K. J. Somaiya College of Engineering, Mumbai-77

(Autonomous College Affiliated to University of Mumbai)

Batch: B1 **Roll No.: 1711072**

Experiment / assignment / tutorial No. 2

Grade: AA / AB / BB / BC / CC / CD /DD

Signature of the Staff In-charge with date

Experiment No.: 2

TITLE: Implementation of 11 bit Hamming Code for Computer Networks

AIM: To implement Layer 2 Error Control schemes: Hamming Codes.

Expected Outcome of Experiment:
CO: Describe Data Link Layer, MAC layer technologies &amp; protocols and implement the functionalities like error control, flow control.

Books/ Journals/ Websites referred:

3. A. S. Tanenbaum, "Computer Networks", Pearson Education, Fourth Edition
4. B. A. Forouzan, "Data Communications and Networking", TMH, Fourth Edition
5.

Pre Lab/ Prior Concepts:
Data Link Layer, Error Correction/Detection, Types of Errors

New Concepts to be learned: Hamming Code.

Stepwise-Procedure:

Consider data stream of 7 bytes. Number it from 1 from left hand side of the data stream.

Add parity bits like P_0 , P_1, and P2 etc. into the positions 2's power like at 1,2,4,8.

1 0 1 P_4 1 1 0 P_3 1 P_2 P_1

D11 D10 D9 D7 D6 D5 D3

P_1 = D3 EXOR D5 EXOR D7 EXOR D9 EXOR D11 = 1

P_2 = D3 EXOR D6 EXOR D7 EXOR D10 EXOR D11= 1

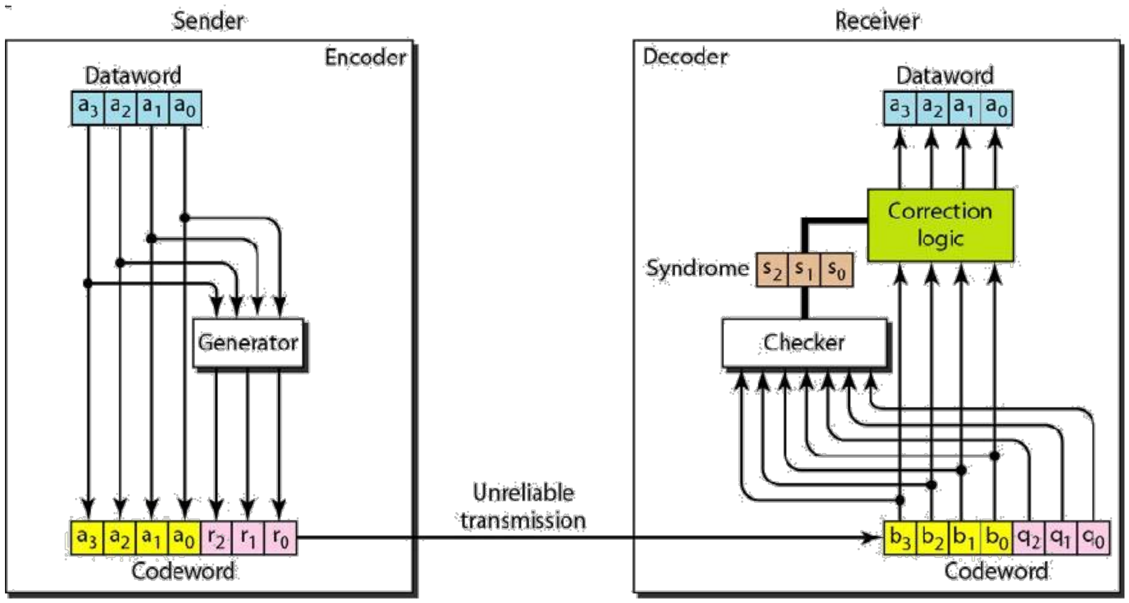
P_3 = D5 EXOR D6 EXOR D7 = 1

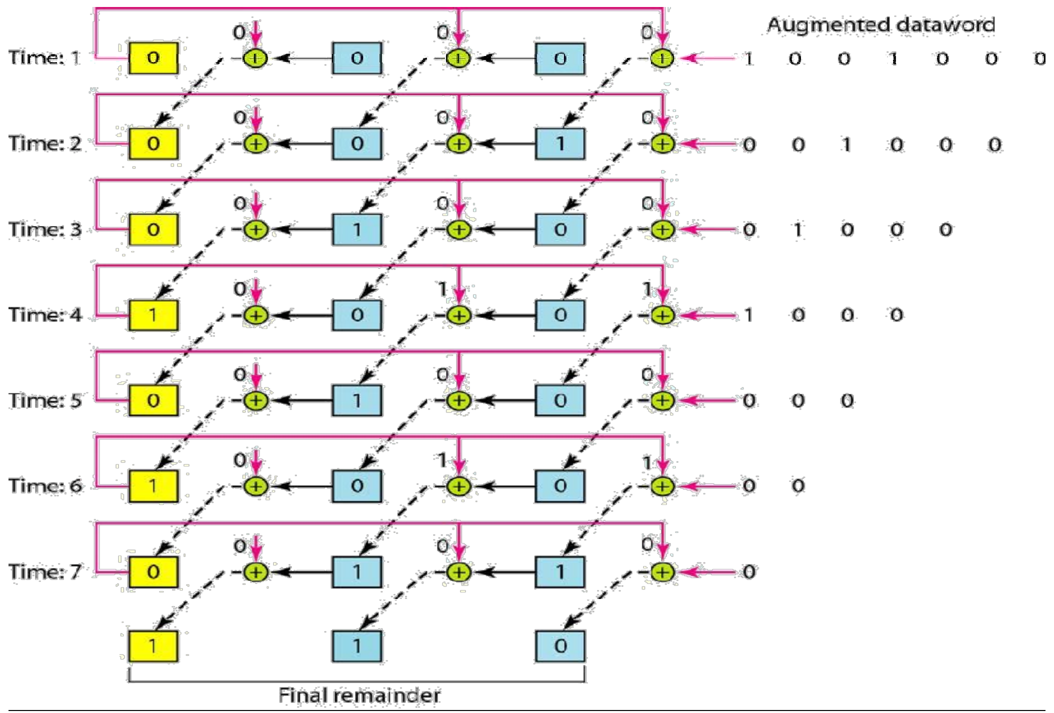
P_4 = D9 EXOR D10 EXOR D11 = 1

10111101111

This is the hamming code which is to be transferred to the destination. Again the same mechanism is implemented at the receiver end.

If we get the values 0000 then no error, if other than 0000 then error.



