East West University

Department of CSE

Individual Project

CSE 453

Wireless Networking

Summer 2022

Submitted To:

Md. Mahir Ashhab

Lecturer

Department of Computer Science and Engineering

Submitted By:

Adri Saha

ID: 2019-1-60-024

Submission Date: 7 September 2022

Project title: CDMA implementation for 8 users using Walsh code on python

Background

In CDMA, we know first we need chip sequence for each station which want to send data bits.

As we have 8 stations, we can generate the chip sequence for 8 stations by implementing Walsh matrix. We know, there are 2 properties need to fulfill for chip sequence in CDMA.

- First is, multiplication of any 2-chip sequence code must be zero. Example: C1 × C2=0.
- And the other is multiplication of same bit sequence number is equal to the base station number. Example: C1 ×C1= 8 [As there are 8 stations]

If this 1st one property is fulfilled it can be said spreading or chip sequence code codes are orthogonal. Only orthogonal codes can be used to implement CDMA.

Theory

CDMA: CDMA (Code-Division Multiple Access) is a spread spectrum multiplexing technique. In second-generation (2G) and third generation (3G) wireless communications, it refers to any of several protocols. As the name suggests, CDMA is a type of multiplexing that enables several signals to share a single transmission channel, making the best use of the available bandwidth.

Orthogonal code:

Spreading codes, which allow a single data bit to be "spread" over a longer sequence of transmitted bits, are used to implement CDMA. To ensure that the data is accurately "despread" at the receiver, these codes are also referred to as chip sequences. Such codes are called orthogonal codes.

If the product of multiplying two codes together sums to zero when the result is added over a period, the two codes are said to be orthogonal. For instance, multiplying the codes 1 -1 -1 1 and 1 -1 1 -1 results in 1 1 -1 -1, which equals zero.

Uses:

- Essential for achieving the highest levels of user density and efficiency.
- CDMA's effectiveness is usefully increased by using orthogonal codes instead of random ones, which also reduces user interference.

Walsh code: One type of orthogonal code is the Walsh code. The Walsh Code is a collection of spreading codes with favorable auto correlation characteristics but unfavorable cross correlation properties. The fundamental building block of CDMA systems, Walsh codes are utilized to create each of the various channels in CDMA.

Walsh matrix: Making a Walsh matrix produces a Walsh code. Starting with a seed of 0, repeating the 0 horizontally and vertically, and then complementing the 1 diagonally, makes it simple to create Walsh codes.

In CDMA, the seed of Walsh code is +1 and compliment is -1.

Uses:

- Direct sequence spread spectrum (DSSS) systems, like Qualcomm's CDMA, use Walsh codes.
- In order to choose the target frequency for the next hop, they are also utilized in frequency-hopping spread spectrum (FHSS) systems.

Problem statement

In this project, the scenario is there is 8 senders and their corresponding receivers. First, we need to generate orthogonal code for each of them. Then, implement the CDMA method on these codes and show the simulation on python.

The input is the 8 data bits which senders want to send. So, bit string of size 8 for 8 users depicting input of each user at the time slot.

On output I need to show, the resultant channel bits and how multiplex and demultiplex the data bits on each spreading codes. And finally verify the exact data bits is received from the sender or not.

Mathematical explanation

As there are 8 stations, so we need to generate 8*8 Walsh matrix first.

By the theory we know, Walsh matrix is represented by,

$W_{2N} = \begin{vmatrix} Wn & Wn \\ \mathrm{Wn} & \overline{Wn} \end{vmatrix}$

Here, the seed is $W_1 = [+1]$

Likewise, $W_{2*1} = \begin{vmatrix} +1 & +1 \\ +1 & -1 \end{vmatrix}$

As, we need to find W_8. So, N=4 here. 2*4=8.

$$\text{So, } W_{2*1} = \begin{vmatrix} +1 & +1 \\ +1 & -1 \end{vmatrix}$$

$$W_4 = W_{2(2)} = \begin{vmatrix} +1 & +1 & +1 & +1 \\ +1 & -1 & +1 & -1 \\ +1 & +1 & -1 & -1 \\ +1 & -1 & -1 & +1 \end{vmatrix}$$

