NETWORK LAB REPORT

NAME: ANURAN CHAKRABORTY
ROLL NO.: 20
CLASS: BCSE-III
SECTION: A1

ASSIGNMENT NUMBER: 1

PROBLEM STATEMENT:
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, codeword will be prepared. Sender will send the codeword to the Receiver. Receiver will extract the data word 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.
(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.]

DEADLINE: 24TH JANUARY, 2019
SUBMITTED ON: 24TH JANUARY, 2019
REPORT SUBMITTED ON: 31ST JANUARY, 2019

DESIGN

The program implements the four error-detection mechanisms namely Checksum, LRC, VRC, CRC. The program consists of four files which handles the various aspects of the program.

1. **errorchecker.py**
   This file contains the implementation of all the above algorithms in different functions.
2. **sender.py**
   This file contains the code to perform the work of the sender. Read from the input file, form codeword from the data word, inject error and write to the output file.
3. **receiver.py**
   This code reads from the file written to by the sender, checks if any error exists and accordingly gives an output.
4. **main.py**
   As a wrapper to run all the above modules

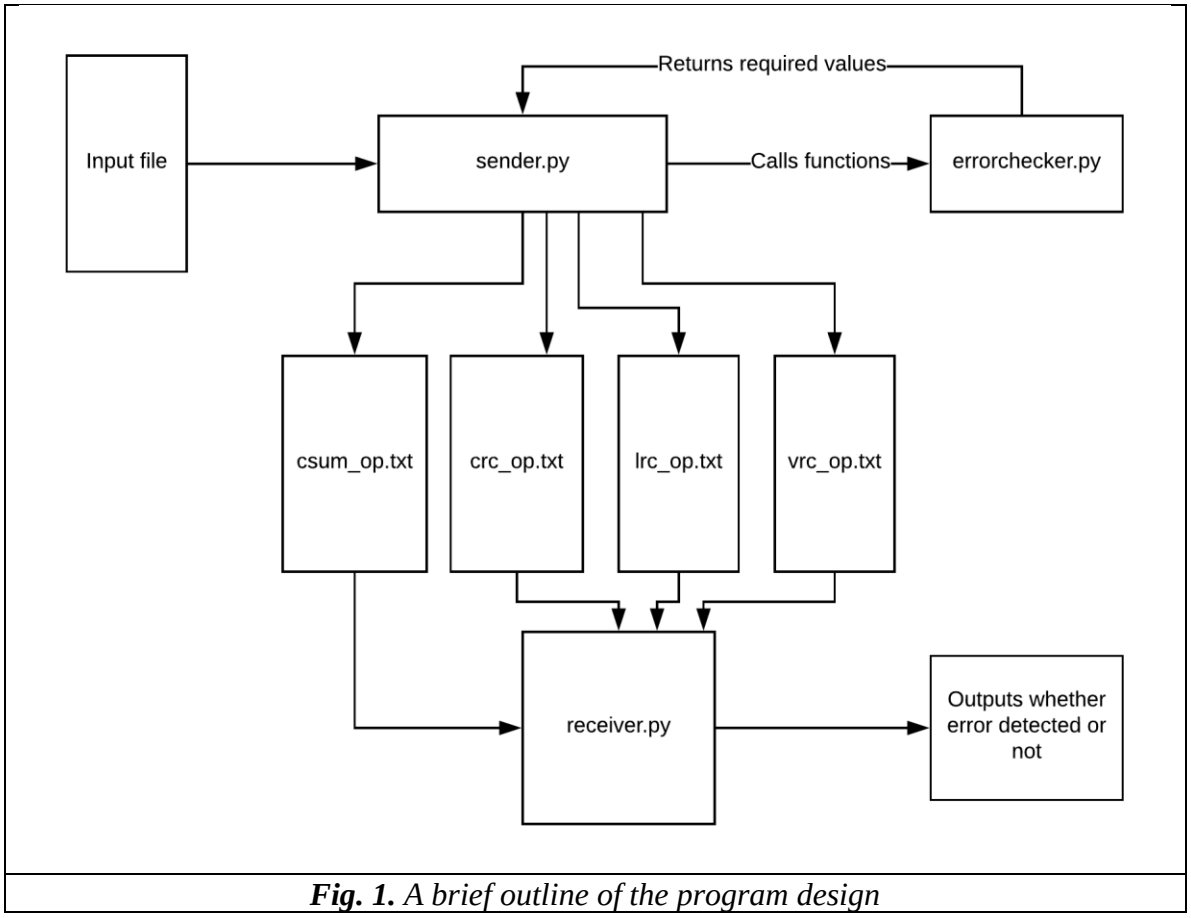

Fig. 1. *A brief outline of the program design*

Fig. 1 gives a brief outline of the program. The sender program reads input from a file, splits it into frames, converts data word to codeword, introduces some random error, and then writes the codeword to the respective files as shown. The receiver then reads from these files, splits it into frames and then applies the algorithms to find whether there is any error introduced or not.

Some important parameters for the design of the program are:
CRC Polynomial: The CRC polynomial which has been considered is CRC-7 ($x^7 + x^3 + 1$).
Frame size: Based on the CRC polynomial chosen, the frame size is chosen to be 8 as it has the same number of terms as in the CRC polynomial.

Assumption: During the design one assumption that has been made is that the number of bits in the input file is a multiple of 8.

Input format: The input for the program is a text file consisting of a string of only 0s and 1s.

Output format: The program outputs whether any error is detected by the respective methods.

IMPLEMENTATION

The assignment has been implemented in Python3. The detailed description is given below.

errorchecker.py

This file has all the error detection algorithms implemented in it.

