Experiment No. 2

Aim: To implement CDMA in Java.

Requirements: Compatible version of JDK.

Theory: Code division multiple access (CDMA) is a form of spread spectrum communications that is being used for, among other applications, some second-generation cellular phone systems (others use TDMA). The method is also known as S-CDMA, where the S stands for *synchronous*. S-CDMA is one of the advanced upstream options specified for DOCSIS 2.0.

Figure A illustrates the principle of CDMA transmission. The basic modulation is often QAM[1] of some level. The data to be modulated on each axis is exclusive ORed with a higher-speed pseudorandom bit sequence (PRBS), which is also known as a *spreading code* or *spreading sequence*. The signal is then modulated, normally using m-ary QAM. The effect is to spread out the carrier over a wider bandwidth than necessary for carriage of the information.

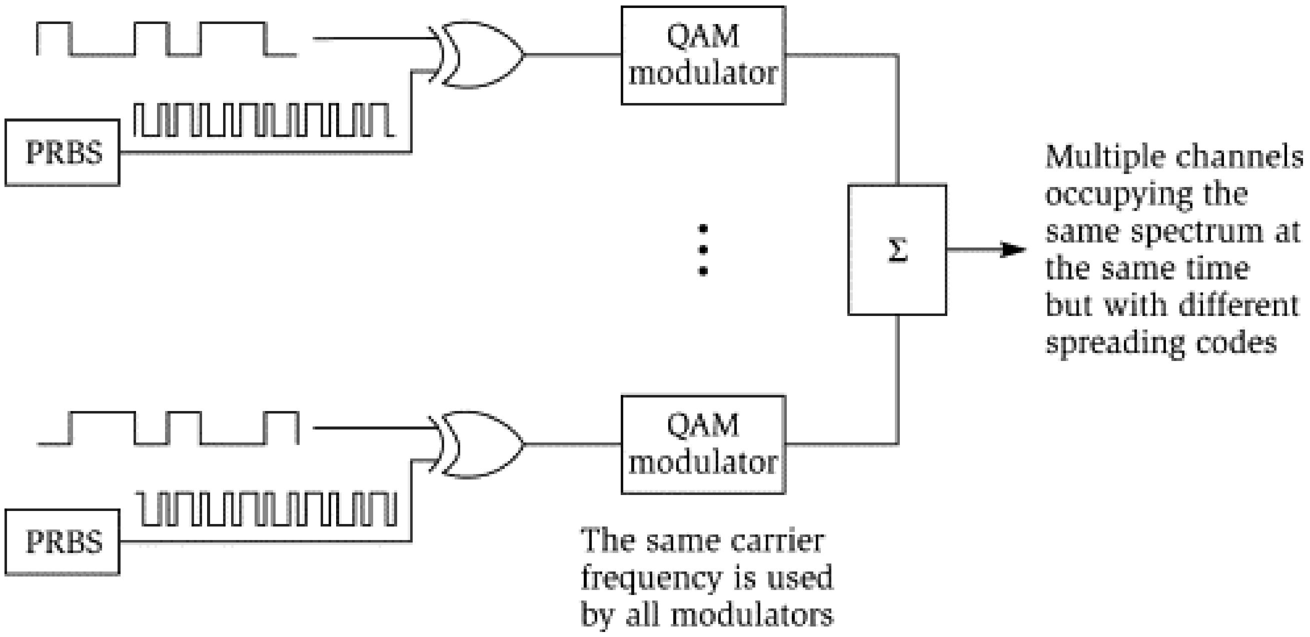

QAM modulator
PRBS
Σ
Multiple channels occupying the same spectrum at the same time but with different spreading codes
QAM modulator
PRBS
The same carrier frequency is used by all modulators

Figure A. Principle of CDMA Transmission.