Similarly, for 8 users, 8 row & column can also be calculated by Walsh matrices.

$$W_8 = W_{2(4)} = \begin{vmatrix} +1 & +1 & +1 & +1 & +1 & +1 & +1 & +1 \\ +1 & -1 & +1 & -1 & +1 & -1 & +1 & -1 \\ +1 & +1 & -1 & -1 & +1 & +1 & -1 & -1 \\ +1 & -1 & -1 & +1 & +1 & -1 & -1 & +1 \\ +1 & +1 & +1 & +1 & -1 & -1 & -1 & -1 \\ +1 & -1 & +1 & -1 & -1 & +1 & -1 & +1 \\ +1 & +1 & -1 & -1 & -1 & -1 & +1 & +1 \\ +1 & -1 & -1 & +1 & -1 & +1 & +1 & -1 \end{vmatrix}$$

So, each row is the spreading code for each 8 users. The spreading code for each row corresponds to eight users. Every row on the channel represents different station's chip sequence.

So now generated 8 stations are:

Station 1: $+1 \quad +1 \quad +1 \quad +1 \quad +1 \quad +1 \quad +1 \quad +1$

Station 2: $+1 \quad -1 \quad +1 \quad -1 \quad +1 \quad -1 \quad +1 \quad -1$

Station 3: $+1 \quad +1 \quad -1 \quad -1 \quad +1 \quad +1 \quad -1 \quad -1$

Station 4: $+1 \quad -1 \quad -1 \quad +1 \quad +1 \quad -1 \quad -1 \quad +1$

Station 5: $+1 \quad +1 \quad +1 \quad +1 \quad -1 \quad -1 \quad -1 \quad -1$

Station 6: $+1 \quad -1 \quad +1 \quad -1 \quad -1 \quad +1 \quad -1 \quad +1$

Station 7: $+1 \quad +1 \quad -1 \quad -1 \quad -1 \quad -1 \quad +1 \quad +1$

Station 8: $+1 \quad -1 \quad -1 \quad +1 \quad -1 \quad +1 \quad +1 \quad -1$

Now, these station needs to send data bit. If a station wants to send 1 bit, data bits are encoded as +1. If a station wants to send 0-bit, data bits are encoded as -1. And if the station is idle with no signal, data bits are encoded as 0.

Data bits or signal	1	0	silent
Encoded bit	+1	-1	0

So, now we must input the data bits for each station. Let's send these above data bits:

Stations	Data bit	bits to be multiplied by chip sequence
Station 1	1	+1
Station 2	1	+1
Station 3	1	+1
Station 4	Silent	0
Station 5	1	+1

Station 6	0	-1
Station 7	1	+1
Station 8	Silent	0

Now, each station will multiply its chip sequence with data in order to transmit data to a shared channel.

Multiplexing with data bits:

Stations	Spreading code (C)	Data(D)	Resultant Code (R=C×D)
Station 1	[+1 +1 +1 +1 +1 +1 +1 +1]	+1	[+1 +1 +1 +1 +1 +1 +1 +1]
Station 2	[+1 −1 +1 −1 +1 −1 +1 −1]	+1	[+1 -1 +1 -1 +1 -1 +1 -1]
Station 3	[+1 +1 −1 −1 +1 +1 −1 −1]	+1	[+1 +1 -1 -1 +1 +1 -1 -1]
Station 4	[+1 −1 −1 +1 +1 −1 −1 +1]	0	[0 0 0 0 0 0 0 0]
Station 5	[+1 +1 +1 +1 −1 −1 −1 −1]	+1	[+1 +1 +1 +1 -1 -1 -1 -1]
Station 6	[+1 −1 +1 −1 −1 +1 −1 +1]	-1	[-1 +1 -1 +1 +1 -1 +1 -1]
Station 7	[+1 +1 −1 −1 −1 −1 +1 +1]	+1	[+1 +1 -1 -1 -1 -1 +1 +1]
Station 8	[+1 −1 −1 +1 −1 +1 +1 −1]	0	[0 0 0 0 0 0 0 0]

Now, we get resultant data on channel by summing all the resultant code. If all stations send data at the same time, then we will get this resultant on common channel.

