Assignment 1: Design and implement an error detection module

Name: Sayantan Biswas

Class: BCSE 3rd year 1st sem

Roll: 001910501057

Group: A2

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 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 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.]

Due on: 09-13 August 2021 (in your respective online lab classes)
Report submission due on: 15 August 2021

DESIGN: The purpose of this program is to detect errors during transmission from sender to receiver using 4 different error detection techniques –

- **VRC (VERTICAL REDUNDANCY CHECK)**
- **LRC (LONGITUDINAL REDUNDANCY CHECK)**
- **CHECKSUM**
- **CRC (CYCLIC REDUNDANCY CHECK)**

The code has been written in Python 3. There are mainly five files(.py) which execute the whole process –

- sender.py : This file performs the role of a sender; reads the frames of the input(raw_input.txt file) which contains a sequence of 0,1 as datawords, then converts it into codewords and then sends them to transmission.py
- transmission.py : After applying the inject error method, this file sends the codeword(from ErrorDetection.py) to the receiver.py
- receiver.py : This file performs the role of a receiver; receives the codeword from transmission.py and rejects or accepts the codeword with the help of suitable error checking methods
- ErrorDetection.py : This file contains all the methods of error detection as different functions
- user.py : This file contains the choices that the user will make to run and check various cases

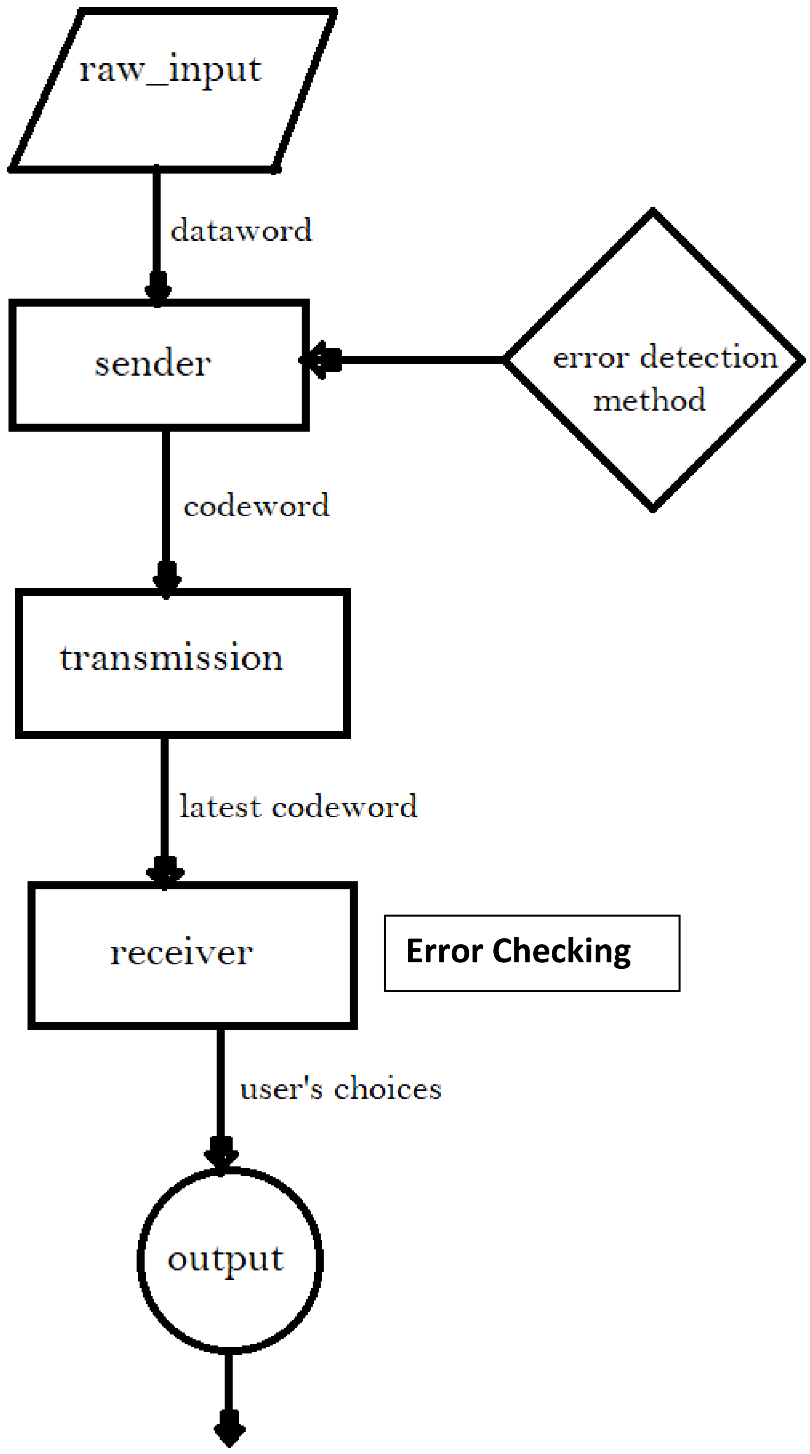


Fig 1: Structure diagram of the program

Input Format: The file ‘raw_input.txt’ is the input. It contains 48 characters in a sequence of 0, 1.

