EAST WEST UNIVERSITY

Project Report

On

CDMA Implementation in MATLAB

Submitted by:

Adnan Saif Dipto
2018-1-60-157

Ashfaq Ahmed Jadeed
2018-1-60-158

Submitted to:

Md. Mahir Ashhab
Lecturer
Dept. of CSE
East West University

CDMA:

Code Division Multiple Access, CDMA is an innovative use of direct sequence spread spectrum technology used to provide a multiple access scheme for mobile telecommunications and other communication systems like wireless communication.

How CDMA Works:

As we know, CDMA is used so that many users can transmit and receive data at the same time. For that we use code. Each user is given a code, and using that code user can transmit that data along with other user. At transmission data bits of different users marge together and a single data string is transmitted. At the receiving end, the single string data bit is then goes through demodulation and using the same code data bits of each user is generated at the receiver.

Project Description:

In this project, we will implement Code Division Multiple Access (CDMA) using MATLAB.
For this, we will have 8 users. A data string will be given for those 8 users. We will generate orthogonal codes and then map the data string in the code. The mapped data string will be transmitted and then receiver will receive that string and then demodulate with the codes for particular users to generate the original data string which was sent.

So, the first thing we need to know is the orthogonal code and how it is generated. For this, we will be using orthogonal codes. An orthogonal code is a set of sequence in which all pairwise cross correlations are zero. In our project we have used the Walsh Codes.

Walsh Codes:

Walsh codes are the most coming orthogonal codes used in CDMA applications. A set of Walsh Codes of length **n** consists of the n rows and an **n x n Walsh Matrix.** That is, there are **n** codes, each of length **n**. The matrix is defined recursively as follows:

$$W_1 = (0) \qquad W_{2n} = \begin{pmatrix} W_n & W_n \\ W_n & \overline{W}_n \end{pmatrix}$$

Where, **n** is the dimension of the matrix and the overscore denotes the logical NOT of the bits in the matrix. The **Walsh matrix** has the property that every row is orthogonal to every other row and to the logical NOT of every other row.

$W_1 = (0)$

(a) 1×1

$$W_2 = \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}$$

(b) 2×2

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

(c) 4×4

$$W_8 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{bmatrix}$$

(d) 8×8

Fig: Walsh Matrix

The above figure shows the Walsh Matrix of dimensions 2, 4 and 8.

Walsh Code in MATLAB:

```
s = 8;
x=[1 1;1 -1];
z=[];
s=log(s)/log(2);
for i=1:s-1
    z=[x x;x -x];
    x=z;
end
```

Here, in our project we have 8 users. We have created the Walsh matrix matrix of dimension 2 and then using a for-loop, we have created the Walsh matrix of dimension 8. The generated Walsh matrix of dimension 8 is shown below.

```
 1     1     1     1     1     1     1     1
 1    -1     1    -1     1    -1     1    -1
 1     1    -1    -1     1     1    -1    -1
 1    -1    -1     1     1    -1    -1     1
 1     1     1     1    -1    -1    -1    -1
 1    -1     1    -1    -1     1    -1     1
 1     1    -1    -1    -1    -1     1     1
 1    -1    -1     1    -1     1     1    -1
```

This is the orthogonal code which will be used.

Bit Data String and Mapping using Walsh code:

The bit data string is:

```
d=[1 0 0 0 1 1 0 1];
```

Now this bit data string will be mapped using the walsh code we have generated earlier. The MATLAB code for this is shown below.

```
value1 = d(1,1).*z(1,:);
value2 = d(1,2).*z(2,:);
value3 = d(1,3).*z(3,:);
value4 = d(1,4).*z(4,:);
value5 = d(1,5).*z(5,:);
value6 = d(1,6).*z(6,:);
value7 = d(1,7).*z(7,:);
value8 = d(1,8).*z(8,:);
```

The mapped data will look like this-

```
 1     1     1     1     1     1     1     1
 0     0     0     0     0     0     0     0
 0     0     0     0     0     0     0     0
 0     0     0     0     0     0     0     0
 1     1     1     1    -1    -1    -1    -1
 1    -1     1    -1    -1     1    -1     1
 0     0     0     0     0     0     0     0
 1    -1    -1     1    -1     1     1    -1
```

Now this mapped data will be converted into a single data string and then will be transmitted. To convert the data into a single data string, the code is:

```
c1 = sum(transmit,1);
```

The single data string which was transmitted is-

```
4     0     2     2    -2     2     0     0
```

Process at the Receiving End:

At the receiving end, the transmitted data string will be multiplied again with the code given for each user and then, summation of the whole strings for each data will be divided with 8 to yield the original bit string. The codes are given below.

```
r1 = c1.*z(1,:);
r2 = c1.*z(2,:);
r3 = c1.*z(3,:);
r4 = c1.*z(4,:);
r5 = c1.*z(5,:);
r6 = c1.*z(6,:);
r7 = c1.*z(7,:);
r8 = c1.*z(8,:);
```

```
rec1 = sum(r1)/8;
rec2 = sum(r2)/8;
rec3 = sum(r3)/8;
rec4 = sum(r4)/8;
rec5 = sum(r5)/8;
rec6 = sum(r6)/8;
rec7 = sum(r7)/8;
rec8 = sum(r8)/8;
```

So the receiver will display the original bit data string as there was no errors-

```
1     0     0     0     1     1     0     1
```

Conclusion:

CDMA is a very famous multiplexing technique. This was a demonstration of how the CDMA works. For the implementation, we have used MATLAB, to write the codes and show the results. We have cautiously written all the codes and understood the process which was happening at the transmission and receiving end. As there was no error, we have shown the original data string which was transmitted at the receiver.

Full MATLAB Code:

```
clc
close all

s = 8;
x=[1 1;1 -1];
z=[];
s=log(s)/log(2);
for i=1:s-1
    z=[x x;x -x];
    x=z;
end

d=[1 0 0 0 1 1 0 1];

value1 = d(1,1).*z(1,:);
value2 = d(1,2).*z(2,:);
value3 = d(1,3).*z(3,:);
value4 = d(1,4).*z(4,:);
value5 = d(1,5).*z(5,:);
value6 = d(1,6).*z(6,:);
value7 = d(1,7).*z(7,:);
value8 = d(1,8).*z(8,:);

transmit = [value1; value2; value3; value4; value5; value6; value7; value8];

c1 = sum(transmit,1);

r1 = c1.*z(1,:);
r2 = c1.*z(2,:);
r3 = c1.*z(3,:);
r4 = c1.*z(4,:);
r5 = c1.*z(5,:);
r6 = c1.*z(6,:);
r7 = c1.*z(7,:);
r8 = c1.*z(8,:);

rec1 = sum(r1)/8;
rec2 = sum(r2)/8;
rec3 = sum(r3)/8;
rec4 = sum(r4)/8;
rec5 = sum(r5)/8;
rec6 = sum(r6)/8;
rec7 = sum(r7)/8;
rec8 = sum(r8)/8;

receiver =[rec1 rec2 rec3 rec4 rec5 rec6 rec7 rec8];
```