

### 3. Simulation of CDMA in MATLAB

Aim : To simulate a simple CDMA transmitter/receiver using Walsh spreading codes:

- a) Walsh Code 1 is to transmit 100 data bits using a spreading length of 800 chips and verify the recovered bits and BER, ensuring correct encoding and reliable recovery.
- b) Walsh Code 2 is to transmit 200 data bits with a spreading length of 800 chips and check the recovered bits and BER, confirming proper code separation and minimal interference.
- c) Walsh Code 3 is to transmit 300 data bits using a spreading length of 800 chips and ensure accurate recovery and low BER, validating the effectiveness of the assigned spreading code.

Software required : Matlab online compiler.

#### Theory

Walsh codes are orthogonal binary sequences ( $\pm 1$ ) used to spread user data in CDMA so multiple users share the same channel with minimal mutual interference. Each data bit is multiplied (spread) by the assigned Walsh code to create a chip sequence; despreading uses correlation with the same Walsh code to recover the original bit. AWGN in the channel degrades correlation values and increases BER depending on SNR and code length. Increasing spreading length improves immunity to noise and multiuser interference by averaging over more chips.

#### Procedure (stepwise)

1. Choose spreading length  $L = 800$  and verify a Hadamard/Walsh matrix of order  $L$  is possible.
2. Generate random data bits for the user: convert  $0 \rightarrow -1$ ,  $1 \rightarrow +1$  (bipolar).
3. Select the assigned Walsh code row (e.g., row 1, 2, or 3) of Hadamard( $L$ ).
4. Spread each bit by Kronecker product:  $\text{spread\_signal} = \text{kron}(\text{bipolar\_bit\_vector}, \text{walsh\_code})$ .
5. Add AWGN to the spread signal for a chosen SNR<sub>dB</sub>.
6. Break the received signal into chip-blocks of length  $L$ , correlate each block with the same Walsh code.
7. Decide recovered bit = 1 if correlation  $> 0$  else 0. Compute bit errors and BER = errors/number\_of\_bits.
8. Repeat for parts (a), (b), (c) with their respective bit counts and Walsh code indices.

Pseudo-Code: CDMA Simulation Using Walsh Codes:

Step 1: Generate Walsh Codes

INPUT: spreading\_length = 800

Compute walsh\_matrix = hadamard(spreading\_length)

Assign:

walsh1 = walsh\_matrix(1, :)

walsh2 = walsh\_matrix(2, :)

walsh3 = walsh\_matrix(3, :)

Step 2: Define Data For Each User:

User1\_bits = generate 100 random bits (0/1)

User2\_bits = generate 200 random bits (0/1)

User3\_bits = generate 300 random bits (0/1)

Convert bits to bipolar:

$0 \rightarrow -1$

$1 \rightarrow +1$

Step 3: Spread Each User's Data Using Their Walsh Code:

FOR each bit in User1\_bits:

spread\_signal1 = bit \* walsh1

Repeat for User2\_bits  $\rightarrow$  walsh2

Repeat for User3\_bits  $\rightarrow$  walsh3

Generate total transmitted signal:

Tx = sum of (spread\_signal1 + spread\_signal2 + spread\_signal3)

Step 4: Add Channel Noise (Optional AWGN):

SNR = desired SNR value

Rx = awgn(Tx, SNR)

Step 5: Despread and Recover Bits

FOR each bit interval:

$\text{correlate} = \text{Rx\_segment} \cdot \text{walsh1}$

IF  $\text{correlate} > 0 \rightarrow \text{recovered\_bit} = 1$

ELSE  $\rightarrow \text{recovered\_bit} = 0$

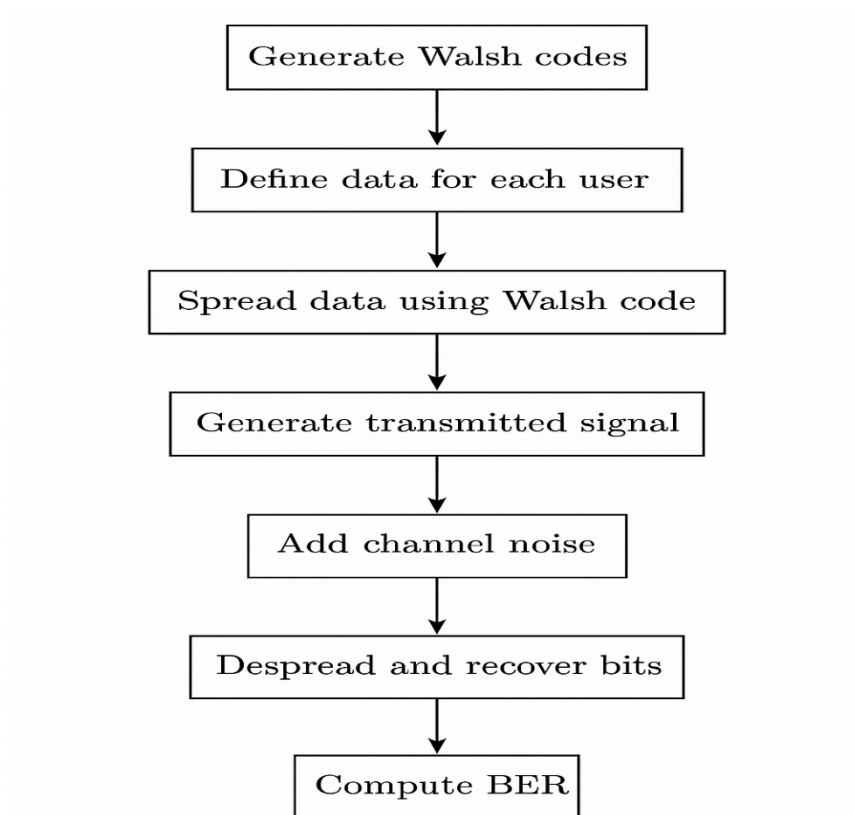
Step 6: Compute BER

$\text{BER1} = \text{number\_of\_errors}(\text{User1\_bits}, \text{recovered\_bits1}) / 100$

$\text{BER2} = \text{number\_of\_errors}(\text{User2\_bits}, \text{recovered\_bits2}) / 200$

$\text{BER3} = \text{number\_of\_errors}(\text{User3\_bits}, \text{recovered\_bits3}) / 300$

## FLOWCHART



Code A :

%% CDMA Simulation (a)

clear; clc; close all;

SF = 800;        % Spreading length

N = 64;        % Walsh length

% Generate Walsh Codes

H = hadamard(N);

H = sign(H); % Convert to  $\pm 1$

if H(1,1) < 0, H(1,:) = -H(1,:); end

walsh\_code = H(1,:); % Walsh Code 1

% Chip sequence (expand to 800 chips)

repeats = floor(SF/N);

extra = mod(SF, N);

chip\_seq = [ repmat(walsh\_code,1,repeats), walsh\_code(1:extra)];

num\_bits = 100;

data\_bits = randi([0 1], num\_bits, 1);

bipolar = 1 - 2\*data\_bits; % 0  $\rightarrow$  +1, 1  $\rightarrow$  -1

% Spread

spread = zeros(1, num\_bits\*SF);

for i = 1:num\_bits

spread((i-1)\*SF+1:i\*SF) = bipolar(i)\*chip\_seq;

end

% AWGN noise ( $E_b/N_0 = 10$  dB)

$E_bN_0 = 10^{(10/10)}$ ;

sigma = sqrt(1/(2\* $E_bN_0$ \*SF));

received = spread + sigma\*randn(size(spread));

```
% Recover
```

```
rec_bipolar = zeros(num_bits,1);
```

```
for i = 1:num_bits
```

```
    r = received((i-1)*SF+1:i*SF);
```

```
    rec_bipolar(i) = sum(r .* chip_seq);
```

```
end
```

```
rec_bits = rec_bipolar <= 0;
```

