NOAKHALI SCIENCE AND TECHNOLOGY UNIVERSITY

Lab Report

Course name: Data Communication Lab.

Course code: CSTE 3102.

Submitted by:	**Submitted to:**
Name: Farman Arefin Tamim **Roll:** ASH2001043M **Year:** 03, **Term:** 01 **Session:** 2019-2020 Dept. of Computer Science and Telecommunication Engineering.	**Dr. Mohammed Humayun Kabir** Professor. Dept. of Computer Science and Telecommunication Engineering. Noakhali Science and Technology University.

Submission Date: 25/09/2023.

1) <u>Experiment Name:</u> Write a program in the programming language you are familiar with - A program to simulate the calculation of Fletcher checksum, use your full roll number as data for the sender (the output of sender is the input of receiver).

Software: CodeBlocks.

Language: C++.

Ans: Fletcher – 16 − The data word is divided into 8-bit blocks. Then, two 8-bit checksums are computed and are appended to form a 16-bit Fletcher checksum

INPUT : data blocks of equal sizes, $b1$,$b2$………… bn

OUTPUT : two checksums, $checksum1$ and $checksum2$, of 1 byte each

Step 1) Initialize partial sums, and sums, $c1$=0 and $c2$=0

Step 2) For each data block, bi

i. Add bi to $c1$

ii. Add updates value of $c1$ to $c2$

Step 3) Compute checksums,

$checksum1$= $c1$ MOD 256 and $checksum2$= $c2$ MOD 256

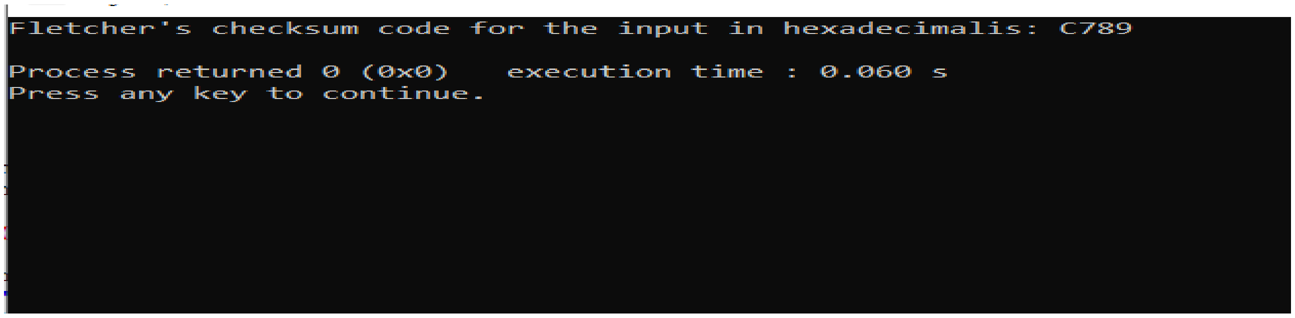
Step 4) Append checksums, $checksum1$ and $checksum2$, to the data blocks, $b1$,2………… bn

<u>Code:</u>

```
#include<bits/stdc++.h>

using namespace std;

uint16_t Fletcher_16(string data, int n)

{

uint16_t R = 0, L = 0, check_sum = 0;

for (int i = 0; i < n; i++) {

R = (R + data[i]) % 256;

L = (L + R) % 256;

}

check_sum = L * 256 + R;

return check_sum;

}

int main()
```

```
{
string input = "ASH2001043M";
cout << "Fletcher's checksum code for the input in hexadecimal
is: ";
printf("%X\n", Fletcher_16(input, input.size()));
}
```

Output:



```
Fletcher's checksum code for the input in hexadecimalis: C789

Process returned 0 (0x0)    execution time : 0.060 s
Press any key to continue.
```

2) Experiment Name: Write a program in the programming language you are familiar with: A program to simulate the calculation of Adler checksum, use your full roll number as data for the sender (the output of sender is the input of receiver).

Software: CodeBlocks.

Language: C++.

Ans: integer. A is the sum of all bytes in the stream plus one, and B is the sum of the individual values of A from each step.

At the beginning of an Adler-32 run, A is initialized to 1, B to 0. The sums are done modulo 65521 (the largest prime number smaller than 216). The bytes are stored in network order (big endian), B occupying the two most significant bytes.

The function may be expressed as

A = 1 + D1 + D2 + ... + Dn (mod 65521)

B = (1 + D1) + (1 + D1 + D2) + ... + (1 + D1 + D2 + ... + Dn) (mod 65521)

  = n×D1 + (n−1)×D2 + (n−2)×D3 + ... + Dn + n (mod 65521)

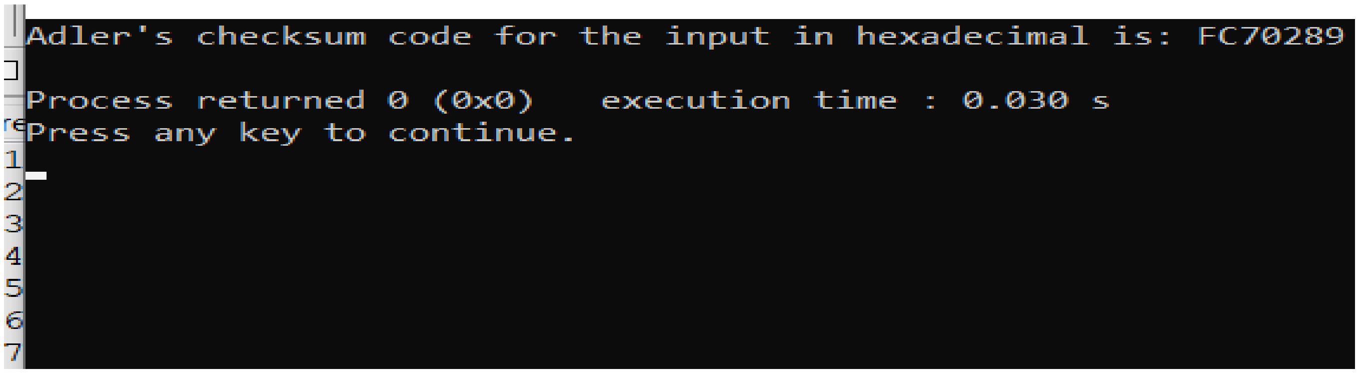
Adler-32(D) = B × 65536 + A *ekhane 16 bit right shift kortesi

where D is the string of bytes for which the checksum is to be calculated, and n is the length of D.