QAM[1]: Quadrature Amplitude Modulation implements combination of Amplitude and Phase shift keying, standard QAM uses three different amplitudes and 12 different phases

At the receiver, the demodulated signal is again exclusive ORed (EXOR) with the same PRBS. The result is a collapse of the transmission bandwidth back to that required to convey the information signal without the PRBS. Recovery of the original signal can then proceed. In the process, any narrowband interference that is added to the signal during transmission is spread, most of it out of the receive channel. In addition, if an impairment, such as a suck-out, affects part of the channel, the signal quality is minimally degraded, because most of the energy lies outside of the affected portion of the passband. (For the benefit of data communications engineers, the term *suck-out* is used in the cable television industry to mean a narrow frequency-selective reduction in the response of a transmission path.)

Figure A illustrates that many intentional transmissions may occupy the same frequency band simultaneously without interfering with one another, so long as each uses a different spreading sequence. All signals may be received simultaneously, with each having its own unique *despreading* sequence applied to separate it from the other signals. As each signal is separated with its despreading sequence, power in the other signals is spread out as if it were background noise. All spreading codes used must have the mutual (i.e., shared) property of *orthogonality*. That is, they must be noninterfering with each other. The practical limitation on the number of signals that can be transmitted in the same bandwidth is the number of orthogonal spreading codes available. In turn, the number of spreading codes depends on the number of bits in the spreading sequence: The longer the sequence, the more orthogonal codes exist. Spreading code design is a complex subject.

Example: Consider, there are 4 stations using CDMA and they all send the data simultaneously

Station$_1$: $D_1 = 0$, Station$_2$: $D_2 = 0$, Station$_3$: $D_3 = 1$, Station$_4$: $D_4 = 1$.

Since, there are 4 stations we will need 4 orthogonal codes of length 4. We will use Walsh table to generate orthogonal codes.

$W_1 = [0]$, $W_{2(1)} = \begin{bmatrix} W_1 & W_1 \\ W_1 & W_1' \end{bmatrix}$, $W_{2(2)} = W_4 = \begin{bmatrix} W_2 & W_2 \\ W_2 & W_2' \end{bmatrix}$ ----- $\{W'$ represents complement of $W\}$.

Using this recursive procedure we will prepare Walsh table of desired length. We need Walsh table of length 4 i.e. W_4.

$$W_4 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix}$$

In our case we will replace 0 with 1 and 1 with -1 (0 can be replaced with -1 and 1 stay as it is). $[0,1] \to [1,-1]$

$$W_4 = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix}$$

Station$_1$: D_1 = 1, Station$_2$: D_2 = 1, Station$_3$: D_3 = -1, Station$_4$: D_4 = -1.

Providing row entries of table as a chipping code to station:

Station$_1$: $C_1 = [1 \quad 1 \quad 1 \quad 1]$, Station$_2$: $C_2 = [1 \quad -1 \quad 1 \quad -1]$,

Station$_3$: $C_3 = [1 \quad 1 \quad -1 \quad -1]$, Station$_4$: $C_4 = [1 \quad -1 \quad -1 \quad 1]$.

If all stations send data together the channel with have sum of scalar multiple of data and corresponding code.

$D_1C_1 + D_2C_2 + D_3C_3 + D_4C_4 = [1 \quad 1 \quad 1 \quad 1] + [1 \quad -1 \quad 1 \quad -1] + [-1 \quad -1 \quad 1 \quad 1] + [-1 \quad 1 \quad 1 \quad -1]$. – (1)

Consider, if Station$_3$ wants the data of Station$_1$ then it will just take dot product of equation (1) and code C_1.

$D_1 = C_1 \cdot (D_1C_1 + D_2C_2 + D_3C_3 + D_4C_4) = 4 + 0 + 0 + 0 = 4$. Normalizing by length 4 (since, we consider code of length 4) we get, $D_1 = 4/4 = 1 = 0$ (Back to binary form). Value of operation is indeed the data of D_1.

