Computer Networks Lab Report – Assignment 1

Name – Adnan Khurshid
Roll – 002010501025
Class – BCSE 3rd year
Group – A1
Assignment Number – 1

Problem Statement – Design and implement an error detection module

Design and implement an error detection module which 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. Sender will send the codeword to the Receiver. Receiver will extract the dataword from codeword and show if there is any error detected. Test the same program to produce a PASS/FAIL result for following cases.
(a) Error is detected by all four schemes. Use a suitable CRC polynomial (list is given in 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.]

DESIGN

A bit, on travel, is subjected to electromagnetic (optical) interference due to noise signals (light sources). Thus, the data transmitted may be prone to errors. In particular, a bit '0' (bit '1') sent by the sender may be delivered as a bit '1' (bit '0') at the receiver. This happens because the voltage present in noise signals either has a direct impact on the voltage present in the data signal or creates a distortion leading to misinterpretation of bits while decoding the signals at the receiver. This calls for a study on error detection and error correction. The receiver must be intelligent enough to detect the error and ask the sender to transmit the data packet or must have the ability to detect and correct the errors.
In this assignment, we shall discuss the following error detection techniques in detail.
1. Vertical Redundancy check
2. Longitudinal Redundancy check
3. Checksum
4. Cyclic Redundancy check

I have implemented the error detection module in a total of 8 program files.

- sender.py (Sender program)
- receiver.py (Receiver program)
- VRC.py (Has a VRC class which has two methods for encoding and decoding in VRC)
- LRC.py (Has a LRC class which has two methods for encoding and decoding in LRC)
- Checksum.py (Has a Checksum class which has two methods for encoding and decoding using Checksum)
- CRC.py (Has a CRC class which has two methods for encoding and decoding in CRC technique)
- config.py (has a dictionary of available polynomials for CRC)
- helper.py (this file has all the helper functions)
    - Binary to string converter
    - String to binary converter
    - XOR calculator for two bits
    - XOR calculator for two strings of bits
    - Function two build frames with size equal to the frame size decided for the program
    - Add two binary strings
    - Add two binary strings by 1’s complement
    - Complement a binary string
    - Generate a divisor for CRC

The individual files fulfil different purposes, following which have been explained in details :

1. **sender.py :** The following are the tasks performed in this Sender program :
    a. Input is read from the terminal or an input file depending on what the user selects
    b. The message sequence is divided into datawords on the basis of frame size taken as the input from the user.

    c. According to the four schemes namely VRC, LRC, Checksum and CRC, redundant bits/dataword are added along with the datawords to form codewords.
    d. The datawords and codewords are displayed before sending.
    e. Sockets are used for sending data
    f. Codewords are then sent to the receiver through the socket

2. **receiver.py :**
   The following are the tasks performed in this Receiver program:
    a. The codewords are received from the sender by socket or are read from the console depending on what the user selects
    b. The received codewords are then decoded according one of the four schemes namely VRC, LRC, Checksum and CRC
    c. The result is checked and shown if there is any error detected.
    d. The codewords and the datawords extracted from the codewords are also displayed.

3. **helper.py :** The helper file has the following helper functions which are used by the sender and receiver classes.
    a. Function to convert string to binary
    b. Function to convert binary to string
    c. Function to find XOR of two bits
    d. Function to find XOR of two binary strings
    e. Function to count zeros and ones in a binary string
    f. Function to build frames according to the frame size given by user
    g. Add two binary strings
    h. Add two binary strings using 1's complement
    i. Function to find complement of a binary string
    j. Function to generate a divisor for CRC

4. **VRC.py :** This file has a VRC class which has two static methods encode and decode
    a. The encode method is called in the sender.py file before sending the data and it encodes the datawords by VRC encoding technique
    b. The decode function is called in the receiver.py file and it decodes the codewords using VRC decoding technique

5. **LRC.py :** This file has a LRC class which has two static methods encode and decode
    a. The encode method is called in the sender.py file before sending the data and it encodes the datawords by LRC encoding technique
    b. The decode function is called in the receiver.py file and it decodes the codewords using LRC decoding technique
6. **Checksum.py :** This file has a Checksum class which has two static methods encode and decode
    a. The encode method is called in the sender.py file before sending the data and it encodes the datawords by Checksum encoding technique
    b. The decode function is called in the receiver.py file and it decodes the codewords using Checksum decoding technique
7. **CRC.py :** This file has a CRC class which has two static methods encode and decode
    a. The encode method is called in the sender.py file before sending the data and it encodes the datawords by CRC encoding technique
    b. The decode function is called in the receiver.py file and it decodes the codewords using CRC decoding technique
8. **config.py :** This file has a dictionary of available CRC polynomials to us.