Code:

```
#include<bits/stdc++.h>
using namespace std;
uint32_t Adler_32(string data, int n)
{
uint32_t R = 0, L = 0, check_sum = 0;
for (int i = 0; i < n; i++) {
R = (R + data[i]) % 65521;
L = (L + R) % 65521;
}
check_sum = L * 65536 + R;
return check_sum;
}
int main()
{
string input = "ASH2001043M";
cout <<"Adler's checksum code for the input in hexadecimal is: ";
printf("%X\n", Adler_32(input, input.size()));
}
```

Output:



```
Adler's checksum code for the input in hexadecimal is: FC70289

Process returned 0 (0x0)   execution time : 0.030 s
Press any key to continue.
```

3) <u>Experiment Name:</u> Write a program in the programming language you are familiar with:- A program to simulate the calculation of cyclic redundancy check (CRC).

Software: CodeBlocks.

Language: C++.

Ans: CRC or Cyclic Redundancy Check is a method of detecting accidental changes/errors in the communication channel.
CRC uses **Generator Polynomial** which is available on both sender and receiver side. An example generator polynomial is of the form like $x^3 + x + 1$. This generator polynomial represents key 1011. Another example is $x^2 + 1$ that represents key 101.
n : Number of bits in data to be sent

from sender side.

k : Number of bits in the key obtained

from generator polynomial.

Data word to be sent - 100100

Key – 1101

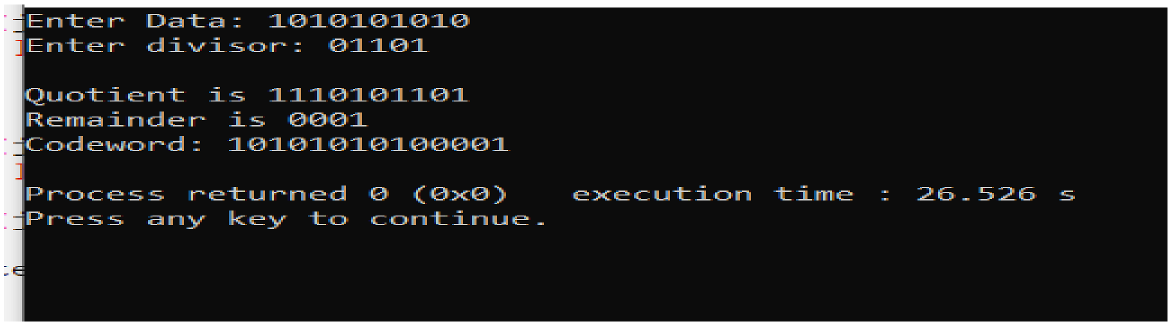
Code:

```
#include <bits/stdc++.h>

using namespace std;

int main()

{

int i, j, divlen, msglen;

char input[100], div[100], temp[100];

char quot[100], rem[100], div1[100];

printf("Enter Data: ");

gets(input);

printf("Enter divisor: ");

gets(div);

divlen = strlen(div);

msglen = strlen(input);

strcpy(div1, div);

for (i = 0; i < divlen - 1; i++)

input[msglen + i] = '0';

for (i = 0; i < divlen; i++)

temp[i] = input[i];
```

```
for (i = 0; i < msglen; i++)
{
quot[i] = temp[0];
if (quot[i] == '0')
{
for (j = 0; j < divlen; j++)
div[j] = '0';
}
else
{
for (j = 0; j < divlen; j++)
div[j] = div1[j];
}
for (j = divlen - 1; j > 0; j--)
{
if (temp[j] == div[j])
rem[j - 1] = '0';
else
rem[j - 1] = '1';
}
rem[divlen - 1] = input[i + divlen];
strcpy(temp, rem);
}
strcpy(rem, temp);
printf("\nQuotient is ");
for (i = 0; i < msglen; i++)
printf("%c", quot[i]);
printf("\nRemainder is ");
```

```
for (i = 0; i < divlen - 1; i++)
printf("%c", rem[i]);
printf("\nCodeword: ");
for (i = 0; i < msglen; i++)
printf("%c", input[i]);
for (i = 0; i < divlen - 1; i++)
printf("%c", rem[i]);
printf("\n");
}
```

Output:



```
Enter Data: 1010101010
Enter divisor: 01101

Quotient is 1110101101
Remainder is 0001
Codeword: 10101010100001

Process returned 0 (0x0)   execution time : 26.526 s
Press any key to continue.
```

4) <u>Experiment Name:</u> Write a program in the programming language you are familiar with:- A program to simulate the calculation of Hamming code generation for 5 bit data (e.g. the data is 01101).

Software: CodeBlocks.

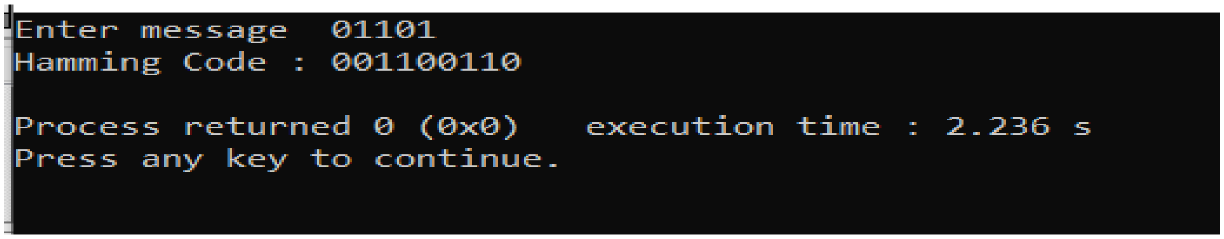
Language: C++.

Ans: Hamming code is an error-correcting code used for detecting and correcting errors in data transmission. It adds redundant bits to the data being transmitted which can be used to detect and correct errors that may occur during transmission. Developed by Richard W. Hamming in the 1950s, it is widely used in applications where reliable data transmission is critical, such as computer networks and telecommunication systems.

