Department of Computer Engineering

Academic Term: Jan-May 2021

**Class : ** T.E IT Sem -VI
Subject: Sensor Network Lab (ITL604)

Practical No:	**5**
Title:	To implement Code Division Multiple Access (CDMA)
Date of Performance:	
Date of Submission:	
Roll No:	8669
Name of the Student:	Mareena Mark Fernandes

Evaluation:

Sr. No	Rubric	Grade
1	**On time Completion & Submission (2)**	
2	**Output (3)**	
3	**Code Optimization (3)**	
4	**Knowledge of the topic (2)**	
5	**Total (10)**	

Signature of the Teacher :

PRACTICAL - 5

Title: : To implement Code Division Multiple Access (CDMA)

Objective: To study code division Multiplexing.

Reference: Mobile communication by Schiller, Mobile Computing by RajKamal

Prerequisite: Knowledge of orthogonal codes and Code Division Multiplexing.

Description:

Code Division Multiple Access (CDMA) is a method of multiplexing that does not divide a channel by time as in TDMA or frequency as in FDMA. Instead all active users use the same frequency at the same time. Separation of channels is now achieved by code . This scheme encodes data using special code associated with each channel called chipping sequence (or Pseudo random Noise sequence). The codes used here are orthogonal and has good auto-correlation property.

CDMA multiples the data being transmitted by a "noise" signal (chipping sequence). This noise signal is a pseudo random sequence of 1 and −1 values, at a frequency much higher than that of the original signal, thereby spreading the energy of the original signal into a much wider band.

De spreading requires the receiver to apply the same PN sequence on the received signal to recover data.

ORTHOGONAL CODES:

Two vectors are said to be orthogonal if their inner product is zero. Consider two vectors (2,0,3) & (3,5,-2). Their inner product is (2*3)+(0*5)+(3*-2)=6+0+(-6)=0. Hence they are orthogonal.
Two codes are orthogonal if following equation is satisfied.

Where ‘n’ is the length if code.
Ex.
Code 1 : 0 1 0 1
Code 2 : 0 1 1 0
Bipolar Code1 : -1 1 -1 1
Bipolar Code2 : -1 1 1 -1

Code1 * Code 2 : 1 1 -1 -1

Sum = (1+1-1-1) = 0
Hence two codes are orthogonal.

WALSH CODES:

- Walsh codes are also known as Walsh Hardmard Codes.
- The Walsh code is a linear code, which maps binary strings of length n to binary codewords of length 2n. Further these codes are mutually orthogonal.
- Walsh codes are most commonly used orthogonal codes in CDMA application.
- Length –> power of 2 (1,2,4,8---)
- Walsh codes are used for spreading in the forward link.

Generation of the Walsh code matrices

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

$$W_{2^n} = \begin{bmatrix} W_{2^{n-1}} & W_{2^{n-1}} \\ W_{2^{n-1}} & \overline{W}_{2^{n-1}} \end{bmatrix}$$

Example of WC sequence generation:

$$W_4 = \begin{bmatrix} W_2 & W_2 \\ W_2 & \overline{W_2} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix}$$

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

- Code is given as a row in WC matrix
- To generate a code
  - "0" -> "1"
  - "1" -> "-1"
- Example: Codes $W_{4,2}$ and $W_{4,3}$
  - $W_{8,2}$: (0,0,1,1,0,0,1,1) -> (1,1,-1,-1,1,1,-1,-1)
  - $W_{8,3}$: (0,1,1,0,0,1,1,0) -> (1,-1,-1,1,1,-1,-1,1)

- When synchronized – codes are orthogonal

$$W_{8,2} \cdot W_{8,3} = (1,1,-1,-1,1,1,-1,-1) \cdot (1,-1,-1,1,1,-1,-1,1) = 0$$

- When out of sync – codes are not orthogonal

$$W_{8,2} \cdot \text{shift}\ (W_{8,3},1) = (1,1,-1,-1,1,1,-1,-1) \cdot (1,1,-1,-1,1,1,-1,-1) = 8$$

CDMA Example:

1) Sender A's data Ad = 1 => Bipolar Ad = +1

   Sender B's data Bd = 0 => Bipolar Bd = -1

2) A's Chip code is codeA[ ] : 0 0 1 1 0 0 1 1 =>

   Bipolar conversion is : -1 -1 +1 +1 -1-1 +1 +1

   B's Chip code is codeB[]: 0 1 1 0 0 1 1 0

Bipolar conversion is : -1 +1 +1 -1 -1 +1 +1 -1

3) Spread A's data

As =1 * (-1 -1 +1 +1 -1 -1 +1 +1) = (-1 -1 +1 +1 -1 -1 +1 +1)