Output Format: Error detected or not by those 4 error detection methods.

IMPLEMENTATION:

The program is written in Python.

Method description is written below with suitable Code snippets:

- **Error Detection:**

  There are 4 techniques in this file (ErrorDetection.py) :

  1) VRC (Vertical Redundancy Check):
  This method takes a list of frames as a parameter, creates codeword by adding one redundant bit at the end of every frame so that the total number of 1s become even.

```python
# 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
```

2) LRC (Longitudinal Redundancy Check):

This method takes a list of frames and a frame size (no_of_bits) as parameters. It returns the LRC code of all frames.

```
# 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
```

3) Checksum:

This method takes a list of frame and frame size as a parameter and returns the checksum of all frames. the checksum is appended at the end of the last frame.

```
# 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) # binary form
    csum=csum[2:] # Adding csum starting from index 2 (0 based indexing)

    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
```

4) CRC (Cyclic Redundancy Check):

This method takes a list of frame, generator (CRC polynomial) and a frame size (no_of_bits) as parameters. In order to create the codeword for CRC; modulo 2 division function is there. This modulo 2 division function takes the dataword and the generator polynomial as parameters which performs the CRC division to return the codewords.

Below code snippet defines the CRC polynomial and frame size (i.e. no_of_bits)

```
# CRC-4-ITU -> x^4 + x + 1
generator_poly = '10011'
no_of_bits_crc = len(generator_poly) # n = 5
```

a xor function for modulo 2 division :

```
# xor function for two binary strings which is typecasted to integer
def xor(a,b):
    a=int(a,2) # returns the integer value which is equivalent to binary string
    b=int(b,2)
    a=a^b
    a=bin(a)[2:]
    return a
```

modulo 2 division function :

```
# 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
```

- **Sender :**
  This file (sender.py) performs the role of a sender; it creates the codewords from datawords with the help of ErrorDetection.py

1) This function reads the input file which passed as argument, splits into frames and then returns a list of frames.

```
# Function to read from the input file and convert it to a list of frames
def readfile(filename, no_of_bits):
    # Open the file in read mode
    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
```

2) This function takes list of frames and the frame size as parameters and it creates the LRC value with the help of Error Detection methods and it appends at the end of the last frame. It also prints the list of final frames which contains codewords.

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

    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) #Append the resulted LRC at the last frame of
datawords

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

    with open('Original_sender.txt', 'w') as fw:
        fw.write("\n\nOriginal LRC ---\n")
        for item in list_of_frames2:

            fw.write("%s | " % item)
```

3) This function takes list of frames and the frame size as parameters and it creates the VRC value with the help of Error Detection methods and it

appends at the end of every frame and also prints the list of final frames which contains codewords.

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

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

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

    with open('Original_sender.txt', 'a') as fw:
        fw.write("\n\nOriginal VRC ---\n")
        for item in list_of_frames2:

            fw.write("%s | " % item)
```

4) This method is responsible for writing the checksum with the help of error detection methods from the checksum algorithm and then finally append at the last frame. Then it creates codewords.

```python
# Function to write the checksum frames to the file
def write_chksum(list_of_frames, no_of_bits):

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

    list_of_frames2=list_of_frames[:] # added all frames
    list_of_frames2.append(chksum) # put checksum at the end of last frame

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

    with open('Original_sender.txt', 'a') as fw:
        fw.write("\n\nOriginal checksum ---\n")
        for item in list_of_frames2:

            fw.write("%s | " % item)
```

5) This method is responsible for creating the CRC value with the help of Error Detection methods from the CRC algorithm. Then, it creates codewords.

```python
# Function to write the crc frames to the file
def write_crc(list_of_frames, generator):
    list_of_frames2=err.crc(list_of_frames=list_of_frames, generator=err.generato
r_poly,
                                no_of_bits=err.no_of_bits_crc)[:]

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

    with open('Original_sender.txt', 'a') as fw:
        fw.write("\n\nOriginal CRC ---\n")
        for item in list_of_frames2:

            fw.write("%s | " % item)
```

Transmission :

This file (transmission.py) injects the error into the codewords, then transmits those codewords to receiver.py

1) This method is used to inject the errors.

```
# Function to inject error
def inject_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
```

