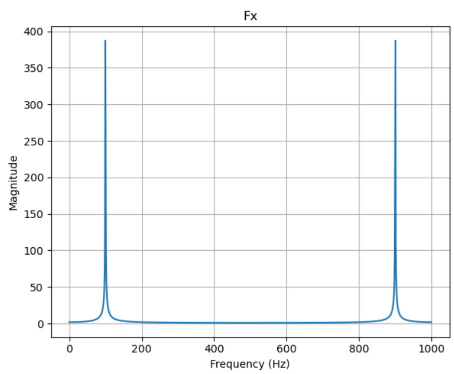
ADSP: HW5

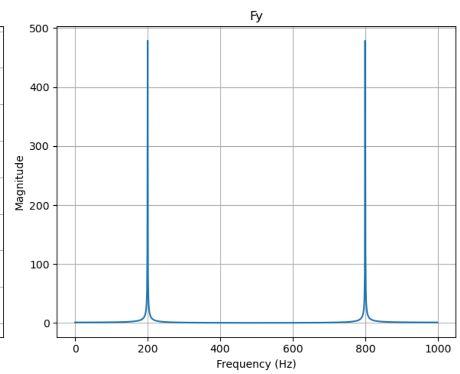
Lo Chun, Chou
R13922136

June 11, 2025

(1)

An example result of the code is shown as the following image:





For the code, please refer to the attached python file `problem_1.py` on NTU-COOL.

(2)

The two main advantages of the sectioned convolution compared with the original non-sectioned convolutions are:

- We can reduce computation by sectioning, since the complexity of sectioned convolution is linear with N ($\theta(N)$), however, for non-sectioned convolution, the complexity is $\theta(N \log N)$
- Since the restriction $P \geq M+N-1$ needs to be satisfied, when the number of N changes, the number of P should also change.

  Even though this could be easily implemented by software, it is not realistic for hardware implementation, because implementing multiple fourier transforms on a chip with different Ns, in order to deal with inputs of different length is too costly.

However, if we split the input signal into multiple sections, with each section of length L, for the number of points of the fourier transform (P) to satisfy the inequality $P \geq M + L - 1$, since L is fixed, P would not be affected by the length of the input signal N.

(3)

To compute the convolution operation of $y[n] = x[n] * h[n]$, where:

$$h[n] = [0.09\ 0.36\ 0.55\ 0.55\ 0.36\ 0.09] \quad -2 \leq n \leq 3$$

by an efficient way that the number of multiplciations is minimized, first observe that $h[n]$ is symmetric, since it fits the two conditions in lecture note "ADSP_Write6.pdf", p.439:

- M is small
- The filter has some symmetric relation

This is the case that we can use "direct computing" to compute the convolution operation:

$$\begin{aligned} y[n] &= 0.09x[n-3] + 0.36x[n-2] + 0.55x[n-1] + 0.55x[n] + 0.36x[n+1] + 0.09x[n+2] \\ &= 0.09(x[n-3] + x[n+2]) + 0.36(x[n-2] + x[n+1]) + 0.55(x[n-1] + x[n]) \end{aligned}$$

(4)

Given that the length$(x[n]) = 1500$, we can find the optimal approach by the cases in lecture note "ADSP_Write6.pdf", p.437, 441. And the following are the formulas showing the amount of multiplications for each case:

- For direct computing, we have:

$$3N \times M, \qquad \text{where } N = \text{length}(x[n]),\ M = \text{length}(h[n])$$

- If we use non-sectioned convolution, which is $y[n] = \text{IFFT}_P(\text{FFT}_P\{x[n]\} \times \text{FFT}_P\{h[n]\})$

$$2 \times \text{MUL}_P + 3P, \qquad \text{where } P \geq N + M - 1$$

- If we use sectioned convolution, we have;

$$S \times (2 \times \text{MUL}_P + 3P), \qquad \text{where } S = \left\lceil \frac{N}{L} \right\rceil$$

Since the values are large and hard to calculate by hand, I wrote a python code to calculate the results of the three approaches.

