Data Communication and Networks

Error Detection and Correction

Prashanth.S (19MID0020)

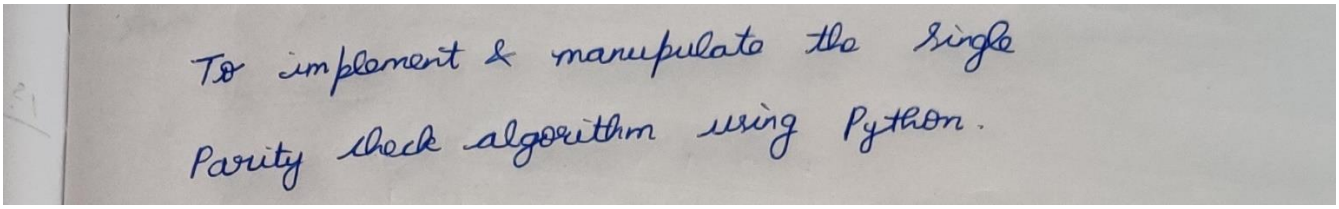
Aim

To implement and manipulate the Error detection and Correction algorithms used in Networking theory using Python

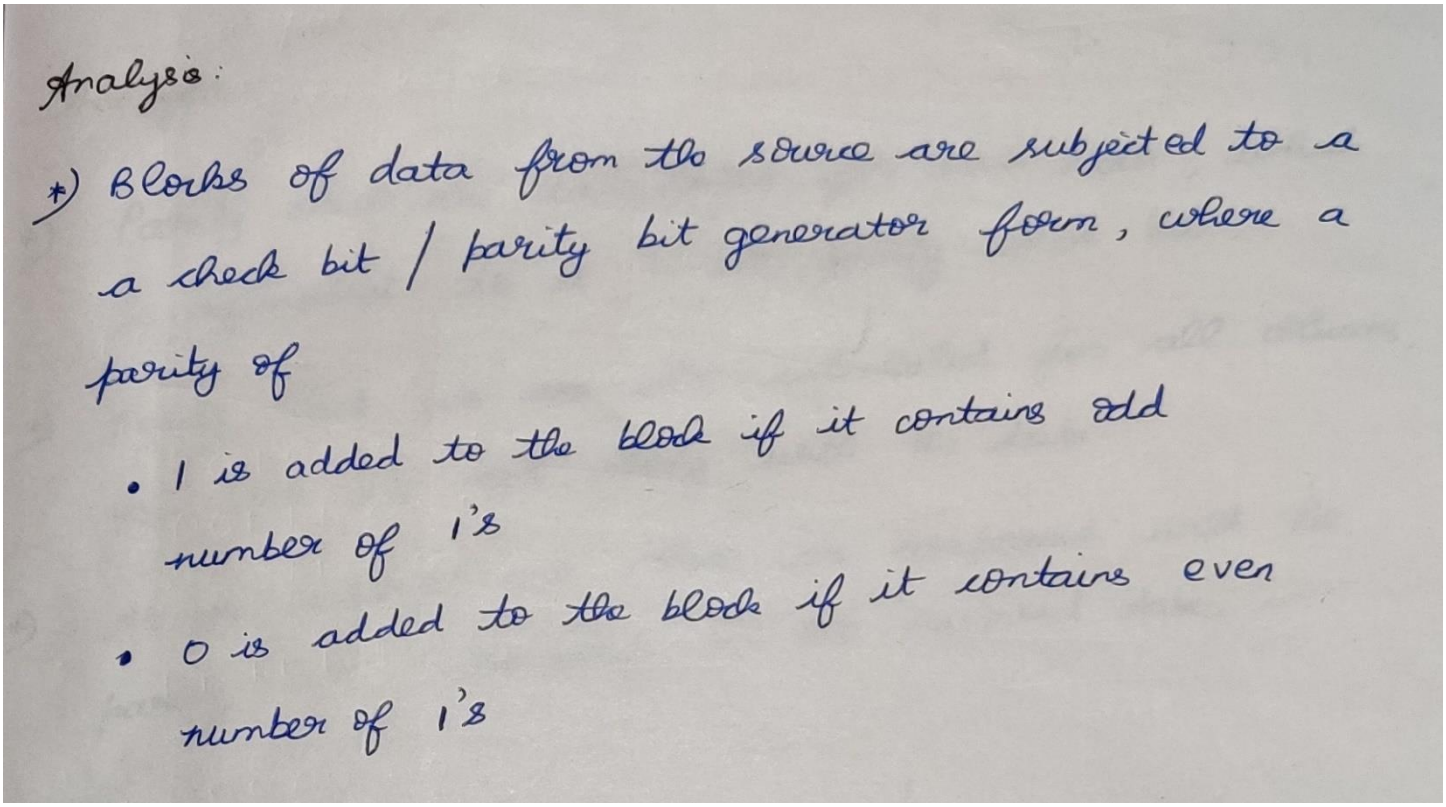
Error Detection Methods

1. Single Parity
2. 2-Dimensional Parity
3. Check-Sum
4. Cyclic Redundancy Check

1)Single Parity


To implement & manipulate the Single Parity check algorithm using Python.

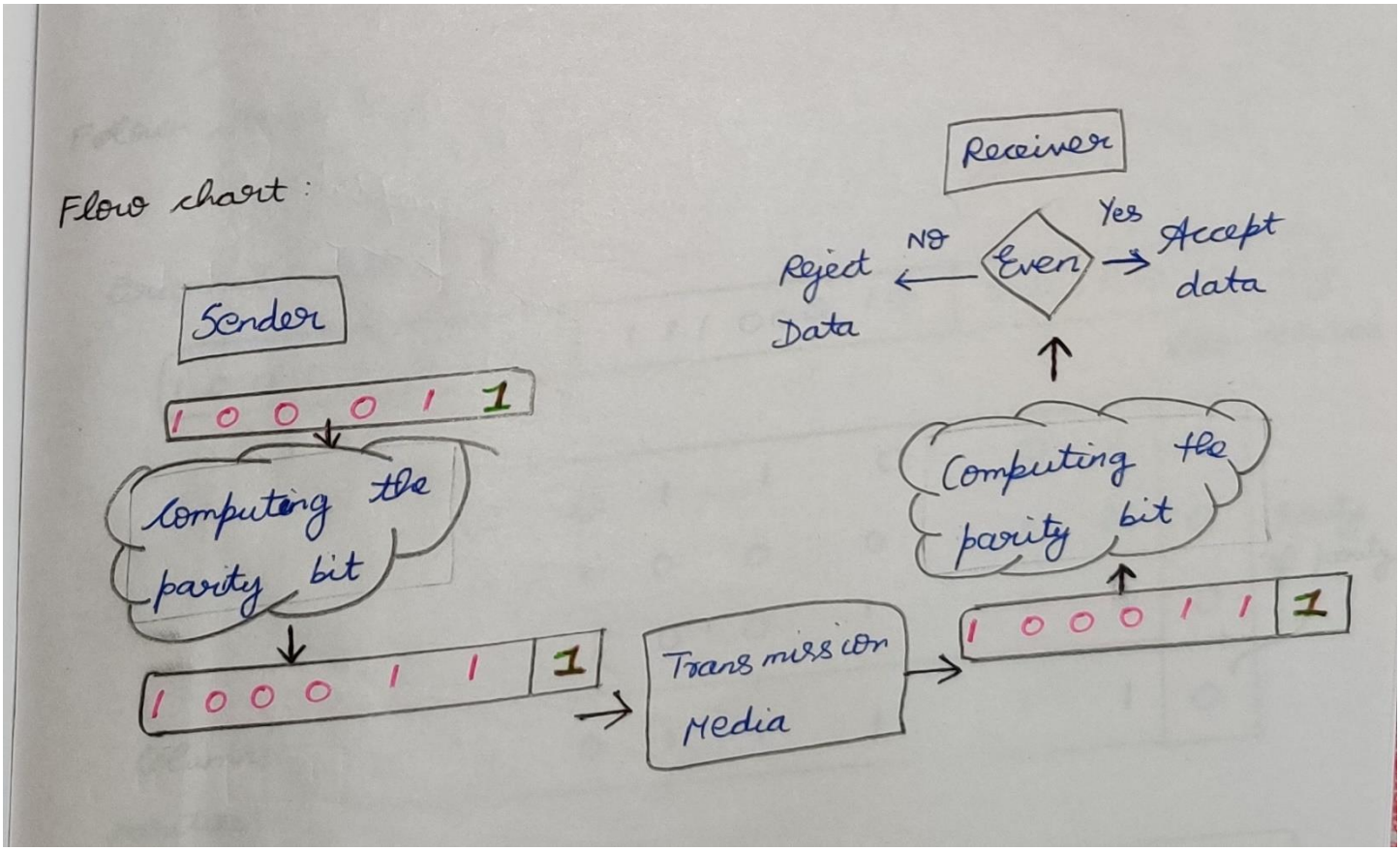
Analysis


Analysis:

*) Blocks of data from the source are subjected to a a check bit / parity bit generator form, where a parity of

- 1 is added to the block if it contains odd number of 1's
- 0 is added to the block if it contains even number of 1's

Flow-chart



Code Structure

```
 1   def Sender(list1,m):...
36
37   def Receiver(ans,str1):...
48
49   if __name__ == '__main__':...)
```

Actual code

```
def Sender(list1,m):
    cnt = 0
    for i in list1:
        if i==1:
            cnt+=1

    if (m=='odd_parity'):
        if (cnt%2!=0):
            print("Input data is odd_parity")

            list1.append(0)
            print("Parity bit 0 is added")
            print("Sender sends the data : ",*list1)
            return list1
        else:
            print("Input data is not in odd parity, so converting")
            list1.append(1)
            print("Parity bit 1 is added")
            print("Sender sends the data : ",*list1)
            return list1

    elif (m=='even_parity'):
        if (cnt%2==0):
            print("Input data is even_parity")

            list1.append(0)
            print("Parity bit 0 is added")
            print("Sender sends the data : ",*list1)
            return list1
        else:
            print("Input data is not in even parity, so converting")
            list1.append(1)
            print("Parity bit 1 is added")
            print("Sender sends the data : ",*list1)
            return list1

```

```
37  def Receiver(ans,str1):
38      cnt = 0
39      for i in ans:
40          if i==1:
41              cnt+=1
42      if (cnt%2==0 and str1=='even_parity'):
43          print("Receiver has received correctly")
44      elif (cnt%2!=0 and str1=='odd_parity'):
45          print("Receiver has received correctly")
46      else :
47          print("Receiver has not received correcly")
48
49  if __name__ == '__main__':
50          list1 = list(map(int,input("Enter the data : ").split()))
51          str1 = input("Enter the mode of parity : ")
52          ans = Sender(list1,str1)
53          print("Receiver receives the data : ",*ans)
54          Receiver(ans,str1)
```

Test-case-1

```
Enter the data : 1 0 0 0 1 1
Enter the mode of parity : odd_parity
Input data is odd_parity
Parity bit 0 is added
Sender sends the data :  1 0 0 0 1 1 0
Receiver receives the data :  1 0 0 0 1 1 0
Receiver has received correctly

***Repl Closed***
```

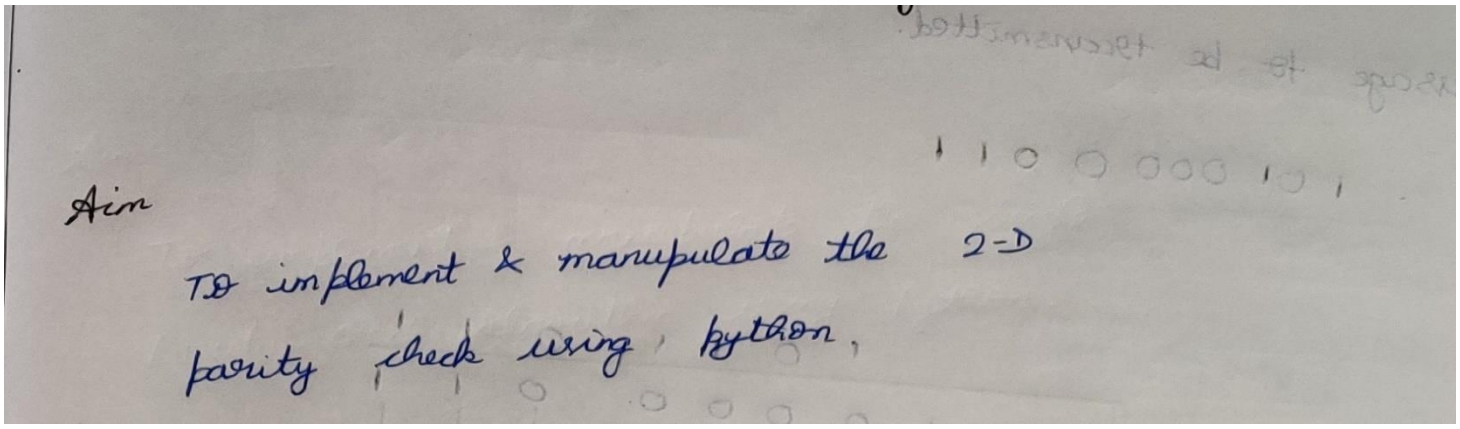
Test-case-2

```
Enter the data : 1 1 0 0 1
Enter the mode of parity : even_parity
Input data is not in even parity, so converting
Parity bit 1 is added
Sender sends the data :  1 1 0 0 1 1
Receiver receives the data :  1 1 0 0 1 1
Receiver has received correctly
```