2) This method is responsible for sending the list of frames where the error was injected in the frame with the appropriate bit position.

```
# sending codewords through transmission.py
# error_bit_list is a list of lists containing bit posiiton of errors in each fra
me
def sending_codeword(list_of_frames, no_of_bits, error_list_frames, error_bit_lis
t):
    global no_of_errors

    medium_lrc(list_of_frames, no_of_bits, error_list_frames, error_bit_list)

    medium_vrc(list_of_frames, no_of_bits, error_list_frames, error_bit_list)

    medium_chksum(list_of_frames, no_of_bits, error_list_frames, error_bit_list)

    medium_crc(list_of_frames, no_of_bits, error_list_frames, error_bit_list)
```

3) This method writes the LRC frames to file.

```
# Function to write the lrc frames to file
def medium_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)

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

        # Reciever will read this txt file
    with open('Transmission_LRC.txt', 'w') as fw:
        for item in list_of_frames2:
            item='0'*(len(err.generator_poly)-
1)+item # for list maintaining order
                                                            # without this, bug arises

            fw.write("%s" % item)

    with open('Latest_reciver.txt', 'w') as fw:
        fw.write('\n\nReciever LRC ---\n')
        for i in list_of_frames2:

            fw.write("%s | " % i)
```

.

4) This method writes the VRC frames to file.

```
# Function to insert error toggling the vrc introduced frames to file
def medium_vrc(list_of_frames, no_of_bits, error_list_frames, error_bit_list):

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

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

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

    with open('Latest_reciver.txt', 'a') as fw:
        fw.write('\n\nReciever VRC ---\n')
        for i in list_of_frames2:

            fw.write("%s | " % i)
```

5) This method writes the checksum frames to file.

```
# Function to insert error toggling the checksum frames to file
def medium_chksum(list_of_frames, no_of_bits, error_list_frames, error_bit_list):

    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)

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

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

    with open('Latest_reciver.txt', 'a') as fw:
        fw.write('\n\nReciever checksum ---\n')
        for i in list_of_frames2:

            fw.write("%s |" % i)
```

6) This method writes the CRC frames to file.

```
# Function to insert error toggling the CRC frames to file
def medium_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_crc)[:]

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

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

    with open('Latest_reciver.txt', 'a') as fw:
        fw.write('\n\nReciever CRC ---\n')
        for i in list_of_frames2:

            fw.write("%s | " % i)
```

Receiver :

This file (receiver.py) performs the role of a receiver and it checks the error(s) is(are) present or not with the help of error checking methods.

1) In this method, it takes a list of frames and size of frames to detect the error for LRC codeword with the help of error checking method.

```
# 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 the appended value is zero
    if(int(lrcval,2)==0):
        print('No error is detected by LRC')
        print('Dataword frames are ')
        print(list_of_frames[0:-1])

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

2) In this method, it takes a list of frames to detect the error for VRC codeword with the help of error checking method.

```
# 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 is detected in frame '+str(i+1)+' by VRC')
            flag=False

    if(flag):
        # No error extract dataword
        print("No error is detected 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)
```

3) In this method, it takes a list of frames and the frame size to detect the error for checksum codeword with the help of error checking method.

```
# 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):
        # In case of no error detected then dataword is printed
        print('No error is detected by checksum')
        print('Dataword frames are ')
        print(list_of_frames[0:-1])

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

4) In this method, it takes a list of frames and the CRC polynomial to detect the error for checksum codeword with the help of error checking method. For every frame, the **modulo2div()** method is applied and if the remainder is 0 then there is no error otherwise there is an error in the frame.

```
# Check for error by crc
def check_crc(list_of_frames, generator):

    flag=True
    for i in range(len(list_of_frames)):
        if(int(err.modulo2div(list_of_frames[i],err.generator_poly),2)!=0):
            print('Error is detected in frame '+str(i+1)+' by CRC')
            flag=False

    if(flag):
        list_of_frames=[list_of_frames[i][0:err.no_of_bits_crc]
        for i in range(len(list_of_frames))]
        print('Dataword frames are ')
        print(list_of_frames)
        print("No error is detected by CRC")
```

User :