Code:

```
#include<bits/stdc++.h>
using namespace std;
vector<char> generate_hamming(string message)
{
 int n = message.size(), p = 0, hammingLength, k = 0;
 int currentPower = 1;
 string str = message;
 while (currentPower < p + n + 1) {
 p++;
 currentPower *= 2;
 }
 hammingLength = n + p;
 reverse(str.begin(), str.end());
 vector<char> ans(hammingLength + 1, '0');
 for (int i = 1; i <= hammingLength; i++) {
 if (i & (i - 1)) { //check if 'i' is a power of 2 or not, if 0
then power of 2 else not
 ans[i] = str[k];
 k++;
 }
 }
 for (int i = 0; i <= p - 1; i++) {
 int XOR = 0, flag = 1, skipBit = (1 << i);
 for (int j = (1 << i); j <= hammingLength; j += skipBit) {
 if (!flag) {
 flag ^= 1;
 continue;
 }
 flag ^= 1;
 int l = j, limit = skipBit;
 while (limit > 0 && l <= hammingLength) { //check limit
and skipBit limit
 XOR ^= (ans[l] - '0');
 l++, limit--;
 }
 }
 if (XOR) ans[(1 << i)] = '1';
 else ans[(1 << i)] = '0';
 }
 return ans;
}
int main()
{
 string message;
 vector<char> hammingCode;
 cin >> message;
 hammingCode = generate_hamming(message);
 for (int i = hammingCode.size() - 1; i >= 1; i--)
 cout << hammingCode[i];
 cout << "\n";
}
```

Output:



```
Enter message  01101
Hamming Code : 001100110

Process returned 0 (0x0)   execution time : 2.236 s
Press any key to continue.
```

5) <u>Experiment Name:</u> Write a program in the programming language you are familiar with - A program to simulate the calculation of error detection from the data (01101) with Hamming code.

Software: CodeBlocks.

Language: C++.

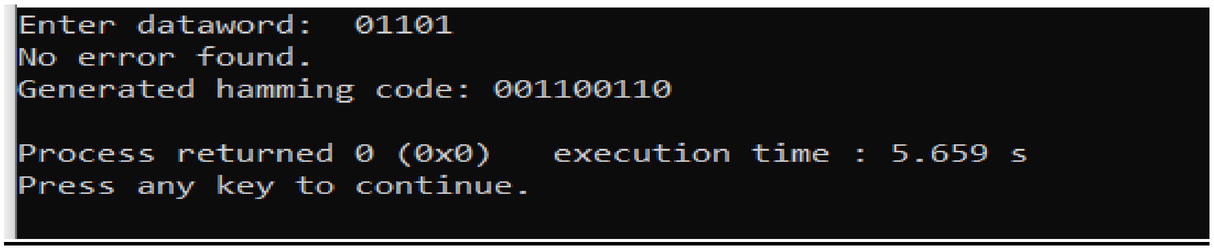
Ans: **Hamming code** is error-detection and error-correction code which is used to find and correct errors in a code while transmission in data communication. The original data bits are mixed with some bits called redundant bits from the sender sides. Then on the receiver side, the Hamming codes are decoded to find the errors while communication.

Hamming code can only detect 2-bit error and can correct a single bit error only.

Code:

```
#include<bits/stdc++.h>
using namespace std;
int p = 0;
vector<char> generate_hamming(string message)
{
 int n = message.size(), hammingLength, k = 0;
 int currentPower = 1;
 string str = message;
 while (currentPower < p + n + 1) {
 p++;
 currentPower *= 2;
 }
 hammingLength = n + p;
 reverse(str.begin(), str.end());
 vector<char> ans(hammingLength + 1, '0');
 for (int i = 1; i <= hammingLength; i++) {
 if (i & (i - 1)) { //check if 'i' is a power of 2 or not, if 0
then power of 2 else not
 ans[i] = str[k];
 k++;
 }
 }
 for (int i = 0; i <= p - 1; i++) {
 int XOR = 0, flag = 1, skipBit = (1 << i);
 for (int j = skipBit; j <= hammingLength; j += skipBit) {
 if (!flag) {
 flag ^= 1;
 continue;
 }
 flag ^= 1;
 int l = j, limit = skipBit;
 while (limit > 0 && l <= hammingLength) { //check limit
and skipBit limit
 XOR ^= (ans[l] - '0');
 l++, limit--;
 }
 }
 if (XOR) ans[(1 << i)] = '1';
 else ans[(1 << i)] = '0';
 }
 return ans;
}
bool check_error(vector<char> v)
{
 int hammingLength = v.size() - 1;
 for (int i = 0; i <= p - 1; i++) {
 int XOR = 0, skipBit = (1 << i);
 for (int j = (1 << i); j <= hammingLength; j += skipBit) {
 int l = j, limit = skipBit;
 while (limit > 0 && l <= hammingLength) { //check limit
and skipBit limit
 XOR ^= (v[l] - '0');
 l++, limit--;
 }
 }
 if (XOR == 1) return true;
 }
 return false;
}
int main()
{
 string message;
 bool error = 0;
 vector<char> hammingCode;
 cout << "Enter dataword: ";
 cin >> message;
 hammingCode = generate_hamming(message);
 error = check_error(hammingCode);
 if (error) cout << "Error found.\n";
 else cout << "No error found.\n";
 cout << "Generated hamming code: ";
 for (int i = hammingCode.size() - 1; i >= 1; i--)
 cout << hammingCode[i];
 cout << "\n";
}
```

Output:



```
Enter dataword:  01101
No error found.
Generated hamming code: 001100110

Process returned 0 (0x0)   execution time : 5.659 s
Press any key to continue.
```

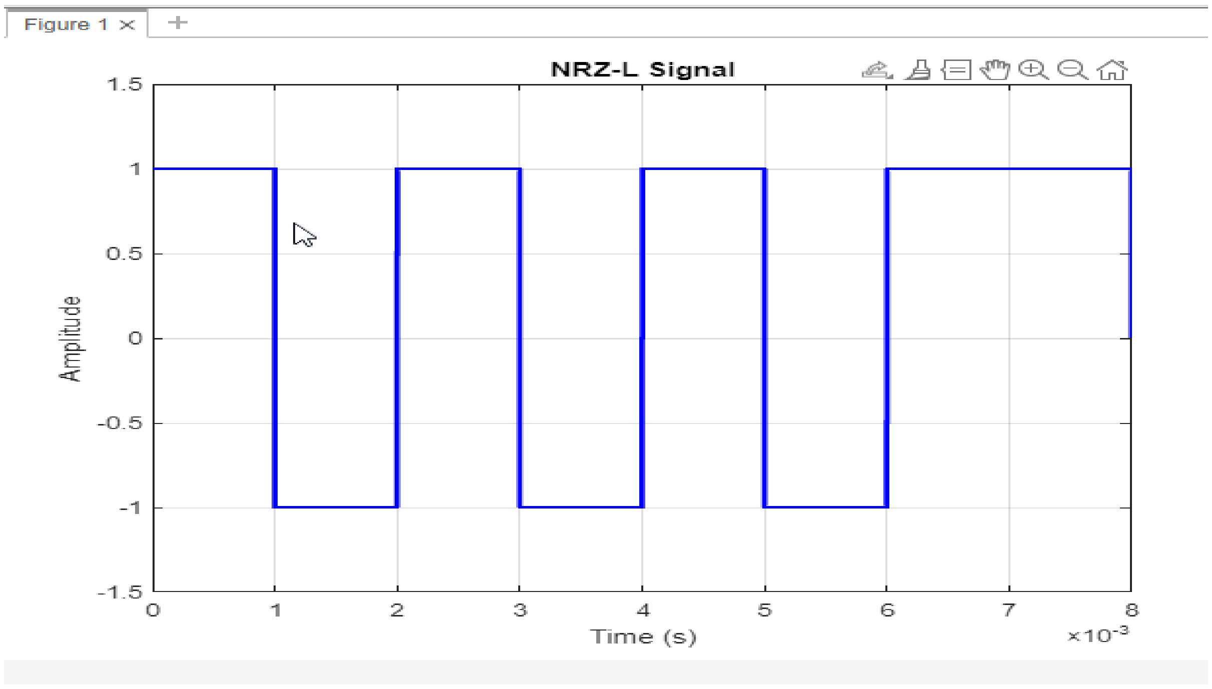
6) **Experiment Name:** using MATLAB generate the following signal NRZ.