IMPLEMENTATION :

Helper.py :

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

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

def BinaryToStr(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 CountOnesZeros(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

def addBinaryStr(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 addBinaryStrUsingOnesComplement(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 = addBinaryStrUsingOnesComplement(result, '1')

    return result.zfill(max_len)

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

def binaryStrComplement(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 polynomialParser(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 degrees 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
```

```python
# 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):

    # initialise 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)
    checkword = tmp
    return checkword
```

Sender.py :

```
import random
import socket

from helper import strTobinary, generateDivisor, polynomialParser
from VRC import VRC
from CRC import CRC
from LRC import LRC
from Checksum import 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 = strTobinary(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 = strTobinary(t.replace('\n', '')) if not binary
else t.replace('\n', '')

    def sendOutput(self):
        if self.output == "": raise Exception("Output is empty !
Nothing to save")
        sender = socket.socket()

        print("Sender successfully created")

        port = 12345

        sender.bind(('', port))
        print("Receiver connected at port %s" % port)

        sender.listen(5)
        print("socket is listening")

        while True:
            c, addr = sender.accept()
            print('Got connection from', addr)
            print('Sending data')
            c.send(self.output.encode())
            c.close()
            break
        sender.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



# ? ################## 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 :
files/sender_input.txt ]: ")
        filename = "files/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=noOfOriginalDataFr
amesPerGroup)
    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=noOfOriginalD
ataFramesPerGroup)
    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 =
polynomialParser(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("Manually inject error [y/n] : ").lower() == "y":
            specificBitsToInjectError = input("Enter the specific bits
to inject error [separate 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)


    sender.sendOutput()

    # ! Print data

print("================================================================
=========================")
    print("Raw Input                : ", sender.rawInput)
    print("Datawords              : ", sender.input)
    print("Encoding technique     : ", encodingMethod+"
"+selectedPolynomial if encodingMethod == "CRC" else encodingMethod)
    print("Codewords [Without error] : ",
sender.outputWithoutAnyError),
    print("Codewords [May have error]: ", sender.output)

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

Receiver.py :

```
from helper import BinaryToStr,generateDivisor, polynomialParser
from VRC import VRC
from CRC import CRC
from LRC import LRC
from Checksum import Checksum
from config import availableCRCPolynomials
import socket


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):
        port = 12345
        receiver = socket.socket()
        receiver.connect(('127.0.0.1', port))
        data = receiver.recv(1024).decode()
        self.input = data


    def writeOutputToFile(self, filename, binary):
        tmp = BinaryToStr(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 = BinaryToStr(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 = BinaryToStr(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 = BinaryToStr(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 = BinaryToStr(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:
        receiver.readInputFromFile()


    # ! 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=noOfOriginalData
FramesPerGroup)
    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=noOfOrigina
lDataFramesPerGroup)
    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 =
polynomialParser(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("================================================================
=========================")
```

TEST CASES

Vertical Redundancy Check (VRC) :

First we will check with data entered from the terminal without any error.

Sender :

```
=========================================================================================
Raw Input               :  ABC abcjhf
Datawords              :  0100000101000010010000110010000001100001011000100110001101101010011
10100001100110
Encoding technique     :  VRC
Codewords [Without error] :  01000001001000010001000011100100000101100001101100010101100011
00110101000110100010110011OO
Codewords [May have error]:  01000001001000010001000011100100000101100001101100010101100011
0011010100011010001011001100
=========================================================================================
PS D:\5th SEM\CN LAB\Assignment 1> 
```

Receiver :

```
=========================================================================================
Input                   :  0100000100100001000100001110010000010110000110110001010110001100
011010100011010001011001100
Decoding technique      :  VRC
Output Binary           :  0100000101000010010000110010000001100001011000100110001101101010
0011010000110O110
Output Data             :  ABC abcjhf
Status                  :  PASS
Written binary output in file :  files/receiver_output_binary.txt
Written binary output in file :  files/receiver_output_data.txt
=========================================================================================
```

Now we will get the data from a file and inject some error in it.

