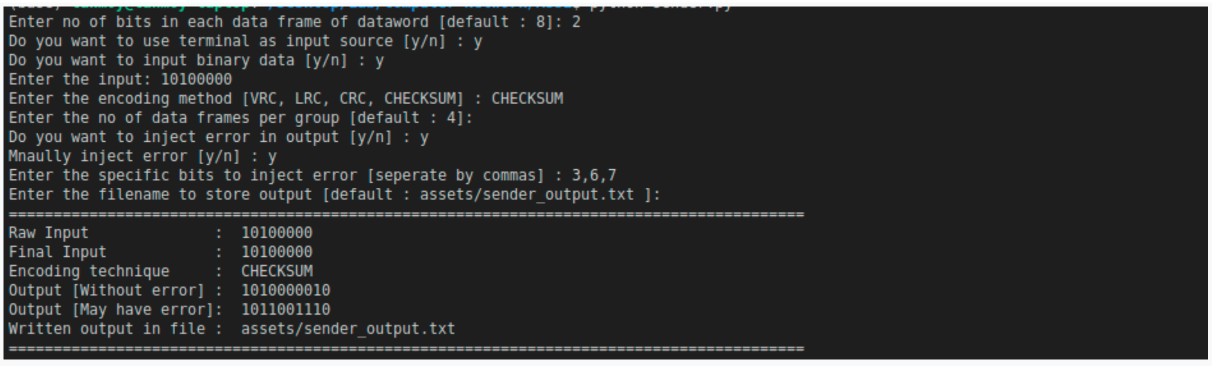


```
Enter no of bits in each data frame of dataword [default : 8]:
Do you want to read input from console? (y/n): n
Enter the filename [default : assets/sender_output.txt]:
Enter the decoding method [VRC, LRC, CRC, CHECKSUM] : CRC
=== Available polynomials ===
CRC_1, CRC_4_ITU, CRC_5_ITU, CRC_5_USB, CRC_6_ITU, CRC_7, CRC_8_ATM, CRC_8_CCITT, CRC_8_MAXIM, CRC_8, CRC_8_SAE, CRC_10, CRC_12,
Enter the polynomial [default : CRC_4_ITU]:
Enter the filename [default : assets/receiver_output.txt]:
=========================================================================================
Input                   :  101100110100
Decoding technique      :  CRC CRC_4_ITU
Output Binary           :  10110011
Output Data             :  ³
Status                  :  PASS
Written binary output in file :  assets/receiver_output_binary.txt
Written binary output in file :  assets/receiver_output_data.txt
=========================================================================================
```