Software: MATLAB.

Code:

```
clc;
close all;
clear all;

% Parameters
num_bits = 8; % Number of bits
bitrate = 1000; % Bit rate (bits per second)

% Define your custom binary data sequence
binary_data = [1 0 1 0 1 0 1 1]; % Replace with your own binary values

% Time vector for the NRZ signal
T = num_bits / bitrate;
t = linspace(0, T, num_bits * 1000); % Higher resolution time vector for plotting

% Create an empty NRZ signal
nrz_signal = zeros(1, length(t));

% Generate NRZ-L signal
bit_duration = 1 / bitrate;
for i = 1:num_bits
    bit_start = (i - 1) * bit_duration;
    bit_end = i * bit_duration;
    if binary_data(i) == 1
        nrz_signal(t >= bit_start & t < bit_end) = 1;
    else
        nrz_signal(t >= bit_start & t < bit_end) = -1;
    end
end

% Plot the NRZ signal
figure;
plot(t, nrz_signal, 'b', 'LineWidth', 2);
title('NRZ-L Signal');
xlabel('Time (s)');
ylabel('Amplitude');
axis([0 T -1.5 1.5]);
grid on;

% Display the binary data
disp('Binary Data:');
disp(binary_data);
```

Output:



7) <u>Experiment Name:</u> using MATLAB generate the following signal RZ.

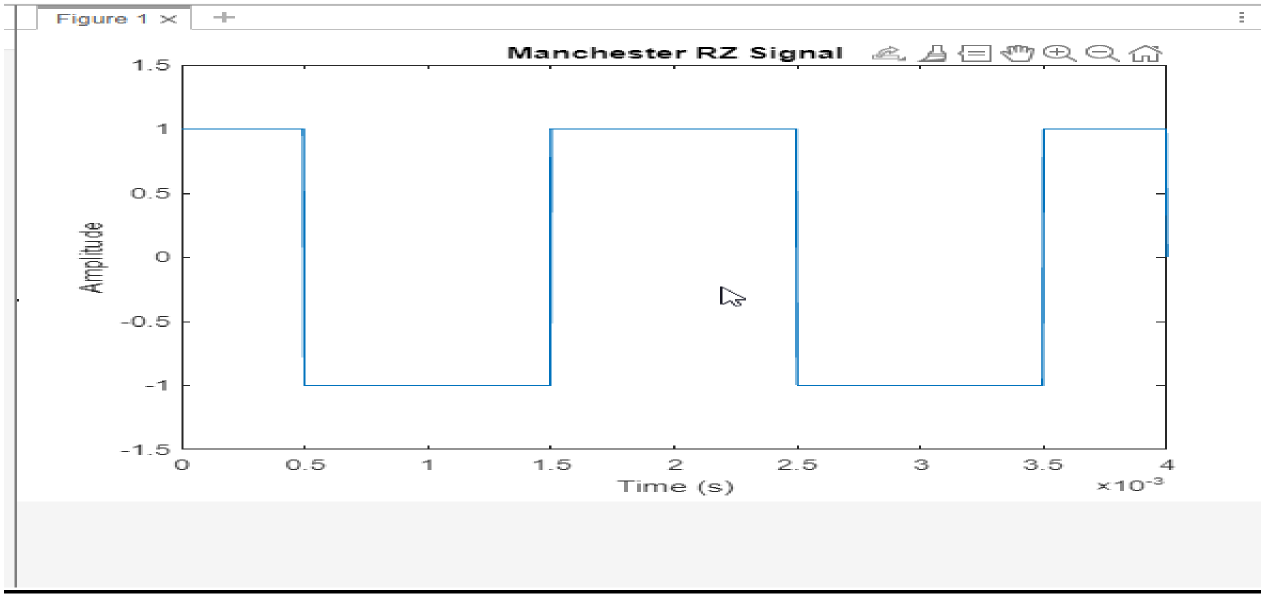
<u>Software:</u> MATLAB.

<u>Code:</u>

```
% Manchester RZ Simulation

% Clear the workspace and close all figures
clc;
close all;
clear all;

% Define the binary data sequence
binary_data = [1 0 1 0];

% Bitrate
bitrate = 1000; % 1000 bits per second

% MANCHESTER Encode bit string using Manchester code
T = length(binary_data) / bitrate; % Full time of bit sequence
n = 200;
N = n * length(binary_data);
dt = T / N;
t = 0:dt:T;
manchester_signal = zeros(1, length(t)); % Output signal

for i = 0:length(binary_data) - 1
    if binary_data(i + 1) == 1
```

```
            manchester_signal((i * n) + 1 : ((i + 0.5) * n)) = 1;
            manchester_signal(((i + 0.5) * n) + 1 : (i + 1) * n) = -1;
        else
            manchester_signal((i * n) + 1 : ((i + 0.5) * n)) = -1;
            manchester_signal(((i + 0.5) * n) + 1 : (i + 1) * n) = 1;
        end
    end

    % Plot the Manchester RZ signal
    figure;
    plot(t, manchester_signal);
    xlabel('Time (s)');
    ylabel('Amplitude');
    title('Manchester RZ Signal');
    axis([0 max(t) -1.5 1.5]);

    % Display binary data
    disp('Binary Data:');
    disp(binary_data);
```

Output:



8) Experiment Name: **using MATLAB generate the following signal PCM.**

Software: MATLAB.