Instead of showing the code here, I attached the code to NTUCOOL since it's too long.
Also, In the following subproblems, I will show the output for executing the code, which shows detailed process for finding the optimal approach. If the code needs to be executed and checked, execute `python 4.py`, and modify the input parameters `N` and `M` to get the results:

```python
def main():
    # Given parameters
    N = 1500  # length(x[n])
    M = 250   # length(h[n])
```

Another thing is that, since the output is also quite lengthy, the answer is written after the output.

(a)

If length$(y[n]) = 250$, then we have $N = 1500$, $M = 250$. The derivation process is shown as the following image:

```
base ~/graduate_stuff/courses/113-2/ADSP/HW/HW5/problem_4 git:(main)±9 (0.555s)
python3 4.py
Given: N = 1500, M = 250

Step 1:
L0 = 1500

Step 2:
Requirements:
- Non-sectioned convolution: P >= N + M - 1 = 1749
- Sectioned convolution: P >= L0 + M - 1 = 1749

Testing P values from table: [1260, 1344, 1440, 1680, 2016, 2048, 2304, 2520, 2688]

Step 3a: Non-sectioned convolution calculations
P       MUL_P    Formula (2xMUL_P + 3xP) Real Multiplications
--------------------------------------------------------------------------
2016    12728    2x12728 + 3x2016                31504
2048    16836    2x16836 + 3x2048                39816
2304    15868    2x15868 + 3x2304                38648
2520    16540    2x16540 + 3x2520                40640
2688    19108    2x19108 + 3x2688                46280
2880    20060    2x20060 + 3x2880                48760
3369    24200    2x24200 + 3x3369                58507
3920    29900    2x29900 + 3x3920                71560
4032    29488    2x29488 + 3x4032                71072
4096    37516    2x37516 + 3x4096                87320

Step 3b: Sectioned convolution calculations
P       L        S        MUL_P    Formula (Sx(2xMUL_P + 3xP))     Real Multiplications
------------------------------------------------------------------------------------
2016    1500     1        12728    1x(2x12728 + 3x2016)                    31504
2048    1500     1        16836    1x(2x16836 + 3x2048)                    39816
2304    1500     1        15868    1x(2x15868 + 3x2304)                    38648
2520    1500     1        16540    1x(2x16540 + 3x2520)                    40640
2688    1500     1        19108    1x(2x19108 + 3x2688)                    46280
2880    1500     1        20060    1x(2x20060 + 3x2880)                    48760
3369    1500     1        24200    1x(2x24200 + 3x3369)                    58507
3920    1500     1        29900    1x(2x29900 + 3x3920)                    71560
4032    1500     1        29488    1x(2x29488 + 3x4032)                    71072
4096    1500     1        37516    1x(2x37516 + 3x4096)                    87320

Direct computation approach:
Number of real multiplications = 1125000

Optimal values for non-sectioned approach:
P = 2016
Table value for P=2016: 12728
Formula: 2 x 12728 + 3 x 2016 = 31504
Number of real multiplications = 31504

Optimal values for sectioned approach:
P = 2016
L = 1500
S = 1
Table value for P=2016: 12728
Formula: 1 x (2 x 12728 + 3 x 2016) = 31504
Number of real multiplications = 31504

Comparison of approaches:
Non-sectioned approach is best with 31504 multiplications

Multiplications required for each approach:
Non-sectioned: 31504
Sectioned: 31504
Direct: 1125000
```

From the image, we can see that the optimal condition is:

- (i) Non-sectioned approach
- (ii) $P = 2016$
- (iii) 31504 real multiplications

(b)