Sender :

```
Enter no of bits in each data frame of dataword [default : 8]: 8
Do you want to use terminal as input source [y/n] : n
Do you want to input binary data [y/n] : n
Enter the filename [default : files/sender_input.txt ]:
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] : 2,3
Sender successfully created
Receiver connected at port 12345
socket is listening
Got connection from ('127.0.0.1', 49937)
Sending data
=====================================================================================
Raw Input              :  I am Adnan Khurshid
Datawords             :  0100100100100000011000010110110100100000010000010110010001101110O1
1000010110111000100000010010110110100001110101011100100111001101101000011010010110O100
Encoding technique     :  VRC
Codewords [Without error] :  01001001100100000101100001101101101100100000101000001001100100
0101101110101100001101101110100100000101001011001101000101110101101110010001110011101101OO
010110100100110O1001
Codewords [May have error]:  01111001100100000101100001101101101100100000101000001001100100
0101101110101100001101101110100100000101001011001101000101110101101110010001110011101101OO
0101101001001100100l
```

Receiver :

```
==========================================================================================
Input                 :  01111001100100000101100001101101101100100000101000001001100100
1
01101110101100001101101110100100000101001011001101000101110101101110010001110011101101000
1011010010011001001
Decoding technique    :  VRC
Output Binary         :  01111001001000000110000101101101001000000100000101100100011011
1
00110000101101110001000000100101101101000011101010111001001110011011010000110100101100100
Output Data           :  y am Adnan Khurshid
Status                :  PASS
Written binary output in file :  files/receiver_output_binary.txt
Written binary output in file :  files/receiver_output_data.txt
==========================================================================================
```

Here we see that there was an error at the 2nd and 3rd bit position and the output data is also not what we sent but VRC failed to check even number of errors.

Longitudinal Redundancy Check (LRC) :

Testing with error

```
Enter no of bits in each data frame of dataword [default : 8]: 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: Rome will fall
Enter the encoding method [VRC, LRC, CRC, CHECKSUM] : LRC
Enter the no of data frames per group [default : 4]: 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] : 1,3,6,11
Sender successfully created
Receiver connected at port 12345
socket is listening
Got connection from ('127.0.0.1', 56881)
Sending data
==========================================================================================
Raw Input             :  Rome will fall
Datawords             :  01010010011011111011010110010100100000011101110110100101101100011
0110000010000001100110011000010110110001101100
Encoding technique    :  LRC
Codewords [Without error] :  01010010011011111011010110010100110101001000000111011011010
0101101100010100100110110000100000011001100110000101001011011011000110110000000000000000
000000000
Codewords [May have error]:  00000000011111110110110101100101001101010010000001110111011010
0101101100010100100110110000100000011001100110000101001011011011000110110000000000000000
000000000
==========================================================================================
```

Receiver :

```
Input                 :  0000000001111111011011010110010100110101001000000111011101101001011011000101001001100111000001000000110011001100001010010110110001101100000000000000000000000000000
Decoding technique    :  LRC
Output Binary         :  0000000001111111011011010110010100110101001000000111011101101001011011000101001001100111000001000000110011001100001010010110110001101100000000000000000000000000000
Output Data           :  me will fall
Status                :  FAILED
Written binary output in file :  files/receiver_output_binary.txt
Written binary output in file :  files/receiver_output_data.txt
================================================================================
PS D:\5th SEM\CN LAB\Assignment 1>
```

Testing with error but not failing

LRC fails when two bits in a codeword are flipped and two bits in another codeword are flipped at the same position

Sender :

```
Enter the input: Adnan
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] : 1,2,9,10
Sender successfully created
Receiver connected at port 12345
socket is listening
Got connection from ('127.0.0.1', 56963)
Sending data
================================================================================
Raw Input             :  Adnan
Datawords           :  01000001011001000110111001100001011011 10
Encoding technique    :  LRC
Codewords [Without error] :  0100000101100100011011100110000100101010011011100000000000000000000000001101110
Codewords [May have error]:  0010000100000100011011100110000100101010011011100000000000000000000000001101110
================================================================================
```