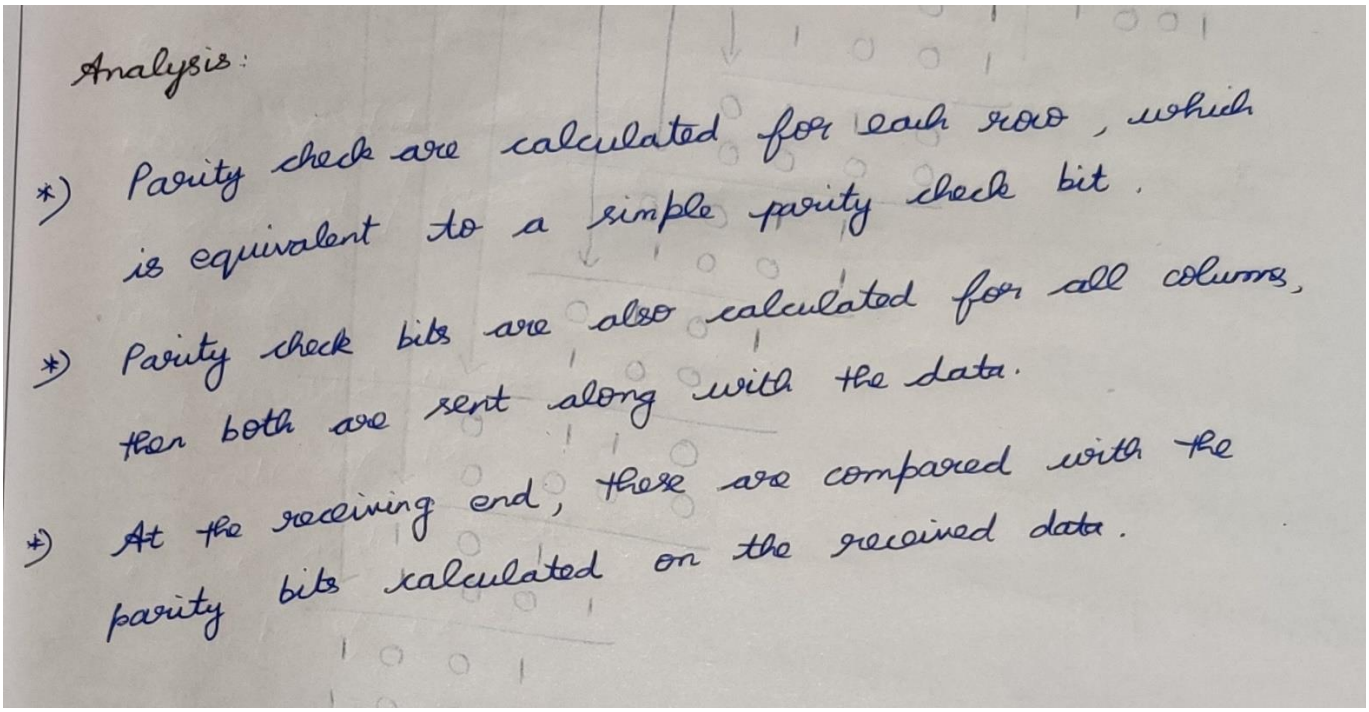
Output

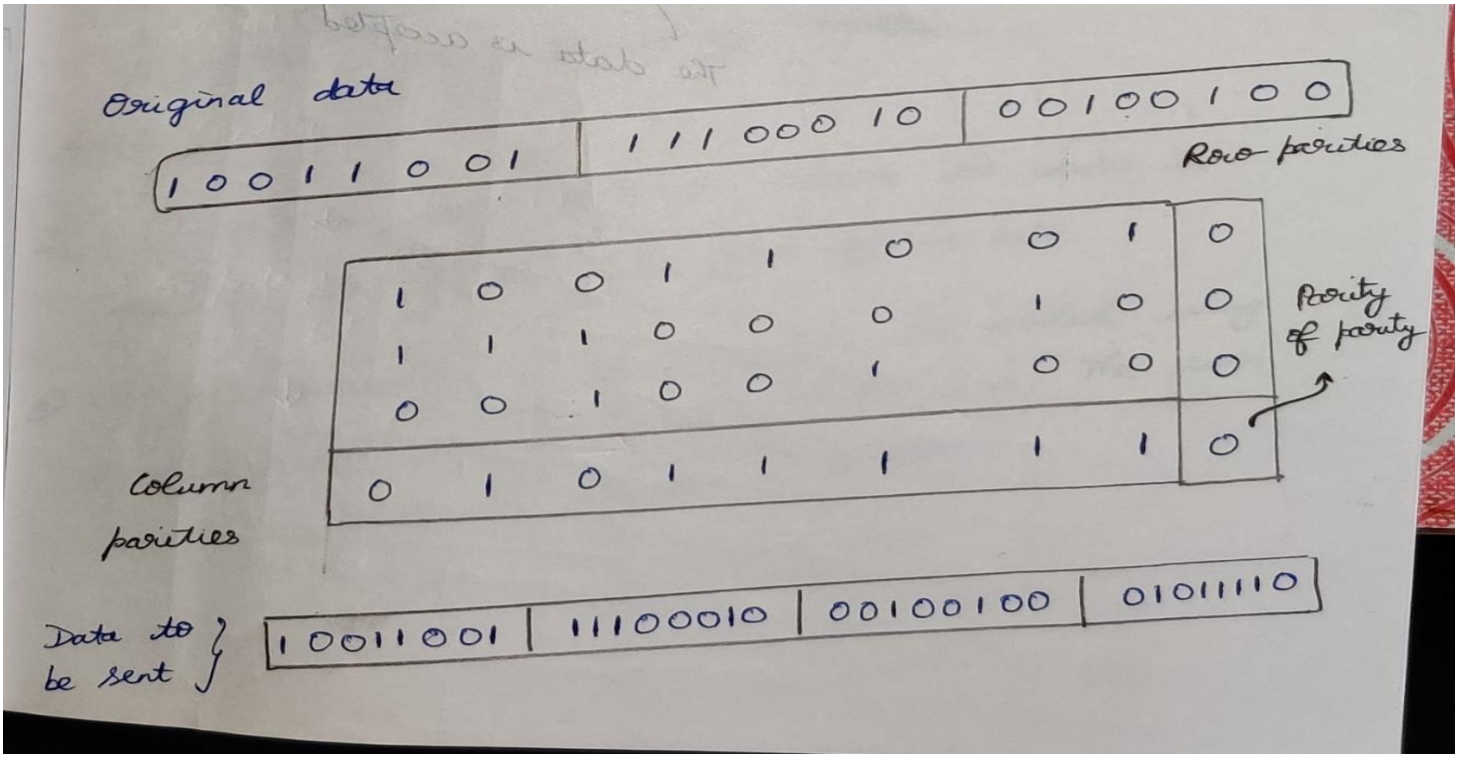
```
Enter the data : 1 0 0 0 1 1
Enter the mode of parity : even_parity
Input data is not in even parity, so converting
Parity bit 1 is added
Sender sends the data :  1 0 0 0 1 1 1
Receiver receives the data :  1 0 0 0 1 1 1
Receiver has received correctly

***Repl Closed***
```

2) Two-Dimensional Parity

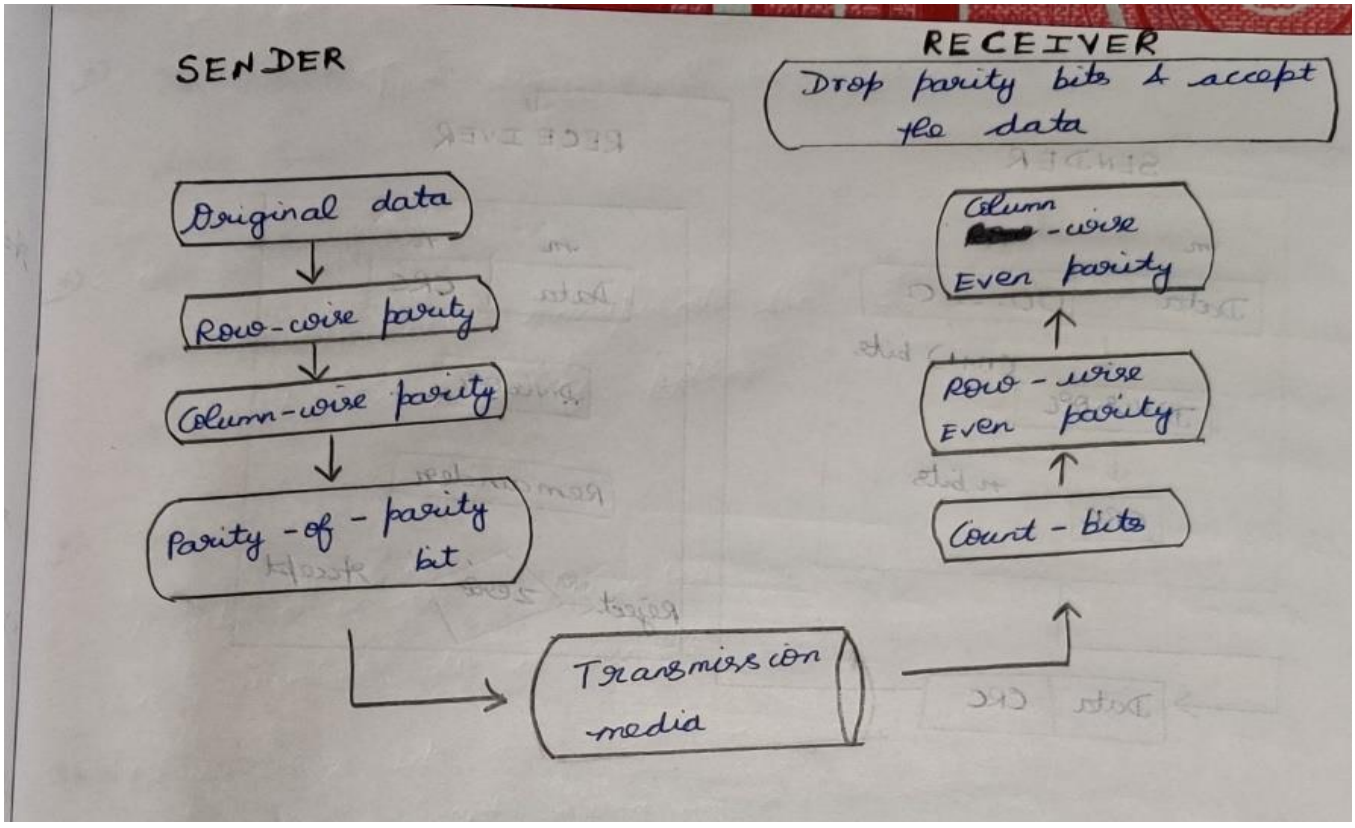


Analysis




Original data
10011001
11100010
00100100
Row parities
Parity of parity
Column parities
Data to be sent
10011001
11100010
00100100
01011110