Note : Error Injected but Receiver side can’t detect the error by CRC 4 ITU

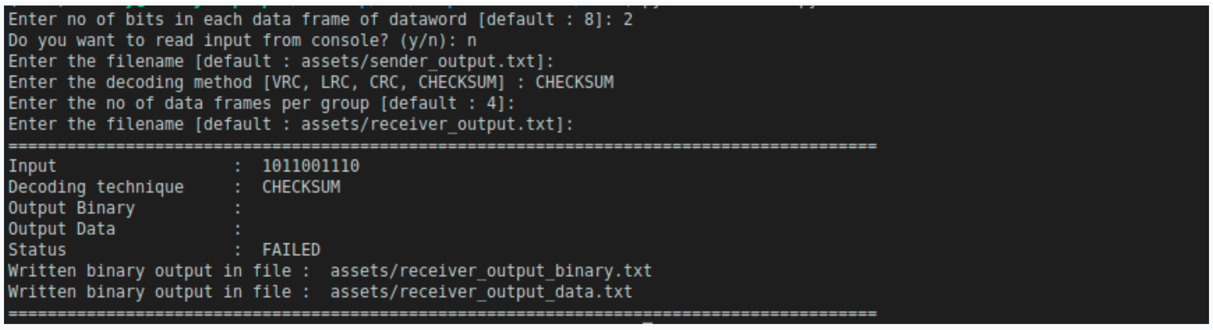
Module used: Checksum

Sender Side



```
Enter no of bits in each data frame of dataword [default : 8]: 2
Do you want to use terminal as input source [y/n] : y
Do you want to input binary data [y/n] : y
Enter the input: 10100000
Enter the encoding method [VRC, LRC, CRC, CHECKSUM] : CHECKSUM
Enter the no of data frames per group [default : 4]:
Do you want to inject error in output [y/n] : y
Mnaully inject error [y/n] : y
Enter the specific bits to inject error [seperate by commas] : 3,6,7
Enter the filename to store output [default : assets/sender_output.txt ]:
=========================================================================================
Raw Input               :  10100000
Final Input             :  10100000
Encoding technique      :  CHECKSUM
Output [Without error] :  1010000010
Output [May have error]:  1011001110
Written output in file :  assets/sender_output.txt
=========================================================================================
```

Receiver Side:



```
Enter no of bits in each data frame of dataword [default : 8]: 2
Do you want to read input from console? (y/n): n
Enter the filename [default : assets/sender_output.txt]:
Enter the decoding method [VRC, LRC, CRC, CHECKSUM] : CHECKSUM
Enter the no of data frames per group [default : 4]:
Enter the filename [default : assets/receiver_output.txt]:
==========================================================================================
Input                   :  1011001110
Decoding technique      :  CHECKSUM
Output Binary           :
Output Data             :
Status                  :  FAILED
Written binary output in file :  assets/receiver_output_binary.txt
Written binary output in file :  assets/receiver_output_data.txt
==========================================================================================
```

Note : Checksum has detected the error in data

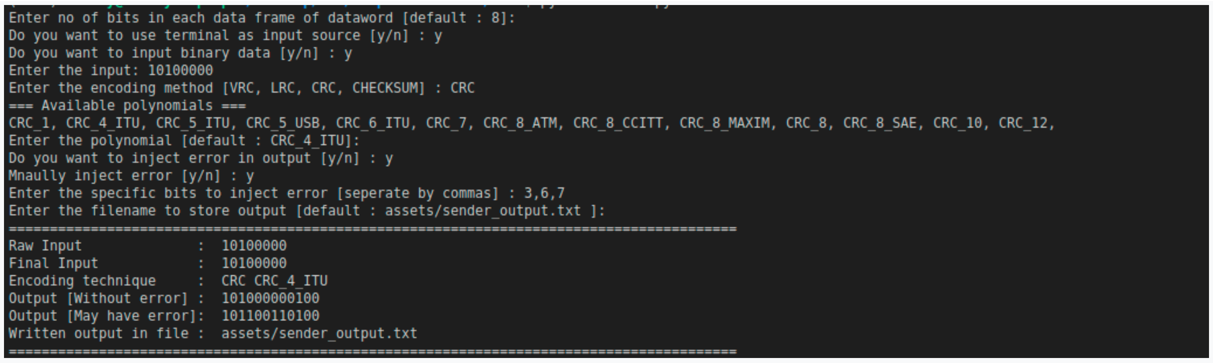
b) Error is detected by VRC but not by CRC