Here we have injected error at first two bits of both first and second frame

Receiver:

```
Enter no of bits in each data frame of dataword [default : 8]: 8
Do you want to read input from console? (y/n): n
Enter the decoding method [VRC, LRC, CRC, CHECKSUM] : LRC
Enter the no of data frames per group [default : 4]:
Enter the filename [default : files/receiver_output.txt]:
================================================================================
Input                 :  0010000100000100011011100110000100101010011011100000000000000000000000001101110
Decoding technique    :  LRC
Output Binary         :  0010000100000100011011100110000101101110000000000000000000000000000000
Output Data           :  !♦nan
Status                :  PASS
Written binary output in file :  files/receiver_output_binary.txt
Written binary output in file :  files/receiver_output_data.txt
================================================================================
```

LRC fails to catch the error although data is corrupted.

Checksum :

Testing with error:

Sender :

```
Enter no of bits in each data frame of dataword [default : 8]: 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: My pain is constant and sharp
Enter the encoding method [VRC, LRC, CRC, CHECKSUM] : CHECKSUM
Enter the no of data frames per group [default : 4]: 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] : 1,2,11,3
Sender successfully created
Receiver connected at port 12345
socket is listening
Got connection from ('127.0.0.1', 56992)
Sending data
================================================================================
Raw Input             :  My pain is constant and sharp
Datawords             :  0100110101111001001000000111000001100001011010010110111000100000011010001011100110010000001100011011011110110111001110011011101000110000101101110011101000010000001100001011011100110010000100000011100110110100001100001011100100111000000
Encoding technique    :  CHECKSUM
Codewords [Without error] :  0100110101111001001000000111000010101000011000010110100101101110001000001010011001101001011100110010000001100011100111111011011110111001110011011101000011110100111000010110111001110100001000001001101101100001011011100110010000100000101010110111001101110100001100001011100100101000001110000000000000000000000000000010001111
Codewords [May have error]:  0011110101101001001000000111000010101000011000010110100101101110001000001010011001101001011100110010000001100011100111111011011110111001110011011101000011110100111000010110111001110100001000001001101101100001011011100110010000100000101010110111001101110100001100001011100100101000001110000000000000000000000000000010001111
================================================================================
PS D:\5th SEM\CN LAB\Assignment 1>
```

Receiver :

```
Enter no of bits in each data frame of dataword [default : 8]: 8
Do you want to read input from console? (y/n): n
Enter the decoding method [VRC, LRC, CRC, CHECKSUM] : CHECKSUM
Enter the no of data frames per group [default : 4]: 4
Enter the filename [default : files/receiver_output.txt]:
================================================================================
Input                 :  0011110101101001001000000111000010101000011000010110100101101110001000001010011001101001011100110010000001100011100111111011011110111001110011011101000011110100111000010110111001110100001000001001101101100001011011100110010000100000101010110111001101110100001100001011100100101000001110000000000000000000000000000010001111
Decoding technique    :  CHECKSUM
Output Binary         :  0011110101101001001000000111000001100001011010010110111000100000101001100110100101110011001000000110001110011111101101111011011100111001101110100001111010011100001011011100111010000100000100110110110000101101110011001000010000010101011011100110111010000110000101110010010100000111000000000000000000000000000001000111100110100101110011001000000110001110011111101101111011011100111001101110100001111010011100001011011100111010000100000100110110110000101101110011001000010000010101011000000000000000
Output Data           :  =i pain is constant and sharp
Status                :  FAILED
Written binary output in file :  files/receiver_output_binary.txt
Written binary output in file :  files/receiver_output_data.txt
================================================================================
```

Testing with error but failing to catch error

We have sent data ‘ab’ and changed the bits at the same position of two consecutive datawords. In one the value is decremented and in other it is incremented so Checksum fails to detect that

