ERROR DETECTTION MECHANISM

ECE 562 - DATA COMMUNICATION NETWORKS

Department of Electrical and Computer Engineering, CSUN

ASSIGNED BY PROF. ALI ZAHID

by

CHINMAY SHUKLA

PRATHAMESH TELI

I. ABSTRACT

When an information message sent by a Sender is not the same as the information message received by the receiver, an error has occurred in the information message transmitted. During transmission, information message can suffer from noise which can introduce errors in the information message to be transmitted. In other words, a 0 can become 1 or a 1 can become a 0. It is important to detect these errors. Amongst a few methods to detect errors, there is Cyclic Redundancy Check (CRC).

This method of Error detection is effective. Initially the Information message (say k-bit sequence) is appended by (n-1) zeroes at the end of it where n is the generator pattern. When this message (along with appended zeroes) is divided by the Generator pattern, we get an n-1 bit remainder. These are nothing but the FCS bits. These FCS bits are appended to the original information message and that forms a Transmitted Sequence. The next step is to take User input in order to enter a Receiver Sequence. After entering this sequence, it is divided by the Generator pattern to get the remainder either zero or non-zero. The next step is to induce an Error Sequence. The Transmitted sequence performs an XOR operation with the Error Sequence which results in a Received Sequence. If this Received sequence is same as the Transmitted sequence, then no errors occur, however, even if one bit in the Received sequence is different from that of the Transmitted sequence, then error has occurred which in this case is 1. The Received sequence is then divided by the Generator Pattern, the remainder of this division may be zero or may not be zero. If the remainder is 0, the Received sequence is accepted by the Receiver irrespective of the fact whether it contains errors or not, however, if the remainder is anything but a 0, the Received sequence won’t be accepted by the Receiver as the receiver will be able to detect that the Received sequence contains error. However, the receiver won’t be able to tell the number of errors occurred.