Dataword -> 10 10 00 00

Insert error at index -> 3, 6, 7

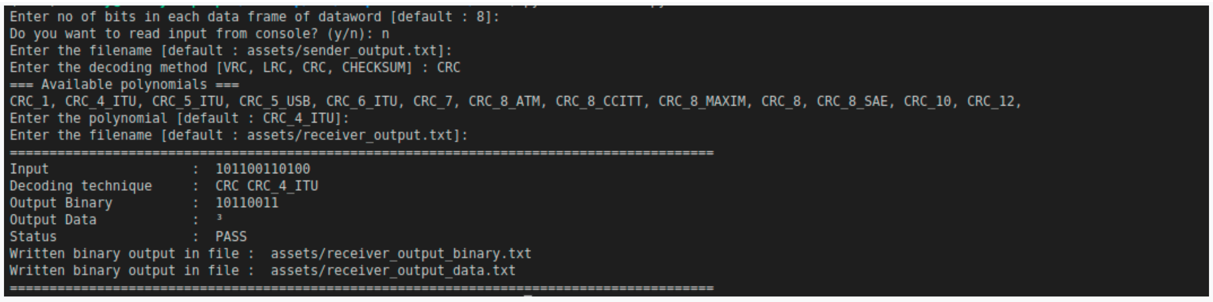
Module used: CRC 4 ITU

Sender Side



```
Enter no of bits in each data frame of dataword [default : 8]:
Do you want to use terminal as input source [y/n] : y
Do you want to input binary data [y/n] : y
Enter the input: 10100000
Enter the encoding method [VRC, LRC, CRC, CHECKSUM] : CRC
=== Available polynomials ===
CRC_1, CRC_4_ITU, CRC_5_ITU, CRC_5_USB, CRC_6_ITU, CRC_7, CRC_8_ATM, CRC_8_CCITT, CRC_8_MAXIM, CRC_8, CRC_8_SAE, CRC_10, CRC_12,
Enter the polynomial [default : CRC_4_ITU]:
Do you want to inject error in output [y/n] : y
Mnaully inject error [y/n] : y
Enter the specific bits to inject error [seperate by commas] : 3,6,7
Enter the filename to store output [default : assets/sender_output.txt ]:
==========================================================================================
Raw Input               :  10100000
Final Input             :  10100000
Encoding technique      :  CRC CRC_4_ITU
Output [Without error] :  101000000100
Output [May have error]:  101100110100
Written output in file :  assets/sender_output.txt
==========================================================================================
```

Receiver Side

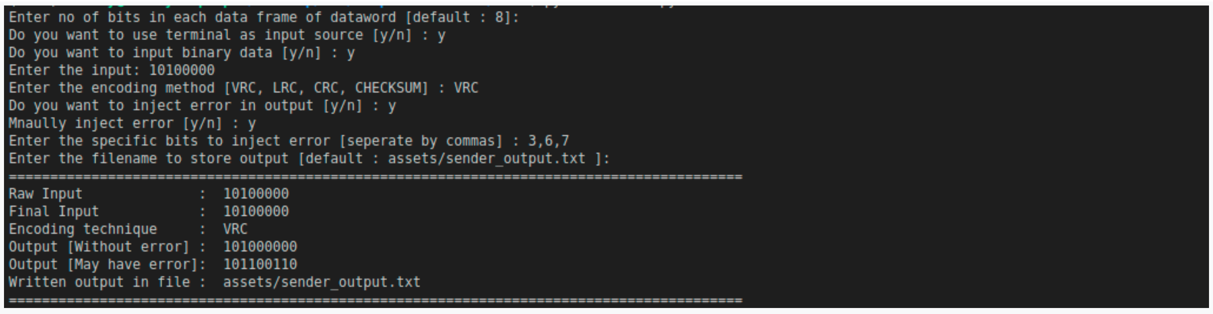


```
Enter no of bits in each data frame of dataword [default : 8]:
Do you want to read input from console? (y/n): n
Enter the filename [default : assets/sender_output.txt]:
Enter the decoding method [VRC, LRC, CRC, CHECKSUM] : CRC
=== Available polynomials ===
CRC_1, CRC_4_ITU, CRC_5_ITU, CRC_5_USB, CRC_6_ITU, CRC_7, CRC_8_ATM, CRC_8_CCITT, CRC_8_MAXIM, CRC_8, CRC_8_SAE, CRC_10, CRC_12,
Enter the polynomial [default : CRC_4_ITU]:
Enter the filename [default : assets/receiver_output.txt]:
==========================================================================================
Input                   :  101100110100
Decoding technique      :  CRC CRC_4_ITU
Output Binary           :  10110011
Output Data             :  ³
Status                  :  PASS
Written binary output in file :  assets/receiver_output_binary.txt
Written binary output in file :  assets/receiver_output_data.txt
==========================================================================================
```