If length($y[n]$) $= 50$, then we have $N = 1500,\ M = 50$. The derivation process is shown as the following image:

```
base ~/graduate_stuff/courses/113-2/ADSP/HW/HW5/problem_4 git:(main)±9 (0.571s)
python3 4.py
Given: N = 1500, M = 50

Step 1:
L0 = 324

Step 2:
Requirements:
- Non-sectioned convolution: P >= N + M - 1 = 1549
- Sectioned convolution: P >= L0 + M - 1 = 373

Testing P values from table: [288, 312, 336, 360, 420, 480, 504, 512, 560]

Step 3a: Non-sectioned convolution calculations
P       MUL_P   Formula (2xMUL_P + 3xP) Real Multiplications
----------------------------------------------------------------------
1680    10420   2x10420 + 3x1680                25880
2016    12728   2x12728 + 3x2016                31504
2048    16836   2x16836 + 3x2048                39816
2304    15868   2x15868 + 3x2304                38648
2520    16540   2x16540 + 3x2520                40640
2688    19108   2x19108 + 3x2688                46280
2880    20060   2x20060 + 3x2880                48760
3369    24200   2x24200 + 3x3369                58507
3920    29900   2x29900 + 3x3920                71560
4032    29488   2x29488 + 3x4032                71072

Step 3b: Sectioned convolution calculations
P       L       S       MUL_P   Formula (Sx(2xMUL_P + 3xP))     Real Multiplications
--------------------------------------------------------------------------------
420     324     5       2080    5x(2x2080 + 3x420)                      27100
480     324     5       2360    5x(2x2360 + 3x480)                      30800
504     324     5       2300    5x(2x2300 + 3x504)                      30560
512     324     5       3180    5x(2x3180 + 3x512)                      39480
560     324     5       3100    5x(2x3100 + 3x560)                      39400
672     324     5       3496    5x(2x3496 + 3x672)                      45040
720     324     5       3620    5x(2x3620 + 3x720)                      47000
784     324     5       4412    5x(2x4412 + 3x784)                      55880
840     324     5       4580    5x(2x4580 + 3x840)                      58400
1008    324     5       5356    5x(2x5356 + 3x1008)                     68680

Direct computation approach:
Number of real multiplications = 225000

Optimal values for non-sectioned approach:
P = 1680
Table value for P=1680: 10420
Formula: 2 x 10420 + 3 x 1680 = 25880
Number of real multiplications = 25880

Optimal values for sectioned approach:
P = 420
L = 324
S = 5
Table value for P=420: 2080
Formula: 5 x (2 x 2080 + 3 x 420) = 27100
Number of real multiplications = 27100

Comparison of approaches:
Non-sectioned approach is best with 25880 multiplications

Multiplications required for each approach:
Non-sectioned: 25880
Sectioned: 27100
Direct: 225000
```

From the image, we can see that the optimal condition is:

- (i) Non-sectioned approach
- (ii) $P = 1680$
- (iii) 25880 real multiplications

(c)

If $\text{length}(y[n]) = 10$, then we have $N = 1500,\ M = 10$. The derivation process is shown as the following image:

```
base ~/graduate_stuff/courses/113-2/ADSP/HW/HW5/problem_4 git:(main)±9 (0.592s)
python3 4.py
Given: N = 1500, M = 10

Step 1:
L0 = 42

Step 2:
Requirements:
- Non-sectioned convolution: P >= N + M - 1 = 1509
- Sectioned convolution: P >= L0 + M - 1 = 51

Testing P values from table: [63, 64, 66, 70, 72]

Step 3a: Non-sectioned convolution calculations
P       MUL_P   Formula (2xMUL_P + 3xP) Real Multiplications
----------------------------------------------------------------------
1680    10420   2x10420 + 3x1680                25880
2016    12728   2x12728 + 3x2016                31504
2048    16836   2x16836 + 3x2048                39816
2304    15868   2x15868 + 3x2304                38648
2520    16540   2x16540 + 3x2520                40640
2688    19108   2x19108 + 3x2688                46280
2880    20060   2x20060 + 3x2880                48760
3369    24200   2x24200 + 3x3369                58507
3920    29900   2x29900 + 3x3920                71560
4032    29488   2x29488 + 3x4032                71072

Step 3b: Sectioned convolution calculations
P       L       S       MUL_P   Formula (Sx(2xMUL_P + 3xP))     Real Multiplications
--------------------------------------------------------------------------------
63      42      36      256     36x(2x256 + 3x63)                       25236
64      42      36      204     36x(2x204 + 3x64)                       21600
66      42      36      284     36x(2x284 + 3x66)                       27576
70      42      36      300     36x(2x300 + 3x70)                       29160
72      42      36      164     36x(2x164 + 3x72)                       19584
80      42      36      260     36x(2x260 + 3x80)                       27360
81      42      36      480     36x(2x480 + 3x81)                       43308
84      42      36      248     36x(2x248 + 3x84)                       26928
88      42      36      364     36x(2x364 + 3x88)                       35712
90      42      36      340     36x(2x340 + 3x90)                       34200

Direct computation approach:
Number of real multiplications = 45000

Optimal values for non-sectioned approach:
P = 1680
Table value for P=1680: 10420
Formula: 2 x 10420 + 3 x 1680 = 25880
Number of real multiplications = 25880

Optimal values for sectioned approach:
P = 72
L = 42
S = 36
Table value for P=72: 164
Formula: 36 x (2 x 164 + 3 x 72) = 19584
Number of real multiplications = 19584

Comparison of approaches:
Sectioned approach is best with 19584 multiplications

Multiplications required for each approach:
Sectioned: 19584
Non-sectioned: 25880
Direct: 45000
```