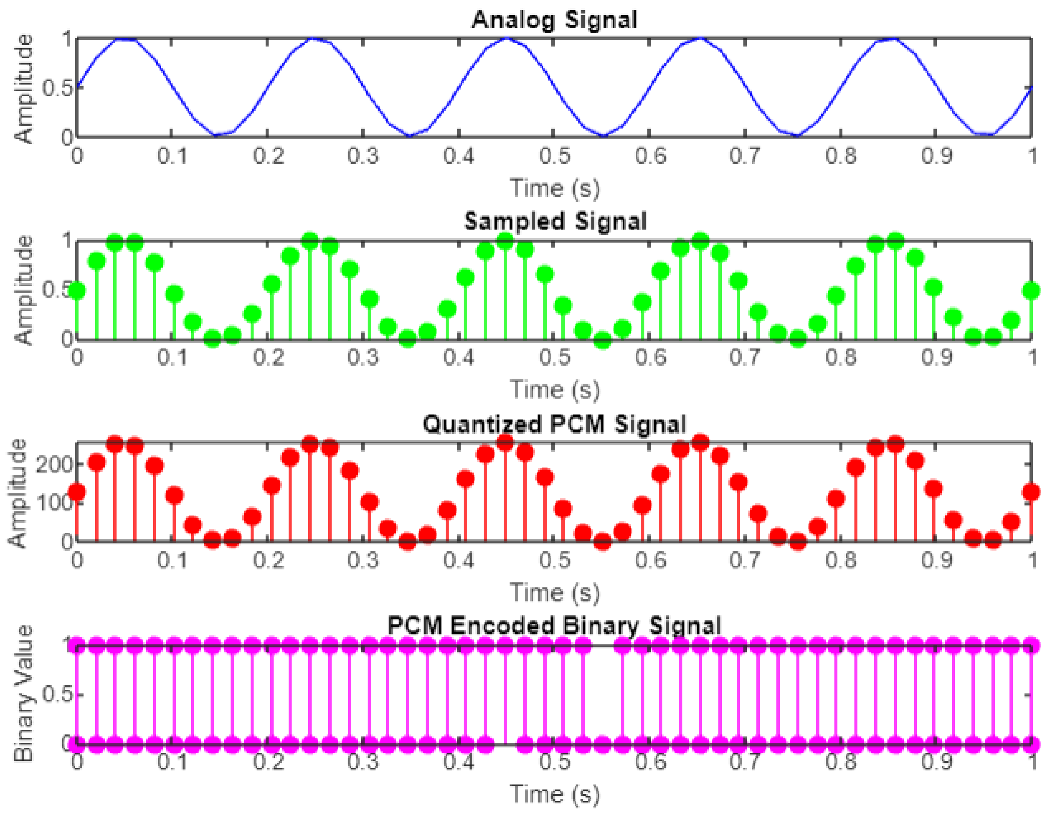
Code:

```
% Clear the workspace and close all figures
```

```
clc;
close all;
clear all;

% Parameters
num_bits = 8; % Number of bits per sample
num_samples = 100; % Number of samples

% Create a continuous analog signal (sine wave) for demonstration
fs = 1000; % Sampling frequency (Hz)
t = linspace(0, 1, num_samples); % Time vector from 0 to 1 second
analog_signal = sin(2 * pi * 5 * t); % Example analog signal (5 Hz sine wave)

% Normalize the analog signal to the range [0, 1]
analog_signal = (analog_signal - min(analog_signal)) / (max(analog_signal) -
min(analog_signal));

% Sample the analog signal
sampled_signal = analog_signal;

% Quantize the sampled signal to num_bits levels (PCM)
quantized_signal = round(sampled_signal * (2^num_bits - 1));

% Manually encode the quantized values into binary (PCM encoding)
encoded_signal = dec2bin(quantized_signal, num_bits) - '0'; % Convert to binary and
subtract '0' to get numeric values

% Manually decode the binary signal back to decimal (for demonstration)
decoded_signal = bin2dec(char(encoded_signal + '0'));

% Plot the analog, sampled, quantized, and encoded signals
figure;
subplot(4, 1, 1);
plot(t, analog_signal, 'b');
title('Analog Signal');
xlabel('Time (s)');
ylabel('Amplitude');

subplot(4, 1, 2);
stem(t, sampled_signal, 'g', 'filled');
title('Sampled Signal');
xlabel('Time (s)');
ylabel('Amplitude');

subplot(4, 1, 3);
stem(t, quantized_signal, 'r', 'filled');
title('Quantized PCM Signal');
xlabel('Time (s)');
ylabel('Amplitude');

subplot(4, 1, 4);
stem(t, encoded_signal, 'm', 'filled');
title('PCM Encoded Binary Signal');
xlabel('Time (s)');
ylabel('Binary Value');
```

```
% Display decoded signal (for demonstration)
disp('Decoded Signal (Quantized Values):');
disp(decoded_signal);
```

Output:



9) Experiment Name: using MATLAB generate the following signal ASK.

Software: MATLAB

Ans : Amplitude Shift Keying

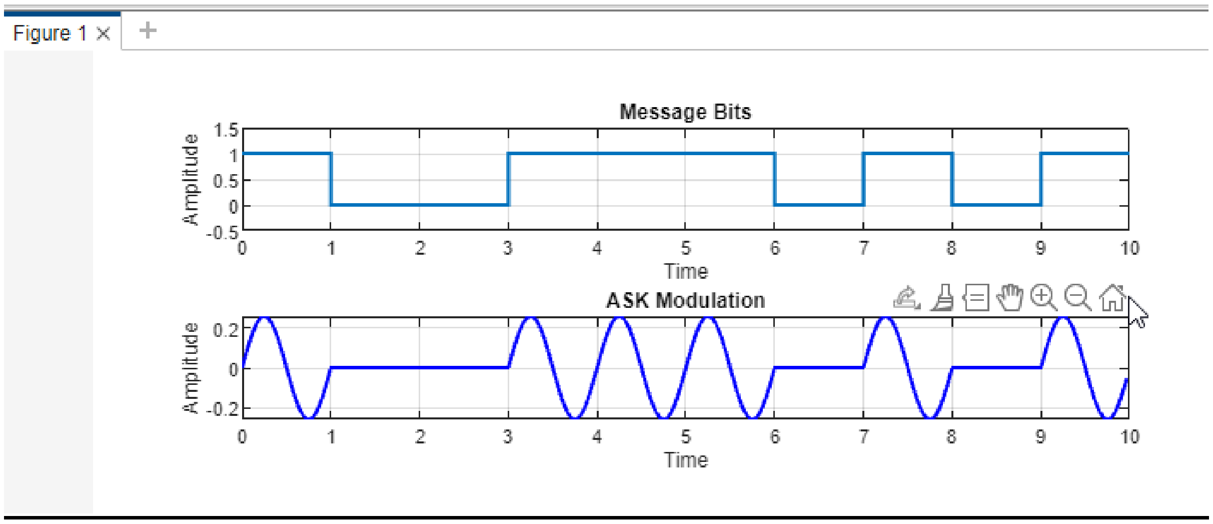
ASK is a type of Amplitude Modulation which represents the binary data in the form of variations in the amplitude of a signal. Any modulated signal has a high frequency carrier. The binary signal when ASK modulated, gives a zero value for Low input while it gives the carrier output for High input.