```
# Customisable parameters
generator_poly='10001001'
no_of_bits=len(generator_poly)
```

The above code snippet defines the CRC polynomial and the frame size.
The following methods are present

checksum(list_of_frames, no_of_bits):

This function takes a list of frames and the frame size as a parameter and return the checksum of all the frames within the list.

```
# Function to find checksum of a number of frames
def checksum(list_of_frames, no_of_bits):
        chksum=0

        for frame in list_of_frames:
                chksum=chksum+int(frame,2) # Computing the sum
        # Wrapping the sum
        csum=bin(chksum)
        csum=csum[2:]

        while(len(csum)>no_of_bits):
                first=csum[0:len(csum)-no_of_bits]
                second=csum[len(csum)-no_of_bits:]
                s=int(first,2)+int(second,2)
                csum=bin(s)
                csum=csum[2:]

        # Perform 1s complement
        while(len(csum)<no_of_bits):
                csum='0'+csum
        chksum=''
        for i in range(len(csum)):
                if(csum[i]=='0'):
                        chksum+='1'
                else:
                        chksum+='0'

        return chksum
```

lrc(list_of_frames, no_of_bits):

This function takes a list of frames and the frame size as a parameter and return the lrc value of all the frames within the list.

```
# Function to generate the LRC code for a list of frames
def lrc(list_of_frames, no_of_bits):
        lsum=0

        for frame in list_of_frames:
                lsum=lsum^int(frame,2)
        lsum=bin(lsum)[2:]

        # Stuffing
        while(len(lsum)<no_of_bits):
                lsum='0'+lsum
        return lsum
```

vrc(list_of_frames):

This function takes a list of frames as a parameter and then modifies the elements of the list i.e. a frame by adding one redundant bit at the end by making sure the entire frame has an even number of 1s (even parity).

```
# Returns codeword for each dataword using vrc
def vrc(list_of_frames):

        codewords=[]
        for i in range(len(list_of_frames)):
                # For every frame check if the parity in even or odd
                dataword=list_of_frames[i]
                if(dataword.count('1')%2==0):
                        dataword+='0'
                else:
                        dataword+='1'
                codewords.append(dataword)

        return codewords
```

crc(list_of_frames, generator, no_of_bits):