This file (user.py) combines all the above files, functions and gives the output with the help of user choices for different cases.

```
def case1() :

    list_of_frames=(se.readfile('raw_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)
    tm.sending_codeword(list_of_frames,no_of_bits=err.no_of_bits, error_list_fram
es=[0, 1], error_bit_list=[[2], [3]])
    re.modules()
    print('--------------------------------------------------------------------
----')

def case2() :

    list_of_frames=(se.readfile('raw_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)
    #tm.sending_codeword(list_of_frames,no_of_bits=err.no_of_bits, error_list_fra
mes=[0], error_bit_list=[[0, 4, 7]])
    tm.sending_codeword(list_of_frames,no_of_bits=err.no_of_bits, error_list_fram
es=[0], error_bit_list=[[0, 3, 4]])
    re.modules()
    print('--------------------------------------------------------------------
----')

def case3() :

    list_of_frames=(se.readfile('raw_input.txt',no_of_bits=err.no_of_bits))
    print('Case3: Error detected by VRC but not by CRC')
    se.dataword_to_codeword(list_of_frames, no_of_bits=err.no_of_bits)
    #tm.sending_codeword(list_of_frames,no_of_bits=err.no_of_bits, error_list_fra
mes=[1], error_bit_list=[[0, 4, 7]])
```

```
    tm.sending_codeword(list_of_frames,no_of_bits=err.no_of_bits, error_list_fram
es=[0, 1], error_bit_list=[[0, 1, 5, 6, 8]])
    re.modules()
    print('-----------------------------------------------------------------------
----')
```

TEST CASES :

The input has been kept in a file (*raw_input*). It consists of 48 bits of binary number (i.e. sequence of 0, 1).

INPUT DATA: 100100011100111100111100101010111100111011010001

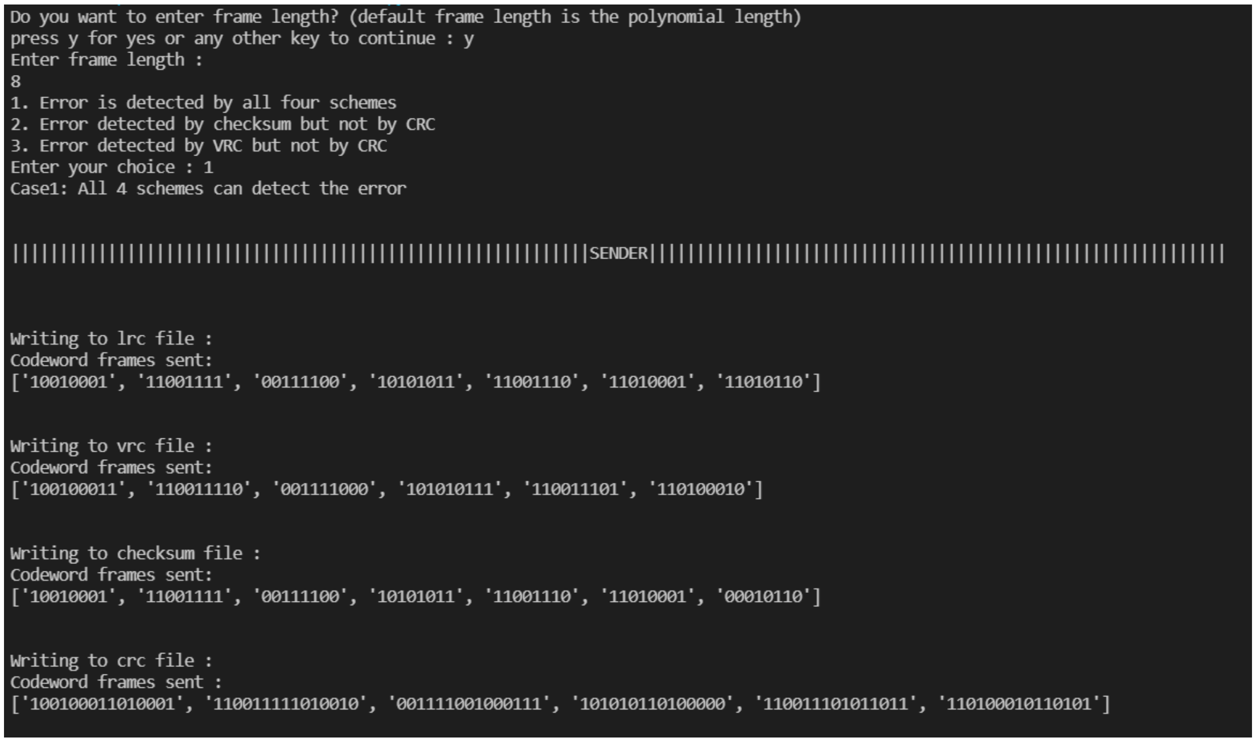
CRC Polynomial: 10001001 (CRC-7 -> *x*^7 + *x*^3 + 1)

I have taken 8 bits frame size. Total number of frames is 6.