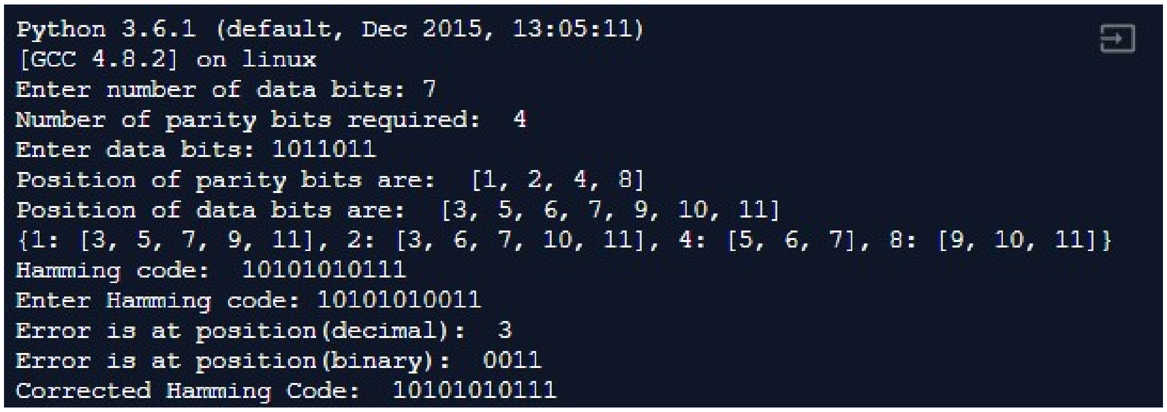


IMPLEMENTATION: (printout of codes)

```
d_bits=int(input("Enter number of data bits: "))
p_bits=2

#finding number of parity bits
while((d_bits+p_bits) > 2**p_bits):
  p_bits+=1
bits=p_bits+d_bits
print("Number of parity bits required: ",p_bits)

#11-bit Hamming Code parity bit generator

d=input("Enter data bits: ")
d=[int(d[i:i+1]) for i in range(0, len(d))]
d.reverse()
#Finding parity positions
parityPos=[]
for i in range(0,p_bits):
  parityPos.append(2**i)
print("Position of parity bits are: ",parityPos)
```

```
#Finding data bit position
finalArr=[i for i in range(1,12)]
dataPos=list(set(finalArr)-set(parityPos))
print("Position of data bits are: ", dataPos)

j=0
for i in dataPos:
  finalArr[i-1]=d[j]
  j+=1
#print("Intermediate stage: ")
#print(finalArr)
sum_dict={3:(1,2),5:(1,4),6:(2,4),7:(1,2,4),9:(1,8),10:(2,8),11:(1,2,8)}

#Finding corresponding positions for individual parity bits
XORList={}

for j in parityPos:
  temp=list()
  for k,v in sum_dict.items():
    if j in v:
      temp.append(k)
  XORList[j]=temp

print(XORList)

for k,v in XORList.items():
  temp=0
  for pos in v:
    temp^=finalArr[pos-1]
  finalArr[k-1]=temp
finalArr.reverse()
print("Hamming code: ",''.join(str(x) for x in finalArr))

#Error Detection in Hamming Code

HammingData=input("Enter Hamming code: ")
HammingData=[int(HammingData[i:i+1]) for i in range(0, len(HammingData))]
HammingData.reverse()

parityCheck=[]
for k,v in XORList.items():
  temp=0
  temp^=HammingData[k-1]
  for i in v:
    temp^=HammingData[i-1]
  parityCheck.append(temp)

parityCheck.reverse()
posi=0
for i in parityCheck:
  posi= posi*10 +i
position=int(str(posi),2)
if(posi==0):
  print("No error.")
else:
  print("Error is at position(decimal): ", position)
  print('Error is at position(binary): ',''.join(str(x) for x in parityCheck))
  HammingData[position-1]+=1
  HammingData[position-1]%=2
HammingData.reverse()
print("Corrected Hamming Code: ",''.join(str(x) for x in HammingData))
```

Output Screen:



```
Python 3.6.1 (default, Dec 2015, 13:05:11)
[GCC 4.8.2] on linux
Enter number of data bits: 7
Number of parity bits required:  4
Enter data bits: 1011011
Position of parity bits are:  [1, 2, 4, 8]
Position of data bits are:  [3, 5, 6, 7, 9, 10, 11]
{1: [3, 5, 7, 9, 11], 2: [3, 6, 7, 10, 11], 4: [5, 6, 7], 8: [9, 10, 11]}
Hamming code:  10101010111
Enter Hamming code: 10101010011
Error is at position(decimal):  3
Error is at position(binary):  0011
Corrected Hamming Code:  10101010111
```

CONCLUSION:
The program for 11-bit hamming code was implemented successfully and the program ran without errors for all test cases.

Post Lab Questions

1. State advantages of hamming code.

Ans.
The advantages of hamming code include:
1. It is useful for a single bit change, which is more probable than two or more-bit changes.
2. The simplicity of hamming codes makes them suitable for use in computer memory and single-error correction.
3. Hamming codes can detect up to two-bit errors or correct one-bit errors without detection of uncorrected errors. By contrast, the simple parity code cannot correct errors, and can detect only an odd number of bits in error.
4. They use a double-error detection variant called SECDED. These codes have a minimum hamming distance of three, where the code detects and corrects single errors while double bit errors are detected only if a correction is not attempted.
5. It works well in low error rate applications telecoms.