Note : Error Injected but Receiver side can’t detect the error by CRC 4 ITU

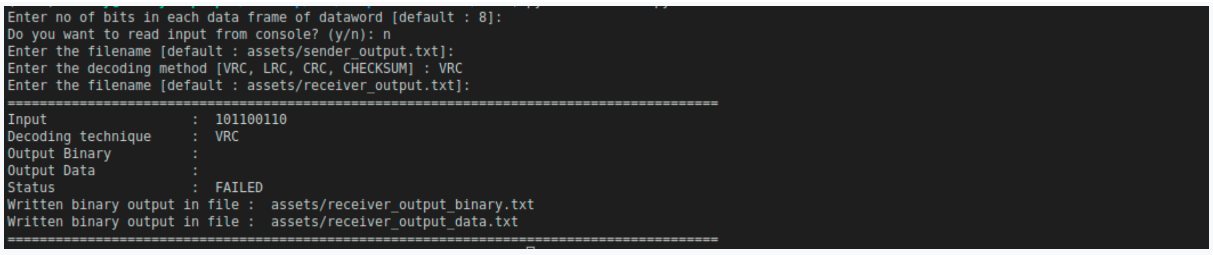
Module used: VRC

Sender Side



```
Enter no of bits in each data frame of dataword [default : 8]:
Do you want to use terminal as input source [y/n] : y
Do you want to input binary data [y/n] : y
Enter the input: 10100000
Enter the encoding method [VRC, LRC, CRC, CHECKSUM] : VRC
Do you want to inject error in output [y/n] : y
Mnaully inject error [y/n] : y
Enter the specific bits to inject error [seperate by commas] : 3,6,7
Enter the filename to store output [default : assets/sender_output.txt ]:
==========================================================================================
Raw Input              :  10100000
Final Input            :  10100000
Encoding technique     :  VRC
Output [Without error] :  101000000
Output [May have error]:  101100110
Written output in file :  assets/sender_output.txt
==========================================================================================
```

Receiver Side:



```
Enter no of bits in each data frame of dataword [default : 8]:
Do you want to read input from console? (y/n): n
Enter the filename [default : assets/sender_output.txt]:
Enter the decoding method [VRC, LRC, CRC, CHECKSUM] : VRC
Enter the filename [default : assets/receiver_output.txt]:
==========================================================================================
Input                  :  101100110
Decoding technique     :  VRC
Output Binary          :
Output Data            :
Status                 :  FAILED
Written binary output in file :  assets/receiver_output_binary.txt
Written binary output in file :  assets/receiver_output_data.txt
==========================================================================================
```

Note : VRC has detected the error in data

ANALYSIS

Vertical Redundancy Check (VRC):

1. This scheme detects all single bit errors. Further, it detects all multiple errors as long as the number of bits corrupted is odd (referred to as odd bit errors). Suppose the message to be transmitted is

   10111 10001 10010

   and the message received at the receiver is

   10011 10001 10010

2. 0 represents that this bit is in error. For the above example, when the receiver performs the parity check, it detects that there was an error in the first nibble during transmission as there is a mismatch between the parity bit and the data in the nibble. However, the receiver does not know which bit in that nibble is in error. Similarly, the following error is also detected by the receiver.

   The message transmitted is

   10111 10001 10010

   and the message received at the receiver is

01111 10011 10010

Three bits are corrupted by the transmission medium and this is detected by the receiver as there is a mismatch between the 2nd nibble and the 2nd parity bit. It is important to note that the error in the first nibble is unnoticed as there is no mismatch between the data and the parity bit.

3. The above example also suggests that not all even bit errors (multiple bit errors with the number of bits corrupted even) are detected by this scheme. If an even bit error is such that the even number of bits are corrupted in each nibble, then such error is unnoticed by the receiver. However, if an even bit is such that at least one of the nibbles has an odd number of bits in error, then a such error is detected by VRC.