Spread B's data

Bs = -1 * (-1 +1 +1 -1 -1 +1 +1 -1) = (+1 -1 -1 +1 +1 -1 -1 +1)

4) Send the sum of As+Bs

Cs = As + Bs = (0 -2 0 2 0 -2 0 2)

5) Recover As Data from received signal Cs

Cs * codeA[ ] = (0 -2 0 2 0 -2 0 2) * (-1 -1 +1 +1 -1-1 +1 +1)

= (0 2 0 2 0 2 0 2)

Sum = 8 > 0 hence A's transmitted data was Ad=1

6) Recover B's Data from received signal Cs

Cs * codeB[ ] = (0 -2 0 2 0 -2 0 2) * (-1 +1 +1 -1 -1 +1 +1 -1)

= (0 -2 0 -2 0 -2 0 -2)

Sum = -8 < 0 hence B's transmitted data was Bd = 0

Algorithm :

1) Start
2) Enter sender A's data : Ad , Convert into bipolar
3) Enter sender B's data : Bd , Convert into bipolar
4) Enter A's PN sequence : codeA[ ] and Convert into bipolar
5) Enter B's PN sequence : codeB[ ] and Convert into bipolar
6) Spread A's data : As[ ] = Ad * codeA[ ]
7) Spread B's data : Bs[ ] = Bd * codeB[ ]
8) Add As[ ] and B[ ] : c[ ] = As[ ] + Bs[ ]
9) De spread A's signals
   ResultA [ ] = c[ ] *
   codeA[ ]Add values of
   ResultA[ ]
   If sum > 0 then A's transmitted data is 1 else 0.
10) De spread B's
   signals ResultB [ ] = c[
   ] * codeB[ ] Add values
   of ResultB[ ]
   If sum > 0 then A's transmitted data is 1 else 0.
11) Stop.

Conclusion: CDMA has been studied.

PostLab assignment:

1. What are advantages of CDMA technology over GSM
2. Compare FDM, TDM ,CDM, and SDM.

CODE:

server.cpp

```
#define WIN32_LEAN_AND_MEAN

#include <iostream> #include <windows.h> #include <winsock2.h> #include <ws2tcpip.h> #include
<stdlib.h> #include <stdio.h>
#pragma comment(lib, "Ws2_32.lib")

#define DEFAULT_BUFLEN 512
#define DEFAULT_PORT "27015"
#define CODE_LEN 4
#define NUM_USERS 2

void printIntArr(int *arr)
{
printf("[ ");
for (int i = 0; i < CODE_LEN; i++)
{
printf("%d ", arr[i]);
}
printf("]\n");
}

void arrCharToInt(int *arr, char *buf, int code_len)
{
for (int i = 0; i < code_len; i++)
{
arr[i] = (int)buf[i];
}
}

void arrIntToChar(int *arr, char *buf, int code_len)
{
for (int i = 0; i < code_len; i++)
{
buf[i] = (char)arr[i];
}
}

int serverResolveError(int error_code)
```

```
{
printf("getaddrinfo failed with error: %d\n", error_code); WSACleanup();
return 1;
}

int socketCreateError(addrinfo *result)
{
printf("socket failed with error: %ld\n", WSAGetLastError()); freeaddrinfo(result);
WSACleanup(); return 1;
}

int bindError(SOCKET socket, addrinfo *result)
{
printf("bind failed with error: %d\n", WSAGetLastError()); freeaddrinfo(result);
closesocket(socket); WSACleanup();
return 1;
}

int listenError(SOCKET socket)
{
printf("listen failed with error: %d\n", WSAGetLastError()); closesocket(socket);
WSACleanup(); return 1;
}

int acceptError(SOCKET socket)
{
printf("accept failed with error: %d\n", WSAGetLastError()); closesocket(socket);
WSACleanup(); return 1;
}

int sendError(SOCKET sockets[])
{
printf("send failed with error: %d\n", WSAGetLastError()); closesocket(sockets[0]);
closesocket(sockets[1]); WSACleanup();
return 1;
}

int recvError(SOCKET sockets[])
{
printf("recv failed with error: %d\n", WSAGetLastError()); closesocket(sockets[0]);
closesocket(sockets[1]); WSACleanup();
return 1;
```

```
}

int dataTransferError(SOCKET sockets[], int code)
{
if (code == -1)
{
return sendError(sockets);
}
else if (code == -2)
{


}
else
{

}
}

return recvError(sockets);

return 1;

int dataTransferIn(SOCKET socket, int *arr)
{
int iResult;
char recvbuf[DEFAULT_BUFLEN];
iResult = recv(socket, recvbuf, DEFAULT_BUFLEN, 0); if (iResult > 0)
arrCharToInt(arr, recvbuf, CODE_LEN); else if (iResult == 0)
return -1;
else
return -2;
return 0;
}

int dataTransferOut(SOCKET socket, int data)
{
int iSendResult;
const char arr[] = {(char)data};
iSendResult = send(socket, arr, sizeof(int), 0); if (iSendResult == SOCKET_ERROR)
return -1;
return 0;
```

```
}


int dataAckOut(SOCKET socket, const char *msg)
{
int iSendResult;
iSendResult = send(socket, msg, strlen(msg), 0); if (iSendResult == SOCKET_ERROR)
return -1;
return 0;
}

void shutdownSockets(SOCKET sockets[])
{
int iResult;
iResult = shutdown(sockets[0], SD_SEND); if (iResult == SOCKET_ERROR)
printf("shutdown failed with error: %d\n", WSAGetLastError()); iResult = shutdown(sockets[1],
SD_SEND);
if (iResult == SOCKET_ERROR)
printf("shutdown failed with error: %d\n", WSAGetLastError());

// cleanup closesocket(sockets[0]); closesocket(sockets[1]); WSACleanup();
}

int initServer(SOCKET ClientSockets[])
{
WSADATA wsaData;
SOCKET ListenSocket = INVALID_SOCKET; struct addrinfo *result = NULL, hints; int iResult;

printf("Socket init in process...\n");
// Initialize Winsock
iResult = WSAStartup(MAKEWORD(2, 2), &wsaData); if (iResult != 0)
{
printf("WSAStartup failed with error: %d\n", iResult); return 1;
}

// Initialize hints attributes to zero and then set socket transmission details
// socketInfoDef(&hints, AF_INET, SOCK_STREAM, IPPROTO_TCP, AI_PASSIVE);
ZeroMemory(&hints, sizeof(hints));
hints.ai_family = AF_INET; hints.ai_socktype = SOCK_STREAM; hints.ai_protocol = IPPROTO_TCP;

hints.ai_flags = AI_PASSIVE;

// Resolve the server address and port
```

```
iResult = getaddrinfo(NULL, DEFAULT_PORT, &hints, &result); if (iResult != 0)
return serverResolveError(iResult);

// Create a SOCKET for connecting to server
ListenSocket = socket(result->ai_family, result->ai_socktype, result->ai_protocol); if (ListenSocket ==
INVALID_SOCKET)
return socketCreateError(result);

// Setup the TCP listening socket
iResult = bind(ListenSocket, result->ai_addr, (int)result->ai_addrlen); if (iResult == SOCKET_ERROR)
return bindError(ListenSocket, result);

freeaddrinfo(result);

printf("Listening for incoming connections...\n"); for (int i = 0; i < NUM_USERS; i++)
{
iResult = listen(ListenSocket, SOMAXCONN); if (iResult == SOCKET_ERROR)
return listenError(ListenSocket);

// Accept a client socket
ClientSockets[i] = accept(ListenSocket, NULL, NULL); if (ClientSockets[i] == INVALID_SOCKET)
return acceptError(ListenSocket);

printf("%d connection(s) received\n", i + 1);
}

printf("All users connected to service!\n");

// No longer need server socket closesocket(ListenSocket); return 0;
}

int cdmaServer()
{

int iResult, iSendResult;

SOCKET ClientSockets[NUM_USERS]; for (int i = 0; i < NUM_USERS; i++)

{
ClientSockets[i] = INVALID_SOCKET;
}
```

```
char recvbuf[DEFAULT_BUFLEN]; int recvbuflen = DEFAULT_BUFLEN;
int acode[CODE_LEN], bcode[CODE_LEN], aspread[CODE_LEN], bspread[CODE_LEN],
spread_sum[CODE_ LEN];

iResult = initServer(ClientSockets); if (iResult == 1)
return 1;

// Practical specific implementation. Change to a loop for actual implementation

// Recieve unique code from User A and send back confirmation to send spread iResult =
dataTransferIn(ClientSockets[0], acode);
if (iResult < 0)
return dataTransferError(ClientSockets, iResult); iSendResult = dataAckOut(ClientSockets[0], "y");
if (iSendResult < 0)
return dataTransferError(ClientSockets, iSendResult); printf("Client A key code: ");
printIntArr(acode);

// Receive unique code from User B and send back confirmation to send spread iResult =
dataTransferIn(ClientSockets[1], bcode);
if (iResult < 0)
return dataTransferError(ClientSockets, iResult); iSendResult = dataAckOut(ClientSockets[1], "y");
if (iSendResult < 0)
return dataTransferError(ClientSockets, iSendResult); printf("Client B key code: ");
printIntArr(bcode);

// Receive spread code from User A and send back code of User B iResult =
dataTransferIn(ClientSockets[0], aspread);
if (iResult < 0)
return dataTransferError(ClientSockets, iResult); iSendResult = dataAckOut(ClientSockets[0], "y");
if (iSendResult < 0)
return dataTransferError(ClientSockets, iSendResult); printf("Client A spread code: ");
printIntArr(aspread);

// Receive spread code from User B and send back code of User A iResult =
dataTransferIn(ClientSockets[1], bspread);
if (iResult < 0)

return dataTransferError(ClientSockets, iResult); iSendResult = dataAckOut(ClientSockets[1], "y");
if (iSendResult < 0)
return dataTransferError(ClientSockets, iSendResult); printf("Client B spread code: ");
printIntArr(bspread);

// Generate spread sum
```

```
for (int i = 0; i < CODE_LEN; i++)
{
spread_sum[i] = aspread[i] + bspread[i];
}
printf("Spread sum: "); printIntArr(spread_sum);

// Extract user data
int a_data = 0, b_data = 0;
for (int i = 0; i < CODE_LEN; i++)
{
a_data += acode[i] * spread_sum[i]; b_data += bcode[i] * spread_sum[i];
}
a_data = a_data > 0 ? 1 : 0;
b_data = b_data > 0 ? 1 : 0;

// Send data to both users
iSendResult = dataTransferOut(ClientSockets[0], b_data); if (iSendResult < 0)
return dataTransferError(ClientSockets, iSendResult); printf("Sent B data bit to A...\n");
iSendResult = dataTransferOut(ClientSockets[1], a_data); if (iSendResult < 0)
return dataTransferError(ClientSockets, iSendResult); printf("Sent A data bit to B...\n");

// shutdown the connection since we're done shutdownSockets(ClientSockets); printf("Socket shutdown
successful. Bye!"); return 0;
}

int main()
{
int code = cdmaServer(); if (code == 1)
{
printf("Server program failed! Exiting..."); return 1;
```