After multiplexing resultant data on channel, R= C_1D_1+ C_2D_2+ C_3D_3+ C_4D_4+ C_5D_5+ C_6D_6+ C_7D_7+ C_8D_8

= [(+1+1+1+0+1-1+1+0),(+1-1+1+0+1+1+1+0), (+1+1-1+0+1-1-1+0), (+1-1-1+1+1-1+0), (+1+1+1+0-1+1-1+0), (+1-1+1+0-1-1-1+0), (+1+1-1+0-1+1+1+0), (+1-1-1+0-1-1+1+0)]

= [4, 4, 0, 0, 2, -2, 2, -2] = resultant channel

Demultiplexing:

To demultiplex, the code for each station will be multiplied with the data on resultant common channel and then divided with the number of stations.

$$\text{Like, } D_i = \frac{Ci * Resultant\ channel\ code}{Number\ of\ stations}$$

Here, number of stations= 8 [For 8 users = 8 stations]

For station 1:

$D_1 = \frac{[+1\ +1\ +1\ +1\ +1\ +1\ +1\ +1] \times [4\ 4\ 0\ 0\ 2\ -2\ 2\ -2]}{8} = \frac{8}{8}$ = +1 which is 1 bit

For station 2:

$D_2 = \frac{[+1\ -1\ +1\ -1\ +1\ -1\ +1\ -1] \times [4\ 4\ 0\ 0\ 2\ -2\ 2\ -2]}{8} = \frac{8}{8}$ = +1 which is 1 bit

For station 3:

$$D_3 = \frac{[+1\ \ +1\ -1\ -1\ \ +1\ +1\ -1\ -1] \times [4\ 4\ 0\ 0\ 2\ -2\ 2\ -2]}{8} = \frac{8}{8} = +1 \text{ which is 1 bit}$$

For station 4:

$$D_4 = \frac{[+1\ -1\ -1\ \ +1\ \ +1\ -1\ -1\ +1] \times [4\ 4\ 0\ 0\ 2\ -2\ 2\ -2]}{8} = \frac{0}{8} = 0 \text{ which is silent}$$

For station 5:

$$D_5 = \frac{[+1\ \ +1\ \ +1\ \ +1\ -1\ -1\ -1\ -1] \times [4\ 4\ 0\ 0\ 2\ -2\ 2\ -2]}{8} = \frac{8}{8} = +1 \text{ which is 1 bit}$$

For station 6:

$$D_6 = \frac{[+1\ -1\ \ +1\ -1\ -1\ \ +1\ -1\ \ +1] \times [4\ 4\ 0\ 0\ 2\ -2\ 2\ -2]}{8} = \frac{-8}{8} = -1 \text{ which is 0 bit}$$

For station 7:

$$D_7 = \frac{[+1\ \ +1\ -1\ -1\ -1\ -1\ \ +1\ \ +1] \times [4\ 4\ 0\ 0\ 2\ -2\ 2\ -2]}{8} = \frac{8}{8} = +1 \text{ which is 1 bit}$$

For station 8:

$$D_8 = \frac{[+1\ -1\ -1\ \ +1\ -1\ \ +1\ \ +1\ -1] \times [4\ 4\ 0\ 0\ 2\ -2\ 2\ -2]}{8} = \frac{0}{8} = 0 \text{ which is silent}$$

DE multiplication complete. Now check these bits with the previously sending bits. We see, both data bits for each station are same. That means, data bits are successfully receiving by sender. So, our CDMA (Code division multiply Access) process is complete.

Implementation

Input: In this project, on user interface we need to input the 8 data bits for 8 specific user which want to be sent by CDMA technique. The messages or 8-bit chip sequences will be auto generated by Walsh code.

Output: From this project we can find step by step simulation of CDMA.