Flowchart


SENDER
Original data
Row-wise parity
Column-wise parity
Parity-of-parity bit
Transmission media
RECEIVER
Drop parity bits & accept the data
Column -wise Even parity
Row-wise Even parity
Count-bits

Code-structure

```
  1  def get_even(list1):...
  7
  8  def get_odd(list1):...
 15
 16  def check_column(f_list):...
 28
 29  def to_string(listfin, m):...
 32
 33  if __name__ == '__main__':...)
```

Actual-Code

```
 1  def get_even(list1):
 2      if sum(list1) % 2 != 0:
 3          list1.append(1)
 4      else:
 5          list1.append(0)
 6      return list1
 7
 8  def get_odd(list1):
 9      if sum(list1) % 2 == 0:
10          list1.append(1)
11      else:
12          list1.append(0)
13      return list1
14
15  def check_column(f_list):
16      global nop, n
17      list_re = []
18      for i in range(nop + 1):
19          list_re.append(sum([x[i] for x in f_list]))
20      if(n == 2):
```

```
        for i in range(len(list_re)):
            list_re[i] = list_re[i] % 2
    else:
        for i in range(len(list_re)):
            list_re[i] = (list_re[i] + 1) % 2
    return list_re

def to_string(listfin, m):
    string2 = m.join([str(y) for y in listfin])
    return string2

if __name__ == '__main__':
    n = int(input("Enter the standard that sender and receiver are gonna follow :
    print("At the Sender End...")
    data = input("Enter the data to send in binary : ")
    nop = int(
        input("Enter the no of bits per packets for your data have to be encrypted
    extra = (nop - ((len(data)) % nop)) % nop
    extra_bits = str('0' * extra)
    data = extra_bits + data

    list_data = [[int(x) for x in data[i:i + nop]]
                 for i in range(0, len(data), nop)]
    if (n == 2):
        for i in list_data:
            get_even(i)
    else:
        for i in list_data:
            get_odd(i)

    list_data.append(check_column(list_data))
    for l in range(len(list_data)):
        list_data[l] = to_string(list_data[l], '')
    Sender_data = to_string(list_data, ' ')
    print("Code generated by the Sender is : ", Sender_data)
    print("At the receiver End : ")
    n = 0
```

```
    n = int(input("Enter the standard that receiver wants to follow : '1' for 'Odd_
    data = input("Enter the data to send in binary : ")
    nop = int(
        input("Enter the no of bits per packets which sender used to encrypt : "))
    list_data = [[int(x) for x in data[i:i + nop]]
                 for i in range(0, len(data), nop)]
    print("Checking....")
    if n == 1:
        result = [sum(i) % 2 == 1 for i in list_data]
    else:
        result = [sum(i) % 2 == 0 for i in list_data]
    if all(result):
        print("Data received without error ")
    else:
        print("Incorrect Data")
```

Test-case-1

```
Enter the standard that sender and receiver are gonna follow : '1' for '
    Odd_parity'  and '2' for 'Even_parity' : 2
At the Sender End...
Enter the data to send in binary : 10011001111000100010010 0
Enter the no of bits per packets for your data have to be encrypted : 8
Code generated by the Sender is :  100110010 111000100 001001000 010111110
At the receiver End :
Enter the standard that receiver wants to follow : '1' for 'Odd_parity'
    and '2' for 'Even_parity' : 2
Enter the data to send in binary : 10011001111000100010010 0
Enter the no of bits per packets which sender used to encrypt : 8
Checking....
Incorrect Data
```

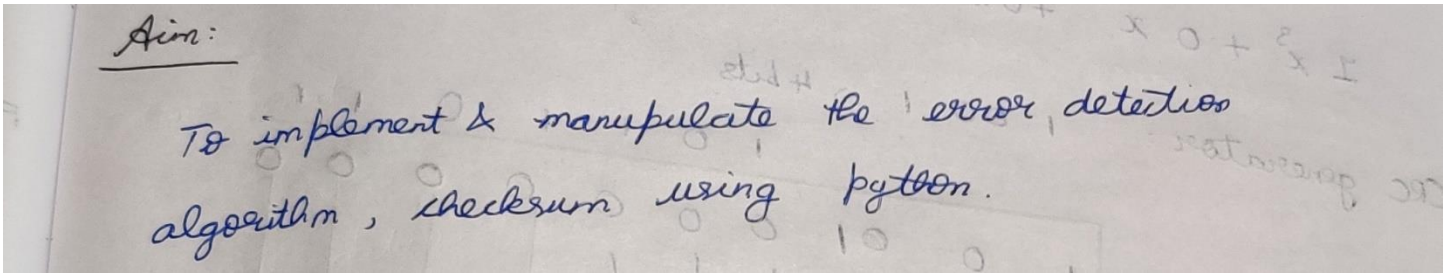
Test-case-2

```
Enter the standard that sender and receiver are gonna follow : '1' for '
    Odd_parity'  and '2' for 'Even_parity' : 1
At the Sender End...
Enter the data to send in binary : 110011001010
Enter the no of bits per packets for your data have to be encrypted : 4
Code generated by the Sender is :  11001 11001 10101 01010
At the receiver End :
Enter the standard that receiver wants to follow : '1' for 'Odd_parity'
    and '2' for 'Even_parity' : 1
Enter the data to send in binary : 11001 11001 10101 01010
Enter the no of bits per packets which sender used to encrypt : 4
```