From the image, we can see that the optimal condition is:

- (i) Sectioned approach
- (ii) $P = 72$
- (iii) 19584 real multiplications

(d)

If $\text{length}(y[n]) = 2$, then we have $N = 1500,\ M = 2$. The derivation process is shown as the following image:

```
base ~/graduate_stuff/courses/113-2/ADSP/HW/HW5/problem_4 git:(main)±10 (0.532s)
python3 4.py
Given: N = 1500, M = 2

Step 1:
L0 = 2

Step 2:
Requirements:
- Non-sectioned convolution: P >= N + M - 1 = 1501
- Sectioned convolution: P >= L0 + M - 1 = 3

Testing P values from table: [63, 64, 66, 70, 72]

Step 3a: Non-sectioned convolution calculations
P       MUL_P    Formula (2xMUL_P + 3xP) Real Multiplications
----------------------------------------------------------------------
1680    10420    2x10420 + 3x1680                25880
2016    12728    2x12728 + 3x2016                31504
2048    16836    2x16836 + 3x2048                39816
2304    15868    2x15868 + 3x2304                38648
2520    16540    2x16540 + 3x2520                40640
2688    19108    2x19108 + 3x2688                46280
2880    20060    2x20060 + 3x2880                48760
3369    24200    2x24200 + 3x3369                58507
3920    29900    2x29900 + 3x3920                71560
4032    29488    2x29488 + 3x4032                71072

Step 3b: Sectioned convolution calculations
P       L       S       MUL_P   Formula (Sx(2xMUL_P + 3xP))     Real Multiplications
--------------------------------------------------------------------------------
63      2       750     256     750x(2x256 + 3x63)                      525750
64      2       750     204     750x(2x204 + 3x64)                      450000
66      2       750     284     750x(2x284 + 3x66)                      574500
70      2       750     300     750x(2x300 + 3x70)                      607500
72      2       750     164     750x(2x164 + 3x72)                      408000
80      2       750     260     750x(2x260 + 3x80)                      570000
81      2       750     480     750x(2x480 + 3x81)                      902250
84      2       750     248     750x(2x248 + 3x84)                      561000
88      2       750     364     750x(2x364 + 3x88)                      744000
90      2       750     340     750x(2x340 + 3x90)                      712500

Direct computation approach:
Number of real multiplications = 9000

Optimal values for non-sectioned approach:
P = 1680
Table value for P=1680: 10420
Formula: 2 x 10420 + 3 x 1680 = 25880
Number of real multiplications = 25880

Optimal values for sectioned approach:
P = 72
L = 2
S = 750
Table value for P=72: 164
Formula: 750 x (2 x 164 + 3 x 72) = 408000
Number of real multiplications = 408000

Comparison of approaches:
Direct approach is best with 9000 multiplications

Multiplications required for each approach:
Direct: 9000
Non-sectioned: 25880
Sectioned: 408000
```