Code explanation: First, I set the number of stations which is 8 for 8 users. Here, n=3; As $2^n = 2^3 = 8$ = number of stations. To generate orthogonal code, I set the seed +1. Seq_mat is a 2-dimensional array. In python which is list of lists to generate the Walsh matrix.

```
no_of_stations=8 #no_of_stations= 2^n = 2^3=8
print('Number of stations:',no_of_stations)
seed=1
n=3
seq_mat=[[seed]] #taking 2 dimentional matrix
```

```
Number of stations: 8
```

Now, I generated Walsh matrix for 8 users and here is the code.

```
#using walsh code to generate chip sequence
#generating walsh matrix
i=1
while(i<=n):
    mat_dim=2**i
    rows, cols = (mat_dim, mat_dim)
    mat = [[0 for i1 in range(cols)] for i2 in range(rows)]

    #top left
    r=0 #on top left starts from row=0
    rt=0
    while(r<mat_dim//2): #loop for filling top left 4 row slots
        c=0 ##on top left starts from col=0
        ct=0
        while(c<mat_dim//2):  #loop for filling top left 4 col slot
            mat[r][c]=seq_mat[rt][ct]
            ct+=1
            c+=1
        rt+=1
        r+=1
```

This is for the top left part where rt and ct represents the recurrent matrix which is every time coming from previous matrix. Here r & c represents the positions of row and column of the matrix. Rest explanations are given on code by commenting out.

On a same way, for top right part of the matrix,

```
#top Right
r=0
rt=0
while(r<mat_dim//2): #for top right, row same but col change
    c=((mat_dim-1)//2)+1
    ct=0
    while(c<mat_dim):
        mat[r][c]=seq_mat[rt][ct]
        ct+=1
        c+=1
    rt+=1
    r+=1
```

For bottom left part of the matrix,

```
#Bottom left
r=((mat_dim-1)//2)+1 #for bottom left, row increases but col starts from 0
rt=0
while(r<mat_dim):
    c=0
    ct=0
    while(c<mat_dim//2):
        mat[r][c]=seq_mat[rt][ct]
        ct+=1
        c+=1
    rt+=1
    r+=1
```

For bottom right part of the matrix,

```
#bottom right
r=((mat_dim-1)//2)+1 #for bottom right, row same and col starts from next division part
rt=0
while(r<mat_dim):
    c=((mat_dim-1)//2)+1
    ct=0
    while(c<mat_dim):
        mat[r][c]=-1*seq_mat[rt][ct] #negate part
        ct+=1
        c+=1
    rt+=1
    r+=1

seq_mat=mat.copy() #filling walsh matrix by copping one by one
i+=1
```

Printing the Walsh matrix with order of 8:

```
#printing chip sequence code
i=1
for item in seq_mat:
    print('Station '+str(i)+'-> '+str(item))
    i+=1
print('\n')
```

```
Station 1-> [1, 1, 1, 1, 1, 1, 1, 1]
Station 2-> [1, -1, 1, -1, 1, -1, 1, -1]
Station 3-> [1, 1, -1, -1, 1, 1, -1, -1]
Station 4-> [1, -1, -1, 1, 1, -1, -1, 1]
Station 5-> [1, 1, 1, 1, -1, -1, -1, -1]
Station 6-> [1, -1, 1, -1, -1, 1, -1, 1]
Station 7-> [1, 1, -1, -1, -1, -1, 1, 1]
Station 8-> [1, -1, -1, 1, -1, 1, 1, -1]
```

So, these are the chip sequences.

Now each user sends specific data bits which is taken as input from user.

```
#data bits
_1bit=1
_0bit=-1
Silent=0
#station_data_bits=[_1bit, _1bit, _1bit, Silent, _1bit, _0bit, _1bit, Silent]
#station_data_bits=[1, 1, 1, 0, 1, -1, 1, 0]
rows, cols = (no_of_stations, no_of_stations)

data_bits = []
for i in range(0, 8):
    bit = int(input('Enter data bits: '))
    data_bits.append(bit)
print('Station data bits are: ',data_bits)
```

```
Enter data bits: 1
Enter data bits: 1
Enter data bits: 1
Enter data bits: 0
Enter data bits: 1
Enter data bits: -1
Enter data bits: 1
Enter data bits: 0
Station data bits are:  [1, 1, 1, 0, 1, -1, 1, 0]
```