Code:

```
clc;
close all;
```

```
clear all;

n=10; %length of bit stream
b= [1 0 0 1 1 1 0 1 0 1]

f1=1;f2=2;
t=0:1/30:1-1/30;

%ASK
sa1=sin(2*pi*f1*t);
E1=sum(sa1.^2);
sa1=sa1/sqrt (E1); %unit energy
sa0=0*sin(2*pi*f1*t);

ask=[];
for i=1:n

if b(i)==1
ask=[ask sa1];
else
ask=[ask sa0];
  end
end
figure (1)
subplot (411)
stairs (0:10,[b(1:10) b(10)], 'linewidth',1.5)
axis([0 10 -0.5 1.5])
title('Message Bits');
grid on
xlabel('Time');
ylabel('Amplitude')


subplot (412)
tb=0:1/30:10-1/30;
plot(tb, ask (1:10*30), 'b', 'linewidth', 1.5)
title('ASK Modulation'); grid on
xlabel('Time');
ylabel('Amplitude')
```

Output:



10) <u>Experiment Name:</u> using MATLAB generate the following signal FSK.

<u>Software:</u> MATLAB

Ans:

Frequency Shift Keying FSK

is the digital modulation technique in which the frequency of the carrier signal varies according to the digital signal changes. FSK is a scheme of frequency modulation.

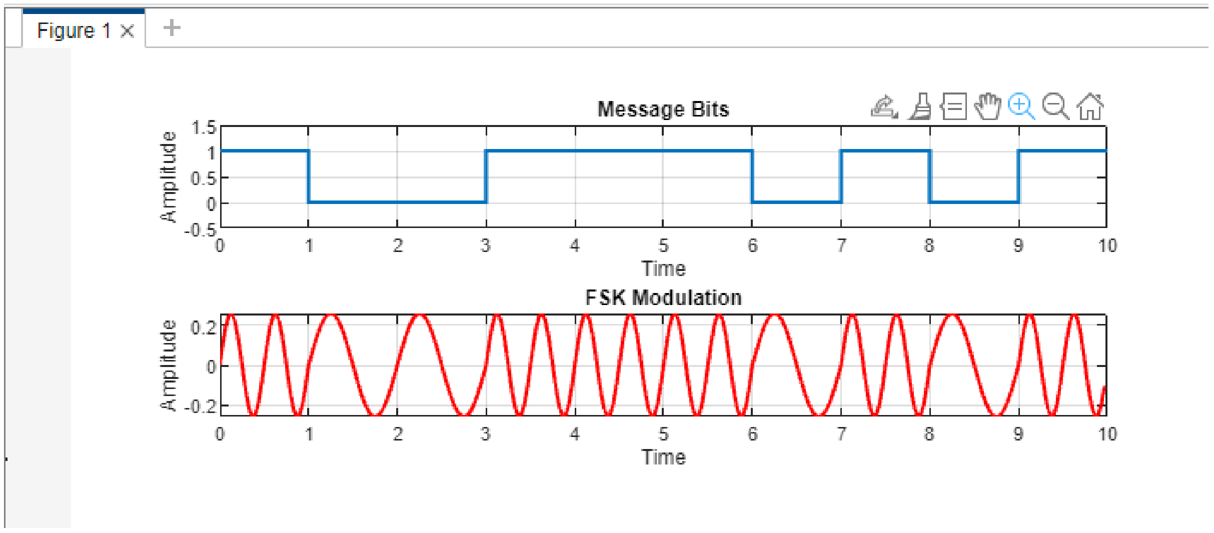
The output of a FSK modulated wave is high in frequency for a binary High input and is low in frequency for a binary Low input. The binary 1s and 0s are called Mark and Space frequencies.

<u>Code:</u>

```
clc;
close all;
clear all;

n = 10; % Length of bit stream
b = [1 0 0 1 1 1 0 1 0 1];

f1 = 1;
f2 = 2;
t = 0:1/30:1-1/30;

% FSK
sf0 = sin(2*pi*f1*t);
E0 = sum(sf0.^2);
sf0 = sf0/sqrt(E0);
sf1 = sin(2*pi*f2*t);
E1 = sum(sf1.^2); % Corrected variable name here
```

```
sf1 = sf1/sqrt(E1);

fsk = [];
for i = 1:n
    if b(i) == 1
        fsk = [fsk sf1];
    else
        fsk = [fsk sf0];
    end
end

figure(1)
subplot(411)
stairs(0:10, [b(1:10) b(10)], 'linewidth', 1.5)
axis([0 10 -0.5 1.5])
title('Message Bits');
grid on
xlabel('Time');
ylabel('Amplitude')

subplot(412)
tb = 0:1/30:10-1/30;
plot(tb, fsk(1:10*30), 'r', 'linewidth', 1.5)
title('FSK Modulation');
grid on
xlabel('Time');
ylabel('Amplitude')
```

<u>Output:</u>



11) <u>Experiment Name:</u> using MATLAB generate the following signal PSK.

Software: MATLAB

Ans:

Phase Shift Keying PSK

is the digital modulation technique in which the phase of the carrier signal is changed by varying the sine and cosine inputs at a particular time. PSK technique is widely used for wireless LANs, bio-metric, contactless operations, along with RFID and Bluetooth

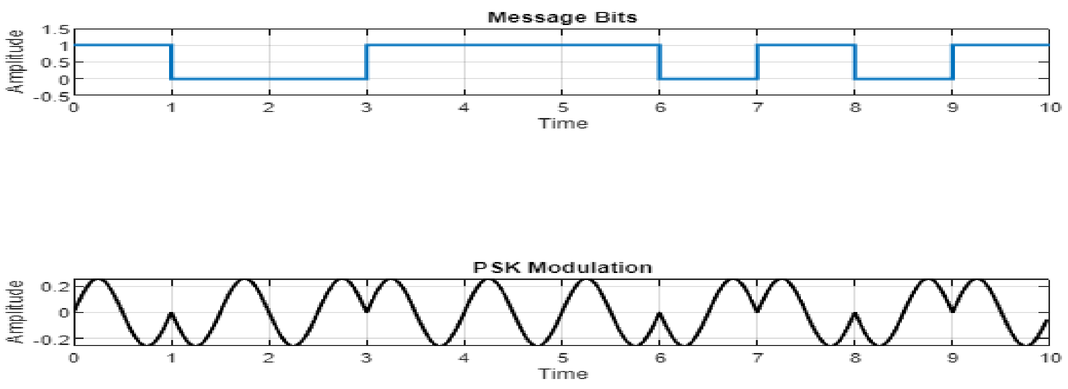
Communications.