This function takes a list of frames and the frame size as a parameter a the creates the CRC codeword for each frame by taking help of the ***modulo2div(dataword, generator)*** function. The ***modulo2div(dataword, generator)*** function takes a dataword and the generator polynomial as input and returns the codeword for the dataword after performing CRC division.

```
# Function to perform modulo 2 division
def modulo2div(dataword, generator):
```

```
        # Number of bits to be XORed a time
        l_xor=len(generator)
        tmp=dataword[0:l_xor]

        while (l_xor<len(dataword)):
                if(tmp[0]=='1'):
                        # If leftmost bit is 1 simply xor and bring the next
bit down
                        tmp=xor(generator,tmp)+dataword[l_xor]
                else:
                        # If leftmost bit is 0 then use all 0 divisor
                        tmp=xor('0'*len(generator),tmp)+dataword[l_xor]
                tmp='0'*(len(generator)-len(tmp))+tmp

                l_xor+=1

        # For the last bit
        if(tmp[0]=='1'):
                tmp=xor(generator,tmp)
        else:
                tmp=xor('0'*len(generator),tmp)
        tmp='0'*(len(generator)-len(tmp)-1)+tmp
        checkword=tmp
        return checkword


# Return the codeword for crc
def crc(list_of_frames, generator, no_of_bits):

        codewords=[]
        for i in range(len(list_of_frames)):
                # For every dataword perform crc division
                dataword=list_of_frames[i]
                # Append length of generator-1 bits to dataword
                aug_dataword=dataword+'0'*(len(generator)-1)

                # Now perform the modulo 2 division
                checkword=modulo2div(aug_dataword,generator)
                # Append the remainder
                codeword=dataword+checkword
                codewords.append(codeword)

        return codewords
```

sender.py

This module is responsible for performing the task of the sender by taking help of the errorchecker module.

readfile(filename, no_of_bits):

The function reads the input file which is passed as argument, splits it into frames, and then returns a list of frames to be used for various error checking algorithms.

```
# Function to read from the input file and convert to a list of frames
def readfile(filename, no_of_bits):
        # Open the file for reading
        f=open(filename,'r')
        data=f.read()

        # Now split the data into frames
        list_of_frames=[data[i:i+no_of_bits] for i in range(0, len(data),
no_of_bits)]
        return list_of_frames
```

ins_error(list_of_frames, frame_no, list_of_bit):

This function takes a list of frames, the frame number and the bit position where error is to be introduced and then returns a new list of frames with the required bits in the required frame toggled. Overall, this function is for inserting error into a particular frame.

```
# Function to introduce error
def ins_error(list_of_frames, frame_no, list_of_bit):

        list_of_frames2=list_of_frames[:]
        frame=list_of_frames2[frame_no]
        new=list(frame)

        # Inserting error in the given bit position here
        for i in range(len(list_of_bit)):

                if(new[list_of_bit[i]]=='0'):
                        new[list_of_bit[i]]='1'
                elif (new[list_of_bit[i]]=='1'):
                        new[list_of_bit[i]]='0'
        list_of_frames2[frame_no]=''.join(new)

        return list_of_frames2
```

write_chksum(list_of_frames, no_of_bits, error_list_frames, error_bit_list):

This function takes as input list of frames, frame size, the positions where any error is to be introduced, and the respective bit position for every frame where error is to be introduced. Then

the checksum is calculated for the frames and the frames (including the checksum frame) are written to the respective file (csum_op.txt). The function ***ins_error()*** is used to insert the errors.

```
# Function to write the checksum frames to file
def write_chksum(list_of_frames, no_of_bits, error_list_frames,
error_bit_list):
        # error_list_frames contains the list of frames to introduce the error
into
        # error_bit_list is a list of lists containing bit posiiton of errrors
in each frame
        chksum=err.checksum(list_of_frames=list_of_frames,
no_of_bits=no_of_bits)

        list_of_frames2=list_of_frames[:]
        list_of_frames2.append(chksum)

        # Printing the frames
        print('Codeword frames sent:')
        print(list_of_frames2)

        # Inserting error
        for i in range(len(error_list_frames)):
                list_of_frames2=ins_error(list_of_frames2,
error_list_frames[i], error_bit_list[i])

        with open('csum_op.txt', 'w') as f:
                for item in list_of_frames2:
                        item=item='0'*(len(err.generator_poly)-1)+item
                        f.write("%s" % item)
```

write_lrc(list_of_frames, no_of_bits, error_list_frames, error_bit_list):