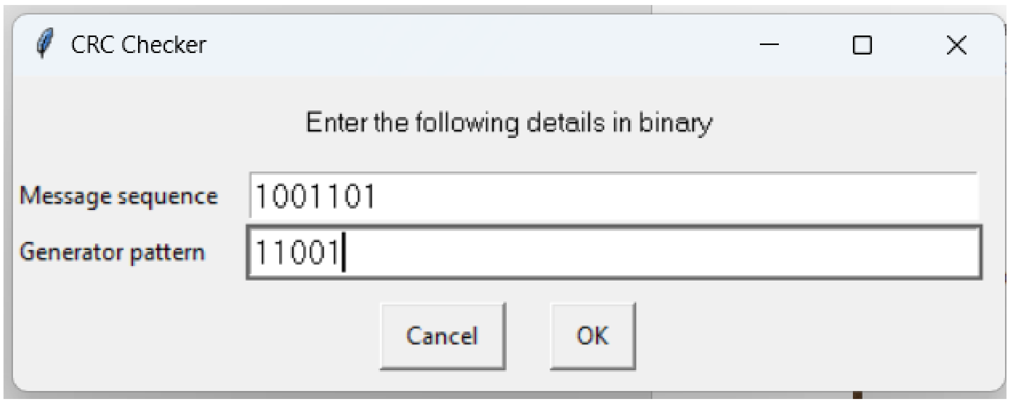
II. COMPLETE CODE :

```
from easygui import *
import re
def xor(a,b): # XOR Operation
  result=[]
  for i,j in zip(a,b):
    if i== j:
      result.append("0")
    else:
      result.append("1")
  return ''.join(result)
def ModuloTwoDiv(dividend, divisor):
  bit_org = len(divisor)
  bit = len(divisor)
  portion = dividend[0:bit]
  while bit<=len(dividend):
    if portion[0]=='0':
      try:
        portion = xor('0'*bit_org,portion)+dividend[bit]
        portion = portion[1:]
      except:
        break
    else:
      if bit<len(dividend):
        portion= xor(divisor,portion)+dividend[bit]
      else:
        portion= xor(divisor,portion)
      portion = portion[1:]
    bit+=1
  if portion[0]=="0":
    return portion[1:]
  else:
    return portion

# codeword = generator pattern
# rx = fcs bits

def Generator(message,codeword): # generatorCode
  addend_len = len(codeword)-1
  dividend = message + '0'*addend_len
  rx = ModuloTwoDiv(dividend,codeword) # 1st division
  op["FCS"] = rx
  return message+rx

# packet = received seq

def Receiver(packet,codeword):
  d = re.compile('[^0]')
  rx = ModuloTwoDiv(packet,codeword)
  op["Remainder"]=rx
  if len(d.findall(rx))==0:
    stat = "Message is error free"
    return stat
  stat = "Message contains Error, receiver is able to detect the error"
  return stat

# error mask = error sequence
# codeword = gen pattern

 # error sequence
def Alter(Err_Sequence,Transmitted_Sequence,Codeword):
  if len(Err_Sequence) != len(Transmitted_Sequence):
    message = "Error_Sequence and Transmitted_Seq are unequal in length"
    return message
  d = re.compile('[^0]')
  if len(d.findall(Err_Sequence))==0:
    message = "Message is error free"
    Received_Seq = xor(Err_Sequence,Transmitted_Sequence)
    op["Received Sequence"]=Received_Seq
    message+= "\nNumber of Error bits: 0"
    addend_len = len(Codeword)-1
    op["Remainder"]= addend_len*"0"
    return message
  Received_Seq = xor(Err_Sequence,Transmitted_Sequence)
  if Received_Seq == Transmitted_Sequence:
    message = "Message contains error, receiver is not able to detect error!"
    op["Received Sequence"]=Received_Seq
    addend_len = len(Codeword)-1
    op["Remainder"]= addend_len*"0"
    message+= "\nNumber of Error bits: 0"
    return message



  if Err_Sequence == Transmitted_Sequence:
    message = "Message contains error, receiver is not able to detect error!"
    op["Received Sequence"]=Received_Seq
    addend_len = len(Codeword)-1
    op["Remainder"]= addend_len*"0"
    find = Compare(Received_Seq,Transmitted_Sequence)
    message+= "\nNumber of Error bits: "+str(find)
    return message



  Received_Seq = xor(Err_Sequence,Transmitted_Sequence)
  op["Received Sequence"]=Received_Seq
  #line = "Received Sequence is: "+Err_Packet
  #message = line
  stat = Receiver(Received_Seq,Codeword)
  found = Compare(Transmitted_Sequence,Err_Sequence)
  if found != 0 and stat == "Message is error free":
    message = "Message contains error, receiver is not able to detect the error, it
accepts the transmitted sequence"
    message += "\nNumber of Error bits:"+str(found)

  else:
    message = stat
    message += "\nNumber of Error bits:"+str(found)
    #printOutput(message)
  return message