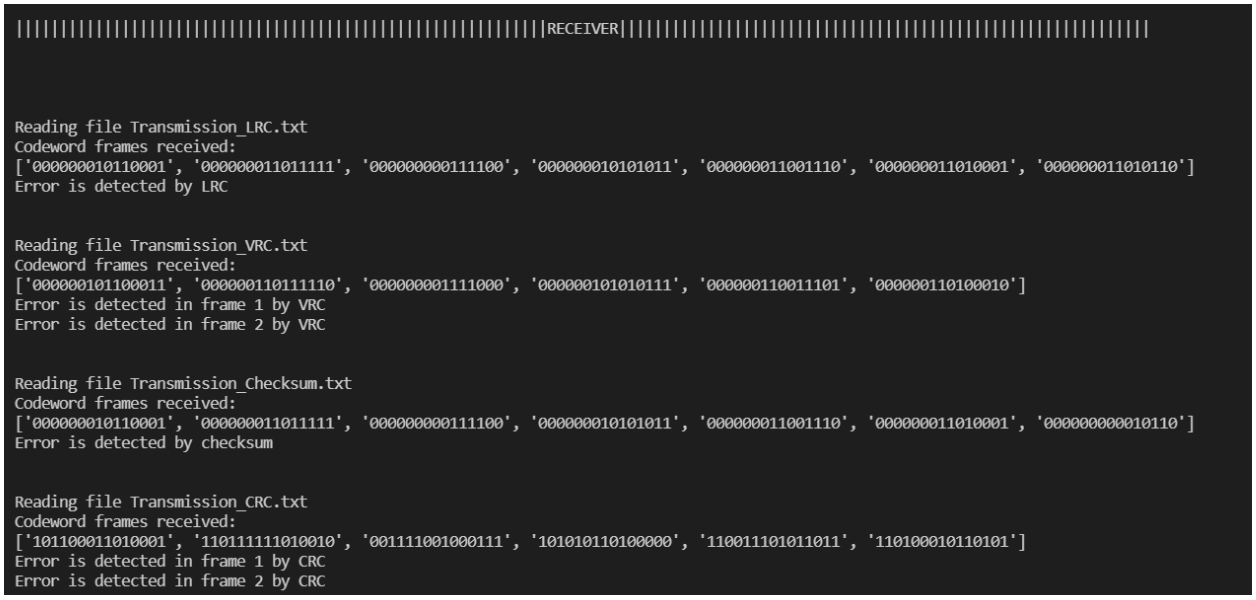
- Case 1: Error is detected by all four schemes.

In transmission.py errors have been injected in frame 0 (bit position 2) and frame 1 (bit position 3). All schemes detect error.

OUTPUT:



```
Do you want to enter frame length? (default frame length is the polynomial length)
press y for yes or any other key to continue : y
Enter frame length :
8
1. Error is detected by all four schemes
2. Error detected by checksum but not by CRC
3. Error detected by VRC but not by CRC
Enter your choice : 1
Case1: All 4 schemes can detect the error


||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||SENDER||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||



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 checksum file :
Codeword frames sent:
['10010001', '11001111', '00111100', '10101011', '11001110', '11010001', '00010110']


Writing to crc file :
Codeword frames sent :
['100100011010001', '110011111010010', '001111001000111', '101010110100000', '110011101011011', '110100010110101']
```

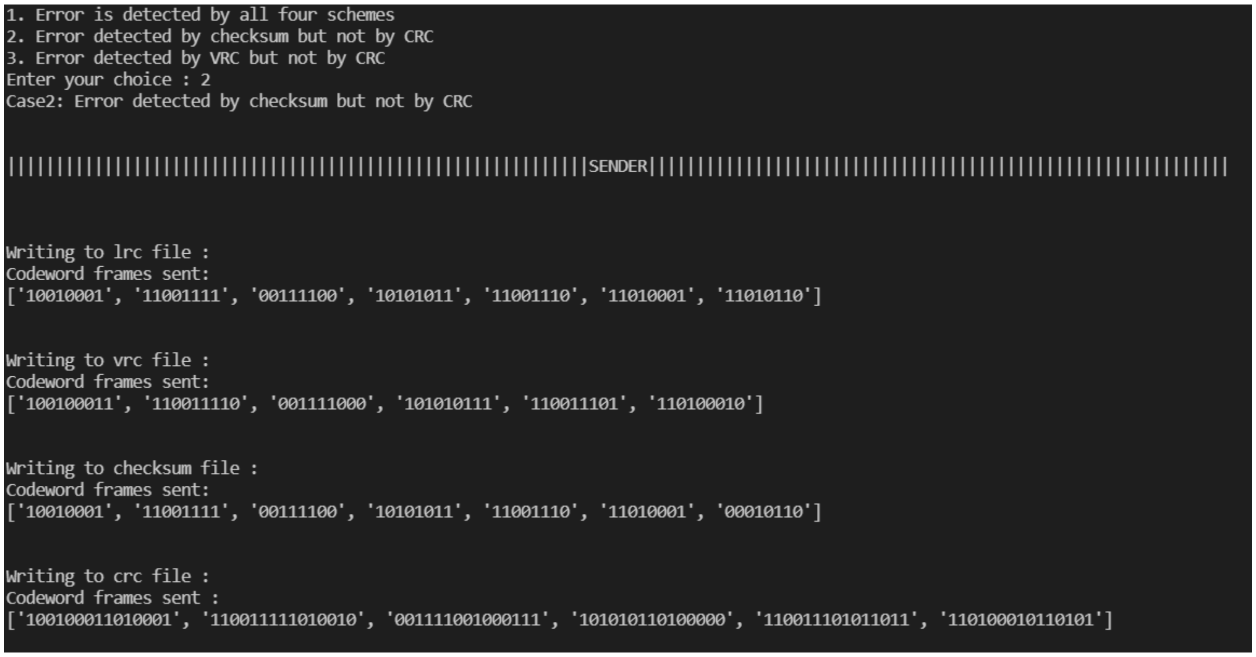


```
|||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||RECEIVER|||||||||||||||||||||||||||||||||||||||||||||||||||||||||||



Reading file Transmission_LRC.txt
Codeword frames received:
['000000010110001', '000000011011111', '000000000111100', '000000010101011', '000000011001110', '000000011010001', '000000011010110']
Error is detected by LRC


Reading file Transmission_VRC.txt
Codeword frames received:
['000000101100011', '000000110111110', '000000001111000', '000000101010111', '000000110011101', '000000110100010']
Error is detected in frame 1 by VRC
Error is detected in frame 2 by VRC


Reading file Transmission_Checksum.txt
Codeword frames received:
['000000010110001', '000000011011111', '000000000111100', '000000010101011', '000000011001110', '000000011010001', '000000000010110']
Error is detected by checksum


Reading file Transmission_CRC.txt
Codeword frames received:
['101100011010001', '110111111010010', '001111001000111', '101010110100000', '110011101011011', '110100010110101']
Error is detected in frame 1 by CRC
Error is detected in frame 2 by CRC
```