This function takes as input list of frames, frame size, the positions where any error is to be introduced, and the respective bit position for every frame where error is to be introduced. Then the LRC value is calculated by using the respective function in the ***errorchecker*** module for the frames and the frames (including the LRC value frame) are written to the respective file (lrc_op.txt). The function ***ins_error()*** is used to insert the errors.

```
# Function to write the lrc frames to file
def write_lrc(list_of_frames, no_of_bits, error_list_frames, error_bit_list):

        lrcval=err.lrc(list_of_frames=list_of_frames, no_of_bits=no_of_bits)

        list_of_frames2=list_of_frames[:]
        list_of_frames2.append(lrcval)
```

```
        # Printing the frames
        print('Codeword frames sent:')
        print(list_of_frames2)

        # Inserting error
        for i in range(len(error_list_frames)):
                list_of_frames2=ins_error(list_of_frames2,
error_list_frames[i], error_bit_list[i])


        with open('lrc_op.txt', 'w') as f:
                for item in list_of_frames2:
                        item='0'*(len(err.generator_poly)-1)+item
                        f.write("%s" % item)
```

write_vrc(list_of_frames, no_of_bits, error_list_frames, error_bit_list):

This function takes as input list of frames, frame size, the positions where any error is to be introduced, and the respective bit position for every frame where error is to be introduced. Then the VRC frames are obtained by using the respective function in the ***errorchecker*** module and the frames are written to the respective file (vrc_op.txt). The function ***ins_error()*** is used to insert the errors.

```
# Function to write the vrc frames to file
def write_vrc(list_of_frames, no_of_bits, error_list_frames, error_bit_list):

        list_of_frames2=err.vrc(list_of_frames=list_of_frames)[:]

        # Printing the frames
        print('Codeword frames sent:')
        print(list_of_frames2)

        # Inserting error
        for i in range(len(error_list_frames)):
                list_of_frames2=ins_error(list_of_frames2,
error_list_frames[i], error_bit_list[i])

        with open('vrc_op.txt', 'w') as f:
                for item in list_of_frames2:
                        item='0'*(len(err.generator_poly)-2)+item
                        f.write("%s" % item)
```

write_crc(list_of_frames, generator, no_of_bits, error_list_frames, error_bit_list):

This function takes as input list of frames, the generator polynomial, frame size, the positions where any error is to be introduced, and the respective bit position for every frame where error is to be introduced. Then the CRC frames are obtained by using the respective function in the

errorchecker module and the frames are written to the respective file (vrc_op.txt). The function ***ins_error()*** is used to insert the errors.

```
# Function to write the crc frames to file
def write_crc(list_of_frames, generator, error_list_frames, error_bit_list):

	list_of_frames2=err.crc(list_of_frames=list_of_frames,
generator=err.generator_poly, no_of_bits=err.no_of_bits)[:]

	# Printing the frames
	print('Codeword frames sent:')
	print(list_of_frames2)

	# Inserting error
	for i in range(len(error_list_frames)):
		list_of_frames2=ins_error(list_of_frames2,
error_list_frames[i], error_bit_list[i])

	with open('crc_op.txt', 'w') as f:
		for item in list_of_frames2:
			f.write("%s" % item)
```

dataword_to_codeword(list_of_frames, no_of_bits, error_list_frames, error_bit_list):
This function is a wrapper for calling all the above functions.

```
# Coverts dataword to codeword and wrote to the appropriate file
def dataword_to_codeword(list_of_frames, no_of_bits, error_list_frames,
error_bit_list):
	global no_of_errors

	print('Writing to checksum file')
	write_chksum(list_of_frames, no_of_bits, error_list_frames,
error_bit_list)
	print('Writing to lrc file')
	write_lrc(list_of_frames, no_of_bits, error_list_frames,
error_bit_list)
	print('Writing to vrc file')
	write_vrc(list_of_frames, no_of_bits, error_list_frames,
error_bit_list)
	print('Writing to crc file')
	write_crc(list_of_frames, no_of_bits, error_list_frames,
error_bit_list)
```

andop(a,b):
A utility function to perform and operation between two strings.

```
# Function to and two numbers
def andop(a,b):
        andstr=''
        for i in range(len(a)):
                if(a[i]=='0' or b[i]=='0'):
                        andstr+='0'
                else:
                        andstr+='1'
        return andstr
```

receiver.py

This module is responsible for performing the task of the receiver by taking help of the errorchecker module.