client.cpp

```
#define WIN32_LEAN_AND_MEAN

#include <iostream> #include <time.h> #include <winsock2.h> #include <windows.h>



#include <ws2tcpip.h> #include <stdlib.h> #include <stdio.h>

#pragma comment(lib, "Ws2_32.lib")

#define DEFAULT_BUFLEN 512
#define DEFAULT_PORT "27015"
#define CODE_LEN 4

#include <stdio.h> #include <cmath>

void printIntArr(int *arr)
{
printf("[ ");
for (int i = 0; i < CODE_LEN; i++)
{
printf("%d ", arr[i]);
}
printf("]\n");
}

void arrCharToInt(int *arr, char *buf, int code_len)
{
for (int i = 0; i < code_len; i++)
{
arr[i] = (int)buf[i];
}
}

void arrIntToChar(int *arr, char *buf, int code_len)
{

for (int i = 0; i < code_len; i++)
{
buf[i] = (char)arr[i];
}
}

void walshGeneration(int *walsh, int final_mat_size)
{
int mat_size = 2;
while (mat_size <= final_mat_size)
{
```

```
bool neg = false; int col_lim = 0;
for (int row = 0; row < mat_size; row++)
{
for (int col = 0; col < mat_size; col++)
{
if (row < (mat_size / 2) && col < (mat_size / 2)) continue;

if (row == col)
{
neg = true; col_lim = col;
}

if (!neg || col < col_lim)
{
if (row > col)
{
walsh[final_mat_size * row + col] = walsh[final_mat_size * (row - (mat_s
ize / 2)) + col];
}
else
{
walsh[final_mat_size * row + col] = walsh[final_mat_size * row + (col -
(mat_size / 2))];
}
}
else
{
if (row > col)
{
walsh[final_mat_size * row + col] = !walsh[final_mat_size * (row - (mat_
size / 2)) + col];
}
else
{

walsh[final_mat_size * row + col] = !walsh[final_mat_size * row + (col -
(mat_size / 2))];
}
}
}
}
mat_size *= 2;
}
}

int serverResolveError(int error_code)
{
printf("getaddrinfo failed with error: %d\n", error_code); WSACleanup();
return 1;
}
```

```
int socketCreateError(addrinfo *result)
{
printf("socket failed with error: %ld\n", WSAGetLastError()); freeaddrinfo(result);
WSACleanup(); return 1;
}

int sendError(SOCKET socket)
{
printf("send failed with error: %d\n", WSAGetLastError()); closesocket(socket);
WSACleanup(); return 1;
}

int recvError(SOCKET socket)
{
printf("recv failed with error: %d\n", WSAGetLastError()); closesocket(socket);
WSACleanup(); return 1;
}

int dataTransferError(SOCKET socket, int code)
{
if (code == -1)
{
return sendError(socket);
}
else if (code == -2)

{

}
else
{

}
}


return recvError(socket);




return 1;


int dataTransferIn(SOCKET socket, int *arr)
{
int iResult;
char recvbuf[DEFAULT_BUFLEN];
iResult = recv(socket, recvbuf, DEFAULT_BUFLEN, 0); if (iResult > 0)
```