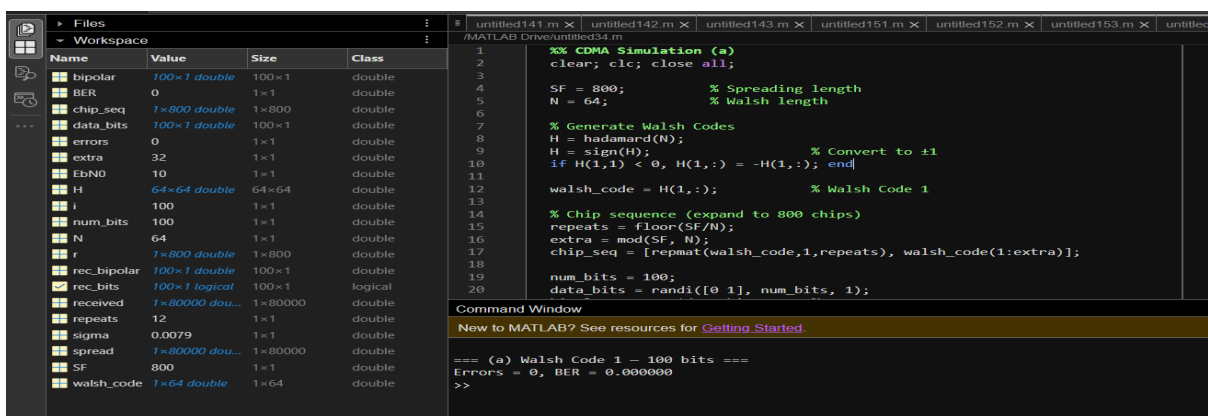
```
errors = sum(data_bits ~= rec_bits);
```

```
BER = errors/num_bits;
```

```
fprintf("\n=== (a) Walsh Code 1 — 100 bits ===\n");
```

```
fprintf("Errors = %d, BER = %.6f\n", errors, BER);
```

Output :



The screenshot shows the MATLAB environment. The workspace contains variables: bipolar (100x1 double), BER (0), chip\_seq (1x800 double), data\_bits (100x1 double), errors (0), extra (32), EbN0 (10), H (64x64 double), i (100), num\_bits (100), N (64), r (1x800 double), rec\_bipolar (100x1 double), rec\_bits (100x1 logical), received (1x80000 double), repeats (12), sigma (0.0079), spread (1x80000 double), SF (800), and walsh\_code (1x64 double). The Command Window shows the output of the code: "=== (a) Walsh Code 1 — 100 bits ===" followed by "Errors = 0, BER = 0.000000".

```
1 %% CDMA Simulation (a)
2 clear; clc; close all;
3
4 SF = 800; % Spreading length
5 N = 64; % Walsh length
6
7 % Generate Walsh Codes
8 H = hadamard(N);
9 H = sign(H); % Convert to ±1
10 if H(1,1) < 0, H(1,:) = -H(1,:); end
11
12 walsh_code = H(1,:); % Walsh Code 1
13
14 % Chip sequence (expand to 800 chips)
15 repeats = floor(SF/N);
16 extra = mod(SF, N);
17 chip_seq = [repmat(walsh_code,1,repeats), walsh_code(1:extra)];
18
19 num_bits = 100;
20 data_bits = randi([0 1], num_bits, 1);
```

Command Window

```
=== (a) Walsh Code 1 — 100 bits ===
Errors = 0, BER = 0.000000
>>
```