def Compare(Packet,Error_Sequence):
    found = 0
    for i in range(len(Packet)):
        if Packet[i] != Error_Sequence[i]:
            found += 1
    return found


def checkBinary(sampleInput):
  c = re.compile('[^01]')
  if(len(c.findall(sampleInput))):
      return False
  else:
      return True


def inputDetails():
    text = "Enter the following details in binary"
    # window title
    title = "CRC Checker"
    # list of multiple inputs
    input_list = ["Message sequence", "Generator pattern"]
    # creating a integer box
    output = multenterbox(text, title, input_list)
    return output


def errorInput():
    # message to be displayed
    text = "Enter Error Sequence"
    # window title
    title = "CRC Checker"
    # creating a enter box
    output = enterbox(text, title)
    return output


def printOutput(message):
    title = "CRC Checker"
    # creating a message box
    msg = msgbox(message, title)


def yesorno():
  message= "Do you want enter the error sequence?"
  title = "CRC Checker"
  output=ynbox(message,title)
  return output

op={}
output= inputDetails()
messagebit=output[0]
op["messagebit"]=messagebit
generatorCode=output[1]
op["generatorCode"]= generatorCode
if len(messagebit) < len(generatorCode):
  message = "Generator pattern is larger than the message sequence"
  title = "CRC Checker"
  msg = msgbox(message, title)

while not(checkBinary(messagebit)) or not(checkBinary(generatorCode)) or
len(messagebit) < len(generatorCode):
  output=inputDetails()
  messagebit=output[0]
  generatorCode=output[1]
```

```
# Packet - Transmitted sequence
# generatorCode -  gen pattern

Packet = Generator(messagebit,generatorCode)
op["Packet"] = Packet
# title for the message box
title = "CRC Checker"
# creating a message
message = "Transmitted Sequence: " + Packet + "\nFCS bits: "+ str(op["FCS"])
# creating a message box
msg = msgbox(message, title)

text = "Enter Received Sequence"
    # window title
title = "CRC Checker"
    # creating a enter box
ReceiverSequence = enterbox(text, title)
while len(ReceiverSequence) != len(Packet):
    ReceiverSequence = enterbox(text, title)

if ReceiverSequence!=Packet:
    find=Compare(Packet,ReceiverSequence)
    d = re.compile('[^0]')
    rx = ModuloTwoDiv(ReceiverSequence,generatorCode)
    op["Remainder"]=rx
    if len(d.findall(rx))==0:
        stat = "Message contains Error, receiver is unable to detect the error"
    else:
        stat = "Message contains Error, receiver is able to detect the error"
    message = "Received Sequence: "+ReceiverSequence+"\nRemainder:
"+op["Remainder"]+"\nNumber of Error bits: "+str(find)+"\n"+stat
    msg = msgbox(message, title)
else:
    message = "Received Sequence: "+ReceiverSequence+"\nMessage is Error
Free\nRemainder: "+op["Remainder"]+"\nNumber of Error bits: 0"
    msg = msgbox(message, title)

response=yesorno()
while response :
  errorMessage = errorInput()
  while not(checkBinary(errorMessage)):
    errorMessage = errorInput()
  outputStat = Alter(errorMessage,Packet,generatorCode)
  message = "Error Sequence: "+errorMessage
  message += "\n"+"Received Sequence: "+op["Received Sequence"]
  message += "\nRemainder: "+op["Remainder"]
  message+="\n"+outputStat
  printOutput(message)
  response=yesorno()
printOutput("Thank you! Goodbye")
```