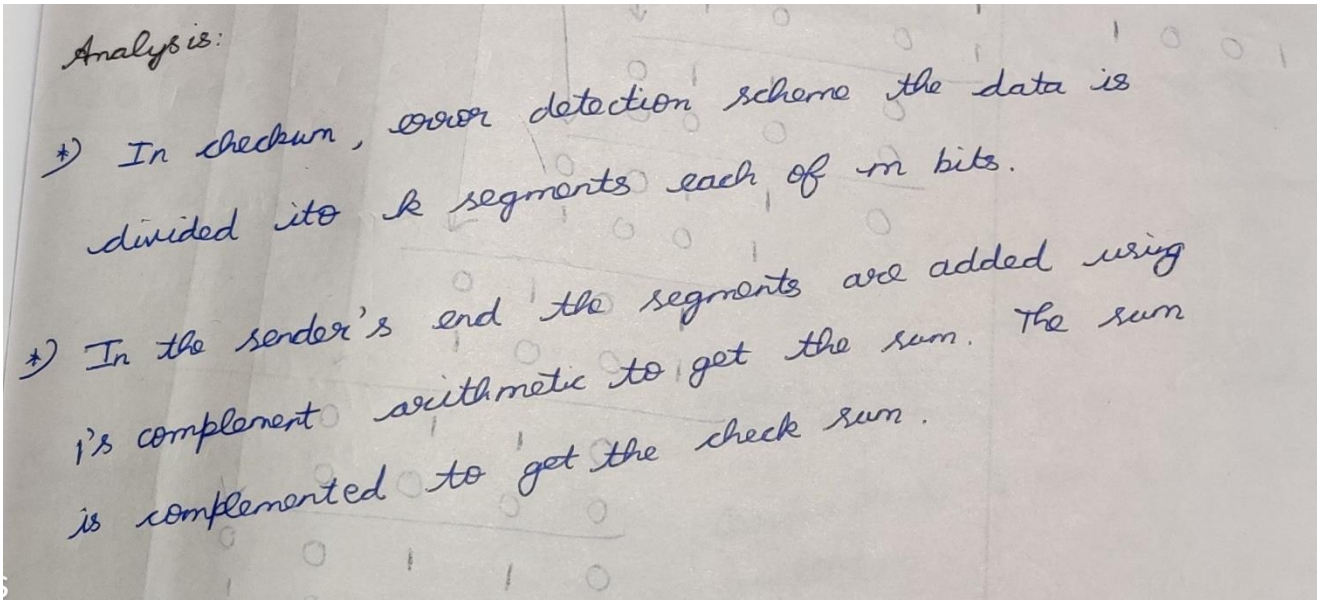
Output

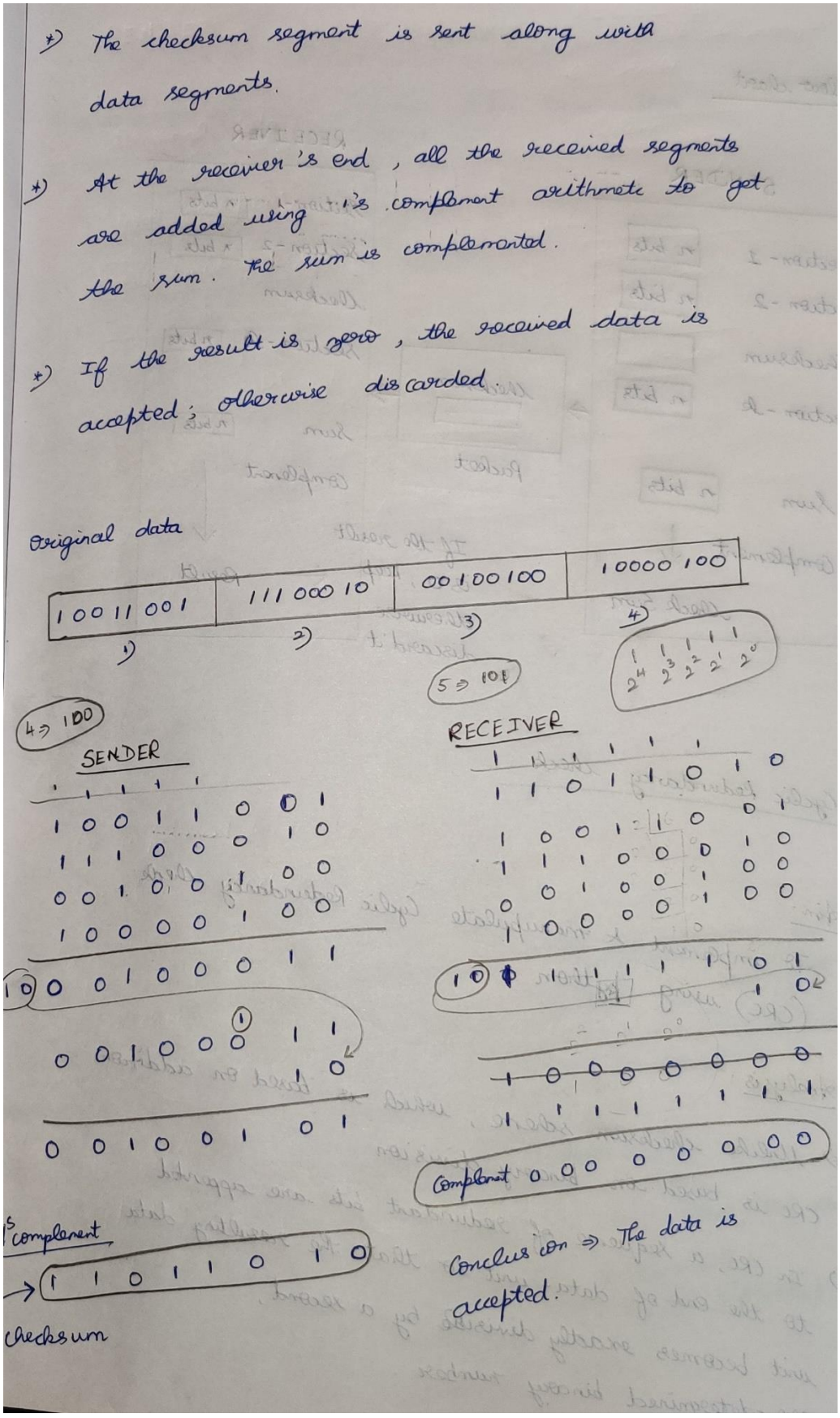
```
Enter the standard that sender and receiver are gonna follow : '1' for '
    Odd_parity'  and '2' for 'Even_parity' : 2
At the Sender End...
Enter the data to send in binary : 100110011110001000100100
Enter the no of bits per packets for your data have to be encrypted : 8
Code generated by the Sender is :  100110010 111000100 001001000 010111110
At the receiver End :
Enter the standard that receiver wants to follow : '1' for 'Odd_parity'
    and '2' for 'Even_parity' : 2
Enter the data to send in binary : 100110010111000100001001000010111110
Enter the no of bits per packets which sender used to encrypt : 8
Checking....
Data received without error
```

3)Checksum

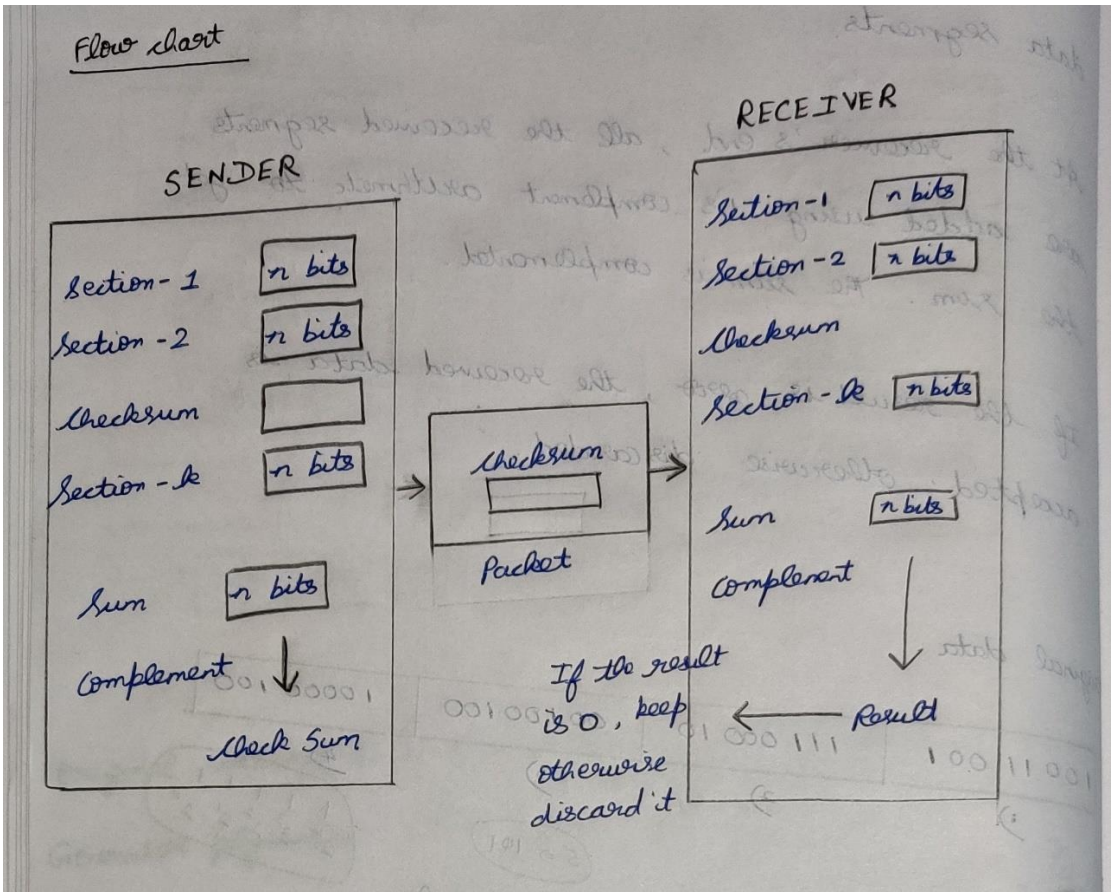


Analysis




*) The checksum segment is sent along with data segments.
*) At the receiver's end, all the received segments are added using 1's complement arithmetic to get the sum. The sum is complemented.
*) If the result is zero, the received data is accepted; otherwise discarded.
Original data
10011001
11100010
00100100
10000100
1)
2)
3)
4)
5 ⇒ 101
4 ⇒ 100
SENDER
RECEIVER
Complement
's complement
Checksum
Conclusion ⇒ The data is accepted.