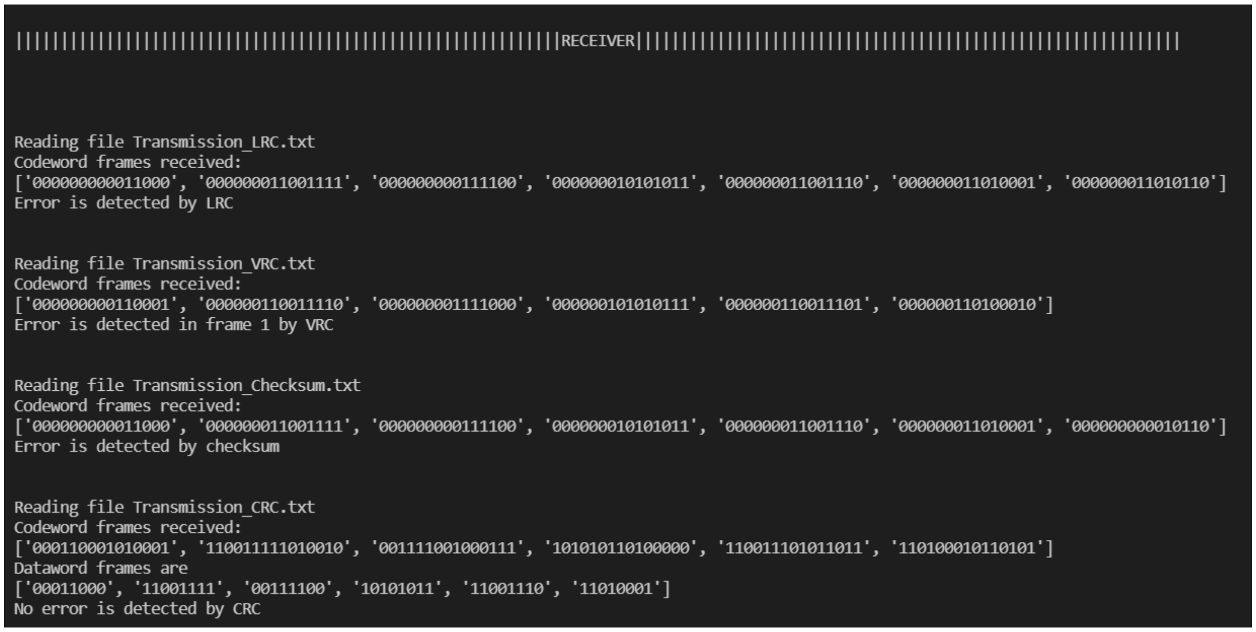
- Case 2: Error is detected by checksum but not by CRC.

In transmission.py errors have been injected in frame 0 (bit position 0, 4, 7). All schemes detect error except CRC.