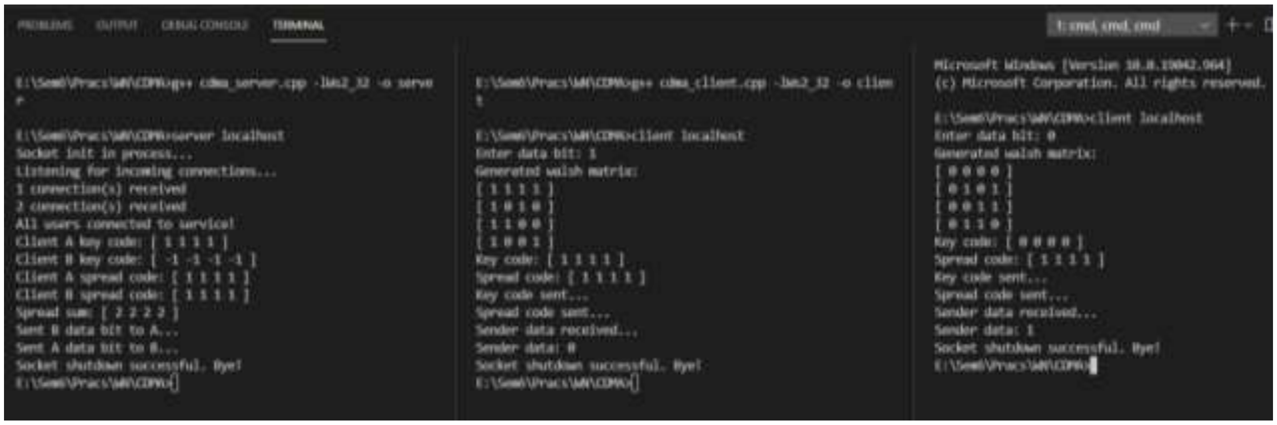
```
arrCharToInt(arr, recvbuf, 1); else if (iResult == 0)
return -1;
else
return -2;
return 0;
}

int dataTransferOut(SOCKET socket, int *arr)
{
int iSendResult; char buf[CODE_LEN];
arrIntToChar(arr, buf, CODE_LEN);
iSendResult = send(socket, buf, sizeof(buf), 0); if (iSendResult == SOCKET_ERROR)
return -1;
return 0;
}

int dataAckIn(SOCKET socket)
{
int iResult;
char recvbuf[DEFAULT_BUFLEN];
iResult = recv(socket, recvbuf, DEFAULT_BUFLEN, 0); if (iResult > 0 && *recvbuf == 'y')
return 0;

else

}



return -1;


void shutdownSocket(SOCKET socket)
{
int iResult;

iResult = shutdown(socket, SD_SEND); if (iResult == SOCKET_ERROR)
{
printf("shutdown failed with error: %d\n", WSAGetLastError());
}

// cleanup closesocket(socket); WSACleanup();
printf("Socket shutdown successful. Bye!");
}

SOCKET initClientSocket(SOCKET ConnectSocket, char *server_name)
{
WSADATA wsaData; int iResult;
struct addrinfo *result = NULL, *ptr = NULL, hints;

// Initialize Winsock
```

```
iResult = WSAStartup(MAKEWORD(2, 2), &wsaData); if (iResult != 0)
{
printf("WSAStartup failed with error: %d\n", iResult); return 1;
}

// Fill memory block with zeros ZeroMemory(&hints, sizeof(hints)); hints.ai_family = AF_UNSPEC;
hints.ai_socktype = SOCK_STREAM; hints.ai_protocol = IPPROTO_TCP;

// Resolve the server address and port
iResult = getaddrinfo(server_name, DEFAULT_PORT, &hints, &result); if (iResult != 0)
return serverResolveError(iResult);

// Attempt to connect to an address until one succeeds for (ptr = result; ptr != NULL; ptr = ptr->ai_next)
{
// Create a SOCKET for connecting to server ConnectSocket = socket(ptr->ai_family, ptr->ai_socktype,
ptr->ai_protocol); if (ConnectSocket == INVALID_SOCKET)
return socketCreateError(result);

// Connect to server.
iResult = connect(ConnectSocket, ptr->ai_addr, (int)ptr->ai_addrlen);

if (iResult == SOCKET_ERROR)
{
closesocket(ConnectSocket); ConnectSocket = INVALID_SOCKET; continue;
}
break;
}

freeaddrinfo(result);

if (ConnectSocket == INVALID_SOCKET)
{
printf("Unable to connect to server!\n"); WSACleanup();
return 1;
}
return ConnectSocket;
}

int cdmaClient(char *server_name)
{
SOCKET ConnectSocket = INVALID_SOCKET; char recvbuf[DEFAULT_BUFLEN];
int iResult;
int recvbuflen = DEFAULT_BUFLEN;
int *walsh = new int[CODE_LEN * CODE_LEN];

int mydata, mycode[CODE_LEN], myspread[CODE_LEN], senderdata[1]; printf("Enter data bit: ");
std::cin >> mydata; walsh[0] = mydata;

if (mydata == 0)
{
```

```
mydata = -1;
}

walshGeneration(walsh, CODE_LEN); printf("Generated walsh matrix: \n"); for (int i = 0; i < CODE_LEN;
i++)
{
printf("[ ");
for (int j = 0; j < CODE_LEN; j++)
{
printf("%d ", walsh[i * CODE_LEN + j]);
}
printf("]\n");

}

srand(time(NULL));
int random_row = rand() % CODE_LEN; printf("Key code: [ ");
for (int col = 0; col < CODE_LEN; col++)
{
mycode[col] = walsh[CODE_LEN * random_row + col]; printf("%d ", mycode[col]);
if (mycode[col] != 0 && mycode[col] != 1)
{
printf("Invalid data format!"); return 1;
}
}
printf("]\n");

for (int i = 0; i < CODE_LEN; i++)
{
if (mycode[i] == 0)
{
mycode[i] = -1;
myspread[i] = mydata * mycode[i];
}
else
{
myspread[i] = mydata * mycode[i];
}
}
printf("Spread code: "); printIntArr(myspread);

ConnectSocket = initClientSocket(ConnectSocket, server_name); if (ConnectSocket ==
INVALID_SOCKET)
return 1;

iResult = dataTransferOut(ConnectSocket, mycode); if (iResult < 0)
return dataTransferError(ConnectSocket, iResult); printf("Key code sent...\n");

iResult = dataAckIn(ConnectSocket); if (iResult != 0)
{
```

```
printf("Bad acknoledgement"); return 1;
}

iResult = dataTransferOut(ConnectSocket, myspread); if (iResult < 0)
return dataTransferError(ConnectSocket, iResult); printf("Spread code sent...\n");
iResult = dataAckIn(ConnectSocket); if (iResult != 0)
{
printf("Bad ack"); return 1;
}

iResult = dataTransferIn(ConnectSocket, senderdata); if (iResult < 0)
return dataTransferError(ConnectSocket, iResult); printf("Sender data received...\n");

printf("Sender data: %d\n", senderdata[0]);

// shutdown the connection since no more data will be sent shutdownSocket(ConnectSocket);

return 0;
}

int main(int argc, char **argv)
{
// Validate the parameters if (argc != 2)
{
printf("Using Server: %s\n", argv[0]); return 1;
}

int code = cdmaClient(argv[1]); if (code == 1)
{
printf("Client program failed"); return 1;
}

return 0;
}
```