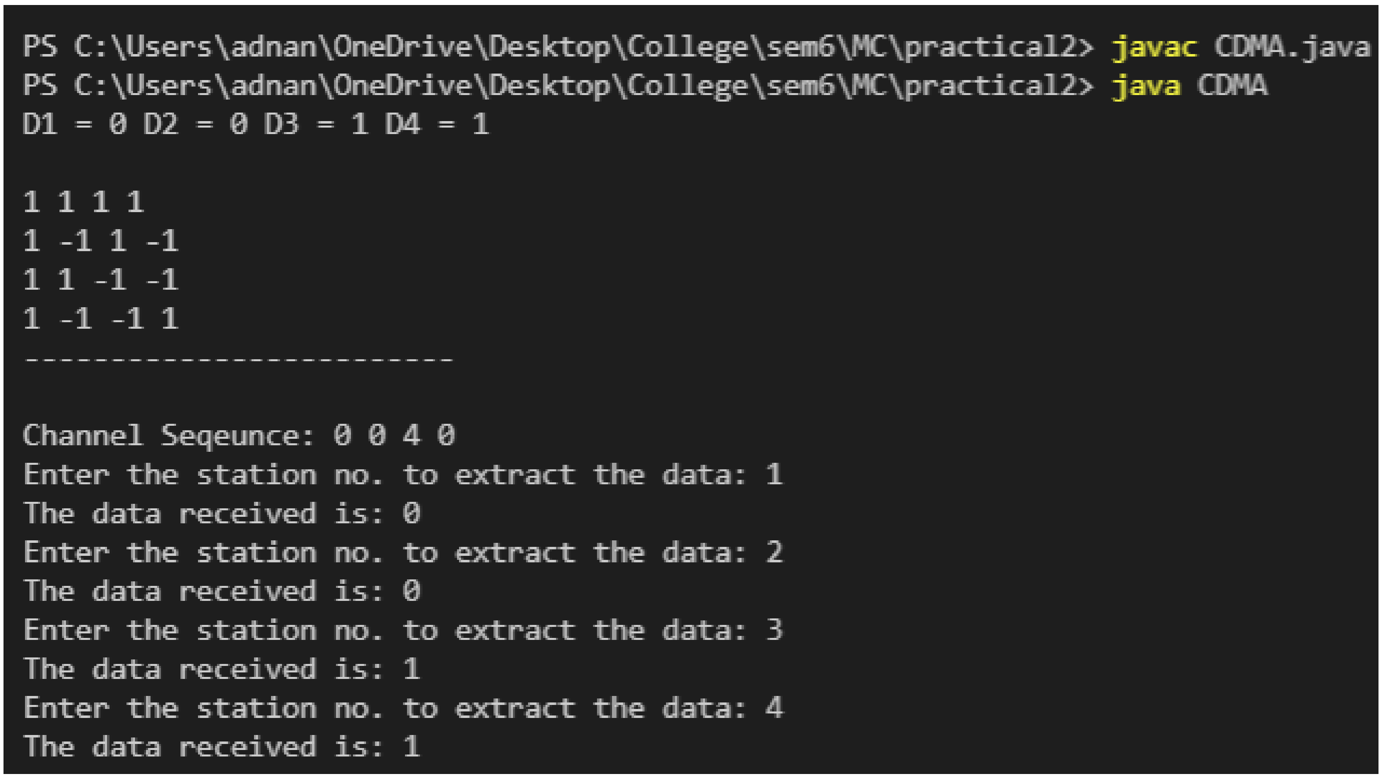
Implementation:

```
import java.util.*;

public class CDMA {

   private int[][] wtable;
   private int[][] copy;
   private int[] channel_sequence;

   public int[] setUp(int[] data, int num_stations)
   {

      wtable = new int[num_stations][num_stations];
      copy = new int[num_stations][num_stations];

      buildWalshTable(num_stations, 0, num_stations - 1, 0,
                              num_stations - 1, false);

      showWalshTable(num_stations);

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

         for (int j = 0; j < num_stations; j++) {

            // Making a copy of walsh table
            // to be used later
            copy[i][j] = wtable[i][j];

            // each row in table is code for one station.
            // So we multiply each row with station data
            wtable[i][j] *= data[i];
         }
      }

      channel_sequence = new int[num_stations];

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

         for (int j = 0; j < num_stations; j++) {
            // Adding all sequences to get channel sequence
            channel_sequence[i] += wtable[j][i];
         }
      }
```

```
    return channel_sequence;
}

public void listenTo(int sourceStation, int num_stations)
{
    int innerProduct = 0;

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

        // multiply channel sequence and source station code
        innerProduct += copy[sourceStation][i] * channel_sequence[i];
    }
    int data = innerProduct / num_stations;
    if(data == -1) data = 1;
    else data = 0;
    System.out.println("The data received is: " + data);
}

public int buildWalshTable(int len, int i1, int i2, int j1,
                                    int j2, boolean isBar)
{
    // len = size of matrix. (i1, j1), (i2, j2) are
    // starting and ending indices of wtable.

    // isBar represents whether we want to add simple entry
    // or complement(southeast submatrix) to wtable.

    if (len == 2) {

        if (!isBar) {

            wtable[i1][j1] = 1;
            wtable[i1][j2] = 1;
            wtable[i2][j1] = 1;
            wtable[i2][j2] = -1;
        }
        else {

            wtable[i1][j1] = -1;
            wtable[i1][j2] = -1;
            wtable[i2][j1] = -1;
            wtable[i2][j2] = 1;
        }

        return 0;
    }

    int midi = (i1 + i2) / 2;