Code:

```
clc;
close all;
clear all;

n=10; %length of bit stream
b= [1 0 0 1 1 1 0 1 0 1]

f1=1;f2=2;
t=0:1/30:1-1/30;

%PSK
sp=sin(2*pi*f1*t);
E1=sum (sp.^2);
sp0=-sin(2*pi*f1*t)/sqrt(E1);
sp1=sin(2*pi*f1*t)/sqrt(E1);

psk=[];
for i=1:n

if b(i)==1
psk=[psk sp1];
else
psk=[psk sp0];
  end
end
figure (1)
subplot (411)
stairs (0:10,[b(1:10) b(10)], 'linewidth',1.5)
axis([0 10 -0.5 1.5])
title('Message Bits');
grid on
xlabel('Time');
ylabel('Amplitude')

subplot (412)
```

```
tb=0:1/30:10-1/30;
plot (tb, psk (1:10*30), 'k', 'linewidth',1.5)
title('PSK Modulation');grid on
xlabel('Time');
ylabel('Amplitude')
```

<u>Output:</u>



12) <u>Experiment Name:</u> using MATLAB generate the following signal QPSK.

<u>Software:</u> MATLAB

Ans:

Quadrature Phase Shift Keying QPSK

Quadrature Amplitude Modulation A modulation technique that employs both phase modulation (PM) and amplitude modulation (AM). Widely used to transmit digital signals such as digital cable TV and cable Internet service. in 8QAM, each three bits of input alters the phase and amplitude of the carrier to derive eight unique modulation states.

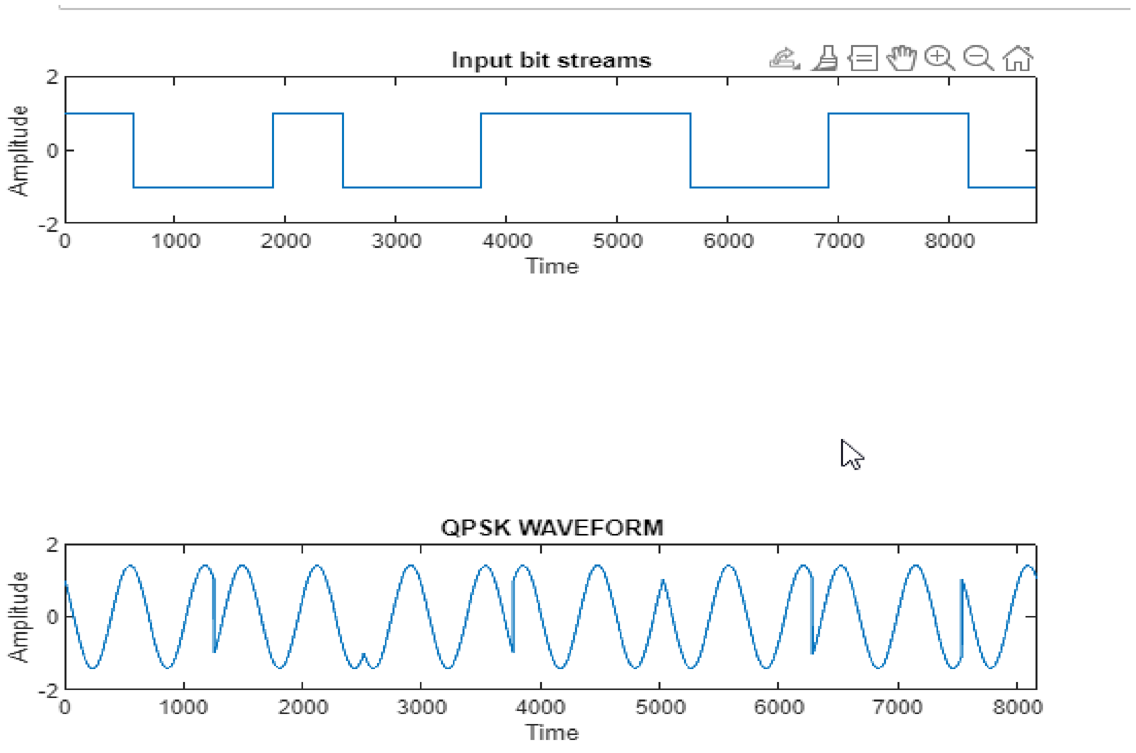
<u>Code:</u>

```
clc;
clear all;
close all;
% data stream input
d=[1 0 0 1 0 0 1 1 1 0 0 1 1];
ln=length(d);
x=0:1/100:ln*2*pi;
cc=cos(x);
cs=sin(x);
k=length(cc);
```

```
k1=k/ln;

% input data odd/even test
if(mod(ln,2)~=0)
    d(ln+1)=0;
    ln=ln+1;
end
for i=1:ln
  if(d(i)==0)
      d(i)=-1;
      i=i+1;
  end
end

% To make odd bit streams and even bit streams
i=1;j=1;
while (i<ln) && (j<ln)
    dd1(j)=d(i);
    dd1(j+1)=d(i);
    dd2(j)=d(i+1);
    dd2(j+1)=d(i+1);
    j=j+2;
    i=i+2;
end

% to make bit streams eqv to sinusoidal waveform
t=1;
for i=1:ln
   for j=1:k1

      dd(t)=d(i);
      d1(t)=dd1(i);
      d2(t)=dd2(i);
      t=t+1;
      j=j+1;
   end
   i=i+1;
end

subplot(3,1,1);
stairs(dd);
axis([0 t -2 2]);
title('Input bit streams');
ylabel('Amplitude');
xlabel('Time');

len=length(d1);
if(k<len)
    len=k;
end;
```

```
% odd streams multiplied with cosine waveform
% even streams multiplied with sine waveform
for i=1:len
    qcc(i)=cc(i)*d1(i);
    qcs(i)=cs(i)*d2(i);
    i=i+1;
end

%QPSK Output
qp=qcc+qcs;
subplot(3,1,3);
plot(qp);
axis([0 len -2 2]);
title('QPSK WAVEFORM');
ylabel('Amplitude');
xlabel('Time');
```

<u>Output:</u>



<u>13) Experiment Name:</u> using MATLAB generate the following signal BPSK.