SENDER :

```
Enter no of bits in each data frame of dataword [default : 8]: 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: ab
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] : 7,15
Sender successfully created
Receiver connected at port 12345
socket is listening
Got connection from ('127.0.0.1', 57011)
Sending data
================================================================================
Raw Input              :  ab
Datawords              :  0110000101100010
Encoding technique     :  CHECKSUM
Codewords [Without error] :  01100001011000100000000000000000111100
Codewords [May have error]:  01100000011000110000000000000000111100
================================================================================
```

RECEIVER :

```
Enter no of bits in each data frame of dataword [default : 8]: 8
Do you want to read input from console? (y/n): n
Enter the decoding method [VRC, LRC, CRC, CHECKSUM] : CHECKSUM
Enter the no of data frames per group [default : 4]:
Enter the filename [default : files/receiver_output.txt]:
================================================================================
Input                  :  01100000011000110000000000000000111100
Decoding technique     :  CHECKSUM
Output Binary          :  01100000011000110000000000000000
Output Data            :  `c
Status                 :  PASS
Written binary output in file :  files/receiver_output_binary.txt
Written binary output in file :  files/receiver_output_data.txt
================================================================================
```

The data is different from what was sent but still it passed.

Longitudinal Redundancy Check (LRC) :

Testing with Error :

SENDER :

```
================================================================================================
Raw Input               :  Patrick Bateman
Datawords               :  01010000011000010111010001110010011010010110001101101011001000000101
0000100110000101110100011001010110110101100001011011110
Encoding technique      :  CRC CRC_4_ITU
Codewords [Without error] :  01010000001001100001111001110100010001110010111001101001010101
1100011100001101011001100100000101001000010000101100001111001110100010001100101001001101101
1100101100001111100110111011100
Codewords [May have error]:  01010000001001100001111001110100010001110010111001101001010101
1100010100001101011101100100000101001000010000101100001111001110100010001000101001001101101
0100101100001111100110111011100
================================================================================================
```

RECEIVER:

```
================================================================================================
Input                   :  01010000001001100001111001110100010001110010111001101001010101011
0001010000110101110110010000010100100001000010110000111100111010001000100010100100110110011
0010111000011110011011101100
Decoding technique      :  CRC CRC_4_ITU
Output Binary           :  01010000011000010111010001110010011010010110001001101011001000000
0010000100110000101110100010001010110110001100000101101110
Output Data             :  Patribk BatElan
Status                  :  FAILED
Written binary output in file :  files/receiver_output_binary.txt
Written binary output in file :  files/receiver_output_data.txt
================================================================================================
```

Testing with error but not catching error:

SENDER :

```
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
Sender successfully created
Receiver connected at port 12345
socket is listening
Got connection from ('127.0.0.1', 57382)
Sending data
================================================================================================
Raw Input               :  10100000
Datawords               :  10100000
Encoding technique      :  CRC CRC_4_ITU
Codewords [Without error] :  101000000100
Codewords [May have error]:  101100110100
================================================================================================
```

RECEIVER :

```
Enter the filename [default : files/receiver_output.txt]:
================================================================================================
Input                   :  101100110100
Decoding technique      :  CRC CRC_4_ITU
Output Binary           :  10110011
Output Data             :  ³
Status                  :  PASS
Written binary output in file :  files/receiver_output_binary.txt
Written binary output in file :  files/receiver_output_data.txt
================================================================================================
```

Error is Detected by CHECKSUM but not CRC :

Input Data : 1 0 1 0 0 0 0 0

Error at bit position : 3,6,7

CRC Polynomial used : `x^4 + x^1 + 1`

Sender in case of Checksum :

```
Enter no of bits in each data frame of dataword [default : 8]: 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] : 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
Sender successfully created
Receiver connected at port 12345
socket is listening
Got connection from ('127.0.0.1', 57451)
Sending data
================================================================================
Raw Input              :  10100000
Datawords            :  10100000
Encoding technique     :  CHECKSUM
Codewords [Without error] :  10100000000000000000000000000001011111
Codewords [May have error]:  10110011000000000000000000000001011111
================================================================================
```

Receiver incase of Checksum :

```
Enter no of bits in each data frame of dataword [default : 8]: 8
Do you want to read input from console? (y/n): n
Enter the decoding method [VRC, LRC, CRC, CHECKSUM] : CHECKSUM
Enter the no of data frames per group [default : 4]:
Enter the filename [default : files/receiver_output.txt]:
================================================================================
Input                :  10110011000000000000000000000001011111
Decoding technique   :  CHECKSUM
Output Binary        :  10110011000000000000000000000000
Output Data          :  ³
Status               :  FAILED
Written binary output in file :  files/receiver_output_binary.txt
Written binary output in file :  files/receiver_output_data.txt
================================================================================
```

Checksum Detects the error.

