Mohiuddin Mondal
Roll- 001910501043 (A2)

Computer Networks
3rd year 1st Sem BCSE UG

Assignment 4

Problem statement: Implement CDMAwith Walsh code

Description: In this assignment you have to implement CDMA for multiple access of a common channel by nstations. Each sender uses a unique code word, given by the Walsh set, to encode its data, send it across the channel, and then perfectly reconstruct the data at n stations.

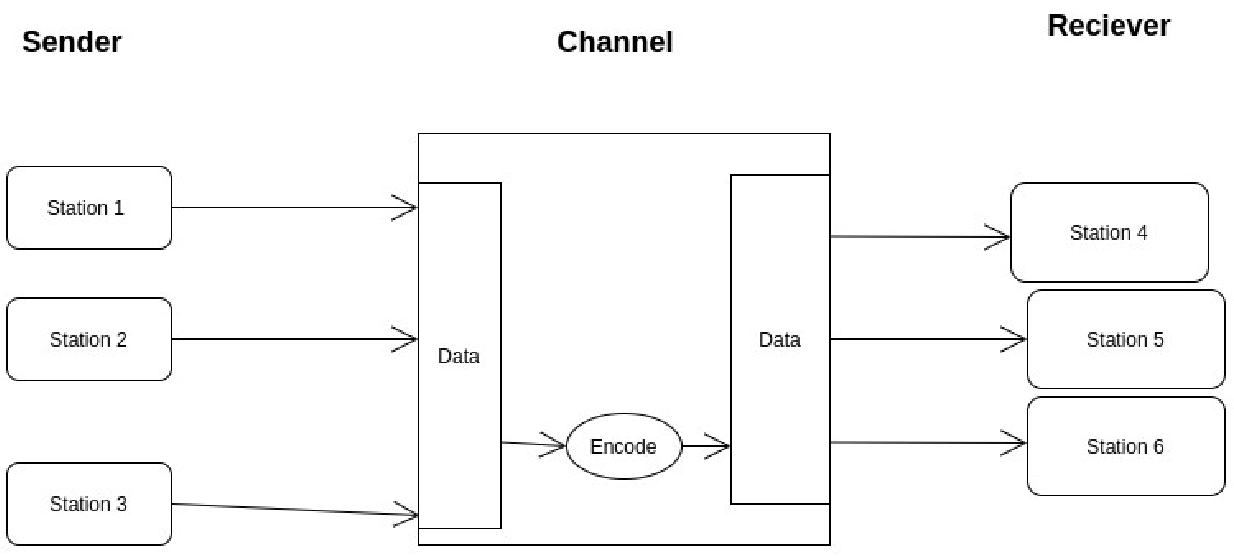
Design: A station model, cdma model is implemented and the driver code acts as the channel here.

- *Station*: It can be both sender or reciever. If it’s sender, it has a 8 bit data (generateed randomly) . Otherwise, it listens to channel and generated the data as recieved from sender and then logs it.
- *Channel*: It’s the main driver code. It first decides total number of stations and initialises CDMA (creates the walsh table). All stations are mentioned in a list. Codeword from walsh table is assigned according to their index number.
- *CDMA*: It generates and stores walsh table. It has support to encode channel data and decode for certain station.
-

N.B. To avoid synchronisation complexity, stations are not spawned as separate thread.

Input/Output : One bit data is sent at a time from all the station and a total of 8 bit data is sent. Sender and reciever source address are pre-determined by index.

Diagram:



Implementation:

driver.cpp :

It’s the main driver code or that implements the channel:

```
int numberOfStations = RS_GAP*2;

    CDMA_FLOW cdma(numberOfStations);
    vector<Station> stations;
    for(int i=0;i<numberOfStations;i++){
        if(i<RS_GAP-1){
            stations.push_back(Station(i,false));
        }
        else stations.push_back(Station(i, true));
    }

    for(int i=0;i<DATA_SIZE;i++){
        vector<int> channelRawData;
        for(int i=0;i<RS_GAP;i++)
            channelRawData.push_back(stations[i].getData());

        vector<int> encodedData = cdma.encodeData(channelRawData);

        for(int i=RS_GAP;i<numberOfStations;i++)
            stations[i].recieveData(cdma,encodedData);
    }
```

CDMA class:

Following is the implementation of cdma class.

```
class CDMA_FLOW{
    vector<vector<int>> walshCalculationBoard;
    vector<vector<int>> walshTable;
    int stationCounts;

public:
    CDMA_FLOW(int stationCounts):walshCalculationBoard(stationCounts,
vector<int>(stationCounts, 0)),
        walshTable(stationCounts, vector<int>(stationCounts, 0)),
stationCounts(stationCounts)
    {
        generateWalshTable(stationCounts, 0, stationCounts - 1, 0,
            stationCounts - 1, false);

        showWalshTable(stationCounts);
    }

    vector<int> encodeData(vector<int> data){
        for (int i = 0; i < stationCounts; i++){
            for (int j = 0; j < stationCounts; j++){
                walshCalculationBoard[i][j] = walshTable[i][j] * data[i];
            }
        }

        vector<int> channelData = vector<int>(stationCounts, 0);

        for (int i = 0; i < stationCounts; i++)
            for (int j = 0; j < stationCounts; j++)
                channelData[i] += walshCalculationBoard[j][i];

        return channelData;
    }

    int getDataOfStation(int sourceStationId,vector<int> channelData){
        int innerProduct = 0;

        for (int i = 0; i < stationCounts; i++) {
            innerProduct += walshTable[sourceStationId][i] * channelData[i];
        }

        return (innerProduct / stationCounts);
    }


    int generateWalshTable(int len, int i1, int i2, int j1,
        int j2, bool isBar)
    {
        if (len == 2) {

            if (!isBar) {

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

            return 0;
        }

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

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

        return 0;
    }

    void showWalshTable(int stationCounts)
    {

        cout<<"\n";

        for (int i = 0; i < stationCounts; i++) {
            for (int j = 0; j < stationCounts; j++) {
                cout << walshTable[i][j] << " ";
            }
            cout<<endl;
        }
        cout<<"-------------------------\n";
        cout<<endl;
    }

};
```