<u>Software:</u> MATLAB

Ans:

Binary Phase Shift Keying BPSK

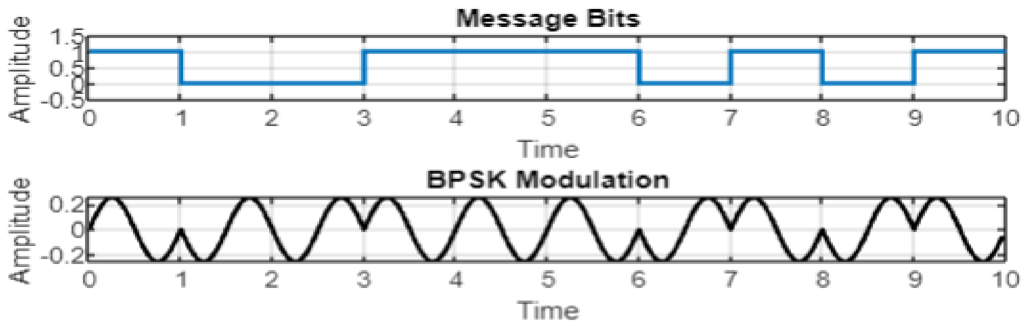
This is also called as 2-phase PSK or Phase Reversal Keying. In this technique, the sine wave carrier takes two phase reversals such as 0° and 180°.

BPSK is basically a Double Side Band Suppressed Carrier DSBSC modulation scheme, for message being the digital information.

Code:

```
 clc;

close all;
clear all;

n=10; %length of bit stream
b= [1 0 0 1 1 1 0 1 0 1]

f1=1;f2=2;
t=0:1/30:1-1/30;

%PSK
sp=sin(2*pi*f1*t);
E1=sum (sp.^2);
sp0=-sin(2*pi*f1*t)/sqrt(E1);
sp1=sin(2*pi*f1*t)/sqrt(E1);

psk=[];
for i=1:n

if b(i)==1
psk=[psk sp1];
else
psk=[psk sp0];
   end
end
figure (1)
subplot (411)
stairs (0:10,[b(1:10) b(10)], 'linewidth',1.5)
axis([0 10 -0.5 1.5])
title('Message Bits');
grid on
xlabel('Time');
ylabel('Amplitude')

subplot (412)
tb=0:1/30:10-1/30;
plot (tb, psk (1:10*30), 'k', 'linewidth',1.5)
title('BPSK Modulation');grid on
xlabel('Time');
ylabel('Amplitude')
```

Output:



---

14) Experiment Name: **using MATLAB simulate the TDM.**

Software: MATLAB

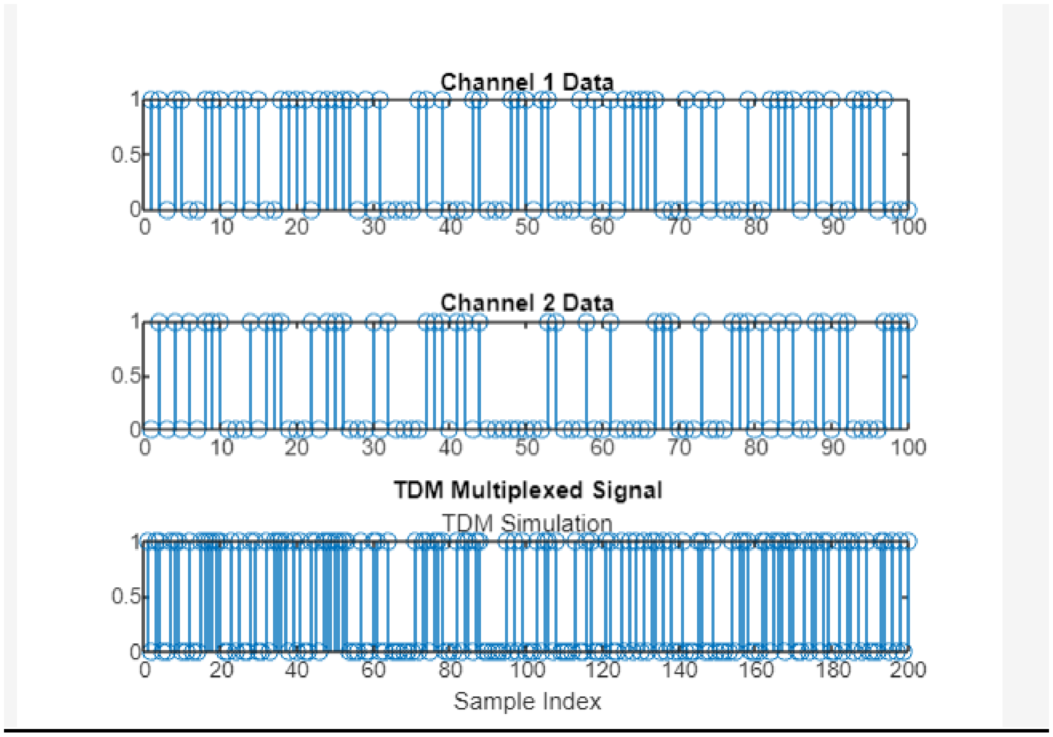
Ans:

Code:

```
% Parameters
num_samples = 100; % Total number of samples
num_channels = 2;  % Number of channels

% Create some example data for two channels
channel1_data = randi([0, 1], 1, num_samples); % Binary data for Channel 1
channel2_data = randi([0, 1], 1, num_samples); % Binary data for Channel 2

% Initialize the TDM multiplexed signal
tdm_signal = zeros(1, num_samples * num_channels);

% Multiplex the data into the TDM signal
for i = 1:num_samples
    tdm_signal((i - 1) * num_channels + 1) = channel1_data(i);
    tdm_signal((i - 1) * num_channels + 2) = channel2_data(i);
end

% Plot the original data for Channel 1
subplot(3, 1, 1);
stem(channel1_data);
title('Channel 1 Data');
```

```
% Plot the original data for Channel 2
subplot(3, 1, 2);
stem(channel2_data);
title('Channel 2 Data');

% Plot the TDM multiplexed signal
subplot(3, 1, 3);
stem(tdm_signal);
title('TDM Multiplexed Signal');

% Label the x-axis
xlabel('Sample Index');

% Add a title to the entire figure
subtitle('TDM Simulation');

% Display the TDM multiplexed signal
disp('TDM Multiplexed Signal:');
disp(tdm_signal);
```

Output:


Channel 1 Data
Channel 2 Data
TDM Multiplexed Signal
TDM Simulation
Sample Index