Sender in case of CRC:

```
Enter no of bits in each data frame of dataword [default : 8]: 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_MA
XIM, 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
Sender successfully created
Receiver connected at port 12345
socket is listening
Got connection from ('127.0.0.1', 57458)
Sending data
================================================================================
Raw Input              :  10100000
Datawords            :  10100000
Encoding technique     :  CRC CRC_4_ITU
Codewords [Without error] :  101000000100
Codewords [May have error]:  101100110100
```

Receiver incase of CRC :

```
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_M
AXIM, CRC_8, CRC_8_SAE, CRC_10, CRC_12,
Enter the polynomial [default : CRC_4_ITU]:
Enter the filename [default : files/receiver_output.txt]:
================================================================================
Input                  :  101100110100
Decoding technique     :  CRC CRC_4_ITU
Output Binary          :  10110011
Output Data            :  ³
Status                 :  PASS
Written binary output in file :  files/receiver_output_binary.txt
Written binary output in file :  files/receiver_output_data.txt
================================================================================
```

CRC does not detect the same error but Checksum does

Error is Detected by VRC but not CRC :

Input Data : 1 0 1 0 0 0 0 0

Error at bit position : 3,6,7

CRC Polynomial used : `x^4 + x^1 + 1`

Sender in case of VRC:

```
Enter no of bits in each data frame of dataword [default : 8]: 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
Sender successfully created
Receiver connected at port 12345
socket is listening
Got connection from ('127.0.0.1', 57476)
Sending data
================================================================================
Raw Input             :  10100000
Datawords             :  10100000
Encoding technique    :  VRC
Codewords [Without error] :  101000000
Codewords [May have error]:  101100110
================================================================================
```

Receiver in case of VRC:

```
Enter no of bits in each data frame of dataword [default : 8]: 8
Do you want to read input from console? (y/n): n
Enter the decoding method [VRC, LRC, CRC, CHECKSUM] : VRC
Enter the filename [default : files/receiver_output.txt]:
================================================================================
Input                 :  101100110
Decoding technique    :  VRC
Output Binary         :  101100110
Output Data           :  ³
Status                :  FAILED
Written binary output in file :  files/receiver_output_binary.txt
Written binary output in file :  files/receiver_output_data.txt
================================================================================
```

Sender in case of CRC:

```
Enter no of bits in each data frame of dataword [default : 8]: 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_MA
XIM, 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
Sender successfully created
Receiver connected at port 12345
socket is listening
Got connection from ('127.0.0.1', 57458)
Sending data
================================================================================
Raw Input             :  10100000
Datawords             :  10100000
Encoding technique    :  CRC CRC_4_ITU
Codewords [Without error] :  101000000100
Codewords [May have error]:  101100110100
```

Receiver in case of CRC:

```
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_M
AXIM, CRC_8, CRC_8_SAE, CRC_10, CRC_12,
Enter the polynomial [default : CRC_4_ITU]:
Enter the filename [default : files/receiver_output.txt]:
================================================================================
Input                 :  101100110100
Decoding technique    :  CRC CRC_4_ITU
Output Binary         :  10110011
Output Data           :  ³
Status                :  PASS
Written binary output in file :  files/receiver_output_binary.txt
Written binary output in file :  files/receiver_output_data.txt
================================================================================
```

VRC detects the error but CRC fails

RESULTS

Vertical Redundancy Check (VRC):
This scheme works as follows: the message to be transmitted is split into frames and with each frame a parity bit is associated before data transmission. Note that with an even parity bit scheme, the number of 1's in the frame including the parity bit must be even. For example, for the message 1 0 0 1 0 1 0 1 with an even parity bit scheme, the message to be transmitted is 1 0 0 1 0 1 0 1 0 . The Right Most bit at the end of the codeword represents the even parity bit corresponding to that dataword.