    int midj = (j1 + j2) / 2;

    buildWalshTable(len / 2, i1, midi, j1, midj, isBar);
```

```
        buildWalshTable(len / 2, i1, midi, midj + 1, j2, isBar);
        buildWalshTable(len / 2, midi + 1, i2, j1, midj, isBar);
        buildWalshTable(len / 2, midi + 1, i2, midj + 1, j2, !isBar);

        return 0;
    }

    public void showWalshTable(int num_stations)
    {

        System.out.print("\n");

        for (int i = 0; i < num_stations; i++) {
            for (int j = 0; j < num_stations; j++) {
                System.out.print(wtable[i][j] + " ");
            }
            System.out.print("\n");
        }
        System.out.println("-------------------------");
        System.out.print("\n");
    }

    // Driver Code
    public static void main(String[] args)
    {
        int num_stations = 4;
        int[] channel_sequence;
        int[] data = new int[num_stations];
        //data bits corresponding to each station
        data[0] = 0;
        data[1] = 0;
        data[2] = 1;
        data[3] = 1;

        System.out.println("D1 = "+data[0]+ " D2 = "+data[1]+" D3 = "+data[2]+" D4 = "+data[3]);
        for(int i=0;i<4;i++) if(data[i]==0) data[i] = 1; else data[i] = -1;
        CDMA channel = new CDMA();

        channel_sequence = channel.setUp(data, num_stations);
        System.out.print("Channel Seqeunce: ");
        for(int C: channel_sequence) System.out.print(C+" ");
        System.out.println();

        while(true){
            System.out.print("Enter the station no. to extract the data: ");
            // station you want to listen to
            int sourceStation = new Scanner(System.in).nextInt() - 1;
            channel.listenTo(sourceStation, num_stations);
        }
    }
}
```

Output:



```
PS C:\Users\adnan\OneDrive\Desktop\College\sem6\MC\practical2> javac CDMA.java
PS C:\Users\adnan\OneDrive\Desktop\College\sem6\MC\practical2> java CDMA
D1 = 0 D2 = 0 D3 = 1 D4 = 1

1 1 1 1
1 -1 1 -1
1 1 -1 -1
1 -1 -1 1
-------------------------

Channel Seqeunce: 0 0 4 0
Enter the station no. to extract the data: 1
The data received is: 0
Enter the station no. to extract the data: 2
The data received is: 0
Enter the station no. to extract the data: 3
The data received is: 1
Enter the station no. to extract the data: 4
The data received is: 1
```

Conclusion: We have understand the concept of CDMA and implemented in JAVA.