Flowchart



Code-structure

```
 1  def check(temp):…
10
11  def concatenate_list_data(list):…
16
17  def complement(temp):…
25
26  def Binary_Addition(x, y): …
45
46  def Cyclic_addition(int1,int2):…
57
58  def Access(main_list,n):…
68
69  if __name__ == '__main__':…n
```

Actual-code

```
def check(temp):
    flag = 0
    for i in temp:
        if temp[i]==0:
            flag +=1
    if len(temp)==flag:
        print("Receiver has received the correct data")
    else:
        print("Reciever has not received the correct data")  ## Will check the fina

def concatenate_list_data(list):
    result= ''
    for element in list:
        result += str(element)
    return str(result)  ## Converting the list elements into a single integer

def complement(temp):
    list1 = []
    for i in range(len(temp)):
        if temp[i]==0:
            list1.append(1)
        else:
            list1.append(0)
    return list1  ## Getting the complement of the elements

def Binary_Addition(x, y):
    max_len = max(len(x), len(y))

    x = x.zfill(max_len)
    y = y.zfill(max_len)

    result = ''
    carry = 0

    for i in range(max_len - 1, -1, -1):
        temp = carry
        temp += 1 if x[i] == '1' else 0
        temp += 1 if y[i] == '1' else 0
```

```
        result = ('1' if temp % 2 == 1 else '0') + result
        carry = 0 if temp < 2 else 1     # Compute the carry.

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

    return result.zfill(max_len) ## Doing the binary addition

def Cyclic_addition(int1,int2):
    result = Binary_Addition(int1,int2)
    if len(result)>len(int1):
        while len(result)>len(int1):
            diff     = abs(len(int1)-len(result))
            extra  = result[0:diff]
            result = result[diff:len(result)]
            result = Binary_Addition(result,extra)
        return result
    else:
        return result  ## Taking care of the last carry


def Access(main_list,n):
    main_list_length = int(len(main_list)/n)
    list1 = [main_list[i:i+main_list_length] for i in range(0, len(main_list), main
    int1 = concatenate_list_data(list1[0])   ## 0th index
    int2 = concatenate_list_data(list1[1])  ## 1st index
    temp_ans = Cyclic_addition(int1,int2)
    for i in range(2,len(list1)):
        int1 = concatenate_list_data(list1[i])
        temp_ans = Cyclic_addition(temp_ans,int1)
    return(temp_ans)   ## Accessing the above function

if __name__ == '__main__':
    #main_list = list(map(int,input().split()))
    main_list = [1,0,0,1,1,0,0,1,1,1,1,0,0,0,1,0, 0,0,1,0,0,1,0,0 ,1, 0, 0, 0, 0,1,
    n = 4
    temp = Access(main_list,n)
```

```
74
75      last = []
76      for i in temp:
77          last.append(int(i))
78
79      last = complement(last)
80      print("checksum elements : ",*last)
81      for i in last:
82          main_list.append(i)
83      print("Sender data : ",*main_list)
84      ################################################
85      temp = Access(main_list,n+1)
86      print("At the receiver end : ",*main_list)
87      temp = complement(temp)
88      check(temp)
```

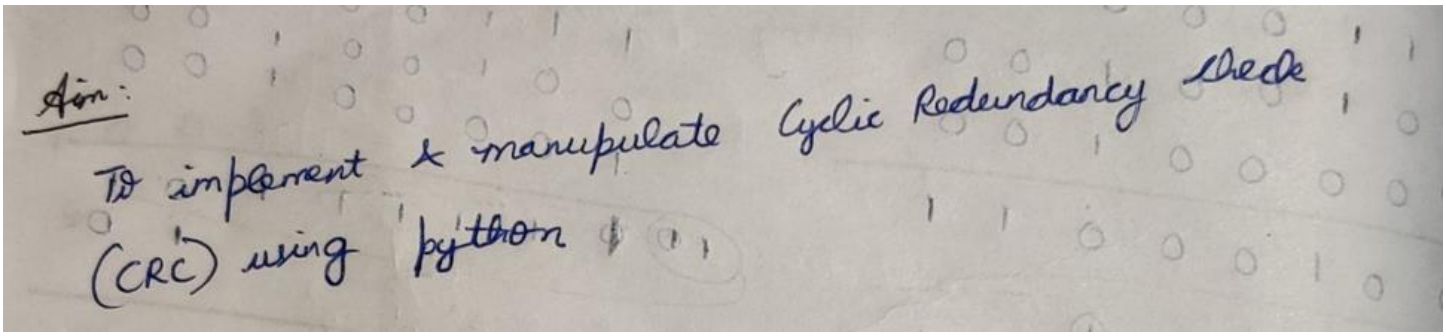
Test-case-1

```
checksum elements :  1 1 0 1 1 0 1 0
Sender data :  1 0 0 1 1 0 0 1 1 1 1 0 0 0 1 0 0 0 1 0 0 1 0 0 1 0 0 0 0 1 0 0 1 1 0 1
    1 0 1 0
At the receiver end :  1 0 0 1 1 0 0 1 1 1 1 0 0 0 1 0 0 0 1 0 0 1 0 0 1 0 0 0 0 1 0 0
    1 1 0 1 1 0 1 0
Receiver has received the correct data
```