Result analysis:

To get result, first I write the code for multiplexing spreading code with given data bits.

```
#Multiplexing (c1*d1....)
mux = [[0 for i1 in range(cols)] for i2 in range(rows)]
i=0
for data in data_bits:
    j=0
    for code in seq_mat[i]:
        mux[i][j]=np.multiply(code,data)
        j+=1
    i+=1

print('After Multiplexing, Resultant Code:')
i=1
for item in mux:
    print('Station '+str(i)+'-> '+str(item))
    i+=1
print('\n')
```

```
After Multiplexing, Resultant Code:
Station 1-> [1, 1, 1, 1, 1, 1, 1, 1]
Station 2-> [1, -1, 1, -1, 1, -1, 1, -1]
Station 3-> [1, 1, -1, -1, 1, 1, -1, -1]
Station 4-> [0, 0, 0, 0, 0, 0, 0, 0]
Station 5-> [1, 1, 1, 1, -1, -1, -1, -1]
Station 6-> [-1, 1, -1, 1, 1, -1, 1, -1]
Station 7-> [1, 1, -1, -1, -1, -1, 1, 1]
Station 8-> [0, 0, 0, 0, 0, 0, 0, 0]
```

So, we got the resultant code for each station. Now, to find the common resultant channel:

```
channel=[0,0,0,0,0,0,0,0]
i=0
while i<no_of_stations:
    j=0
    while j<no_of_stations:
        channel[i]=channel[i]+mux[j][i]
        j+=1
    i+=1
print('After Multiplexing, we get data on channel: '+str(channel))
```

```
After Multiplexing, we get data on channel: [4, 4, 0, 0, 2, -2, 2, -2]
```

Now, start demultiplexing to find out if the same bits have been received to receiver or not:

```
#Demultiplexing
demux=[]
print('Demultiplexed value for:')
st_no=1 #to define string numbering
result = []
print('After demultiplexed bits are: ')
for station in seq_mat:
    i=0
    add=0
    while i<no_of_stations:
        add=add+station[i]*channel[i]
        i+=1
    res=add/8
    result.append(res)

    if res==1:
        val='1 bit'
    elif res==-1:
        val='0 bit'
    elif res==0:
        val='Silent'
    else:
        val='Error'
    demux.append(val)
    print('Station '+str(st_no)+'-> '+val)
    st_no+=1
```

After demultiplexing I got from the code,

```
Demultiplexed value for:
After demultiplexed bits are:
Station 1-> 1 bit
Station 2-> 1 bit
Station 3-> 1 bit
Station 4-> Silent
Station 5-> 1 bit
Station 6-> 0 bit
Station 7-> 1 bit
Station 8-> Silent
```

Lastly, verify if the sender bits and receiver bits are same:

```
#verify part
bit_transfer=0
for i in range(0, 8):
    if(result[i]==data_bits[i]):
        bit_transfer=1
if(bit_transfer==1):
    print('Successfully sent correct bits!')
else:
    print('Error in transfering bits!')
print('\n')
```

```
Successfully sent correct bits!
```

Discussion

Above analysis, I found sender and receiver data bits are same. So, we can say messages has been sent successfully by CDMA. I gave other input bits as well and got successful in both receiving and sending. So, we can say, the CDMA simulation is completed using python but no auto generated library.

Conclusion

First, I want to thank to our honorable faculty Mahir Ashrab sir to give us this excellent, problematic, and analytic simulation task as project to explore ourselves. Power control is used by CDMA systems to reduce interference and noise, which enhances the network's performance. So, implementing CDMA with generating Walsh code in python is a very effective problem for us. To implement CDMA, I could generate orthogonal code by using tile library of Numpy module also. But I generated Walsh matrix manually for better understanding here. Finally, we can show the resultant channel and ensure data bits signals has been correctly sent by CDMA.