Station:

Following is the implementation of Stations:

```
class Station{
    const int stationId;
    string data;

public:
    Station(int index,bool willRecieve):stationId(index){
        if(willRecieve) data="";
        else{
            data = randomData();
        }
        cout<<"Station Id: "<<index<<"  Generated data to be sent: "<<data<<"\n";

    }

    int getData(){
        if (data.length() > 0){
            int r;
            if (data[0] == '0') r = -1;
            else r = 1;

            data = data.substr(1);
            return r;
        }
        else return 0;
    }

    void recieveData(CDMA_FLOW& cdma, vector<int> encodedData){
        int ch = cdma.getDataOfStation(stationId - RS_GAP, encodedData);
        if(ch>0) data+='1';
        else if(ch<0) data += '0';
        else data+='N';

        cout<<"StationId: "<<stationId<<"\t recieved,total: "<<data<<"\n";
    }

};
```

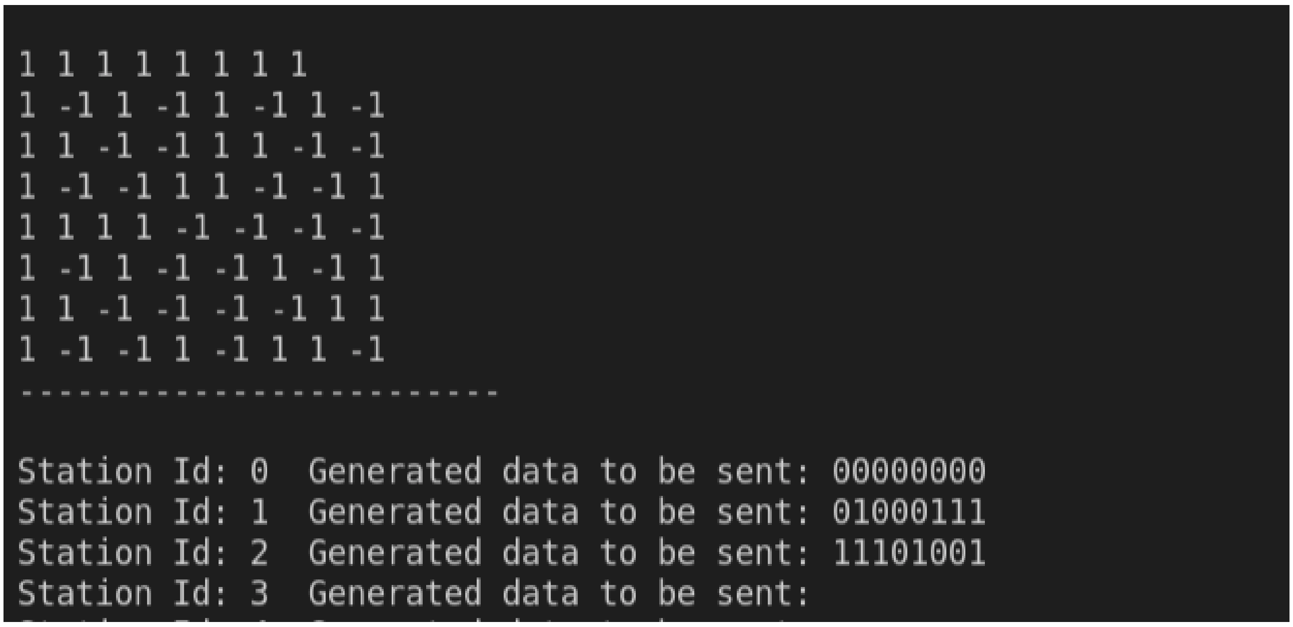
Test cases:

Each sender nodes sends generates a 8 bit data randomly. That is sent bit by bit through the channel. First 4 nodes acts as a sender though 4 th nodes does'nt sends anything.

Reuslts and Analysis:

Results:

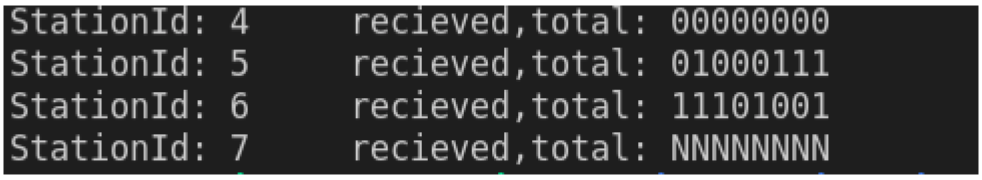
Following screenshot shows initial walsh table and sender side data.



```
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
------------------------

Station Id: 0  Generated data to be sent: 00000000
Station Id: 1  Generated data to be sent: 01000111
Station Id: 2  Generated data to be sent: 11101001
Station Id: 3  Generated data to be sent:
```

Following screen shot shows the recieved data. If sender doesn't send any data, "N" is printed.



```
StationId: 4     recieved,total: 00000000
StationId: 5     recieved,total: 01000111
StationId: 6     recieved,total: 11101001
StationId: 7     recieved,total: NNNNNNNN
```