readfile(filename, no_of_bits):

This function reads from the output files generated by sender and splits it into frames and returns a list of frames for later error checking.

```
# Function to read from the input file and convert to a list of frames
def readfile(filename, no_of_bits):
        # Open the file for reading
        print('Reading file '+filename)
        f=open(filename,'r')
        data=f.read()

        # Now split the data into frames
        list_of_frames=[data[i:i+no_of_bits] for i in range(0, len(data),
no_of_bits)]

        # Printing the frames
        print('Codeword frames received:')
        print(list_of_frames)

        # list_of_frames=data.split('\n')
        # list_of_frames=list_of_frames[0:-1]
        return list_of_frames
```

check_checksum(list_of_frames, no_of_bits):

This function checks if there is any error in the list of frames passed as a parameter by using checksum. The checksum method of the errorchecker module is used to calculate the checksum and if it returns zero there is no error.

```
# Check for error by checksum
def check_checksum(list_of_frames, no_of_bits):
```

```
        # Removing padding
        list_of_frames=[list_of_frames[i][len(err.generator_poly)-1:] for i in
range(len(list_of_frames))]

        chksum=err.checksum(list_of_frames=list_of_frames,
no_of_bits=no_of_bits)

        if(int(chksum,2)==0):
                # Case of no error extract dataword
                print('No error in data detected by checksum')
                print('Dataword frames are')
                print(list_of_frames[0:-1])

        else:
                print('*** Error detected by checksum')
```

check_lrc(list_of_frames, no_of_bits):

This function checks if there is any error in the list of frames passed as a parameter by using LRC. The lrc method of the errorchecker module is used to calculate the LRC value and if it returns zero there is no error.

```
# Check for error by lrc
def check_lrc(list_of_frames, no_of_bits):

        # Removing padding
        list_of_frames=[list_of_frames[i][len(err.generator_poly)-1:] for i in
range(len(list_of_frames))]
        lrcval=err.lrc(list_of_frames=list_of_frames, no_of_bits=no_of_bits)

        if(int(lrcval,2)==0):
                print('No error in data detected by LRC')
                print('Dataword frames are')
                print(list_of_frames[0:-1])

        else:
                print('*** Error detected by LRC')
```

check_vrc(list_of_frames):

This function checks if there is any error in the list of frames passed as a parameter by using VRC. The parity of 1 for every frame is checked and if it 0 then there is no error in that frame else there is an error in the frame.

```
# Check for error by vrc
def check_vrc(list_of_frames):
```

```
        # Removing padding
        list_of_frames=[list_of_frames[i][len(err.generator_poly)-2:] for i in
range(len(list_of_frames))]
        flag=True

        for i in range(len(list_of_frames)):
                if(list_of_frames[i].count('1')%2!=0):
                        print('*** Error detected in frame '+str(i+1)+' by
VRC')
                        flag=False

        if(flag):
                # No error extract dataword
                print("No error detected in data by VRC")
                list_of_frames=[list_of_frames[i][0:-1] for i in
range(len(list_of_frames))]
                print('Dataword frames are')
                print(list_of_frames)
```

check_crc(list_of_frames, generator):

This function checks if there is any error in the list of frames passed as a parameter by using CRC. For every frame the ***modulo2div***() method of errorchecker module is applied with the generator polynomial passed as an argument. If the remainder is 0 then there is no error in the frame else there is an error in the frame.

combiner():

Used as a wrapper method for calling all the functions.

```
# Function which combines all module
def combiner():
        no_of_bits=err.no_of_bits

        print('============================================================')
        list_of_frames=readfile('csum_op.txt',no_of_bits+len(err.generator_pol
y)-1)
        check_checksum(list_of_frames, no_of_bits)

        list_of_frames=readfile('lrc_op.txt',no_of_bits+len(err.generator_poly
)-1)
        check_lrc(list_of_frames, no_of_bits)

        list_of_frames=readfile('vrc_op.txt',no_of_bits+len(err.generator_poly
)-1)
        check_vrc(list_of_frames)
```

```
        list_of_frames=readfile('crc_op.txt',no_of_bits=no_of_bits+len(err.gen
erator_poly)-1)
        check_crc(list_of_frames,generator=err.generator_poly)
```