OUTPUT:



```
E:\Sem6\Pracs\WN\CDMA>g++ cdma_server.cpp -lws2_32 -o serve
r

E:\Sem6\Pracs\WN\CDMA>server localhost
Socket init in process...
Listening for incoming connections...
1 connection(s) received
2 connection(s) received
All users connected to service!
Client A key code: [ 1 1 1 1 ]
Client B key code: [ -1 -1 -1 -1 ]
Client A spread code: [ 1 1 1 1 ]
Client B spread code: [ 1 1 1 1 ]
Spread sum: [ 2 2 2 2 ]
Sent B data bit to A...
Sent A data bit to B...
Socket shutdown successful. Bye!
E:\Sem6\Pracs\WN\CDMA>
```

```
E:\Sem6\Pracs\WN\CDMA>g++ cdma_client.cpp -lws2_32 -o clien
t

E:\Sem6\Pracs\WN\CDMA>client localhost
Enter data bit: 1
Generated walsh matrix:
[ 1 1 1 1 ]
[ 1 0 1 0 ]
[ 1 1 0 0 ]
[ 1 0 0 1 ]
Key code: [ 1 1 1 1 ]
Spread code: [ 1 1 1 1 ]
Key code sent...
Spread code sent...
Sender data received...
Sender data: 0
Socket shutdown successful. Bye!
E:\Sem6\Pracs\WN\CDMA>
```

```
Microsoft Windows [Version 10.0.19042.964]
(c) Microsoft Corporation. All rights reserved.

E:\Sem6\Pracs\WN\CDMA>client localhost
Enter data bit: 0
Generated walsh matrix:
[ 0 0 0 0 ]
[ 0 1 0 1 ]
[ 0 0 1 1 ]
[ 0 1 1 0 ]
Key code: [ 0 0 0 0 ]
Spread code: [ 1 1 1 1 ]
Key code sent...
Spread code sent...
Sender data received...
Sender data: 1
Socket shutdown successful. Bye!
E:\Sem6\Pracs\WN\CDMA>
```