OUTPUT:



```
1. Error is detected by all four schemes
2. Error detected by checksum but not by CRC
3. Error detected by VRC but not by CRC
Enter your choice : 2
Case2: Error detected by checksum but not by CRC

||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||SENDER||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||


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 checksum file :
Codeword frames sent:
['10010001', '11001111', '00111100', '10101011', '11001110', '11010001', '00010110']


Writing to crc file :
Codeword frames sent :
['100100011010001', '110011111010010', '001111001000111', '101010110100000', '110011101011011', '110100010110101']
```

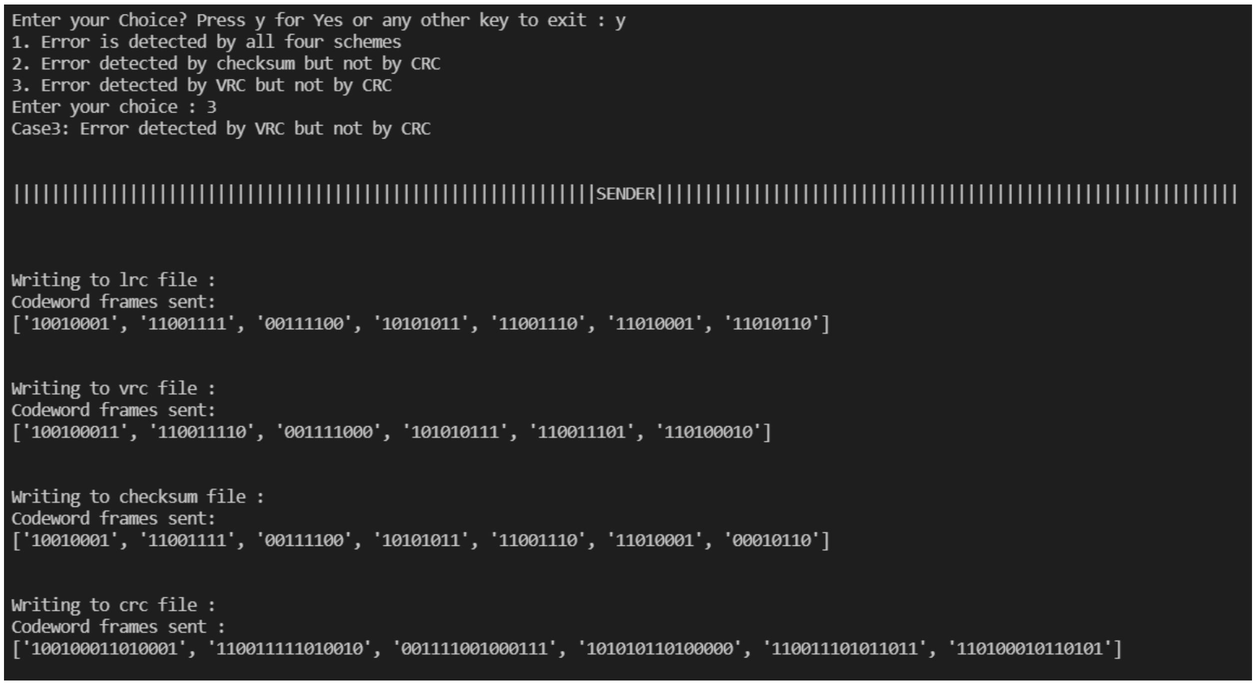


```
||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||RECEIVER||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||



Reading file Transmission_LRC.txt
Codeword frames received:
['000000000011000', '000000011001111', '000000000111100', '000000010101011', '000000011001110', '000000011010001', '000000011010110']
Error is detected by LRC


Reading file Transmission_VRC.txt
Codeword frames received:
['000000000110001', '000000110011110', '000000001111000', '000000101010111', '000000110011101', '000000110100010']
Error is detected in frame 1 by VRC


Reading file Transmission_Checksum.txt
Codeword frames received:
['000000000011000', '000000011001111', '000000000111100', '000000010101011', '000000011001110', '000000011010001', '000000000010110']
Error is detected by checksum


Reading file Transmission_CRC.txt
Codeword frames received:
['000110001010001', '110011111010010', '001111001000111', '101010110100000', '110011101011011', '110100010110101']
Dataword frames are
['00011000', '11001111', '00111100', '10101011', '11001110', '11010001']
No error is detected by CRC
```