From the image, we can see that the optimal condition is:

- (i) Direct approach
- (ii) No use of P
- (iii) 9000 real multiplications

(5)

(a)

From the lecture note "ADSP_Write7.pdf", p.483, we know that Walsh transform will only become multiplication under logical convolution, which is shown as follows:

Walsh transform: Convolution property

Let $\Rightarrow$ denote the Walsh transform, and $\star$ denote the logical convolution, then we have:

$$\text{If } f[n] \Rightarrow F[m],\ g[n] \Rightarrow G[m],\ \text{then } f[n] \star g[n] \Rightarrow F[m] \times G[m]$$

While we do not have this property under linear convolution, thus, Walsh transform is <u>not suitable</u> for calculating the linear convolution.

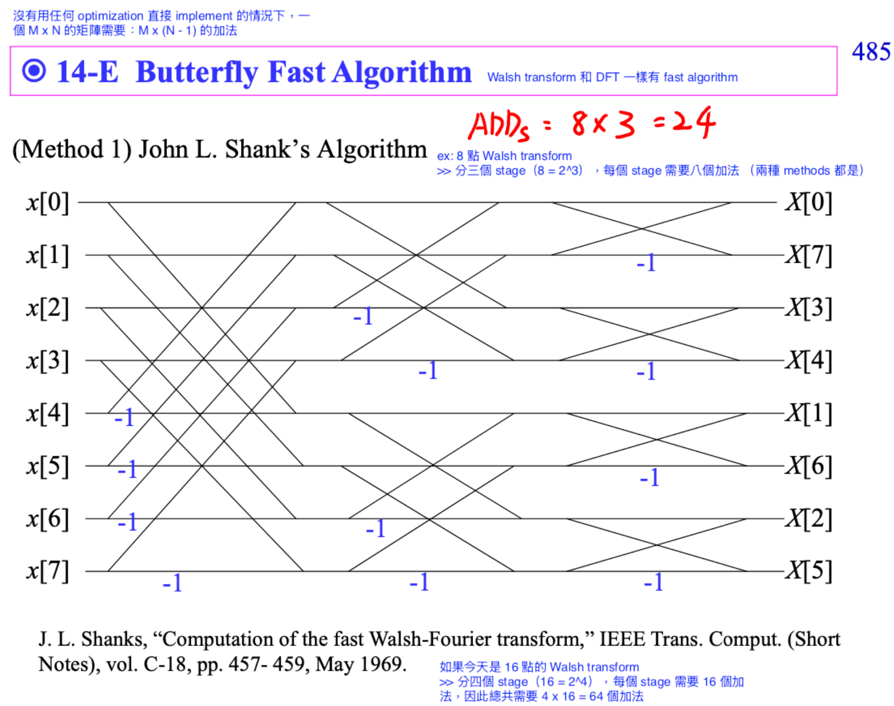
(b)

Stair-like signal anaylsis is <u>suitable</u> for the Walsh transform, since the Walsh transform is a set of orthogonal functions, and the stair-like signal is a combination of step functions.

(6)

(a)

For the 32 point Walsh transform, by using the similar method as in the lecture note "ADSP_Write7.pdf", p.485, which is shown below:

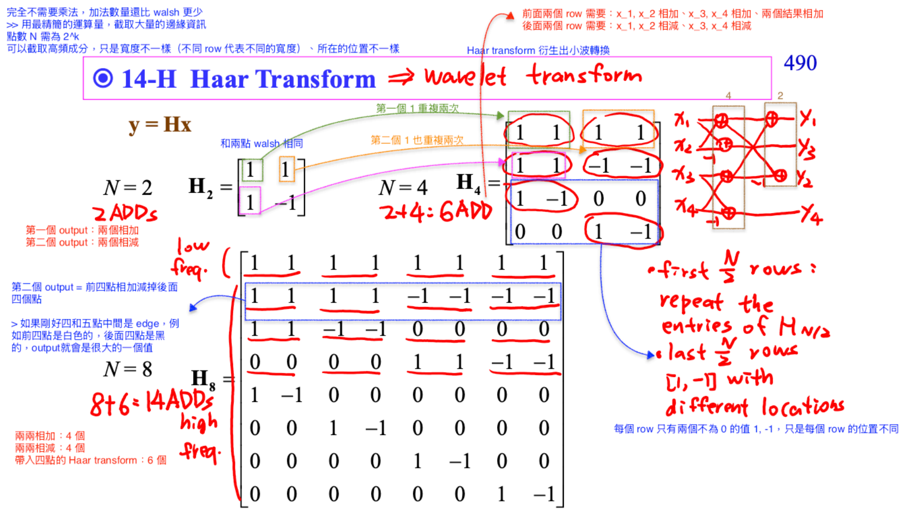


We first need to determine the number of stages needed, then determine the number of additions required for each stage.
Since we have 32 points, which is 2^5, we need 5 stages, and for each stage, we need 32 additions, so the total number of additions needed is:

$$5 \times 32 = 160 \qquad \square$$

(b)

For the 16 point Haar transform, we formulate the matrix as shown in the following lecture note "ADSP_Write7.pdf", p.490:



We need the 8-point Haar transform to realize the 16-point Haar transform, and before applying the 8-point Haar transform, we need every two points to be added and subtracted together, which requires:

1. $16 \div 2 = 8$ (for additions $x_i + x_{i+1},\ i = 0, 2, 4, 6, 8, 10, 12, 14$)
2. $16 \div 2 = 8$ (for subtractions $x_i - x_{i+1},\ i = 1, 3, 5, 7, 9, 11, 13, 15$)

Thus, the total number of additions needed is:

$$\underbrace{8}_{x_i+x_{i+1}} + \underbrace{8}_{x_i-x_{i+1}} + \underbrace{8}_{\text{8 point Haar}} = 24 \qquad \square$$

7

For the subproblems (a) and (b), I use the code as the attached images to generate the results. Simple explanations are shown in the markdown cells.

(a)

Generate the 16 point Walsh matrix

> using scipy

`hadamard(n, dtype=<class 'int'>)`

- `n` is the order of the matrix (must be power of 2)

```python
import numpy as np
from scipy.linalg import hadamard
V = hadamard(16)
print(V)
```

[44] ✓ 0.0s Python

```
[[ 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]
 [ 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]
 [ 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]
 [ 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]]
```

Get the required row values:

row: 1, 4, 10

```python
W1 = V[0]
W4 = V[3]
W10 = V[9]

print(f"W1: {W1}\n")
print(f"W4: {W4}\n")
print(f"W10: {W10}\n")
```

[45] ✓ 0.0s

```
W1: [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]

W4: [ 1 -1 -1  1  1 -1 -1  1  1 -1 -1  1  1 -1 -1  1]

W10: [ 1 -1  1 -1  1 -1  1 -1 -1  1 -1  1 -1  1 -1  1]
```

Modulate

> Check `ADSP_Write7.pdf` p.520 for more info.

Step (1): Change 0 to -1.

Step (2): Modulate

Modulate `[1 0 1]` by `W1`

```python
W1_101 = np.concatenate([W1, -W1, W1])

print(W1_101)
```

```
[ 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]
```

Modulate `[1 1 0]` by `W4`

```python
W4_110 = np.concatenate([W4, W4, -W4])

print(W4_110)
```

```
[ 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]
```

Modulate `[0 1 1]` by `W10`

```python
W10_011 = np.concatenate([-W10, W10, W10])

print(W10_011)
```

```
[-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]
```

And the answer of (a) is as the result of the image below:

Step (3): 相合

```python
x = W1_101 + W4_110 + W10_011
print(f"Step 3 result (answer of  7 (a)):\n {x}\n")
```

```
Step 3 result (answer of  7 (a)):
 [ 1  1 -1  3  1  1 -1  3  3 -1  1  1  3 -1  1  1  1 -3 -1 -1  1 -3 -1 -1
 -1 -1 -3  1 -1 -1 -3  1  1  1  3 -1  1  1  3 -1 -1  3  1  1 -1  3  1  1]
```