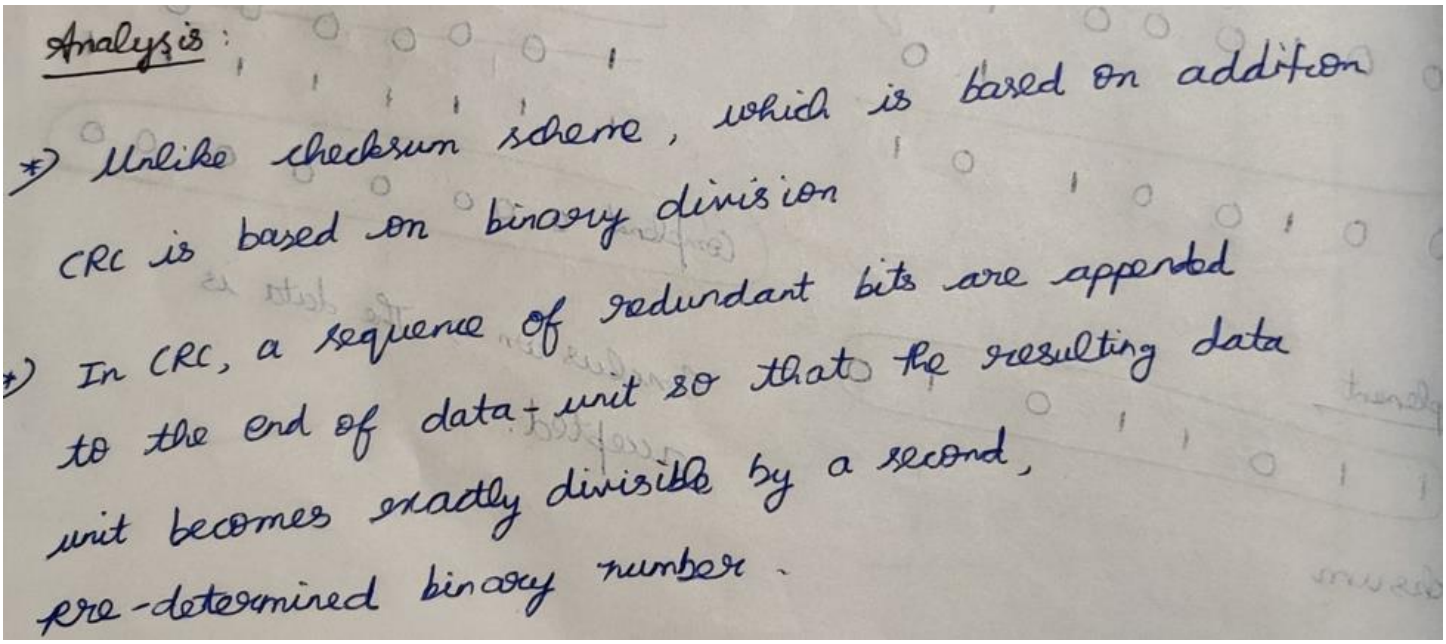
Output

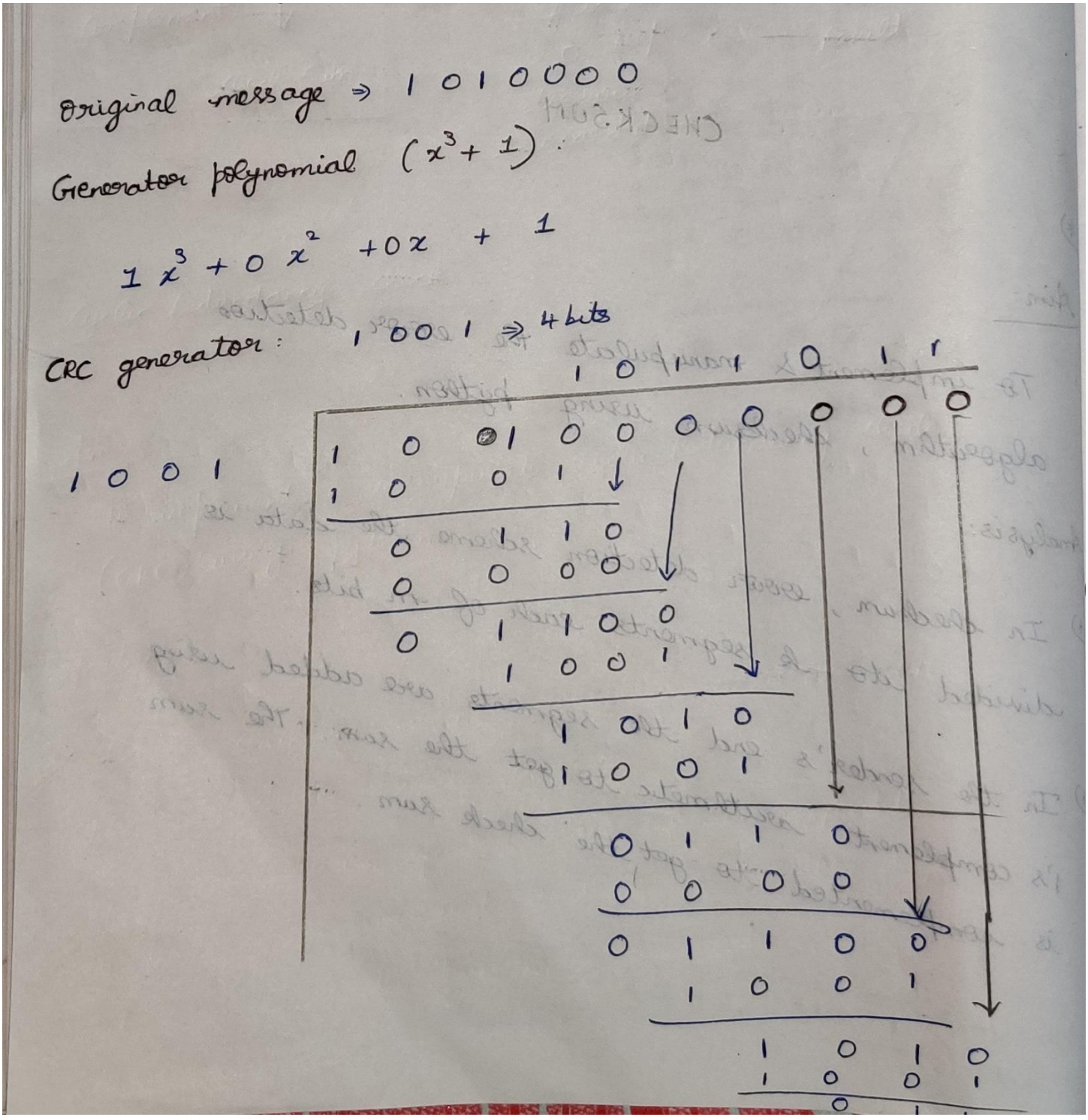
```
checksum elements :  1 1 0 1 1 0 1 0
Sender data :  1 0 0 1 1 0 0 1 1 1 1 0 0 0 1 0 0 0 1 0 0 1 0 0 1 0
     0 0 0 1 0 0 1 1 0 1 1 0 1 0
At the receiver end :  1 0 0 1 1 0 0 1 1 1 1 0 0 0 1 0 0 0 1 0 0 1
     0 0 1 0 0 0 0 1 0 0 1 1 0 1 1 0 1 0
Receiver has received the correct data
```