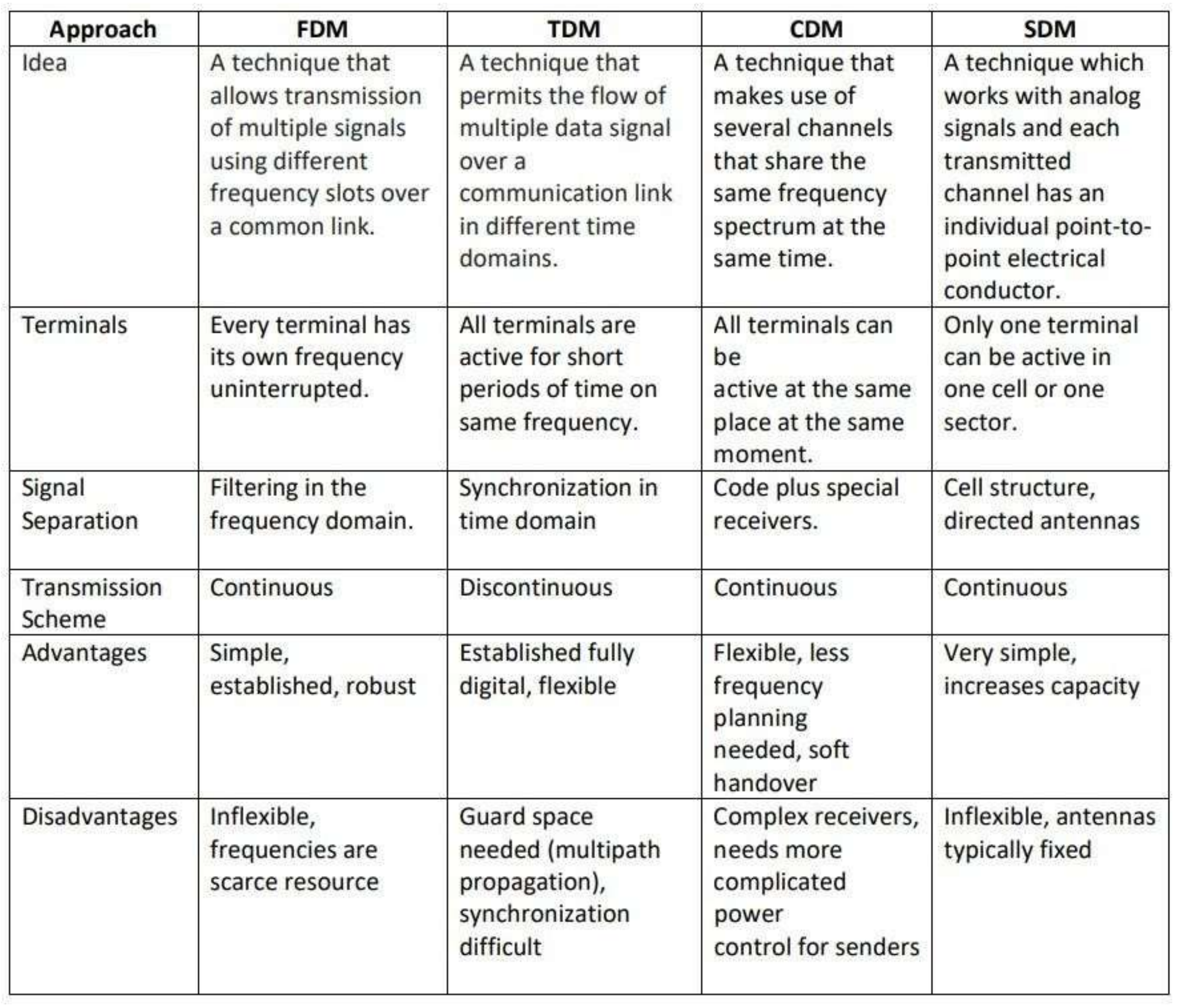
Post labs:

1. What are advantages of CDMA technology over GSM

Ans:

a) High Capacity: The CDMA is based on spread spectrum technology which makes the optimal use of available bandwidth. It allows each user to transfer over the entire frequency spectrum all the time. The GSM operates on the wedge spectrum called a carrier. This carrier is divided into a number of time slots and each user is assigned a different time slot so that until the ongoing call is finished, no other subscriber can have access to this.
b) Strong Security: In CDMA technology, more security is provided as compared with the GSM technology because encryption is inbuilt in the CDMA. A unique code is provided to each and every user and all the conversations between two users are encoded ensuring a greater level of security for CDMA users. The signal cannot be traced easily in CDMA as compared to the signals of GSM, which are concentrated in the narrow bandwidth.
c) Energy Efficiency: A CDMA cell phone employs a dynamic power control system to limit radio interference. Because nearby phones use the same frequency, they will interfere with each other's signal if the transmitting power is too high. CDMA automatically adjusts the power level to one that's powerful enough to provide a clear call but not so powerful that it creates excessive interference. Because it is not always transmitting at peak power, the call uses less energy from the phone's battery.
d) Radiation Exposure: The GSM phones emit continuous wave pulses, so there is a large need to reduce the exposures to electromagnetic fields focused on cell phones with continuous wave pulses. The CDMA cell phones do not produce these pulses. GSM phones emit about 28 times more radiation on average as compared to CDMA phones. Moreover, GSM phones are more biologically reactive as compared to CDMA.

2. Compare FDM, TDM, CDM, and SDM.
Ans:



Approach	FDM	TDM	CDM	SDM
Idea	A technique that allows transmission of multiple signals using different frequency slots over a common link.	A technique that permits the flow of multiple data signal over a communication link in different time domains.	A technique that makes use of several channels that share the same frequency spectrum at the same time.	A technique which works with analog signals and each transmitted channel has an individual point-to-point electrical conductor.
Terminals	Every terminal has its own frequency uninterrupted.	All terminals are active for short periods of time on same frequency.	All terminals can be active at the same place at the same moment.	Only one terminal can be active in one cell or one sector.
Signal Separation	Filtering in the frequency domain.	Synchronization in time domain	Code plus special receivers.	Cell structure, directed antennas
Transmission Scheme	Continuous	Discontinuous	Continuous	Continuous
Advantages	Simple, established, robust	Established fully digital, flexible	Flexible, less frequency planning needed, soft handover	Very simple, increases capacity
Disadvantages	Inflexible, frequencies are scarce resource	Guard space needed (multipath propagation), synchronization difficult	Complex receivers, needs more complicated power control for senders	Inflexible, antennas typically fixed