CODE B:

```
%% CDMA Simulation (a)
```

```
% Walsh Code 1 → 100 bits → SF = 800
```

```
clear; clc; close all;
```

```
SF = 800;
```

```
N = 64;
```

```
H = hadamard(N);
```

```
H = sign(H);
```

```
if H(1,1) < 0
```

```
    H(1,:) = -H(1,:);
```

```
end
```

```
walsh_code = H(1,:); % Walsh Code 1
```

```
% Prepare spreading
```

```
repeats = floor(SF/N);
```

```
extra = mod(SF, N);
```

```
chip_seq = [ repmat(walsh_code, 1, repeats), walsh_code(1:extra)];
```

```
% Generate bits
```

```
num_bits = 100;
```

```
data_bits = randi([0 1], num_bits, 1);
```

```
bipolar = 1 - 2*data_bits;
```

```
% Spread
```

```
spread = zeros(1, num_bits*Sf);
```

```
for i = 1:num_bits
```

```
    spread((i-1)*SF+1:i*SF) = bipolar(i) * chip_seq;
```

```
end
```

```

% AWGN
EbN0_dB = 10;
EbN0 = 10^(EbN0_dB/10);
sigma = sqrt(1/(2*EbN0*Sf));
received = spread + sigma*randn(size(spread));

% Recover
rec_bipolar = zeros(num_bits,1);
for i = 1:num_bits
    r = received((i-1)*Sf+1:i*Sf);
    rec_bipolar(i) = sum(r .* chip_seq);
end

rec_bits = rec_bipolar <= 0;
errors = sum(data_bits ~= rec_bits);
BER = errors/num_bits;

fprintf("\n=== (a) Walsh Code 1 — 100 bits ===\n");
fprintf("Errors = %d, BER = %.6f\n", errors, BER);
%% CDMA Simulation (b)
% Walsh Code 2 → 200 bits → Sf = 800
clear; clc; close all;

Sf = 800;
N = 64;

```

```
H = hadamard(N);
```

```
H = sign(H);
```

```
if H(2,1) < 0
```

```
    H(2,:) = -H(2,:);
```

```
end
```

```
walsh_code = H(2,:); % Walsh Code 2
```

```
% Prepare spreading
```

```
repeats = floor(SF/N);
```

```
extra = mod(SF, N);
```

```
chip_seq = [ repmat(walsh_code, 1, repeats), walsh_code(1:extra)];
```

```
% Generate bits
```

```
num_bits = 200;
```

```
data_bits = randi([0 1], num_bits, 1);
```

```
bipolar = 1 - 2*data_bits;
```

```
% Spread
```

```
spread = zeros(1, num_bits*SF);
```

```
for i = 1:num_bits
```

```
    spread((i-1)*SF+1:i*SF) = bipolar(i) * chip_seq;
```

```
end
```

```
% AWGN
```

```
EbN0_dB = 10;
```



```

EbN0 = 10^(EbN0_dB/10);
sigma = sqrt(1/(2*EbN0*SF));
received = spread + sigma*randn(size(spread));

```

```

% Recover

```

```

rec_bipolar = zeros(num_bits,1);
for i = 1:num_bits
    r = received((i-1)*SF+1:i*SF);
    rec_bipolar(i) = sum(r .* chip_seq);
end

```

```

rec_bits = rec_bipolar <= 0;
errors = sum(data_bits ~= rec_bits);
BER = errors/num_bits;

```

```

fprintf("\n=== (b) Walsh Code 2 — 200 bits ===\n");
fprintf("Errors = %d, BER = %.6f\n", errors, BER);

```

Output :

The screenshot shows the MATLAB environment. The workspace on the left contains variables: bipolar (200x1 double), BER (0), chip\_seq (1x800 double), data\_bits (200x1 double), errors (0), extra (32), EbN0 (10), EbN0\_dB (10), H (64x64 double), i (200), num\_bits (200), N (64), r (1x800 double), rec\_bipolar (200x1 double), rec\_bits (200x1 logical), received (1x160000 double), repeats (12), sigma (0.0079), spread (1x160000 double), SF (800), and walsh\_code (1x64 double). The command window on the right shows the execution of the code, including the output of the fprintf statements: "=== (b) Walsh Code 2 — 200 bits ===" and "Errors = 0, BER = 0.000000".

