LAB - 7

Cyclic Encoder:

Cyclic codes are a class of error-correcting codes used in digital communication systems to detect and correct errors in transmitted data. These codes are linear block codes that have the property that any cyclic shift of a code word is also a code word.

To encode a message using a cyclic code, the message is first padded with additional bits to form a code word that is a multiple of the generator polynomial. The code word is then divided by the generator polynomial to obtain the remainder, which is the encoded message.

Cyclic codes can correct errors by using a technique called syndrome decoding. When a received code word contains errors, the syndrome is calculated by dividing the received code word by the generator polynomial. The syndrome is then used to locate and correct the errors in the code word.

Cyclic codes are widely used in digital communication systems, such as satellite communication, cellular networks, and storage systems, due to their simplicity and efficiency in error correction. They are also used in applications where data integrity is critical, such as in financial transactions, medical records, and military communications.

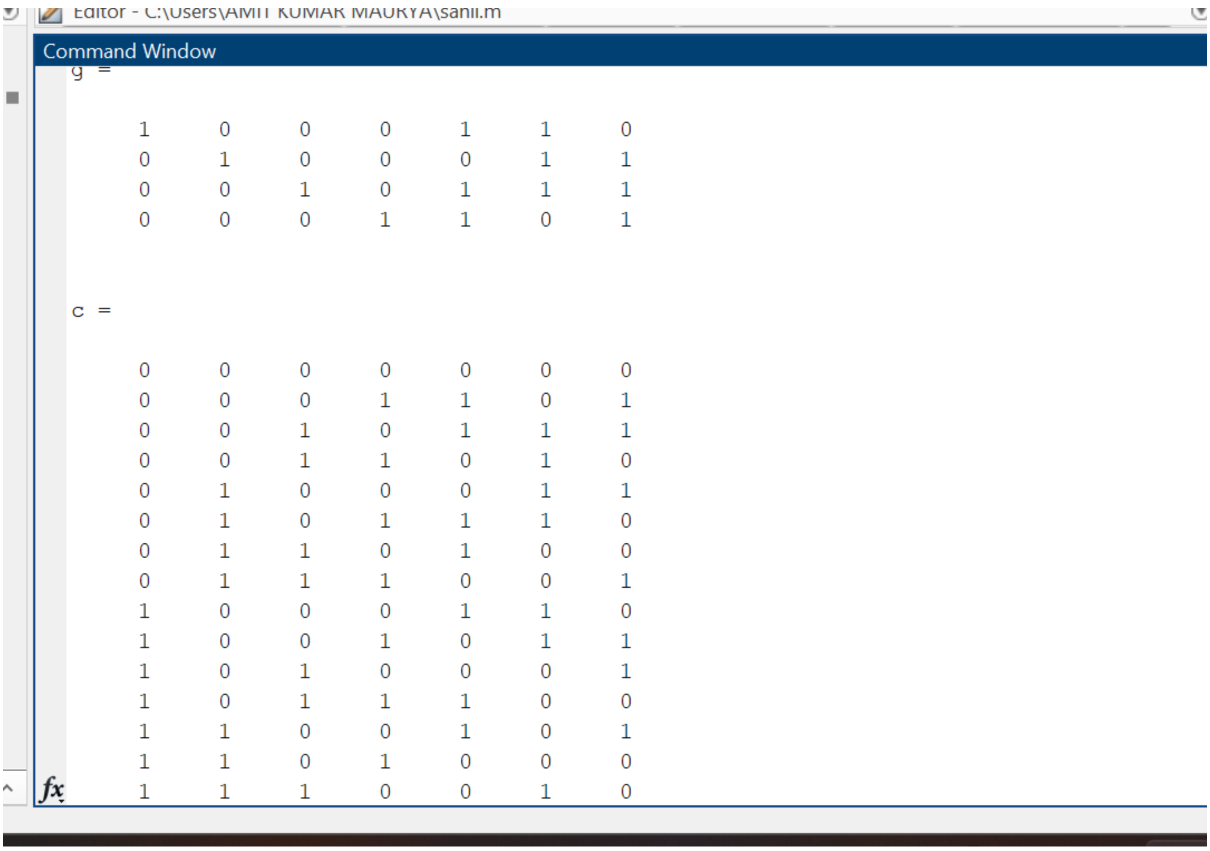
MATLAB CODE:

```
clc;

 close all;

n=7;
k=4;
gx= [1 1 0 1]
d = de2bi(0:2^k-1,'left-msb');
[i,px]= cyclgen(7,gx,'system');
g=circshift(px,[0,4])
c=rem(d*g,2)
```

OUTPUT:



```
Editor - C:\Users\AMIT KUMAR MAURYA\sanil.m
Command Window
g =

     1     0     0     0     1     1     0
     0     1     0     0     0     1     1
     0     0     1     0     1     1     1
     0     0     0     1     1     0     1


c =

     0     0     0     0     0     0     0
     0     0     0     1     1     0     1
     0     0     1     0     1     1     1
     0     0     1     1     0     1     0
     0     1     0     0     0     1     1
     0     1     0     1     1     1     0
     0     1     1     0     1     0     0
     0     1     1     1     0     0     1
     1     0     0     0     1     1     0
     1     0     0     1     0     1     1
     1     0     1     0     0     0     1
     1     0     1     1     1     0     0
     1     1     0     0     1     0     1
     1     1     0     1     0     0     0
fx   1     1     1     0     0     1     0
```

Determine the error correcting capability of given (n,k) code using hamming bound:

The Hamming bound is a theoretical upper bound on the error-correcting capability of a linear block code. It provides an estimate of the maximum number of errors that a code can correct based on its code length (n) and dimension (k). The Hamming bound is given by the formula:

2^(n-k) >= n + 1

where "^" denotes exponentiation, and ">= " denotes "greater than or equal to".

To determine the error-correcting capability of a given (n, k) code using the Hamming bound theory, you can follow these steps:

Input the values of n and k for the given code.

Calculate 2^(n-k).

Compare 2^(n-k) with n + 1.

If 2^(n-k) is greater than or equal to n + 1, then the code satisfies the Hamming bound and is capable of correcting at least one error. Otherwise, it does not satisfy the Hamming bound and may not be capable of correcting errors.

In other words, if 2^(n-k) is greater than or equal to n + 1, then the code can correct at least (2^(n-k) - 1) errors. If 2^(n-k) is less than n + 1, then the code may not have enough error-correcting capability to correct any errors.

Please note that the Hamming bound is a theoretical limit and does not guarantee the actual performance of a specific code in practice. Actual error-correction performance of a code depends on its specific implementation and the characteristics of the channel or medium over which the codeword is transmitted. It is always recommended to test and verify the error-correction performance of a specific code using simulations or experiments in the specific context of its intended application.

MATLAB CODE:

```
clc;
clear all;
close all;
n  = input('enter the code bits: n:');
k=input('enter the data bits:k:');
m=n-k;
disp(m);
z=2^m;
disp(z);
i=0;
for j=0:1:n
    sum=0;
    for i=0:1:j
        c=factorial(n)/factorial(n-i)*factorial(i);
        sum=sum+c;
    end
    if sum >=z
        j=j-1;
        sum=sum-c;
        break;
    end
end
ans=sprintf('(%d,%d) can correct all the combinations of %d error and it can also correct %d combinational of %d error',n,k,j,z-sum,j+1);

             disp(ans);
```

OUTPUT:

```
Command Window
enter the code bits: n:7
enter the data bits:k:4
     3

     8

(7,4) can correct all the combinations of 0 error and it can also correct 7 combinationa
fx >>
```