III. RESULTS:

Part 1. Entering Information message an Generator sequence.

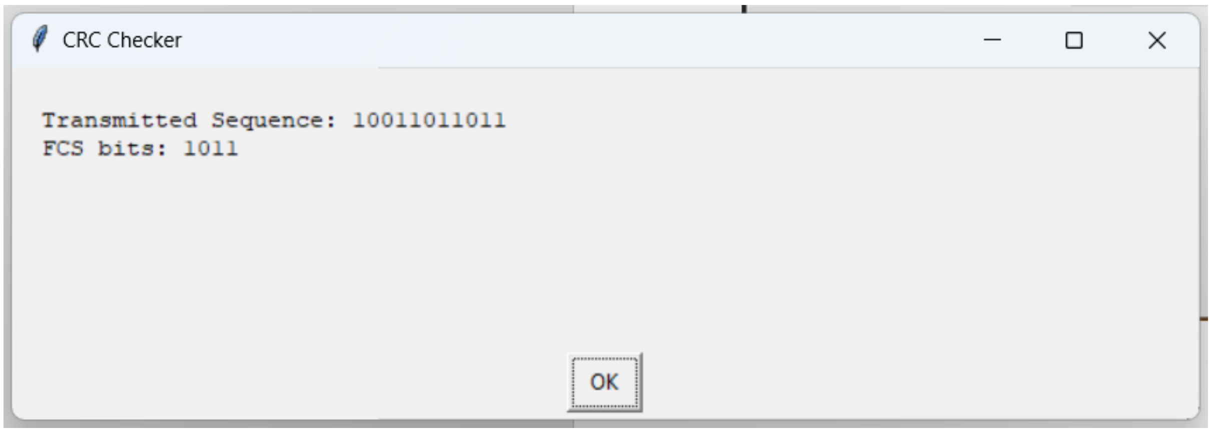


Enter the following details in binary

Message sequence 1001101

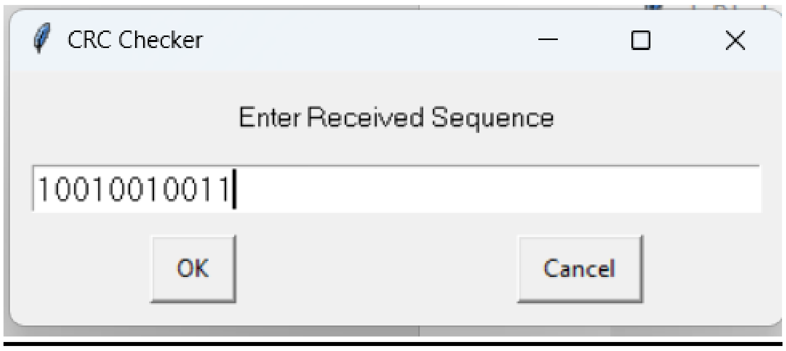
Generator pattern 11001

Part 2. Output of first part



```
Transmitted Sequence: 10011011011
FCS bits: 1011
```

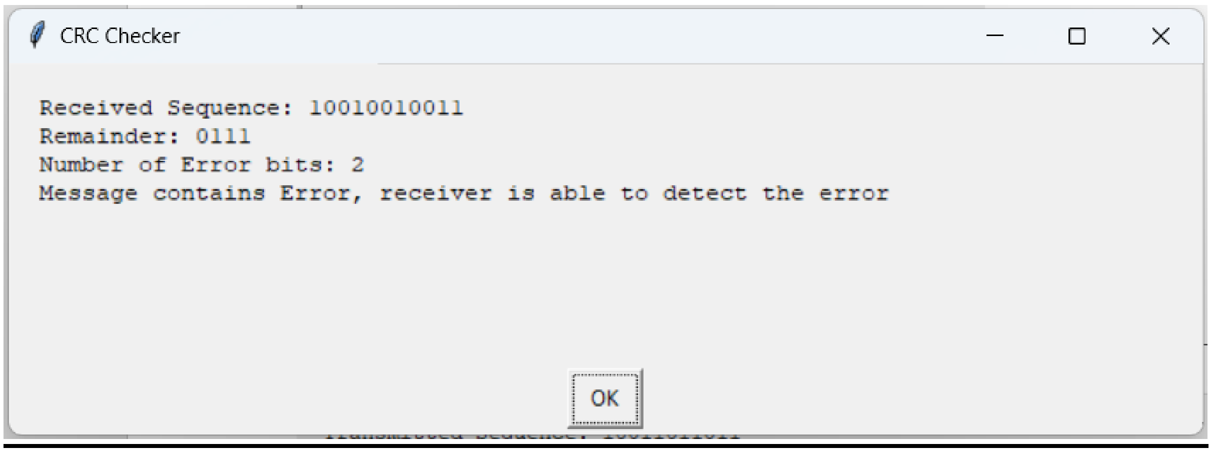
Part 3. Requesting User Input to enter a Received Sequence