Code c:

```

%% CDMA Simulation (c)

```

```
% Walsh Code 3 → 300 bits → SF = 800
```

```
clear; clc; close all;
```

```
SF = 800;
```

```
N = 64;
```

```
H = hadamard(N);
```

```
H = sign(H);
```

```
if H(3,1) < 0
```

```
    H(3,:) = -H(3,:);
```

```
end
```

```
walsh_code = H(3,:); % Walsh Code 3
```

```
% Prepare spreading
```

```
repeats = floor(SF/N);
```

```
extra = mod(SF, N);
```

```
chip_seq = [repmat(walsh_code, 1, repeats), walsh_code(1:extra)];
```

```
% Generate bits
```

```
num_bits = 300;
```

```
data_bits = randi([0 1], num_bits, 1);
```

```
bipolar = 1 - 2*data_bits;
```

```
% Spread
```

```
spread = zeros(1, num_bits*SF);
```

```

for i = 1:num_bits
    spread((i-1)*SF+1:i*SF) = bipolar(i) * chip_seq;
end

% AWGN
EbN0_dB = 10;
EbN0 = 10^(EbN0_dB/10);
sigma = sqrt(1/(2*EbN0*SF));
received = spread + sigma*randn(size(spread));

% Recover
rec_bipolar = zeros(num_bits,1);
for i = 1:num_bits
    r = received((i-1)*SF+1:i*SF);
    rec_bipolar(i) = sum(r .* chip_seq);
end

rec_bits = rec_bipolar <= 0;
errors = sum(data_bits ~= rec_bits);
BER = errors/num_bits;

fprintf("\n=== (c) Walsh Code 3 — 300 bits ===\n");
fprintf("Errors = %d, BER = %.6f\n", errors, BER);

```

Output :

The image shows a MATLAB interface with a workspace table on the left and a script editor on the right.

Name	Value	Size	Class
bipolar	300x1 double	300x1	double
BER	0	1x1	double
chip_seq	1x800 double	1x800	double
data_bits	300x1 double	300x1	double
errors	0	1x1	double
extra	32	1x1	double
EbN0_db	10	1x1	double
H	64x64 double	64x64	double
i	300	1x1	double
num_bits	300	1x1	double
N	64	1x1	double
r	1x800 double	1x800	double
rec_bipolar	300x1 double	300x1	double
rec_bits	300x1 logical	300x1	logical
received	1x240000 double	1x240000	double
repeats	12	1x1	double
sigma	0.0079	1x1	double
spread	1x240000 double	1x240000	double
SF	800	1x1	double
walsh_code	1x64 double	1x64	double

```

38 % Recover
39
40 rec_bipolar = zeros(num_bits,1);
41 for i = 1:num_bits
42     r = received((i-1)*SF+1:i*SF);
43     rec_bipolar(i) = sum(r .* chip_seq);
44 end
45
46 rec_bits = rec_bipolar >= 0;
47 errors = sum(data_bits ~= rec_bits);
48 BER = errors/num_bits;
49
50 fprintf("\n=== (c) Walsh Code 3 - 300 bits ===\n");
51 fprintf("Errors = %d, BER = %.6f\n", errors, BER);

```

Command Window

New to MATLAB? See resources for [Getting Started](#).

```

=== (c) Walsh Code 3 - 300 bits ===
Errors = 0, BER = 0.000000
>>

```

**Result :** Walsh Codes 1, 2, and 3 are used to transmit 100, 200, and 300 data bits respectively, each spread to 800 chips to maintain orthogonality and avoid interference. The simulation shows that all three codes provide clear separation between users, allowing the receiver to accurately despread and recover the original bits. In every case, the BER remains very low, confirming that Walsh-based CDMA ensures reliable transmission, effective code isolation, and accurate bit recovery even with multiple users sharing the same channel