Since the image might not be that clear, the answer is:

$$\begin{bmatrix} 1 & 1 & -1 & 3 & 1 & 1 & -1 & 3 & 3 & -1 & 1 & 1 & 3 & -1 & 1 & 1 \\ 1 & -3 & -1 & -1 & 1 & -3 & -1 & -1 & -1 & -1 & -3 & 1 & -1 & -1 & -3 & 1 \\ 1 & 1 & 3 & -1 & 1 & 1 & 3 & -1 & -1 & 3 & 1 & 1 & -1 & 3 & 1 & 1 \end{bmatrix} \qquad \square$$

(b)

Demodulation

> Check `ADSP_Write7.pdf` p.521 for more info.

```python
x1 = np.split(x, 3)[0]
x2 = np.split(x, 3)[1]
x3 = np.split(x, 3)[2]

print(f"x1: {x1}\n")
print(f"x2: {x2}\n")
print(f"x3: {x3}\n")
```

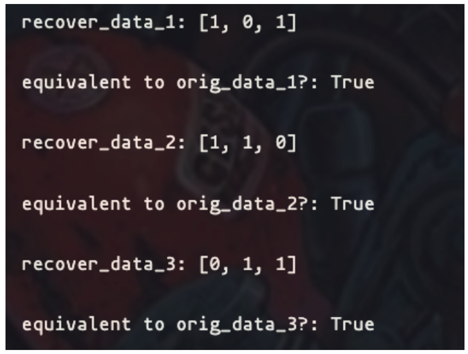
[50] ✓ 0.0s

```
x1: [ 1  1 -1  3  1  1 -1  3  3 -1  1  1  3 -1  1  1]

x2: [ 1 -3 -1 -1  1 -3 -1 -1 -1 -1 -3  1 -1 -1 -3  1]

x3: [ 1  1  3 -1  1  1  3 -1 -1  3  1  1 -1  3  1  1]
```

After getting the values of `x1`, `x2`, `x3`, we try to recover the original data by the following process, note that only the recovery process of `[1 0 1]` is shown since the rest are the same:

```python
orig_data_1 = [1, 0, 1]
orig_data_2 = [1, 1, 0]
orig_data_3 = [0, 1, 1]

recover_data_1 = []
res = np.inner(x1, W1) / 16
if res > 0:
    recover_data_1.append(1)
else:
    recover_data_1.append(0)

res = np.inner(x2, W1) / 16
if res > 0:
    recover_data_1.append(1)
else:
    recover_data_1.append(0)

res = np.inner(x3, W1) / 16
if res > 0:
    recover_data_1.append(1)
else:
    recover_data_1.append(0)

print(f"recover_data_1: {recover_data_1}\n")
print(f"equivalent to orig_data_1?: {recover_data_1 == orig_data_1}\n")
```

The recover result is as follows:



Thus, we can recover the original data by the process shown in the image above.

(c)

In lecture slide ”ADSP_Write7.pdf”, p.522, it is said that we can replace Walsh transform with other orthogonal transforms in CDMA, hence, it is suitable using Haar transform.

Extra problem (ID ends with $1, 6$)

How many real multiplications are needed when the length of the input function is 100 points ($N = \text{length}(x[n]) = 100$), the filter is 19 points ($M = \text{length}(h[n]) = 19$), and we want to implement it by a 120 points Fourier transform ($P = 120$)?

$$\begin{aligned} & 2\text{MUL}_{120} + 120 \times 3 \\ &= 2 \times 380 + 120 \times 3 \\ &= 760 + 360 \\ &= 1120 \end{aligned}$$

Note that we multiplied 120 by 3 since one complex multiplication requires 3 real multiplications, and we got the value of $\text{MUL}_{120} = 380$ by the table in lecture note ”ADSP_Write5.pdf”, p.378.