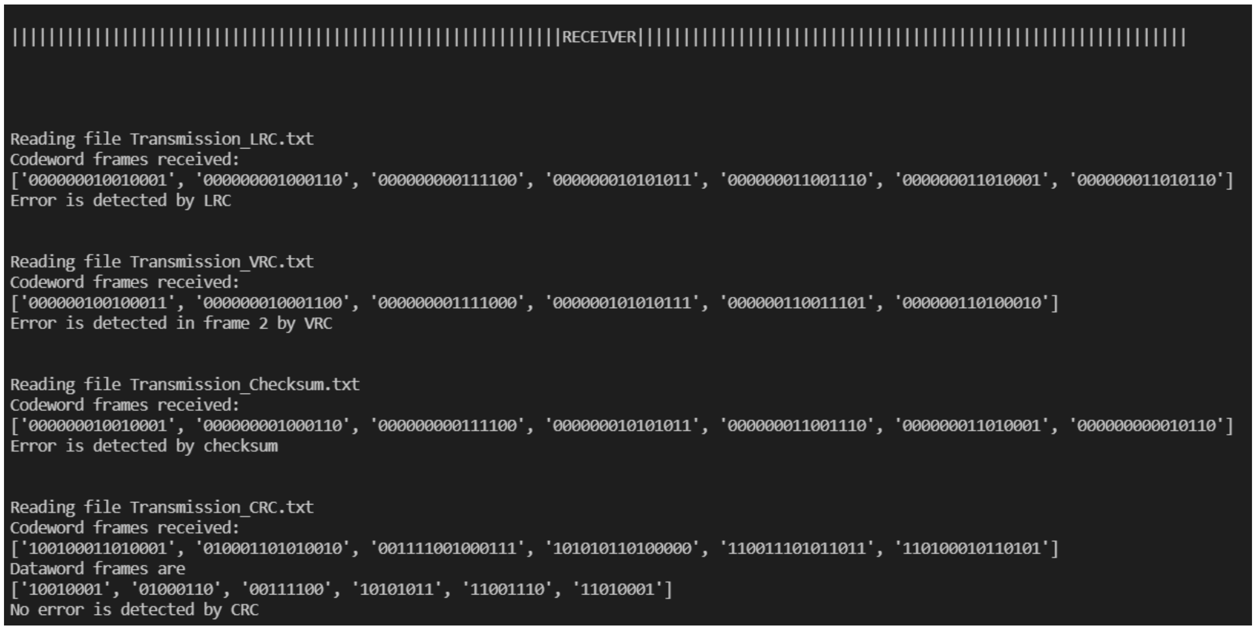
- Case 3: Error is detected by VRC but not by CRC.

In transmission.py errors have been injected in frame 1 (bit position 0, 4, 7). All schemes detect error except CRC.

OUTPUT:



```
Enter your Choice? Press y for Yes or any other key to exit : y
1. Error is detected by all four schemes
2. Error detected by checksum but not by CRC
3. Error detected by VRC but not by CRC
Enter your choice : 3
Case3: Error detected by VRC but not by CRC

||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||SENDER||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||


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 checksum file :
Codeword frames sent:
['10010001', '11001111', '00111100', '10101011', '11001110', '11010001', '00010110']


Writing to crc file :
Codeword frames sent :
['100100011010001', '110011111010010', '001111001000111', '101010110100000', '110011101011011', '110100010110101']
```



```
|||||||||||||||||||||||||||||||||||||||||||||||||||||||||RECEIVER|||||||||||||||||||||||||||||||||||||||||||||||||||||||||


Reading file Transmission_LRC.txt
Codeword frames received:
['000000010010001', '000000001000110', '000000000111100', '000000010101011', '000000011001110', '000000011010001', '000000011010110']
Error is detected by LRC


Reading file Transmission_VRC.txt
Codeword frames received:
['000000100100011', '000000010001100', '000000001111000', '000000101010111', '000000110011101', '000000110100010']
Error is detected in frame 2 by VRC


Reading file Transmission_Checksum.txt
Codeword frames received:
['000000010010001', '000000001000110', '000000000111100', '000000010101011', '000000011001110', '000000011010001', '000000000010110']
Error is detected by checksum


Reading file Transmission_CRC.txt
Codeword frames received:
['100100011010001', '010001101010010', '001111001000111', '101010110100000', '110011101011011', '110100010110101']
Dataword frames are
['10010001', '01000110', '00111100', '10101011', '11001110', '11010001']
No error is detected by CRC
```

Results :

Performance Metric :

Error Ratio = (False Positive) / (Frames Transmitted)

Where, False Positive means the error is injected (i.e. the frame has been changed) during the transmission but still accepted by scheme.

Evaluation Table:

Methods	**Error Ratio**
LRC	0.5
VRC	0.5
Checksum	0.5
CRC	0.16

10 independent executions have been taken. Total number of frames (dataword) for each scheme per execution is 60 (6*10) and frame size is 8 bit. Data is 48 bit.

Analysis :

- After evaluating the results, we can say CRC is sturdier and reliable than most other known schemes.
- Error detection capabilities of the code are increased significantly when all 4 schemes are used.

Comments :

The assignment has helped in understanding and implementing the various error detection schemes. I have analyzed the performance of each of the schemes after introducing random errors. The difficulty level was moderate.