Enter Received Sequence

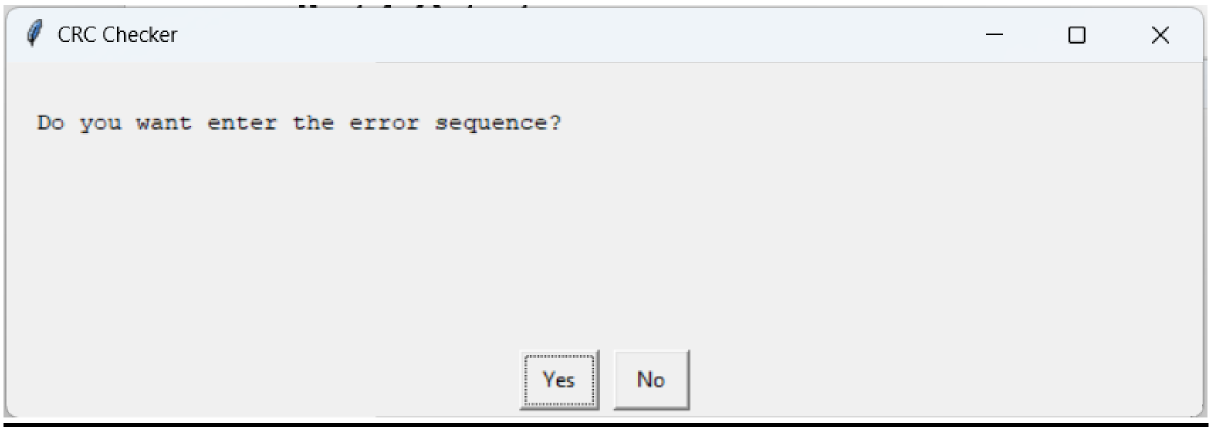
10010010011

Part 4. Output of second part



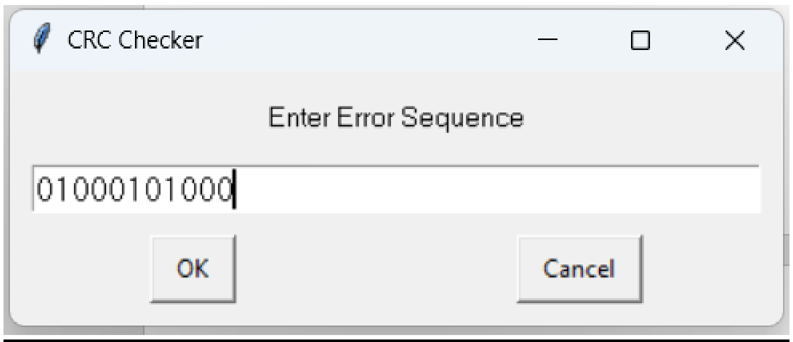
```
Received Sequence: 10010010011
Remainder: 0111
Number of Error bits: 2
Message contains Error, receiver is able to detect the error
```

Part 5. Requesting User to enter Error Sequence



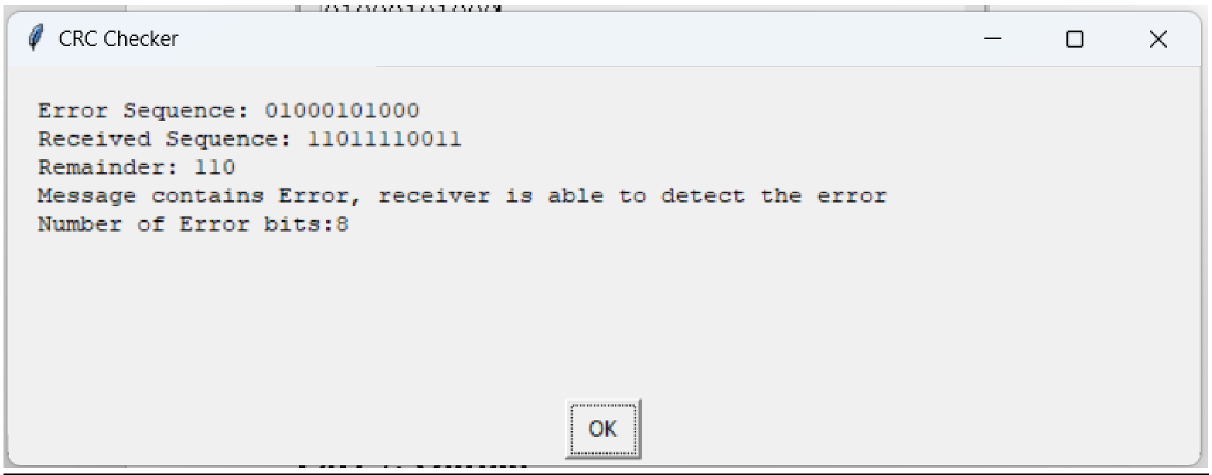
Do you want enter the error sequence?