main.py

This module combines all the above modules and gives a suitable output for all the cases.

```
import sender as se
import receiver as re
import errorchecker as err

# Main module to show all 3 cases
print('-------------------------------------------------------------------
----')
list_of_frames=(se.readfile('input.txt',no_of_bits=err.no_of_bits))
print('Case1: All 4 schemes can detect the error')
se.dataword_to_codeword(list_of_frames,no_of_bits=err.no_of_bits,
error_list_frames=[0, 1], error_bit_list=[[6], [4]])
re.combiner()
print('-------------------------------------------------------------------
----')

print('-------------------------------------------------------------------
----')
list_of_frames=(se.readfile('input.txt',no_of_bits=err.no_of_bits))
print('Case2: Error detected by checksum but not by crc')
se.dataword_to_codeword(list_of_frames,no_of_bits=err.no_of_bits,
error_list_frames=[0], error_bit_list=[[0, 4, 7]])
re.combiner()
print('-------------------------------------------------------------------
----')

print('-------------------------------------------------------------------
----')
list_of_frames=(se.readfile('input.txt',no_of_bits=err.no_of_bits))
print('Case3: Error detected by VRC not by CRC')
se.dataword_to_codeword(list_of_frames,no_of_bits=err.no_of_bits,
error_list_frames=[1], error_bit_list=[[0, 4, 7]])
re.combiner()
print('-------------------------------------------------------------------
----')

print('-------------------------------------------------------------------
----')
list_of_frames=(se.readfile('input.txt',no_of_bits=err.no_of_bits))
```

```
print('Case4: Error detected by VRC not by LRC')
se.dataword_to_codeword(list_of_frames,no_of_bits=err.no_of_bits,
error_list_frames=[1,2], error_bit_list=[[0],[0]])
re.combiner()
print('-------------------------------------------------------------------
----')

print('-------------------------------------------------------------------
----')
list_of_frames=(se.readfile('input.txt',no_of_bits=err.no_of_bits))
print('Case5: Error detected by LRC not by VRC')
se.dataword_to_codeword(list_of_frames,no_of_bits=err.no_of_bits,
error_list_frames=[1], error_bit_list=[[0, 4]])
re.combiner()
print('-------------------------------------------------------------------
----')
```

TEST CASES

For the 3 cases errors have been inserted manually in order to get the required output. The input file used consists of the following sequence of 0s and 1s

Input: 1001000111001111001111001010101111001110110100 01

With a frame size of 8 bits the given input file has a total of 6 frames.
For the different cases errors have been introduced in different bit positions (0 based indexing from left hand side) as described below

Case 1: Error is detected by all four schemes.

Error introduced in:
Frame 0; Bit position 6
Frame 1; Bit position 4

Output:

```
--------------------------------------------------------------------------------
Case1: All 4 schemes can detect the error
Writing to checksum file
Codeword frames sent:
['10010001', '11001111', '00111100', '10101011', '11001110', '11010001', '00010110']
Writing to lrc file
Codeword frames sent:
['10010001', '11001111', '00111100', '10101011', '11001110', '11010001', '11010110']
Writing to vrc file
Codeword frames sent:
['100100011', '110011110', '001111000', '101010111', '110011101', '110100010']
Writing to crc file
Codeword frames sent:
['100100011010001', '110011111010010', '001111001000111', '101010110100000', '110011101011011', '110100010110101']
============================================================
Reading file csum_op.txt
Codeword frames received:
['000000010010011', '000000011000111', '000000000111100', '000000010101011', '000000011001110', '000000011010001', '000000000010110']
*** Error detected by checksum
Reading file lrc_op.txt
Codeword frames received:
['000000010010011', '000000011000111', '000000000111100', '000000010101011', '000000011001110', '000000011010001', '000000011010110']
*** Error detected by LRC
Reading file vrc_op.txt
Codeword frames received:
['000000100100111', '000000110001110', '000000001111000', '000000101010111', '000000110011101', '000000110100010']
*** Error detected in frame 1 by VRC
*** Error detected in frame 2 by VRC
Reading file crc_op.txt
Codeword frames received:
['100100111010001', '110001111010010', '001111001000111', '101010110100000', '110011101011011', '110100010110101']
*** Error detected in frame 1 by CRC
*** Error detected in frame 2 by CRC
--------------------------------------------------------------------------------
```

Case 2: Error is detected by checksum but not by CRC.