Longitudinal Redundancy Check (LRC):
In contrast to VRC, LRC assigns a parity byte along with the message to be transmitted. Suppose the message to be transmitted is 1 0 1 0 1 1 0 1 1 0 0 1 1 0 1 0
Then, we compute the even parity nibble as follows:

1 0 1 0 1 1 0 1
1 0 0 1 1 0 1 0
——————————
0 0 1 1 0 1 1 1

We note that in this scheme, the number of 1's in each column including the bit in the parity byte must be even.

Checksum:

In this scheme we perform 1's complement arithmetic on the data to be transmitted. For the message 1 0 1 1 0 1 1 0 1 0 0 1 0 1 0 0 , we compute checksum in two steps.
The first step performs 1's complement addition on the data and the second step performs the complement on the result of Step 1. The result of Step 2 is the checksum which would be augmented along with the data to be transmitted.

Cyclic Redundancy Check (CRC):
CRC performs mod 2 arithmetic (exclusive-OR) on the message using a divisor polynomial. Firstly, the message to be transmitted is appended with CRC bits and the number of such bits is the degree of the divisor polynomial. The divisor polynomial 1 1 0 1 corresponds to the x^3 + x^2 + 1. For example, for the message 1 0 0 1 0 0 with the divisor polynomial 1 1 0 1, the message after appending CRC bits is 1 0 0 1 0 0 0 0 0. We compute CRC on the modified message M.

ANALYSIS

Vertical Redundancy Check (VRC):

1. This scheme detects all single bit errors. Further, it dete 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 1 0 1 1 1 1 0 0 and the message received at the receiver is 1 0 0* 1 1 1 0 0 1

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 byte during transmission as there is a mismatch between the parity bit and the data in the byte. However, the receiver does not know which bit in that byte is in error.

3. Lets take a byte 1 0 1 0 0 1 0 0. And the transmitted codeword is 0* 1* 1 0 0 1 0 0 1. Here VRC will not notice the error at the 1st and 2nd bit as the parity bit is correct.

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 are unnoticed by the receiver.

2. The following error is detected by LRC but not by VRC. For the message 1011 1000 1001, suppose the received message is

1 1 0 1
1 0 0 0
1 0 0 1
—————
1 0 1 0

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 1 1 0 0*

1 0 0 1*
1 0 0 1
—————
1 0 1 0

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

1 0 1 0*1 1 0 0 1*1 1 0 0 1 0

5. Here again, not all even bit errors are detected by this scheme. If the error is such that each column has an 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 1 1 1 1 1 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 of receiver's data is 1 1 1 1, 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. For any binary data such that a != 0 0 0 0, the value of a + (1 1 1 1) in 1's complement arithmetic is a. On the similar line, if a1,...,an are nibbles, then a1+...+ak+(1 1 1 1) = a1+...+ak This is true

because, a + 1 1 1 1 gives a – 1 with a carry ‘1’ and in turn 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 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 1 0 0 and the message to be transmitted is 1 0 1 0 1 0. After appending CRC bits (in this case 0 0) we get 1 0 1 0 1 0 0 0. Assuming during the data transmission, the 3rd bit is in error and the message received at the receiver is 1 0 1 0 1 1*0 0. On performing CRC check on 1 0 1 0 1 1 0 0, 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 visualise the received message as the xor of the original message and the error polynomial, then we see that the error polynomial is divisible by 1 0 0.

2. Message received = 1 0 1 0 1 1 0 0 Message received = message transmitted + error polynomial, i.e., 1 0 1 0 1 1 0 0 = 1 0 1 0 1 0 0 0 + 0 0 0 0 0 1 0 0 Clearly both are divisible by the divisor 1 0 0. 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 0 0 0 0 1 1 0 0 (3rd and 4th bits in the message in error), then the receiver is unaware of this error.

However, if the divisor is 1 0 1, 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

This assignment has helped me in understanding the different error detection schemes immensely, by researching and implementing them. It has also helped in understanding the demerits of a detection scheme, and how such demerits are overcome by other detection scheme