Part 6. Error Sequence entered by the User



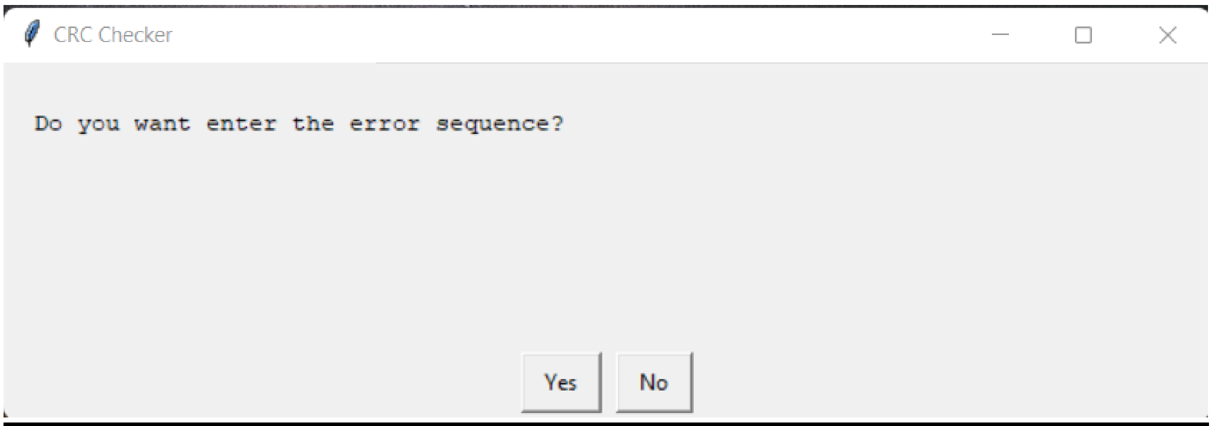
```
01000101000
```

Part 7. Output of third part



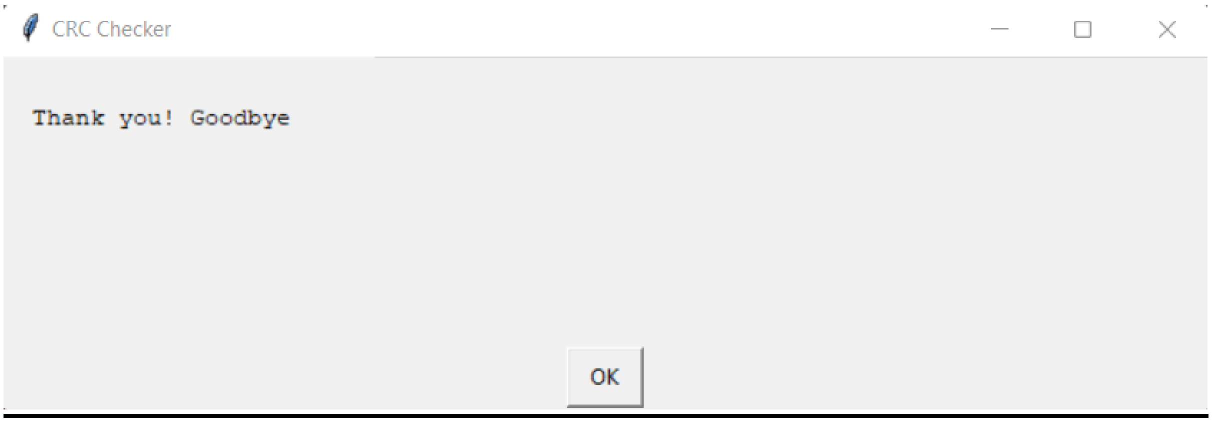
```
Error Sequence: 01000101000
Received Sequence: 11011110011
Remainder: 110
Message contains Error, receiver is able to detect the error
Number of Error bits:8
```

Part 8. To enter another Error Sequence



```
Do you want enter the error sequence?
```

<u>Part 9. If User does chooses “No”</u>



```
Thank you! Goodbye
```

IV. CONCLUSION

This Error Detection Mechanism is capable of detecting all types of data errors. For example, single bit errors, multi-bit errors. The implementation of this mechanism is simple in binary hardware and is easy to analyze mathematically.