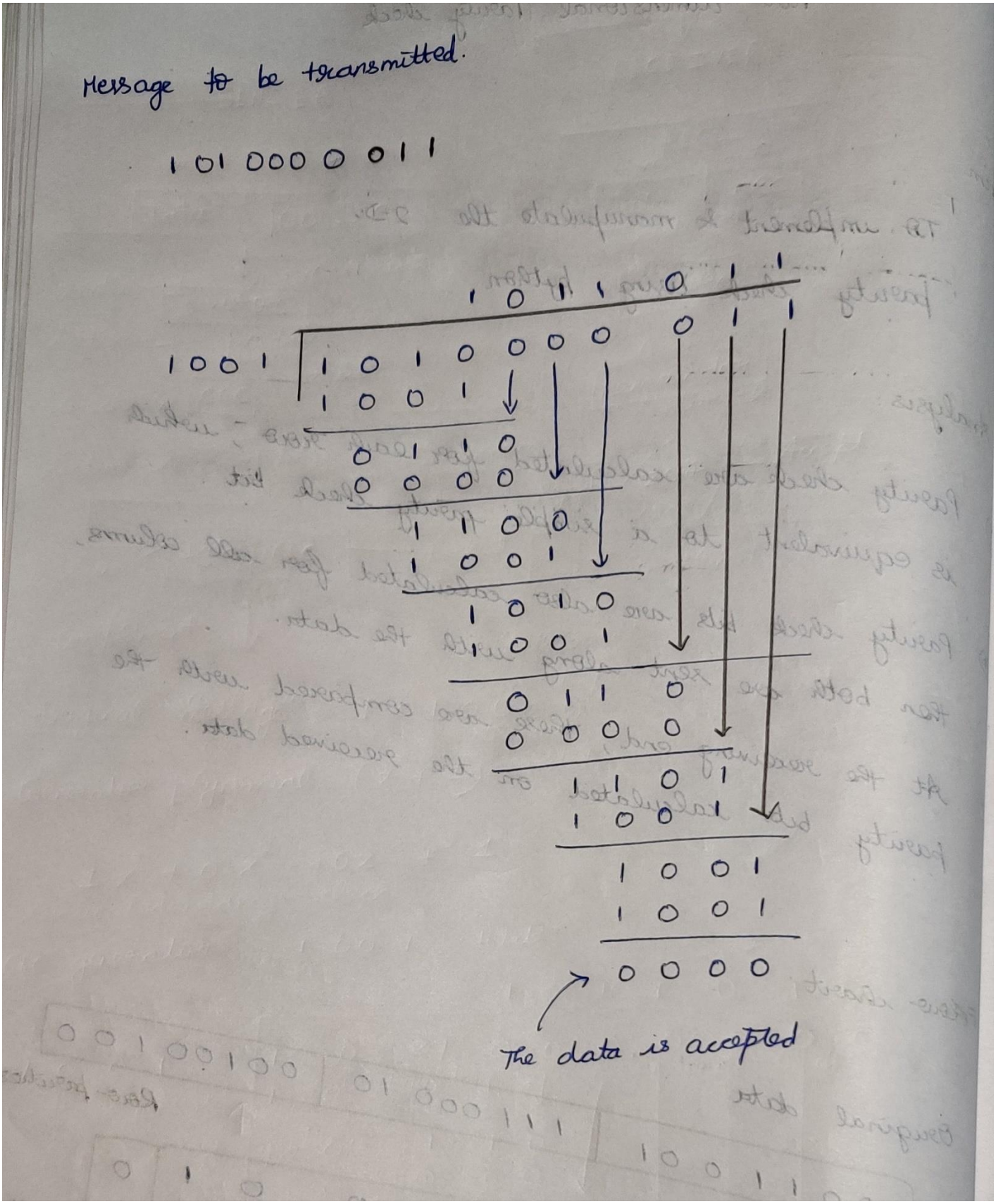
4)Cyclic Redundancy Check


Aim:
To implement & manipulate Cyclic Redundancy check (CRC) using python

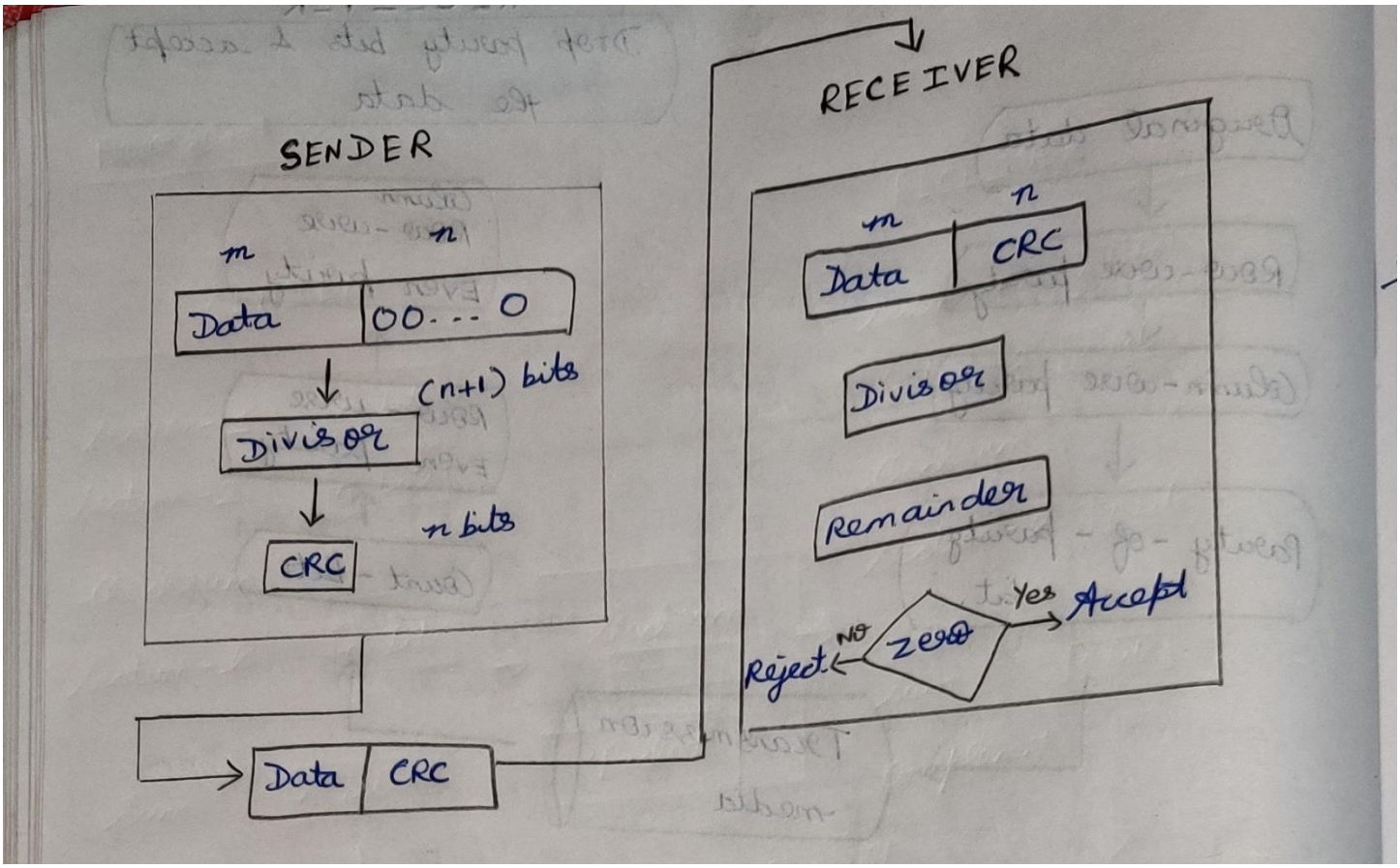
Analysis


Analysis:
* Unlike checksum scheme, which is based on addition CRC is based on binary division
* In CRC, a sequence of redundant bits are appended to the end of data unit so that the resulting data unit becomes exactly divisible by a second, pre-determined binary number.


Original message ⇒ 1 0 1 0 0 0 0
Generator polynomial $(x^3 + 1)$:
$1x^3 + 0x^2 + 0x + 1$
CRC generator: 1 0 0 1 ⇒ 4 bits
1 0 0 1


Message to be transmitted.
1 0 1 0 0 0 0 1 1
1 0 0 1
The data is accepted

Flow-chart



Code-structure

```
 1  def Add_bits(divident,divisor):...
 8
 9  def xor(num1,num2):...
19
20  def Modulo2Division(divident,divisor):...
40
41  def Receiver_check(sender_data,input_divisor):...
50
51  # input
52  # 1) data (divident)
53  # 2) divisor
54  if __name__ == '__main__':...
```

Code

```
def Add_bits(divident,divisor):
    len_divident = len(divident)
    len_divisor   = len(divisor)

    for i in range(len_divisor,(len_divident)+1):
        divident = divident + '0'
    return divident

def xor(num1,num2):
    result = []

    for i in range(1, len(num2)):  ## should go from 1, since
        if num1[i] == num2[i]:
            result.append('0')
        else:
            result.append('1')

    return ''.join(result)

def Modulo2Division(divident,divisor):

    count = len(divisor)
    xor_answer = divident[0:count]  ## Initially slicing out the divident

    while count<len(divident): ## xor division must take place till the en

        if xor_answer[0] == '1':  ## left-most bit is 1
            xor_answer = xor(divisor, xor_answer) + divident[count]  ## st
        elif xor_answer[0]  == '0':  ## Right-most bit is 0
            xor_answer = xor('0'*count, xor_answer) + divident[count]  ##
        count+=1
```

```
        count+=1

    ## Final division
    if xor_answer [0] == '1':
        xor_answer = xor(divisor, xor_answer)
    else:
        xor_answer = xor('0'*count, xor_answer)

    return xor_answer

def Receiver_check(sender_data,input_divisor):
    temp_str = ""
    for i in range(len(input_divisor)-1):
        temp_str = temp_str + '0'
    print("Receiver data : ",sender_data)
    if (Modulo2Division(sender_data,input_divisor)) == (temp_str):
        print("Receiver has received the correct data")
    else:
        print("Receiver has not received the correct data")

# input
# 1) data (divident)
# 2) divisor
if __name__ == '__main__':
    input_divident = input("Enter the data (divident) : ")
    input_divisor  = input("Enter the data (divisor)  : ")
    print("Sender data : ",input_divident)
    extra_bits_divident = Add_bits(input_divident,input_divisor)
    crc = Modulo2Division(extra_bits_divident,input_divisor)

    sender_data = input_divident  + crc

    print("CRC : ",crc)
    print("Sender data along with CRC : ",sender_data)

    Receiver_check(sender_data,input_divisor)
```

Test-case-1

```
Enter the data (divident) : 1010000
Enter the data (divisor)  : 1101
Sender data :  1010000
CRC :  101
Sender data along with CRC :  1010000101
Receiver data :  1010000101
Receiver has not received the correct data
```

Test-case-2

```
Enter the data (divident) : 11000
Enter the data (divisor)  : 1101
Sender data :  11000
CRC :  100
Sender data along with CRC :  11000100
Receiver data :  11000100
Receiver has not received the correct data

***Repl Closed***
```

Result

```
Enter the data (divident) : 100100
Enter the data (divisor)  : 1101
Sender data :  100100
CRC :  001
Sender data along with CRC :  100100001
Receiver data :  100100001
Receiver has received the correct data
```

Sir, the above program code are also available in my github,
https://github.com/PrashanthSingaravelan/WinterSemester-2021/tree/main/CSI2007%20Data%20Communication%20and%20Network/Lab%20Assignment/Assignment-2