Longitudinal Redundancy Check (LRC):

1. Similar to VRC, LRC detects all single bit and odd bit errors. Some even bit errors are detected and the rest is unnoticed by the receiver.
2. The following error is detected by LRC but not by VRC. For the message 1011 1000 1001, supoose the received message is

```
1101
1000
1001
-----
1010
```

3. In the above example, there is a mismatch between the number of 1’s and the parity bit in Columns 2 and 3.
4. The following error is detected by VRC but not by LRC. For the message 1011 1000 1001, suppose the received message is

```
1100
1001
1001
-----
1010
```

The above error is unnoticed by the receiver if we follow VRC scheme whereas it is detected by LRC as illustrated below.

10101 10011 10010

5. Here again, not all even bit errors are detected by this scheme. If the error is such that each column has even number of bits in error, then such error is undetected. However, if the distribution is such that at least one column contains an odd number of bits in error, then such errors are always detected at the receiver.

Checksum:

1. If multiple bit error is such that in each column, a bit ‘0’ is flipped to bit ‘1’, then such an error is undetected by this scheme. Essentially, the message received at the receiver has lost the value 11111 with respect to the sum. Although, it loses this value, this error is unnoticed at the receiver.
2. Also, multiple bit error is such that the difference between the sum of the sender’s data and the sum ofthe receiver’s data is 1111, then this error is unnoticed by the receiver.
3. The above two errors are undetected by the receiver due to the following interesting observations.
4. 4. For any binary data a such that a != 0000, the value of a + (1111) in 1’s complement arithmetic is a. On the similar line, if a1,...,an are nibbles, then a1+...+ak+(1111) = a1+...+ak This is true because, a + 1 1 1 1 gives a – 1 with a carry ‘1’ and inturn this ‘1’ is added with a – 1 as part of 1’s complement addition, yielding a.
5. Consider the scenario in which the sender transmits a1,...,ak along with the checksum and the transmission line corrupts multiple bits due to which we lose 1 1 1 1 on the sum. Although, the receiver received a1+...+ak-(1 1 1 1), it follows from the above observation that a1+...+ak-(1 1 1 1) is still a1+...+ak, thus the error is undetected.
6. Similar to other error detection schemes, checksum detects all odd bit errors and most of even bit errors.

Cyclic Redundancy Check (CRC):

1. The answer to whether CRC detects all errors depends on the divisor polynomial used as a part of CRC computation. Consider the divisor polynomial x^2 which corresponds to 100 and the message to be transmitted is 101010. After appending CRC bits (in this case 0 0) we get 10101000. Assuming during the data transmission, the 3rd bit is in error and the message received at the receiver is 10101100. On performing CRC check on 10101100, we see that the remainder is zero and the receiver wrongly concludes that there is no error in transmission. The reason this error is undetected by the receiver is that the message received is perfectly divisible by the divisor 1 0 0. More appropriately, if we visualize the received message as the xor of the original message and the error polynomial, then we see that the error polynomial is divisible by 100.
2. Message received = 10101100

   Message received = message transmitted + error polynomial, i.e.,

   10101100 = 10101000 + 00000100

   Clearly both are divisible by the divisor 100. In general, if the error polynomial is divisible by the divisor, then such errors are undetected by the receiver. The receiver wrongly concludes that there is no error in transmission. For example, if the error polynomial is 00001100 (3rd and 4th bits in the message in error), then the receiver is unaware of this error. However, if the divisor is 101, then both errors are detected by the receiver. Thus, choosing an appropriate divisor is crucial in detecting errors at the receiver if any during the transmission.

COMMENTS

The assignment has helped me to understand more about the error detection methods in-depth and gave me an in-hand experience and idea about real-world usage in networking by implementing those in python. I am looking forward to implement socket based communication between the sender and receiver