Error introduced in:
Frame 0; Bit positions 0,4,7

Output:

```
--------------------------------------------------------------------------------
Case2: Error detected by checksum but not by crc
Writing to checksum file
Codeword frames sent:
['10010001', '11001111', '00111100', '10101011', '11001110', '11010001', '00010110']
Writing to lrc file
Codeword frames sent:
['10010001', '11001111', '00111100', '10101011', '11001110', '11010001', '11010110']
Writing to vrc file
Codeword frames sent:
['100100011', '110011110', '001111000', '101010111', '110011101', '110100010']
Writing to crc file
Codeword frames sent:
['100100011010001', '110011111010010', '001111001000111', '101010110100000', '110011101011011', '110100010110101']
============================================================
Reading file csum_op.txt
Codeword frames received:
['000000000011000', '000000011001111', '000000000111100', '000000010101011', '000000011001110', '000000011010001', '000000000010110']
*** Error detected by checksum
Reading file lrc_op.txt
Codeword frames received:
['000000000011000', '000000011001111', '000000000111100', '000000010101011', '000000011001110', '000000011010001', '000000011010110']
*** Error detected by LRC
Reading file vrc_op.txt
Codeword frames received:
['000000000110001', '000000110011110', '000000001111000', '000000101010111', '000000110011101', '000000110100010']
*** Error detected in frame 1 by VRC
Reading file crc_op.txt
Codeword frames received:
['000110001010001', '110011111010010', '001111001000111', '101010110100000', '110011101011011', '110100010110101']
Dataword frames are
['00011000', '11001111', '00111100', '10101011', '11001110', '11010001']
No error detected in data by CRC
--------------------------------------------------------------------------------
```

Case 3: Error is detected by VRC but not by CRC.

Error introduced in:
Frame 0; Bit positions 0,4,7

Output:

```
--------------------------------------------------------------------------------
Case3: Error detected by VRC not by CRC
Writing to checksum file
Codeword frames sent:
['10010001', '11001111', '00111100', '10101011', '11001110', '11010001', '00010110']
Writing to lrc file
Codeword frames sent:
['10010001', '11001111', '00111100', '10101011', '11001110', '11010001', '11010110']
Writing to vrc file
Codeword frames sent:
['100100011', '110011110', '001111000', '101010111', '110011101', '110100010']
Writing to crc file
Codeword frames sent:
['100100011010001', '110011111010010', '001111001000111', '101010110100000', '110011101011011', '110100010110101']
============================================================
Reading file csum_op.txt
Codeword frames received:
['000000010010001', '000000001000110', '000000000111100', '000000010101011', '000000011001110', '000000011010001', '000000000010110']
*** Error detected by checksum
Reading file lrc_op.txt
Codeword frames received:
['000000010010001', '000000001000110', '000000000111100', '000000010101011', '000000011001110', '000000011010001', '000000011010110']
*** Error detected by LRC
Reading file vrc_op.txt
Codeword frames received:
['000000100100011', '000000010001100', '000000001111000', '000000101010111', '000000110011101', '000000110100010']
*** Error detected in frame 2 by VRC
Reading file crc_op.txt
Codeword frames received:
['100100011010001', '010001101010010', '001111001000111', '101010110100000', '110011101011011', '110100010110101']
Dataword frames are
['10010001', '01000110', '00111100', '10101011', '11001110', '11010001']
No error detected in data by CRC
--------------------------------------------------------------------------------
```

Case 4: Error is detected by VRC but not by LRC.

