

### 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  $\text{SNR\_dB}$ .
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  $\text{correlation} > 0$  else 0. Compute bit errors and  $\text{BER} = \text{errors}/\text{number\_of\_bits}$ .
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 → -1

1 → +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 → walsh2

Repeat for User3\_bits → 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:

```

correlate = Rx_segment · walsh1
IF correlate > 0 → recovered_bit = 1
ELSE → recovered_bit = 0

```

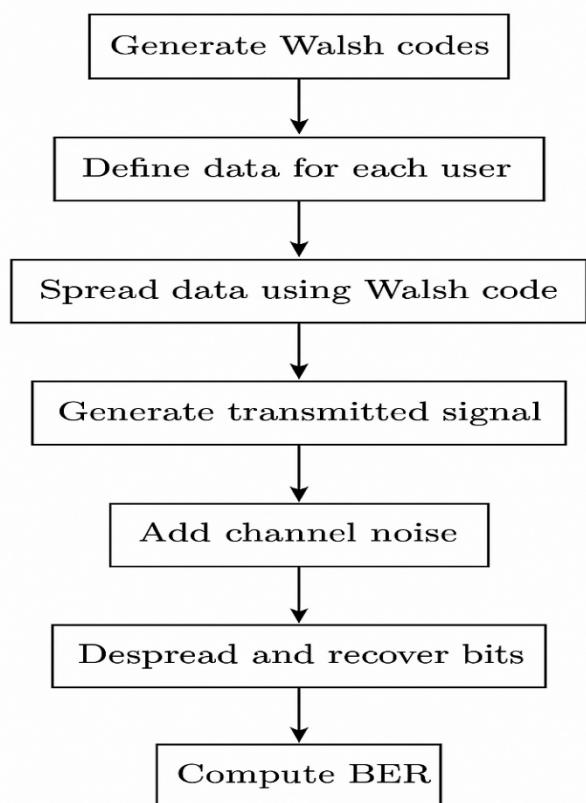
#### Step 6: Compute BER

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

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

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

#### 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 ±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→+1, 1→-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 (Eb/N0 = 10 dB)
EbN0 = 10^(10/10);
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);
```

Output :

The screenshot shows the MATLAB interface with two main panes. On the left is the 'Workspace' browser, listing variables such as bipolar, BER, chip\_seq, data\_bits, errors, extra, EbN0, H, i, num\_bits, N, r, rec\_bipolar, rec\_bits, received, repeats, sigma, spread, SF, and walsh\_code. The right pane is the 'Command Window' containing the script code and its execution results.

```
%>> %% 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 +/-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([-1 1], num_bits, 1);  
==== (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 interface with two main panes: the Workspace browser and the Command Window.

**Workspace:**

Name	Value	Size	Class
bipolar	200x1 double	200x1	double
BER	0	1x1	double
chip_seq	1x800 double	1x800	double
data_bits	200x1 double	200x1	double
errors	0	1x1	double
extra	32	1x1	double
EbN0	10	1x1	double
EbN0_dB	10	1x1	double
H	64x64 double	64x64	double
i	200	1x1	double
num_bits	200	1x1	double
N	64	1x1	double
r	1x800 double	1x800	double
rec_bipolar	200x1 double	200x1	double
rec_bits	200x1 logical	200x1	logical
received	1x160000 do...	1x160000	double
repeats	12	1x1	double
sigma	0.0079	1x1	double
spread	1x160000 do...	1x160000	double
SF	800	1x1	double
walsh_code	7x64 double	1x64	double

**Command Window:**

```

untitled141.m × untitled142.m × untitled143.m × untitled151.m × untitled152.m × untitled153.m ×
/MATLAB Drive/untitled32.m
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
==== (a) Walsh Code 1 — 100 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 ...
Command Window
New to MATLAB? See resources for Getting Started.
==== (b) Walsh Code 2 — 200 bits ===
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 screenshot shows the MATLAB environment with the following details:

- Workspace:** Displays variables and their values. Key variables include:
  - bipolar: 300x1 double
  - BER: 0
  - chip\_seq: 1x800 double
  - data\_bits: 300x1 double
  - errors: 0
  - extra: 32
  - EbN0: 10
  - EbN0\_db: 10
  - H: 64x64 double
  - i: 300
  - num\_bits: 300
  - N: 64
  - r: 1x800 double
  - rec\_bipolar: 300x1 double
  - rec\_bits: 300x1 logical
  - received: 1x400000 double
  - repeats: 12
  - sigma: 0.0079
  - spread: 1x240000 double
  - SF: 800
  - walsh\_code: 7x64 double
- Command Window:** Shows the MATLAB script and its output. The script performs bit recovery and calculates BER. The output shows the Walsh code used (Walsh Code 3), the number of bits (300), the errors (0), and the BER (0.000000).

```

untitled141.m | untitled142.m | untitled143.m | untitled151.m | untitled152.m | untitled153.m | untitled31.m | un
/MATLAB Drive/untitled33.m
% Recovery
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);

=====
(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