Error introduced in:
Frame 1; Bit position 0
Frame 2; Bit position 0

Output:

```
--------------------------------------------------------------------------
Case4: Error detected by VRC not by LRC
Writing to checksum file
Codeword frames sent:
['10010001', '11001111', '00111100', '10101011', '11001110', '11010001', '00010110']
Writing to lrc file
Codeword frames sent:
['10010001', '11001111', '00111100', '10101011', '11001110', '11010001', '11010110']
Writing to vrc file
Codeword frames sent:
['100100011', '110011110', '001111000', '101010111', '110011101', '110100010']
Writing to crc file
Codeword frames sent:
['100100011010001', '110011111010010', '001111001000111', '101010110100000', '110011101011011', '110100010110101']
============================================================
Reading file csum_op.txt
Codeword frames received:
['000000010010001', '000000001001111', '000000010111100', '000000010101011', '000000011001110', '000000011010001', '000000000010110']
No error in data detected by checksum
Dataword frames are
['10010001', '01001111', '10111100', '10101011', '11001110', '11010001']
Reading file lrc_op.txt
Codeword frames received:
['000000010010001', '000000001001111', '000000010111100', '000000010101011', '000000011001110', '000000011010001', '000000011010110']
No error in data detected by LRC
Dataword frames are
['10010001', '01001111', '10111100', '10101011', '11001110', '11010001']
Reading file vrc_op.txt
Codeword frames received:
['000000100100011', '000000010011110', '000000101111000', '000000101010111', '000000110011101', '000000110100010']
*** Error detected in frame 2 by VRC
*** Error detected in frame 3 by VRC
Reading file crc_op.txt
Codeword frames received:
['100100011010001', '010011111010010', '101111001000111', '101010110100000', '110011101011011', '110100010110101']
*** Error detected in frame 2 by CRC
*** Error detected in frame 3 by CRC
--------------------------------------------------------------------------
```

Case 5: Error is detected by LRC but not by VRC.

Error introduced in:
Frame 1; Bit positions 0, 4

Output:

```
--------------------------------------------------------------------------
Case5: Error detected by LRC not by VRC
Writing to checksum file
Codeword frames sent:
['10010001', '11001111', '00111100', '10101011', '11001110', '11010001', '00010110']
Writing to lrc file
Codeword frames sent:
['10010001', '11001111', '00111100', '10101011', '11001110', '11010001', '11010110']
Writing to vrc file
Codeword frames sent:
['100100011', '110011110', '001111000', '101010111', '110011101', '110100010']
Writing to crc file
Codeword frames sent:
['100100011010001', '110011111010010', '001111001000111', '101010110100000', '110011101011011', '110100010110101']
============================================================
Reading file csum_op.txt
Codeword frames received:
['000000010010001', '000000001000111', '000000000111100', '000000010101011', '000000011001110', '000000011010001', '000000000010110']
*** Error detected by checksum
Reading file lrc_op.txt
Codeword frames received:
['000000010010001', '000000001000111', '000000000111100', '000000010101011', '000000011001110', '000000011010001', '000000011010110']
*** Error detected by LRC
Reading file vrc_op.txt
Codeword frames received:
['000000100100011', '000000010001110', '000000001111000', '000000101010111', '000000110011101', '000000110100010']
No error detected in data by VRC
Dataword frames are
['10010001', '01000111', '00111100', '10101011', '11001110', '11010001']
Reading file crc_op.txt
Codeword frames received:
['100100011010001', '010001111010010', '001111001000111', '101010110100000', '110011101011011', '110100010110101']
*** Error detected in frame 2 by CRC
--------------------------------------------------------------------------
```

RESULTS

The performance metric for the evaluation of the above methods is robustness i.e., how well can an algorithm detect errors. For robustness the algorithms were run 10 times with random error injection. Out of the 10 times, CRC failed to detect the error only once, checksum failed about 3 times, LRC about 3 times and VRC about 4 times. For about 4 runs all 4 schemes could detect the error. Thus, we can say CRC is very robust compared to others as the polynomial used has no common factors.

ANALYSIS

Overall the implementation of the assignment is more or less correct. Some possible bugs can arise due to the assumption that the input size is a multiple of the frame size. However, this can easily be overcome by padding the last frame of the input data with 0s so that it is a multiple of the frame size. Also, the program may be modified to work on inter process communication or communication over an unreliable network.

COMMENTS

Overall the lab assignment was a great learning experience as we got to implement the well-known error detection algorithms ourselves. The assignment can be rated as moderately difficult.