**Experiment 4 : Fast Fourier transform**

|  |  |
| --- | --- |
| **Name** | Prathamesh Mane |
| **UID no. & Branch** | **2022200078 (B1)** |
| **Experiment No.** | 4 |

|  |  |
| --- | --- |
| **AIM:** | The aim of this experiment is to  implement computationally Fast Algorithms. |
| **OBJECTIVE:** | 1. Develop a program to perform FFT of N point Signal. 2. Calculate FFT of a given DT signal and verify the results using mathematical formula. 3. Computational efficiency of FFT. |
| **INPUT SPECIFICATION:** | 1. Length of first Signal  N 2. DT Signal values |
| **PROBLEM DEFINITION:** | (1)  Take any four-point sequence x[n]. Find FFT of x[n] and IFFT of {X[k]}.  (2) Calculate Real and Complex Additions &  Multiplications involved to find X[k] |
| **THEORY:** | **Discrete Fourier Transform (DFT)**   1. **Definition:**    * The DFT is a mathematical algorithm used to convert a discrete sequence of values into its frequency domain representation. The DFT computes the Fourier coefficients for a sequence of NNN discrete points. 2. **Formula:**    * The DFT of a sequence x[n] is given by:      * + Here, X[k] is the frequency domain representation of the sequence, and the summation involves all Npoints of the input sequence.  1. **Complexity:**    * The direct computation of the DFT requires O(N^2) operations because each of the N output values requires a summation over N input values. 2. **Usage:**    * The DFT is used to analyze the frequency components of a discrete signal. However, due to its computational complexity, it is often impractical for large datasets.   **Fast Fourier Transform (FFT)**   1. **Definition:**    * The FFT is an efficient algorithm for computing the DFT. It significantly reduces the computational complexity of the DFT, making it practical for larger datasets. 2. **Algorithm:**    * The FFT utilizes the divide-and-conquer strategy to break down the DFT computation into smaller DFTs. It reorganizes the computation to reduce redundant operations and improve efficiency. 3. **Complexity:**    * The FFT reduces the complexity of computing the DFT from O(N^2) to O(NlogN), making it much faster for large N. This efficiency gain is achieved by exploiting symmetries and periodicities in the DFT computation. 4. **Variants:**    * There are various FFT algorithms, such as the Cooley-Tukey algorithm, which is the most commonly used, and others like the Radix-2, Radix-4, and mixed-radix FFT algorithms. Each variant is optimized for different types of inputs and applications. 5. **Usage:**    * The FFT is widely used in practical applications where large datasets need to be transformed quickly, such as in real-time signal processing, audio analysis, image processing, and communication systems |
| **Theoretical solution** | |
| **Question : A= [6,12,7,14]**  **Solution :**  N=4  X[k] = [39, -1 + 2j, -13, -1 - 2j]  Magnitude : [ 39 , 2.24 , 13 , 2.24]  **Question : A= [6,12,7,14,8,16,9,18]**  **Solution :**  N=8  X[k] = [90.000 + j 0.000  -2.000 + j7.657  -2.000 + j4.000  -2.000 + j3.657  -30.000 + j0.000  -2.000 + j-3.657  -2.000 + j-4.000  -2.000 + j-7.657]    Magnitude : [ 90,7.91,4.47,4.1681,30,4.168,4.47,7.91] | |
| **Developing algorithm using programming language** | |
| **PROBLEM STATEMENT:** | Write a program in any programming language to perform the linear convolution of two signals with length L and M respectively. |
| **PROGRAM:** | #include <stdio.h>  #include <math.h>  #define SIZE 8  void FFT\_4\_Point(int N, float input[SIZE][2], float output[SIZE][2]);  void FFT\_8\_Point(int N, float input[SIZE][2], float output[SIZE][2]);  void Inverse\_FFT(int N, float input[SIZE][2], float output[SIZE][2]);  int main() {  int i, n;  float input[SIZE][2], output[SIZE][2];  // Initialize input and output arrays  for(i = 0; i < SIZE; i++) {  output[i][0] = 0;  output[i][1] = 0;  input[i][0] = 0;  input[i][1] = 0;  }  // Get the input signal length  printf("\nEnter the length of x[n] (4 pt or 8 pt) = : ");  scanf("%d", &n);  if (n != 4 && n != 8) {  printf("Length must be 4 or 8 for this implementation.\n");  return 1;  }  // Get the input signal values  printf("Enter the values of x[n]: ");  for(i = 0; i < n; i++) {  scanf("%f", &input[i][0]);  }  // Display the input signal  printf("\nInput signal x[n] = ");  for(i = 0; i < n; i++) {  printf(" %4.2f ", input[i][0]);  }  // Perform FFT  if (n == 4) {  FFT\_4\_Point(n, input, output);  } else {  FFT\_8\_Point(n, input, output);  }  // Display FFT results  printf("\n\nFFT results X[k] = :\n");  for(i = 0; i < n; i++) {  printf("\n %7.3f + j %7.3f", output[i][0], output[i][1]);  }  printf("\n\n");  // Perform Inverse FFT  Inverse\_FFT(n, output, input);  // Display Inverse FFT results  printf("\nInverse FFT results x[n] = :\n");  for(i = 0; i < n; i++) {  printf("\n %7.3f + j %7.3f", input[i][0], input[i][1]);  }  printf("\n\n");  return 0;  }  void FFT\_4\_Point(int N, float input[SIZE][2], float output[SIZE][2]) {  float temp[SIZE][2];  int k;  float angle = 2 \* M\_PI / N;  // Stage 1  temp[0][0] = input[0][0] + input[2][0];  temp[0][1] = input[0][1] + input[2][1];  temp[1][0] = input[0][0] - input[2][0];  temp[1][1] = input[0][1] - input[2][1];  temp[2][0] = input[1][0] + input[3][0];  temp[2][1] = input[1][1] + input[3][1];  temp[3][0] = input[1][0] - input[3][0];  temp[3][1] = input[1][1] - input[3][1];  // Stage 2  for (k = 0; k < N; k++) {  output[k][0] = temp[k % 2][0] + (temp[k % 2 + 2][0] \* cos(angle \* k) + temp[k % 2 + 2][1] \* sin(angle \* k));  output[k][1] = temp[k % 2][1] + (temp[k % 2 + 2][1] \* cos(angle \* k) - temp[k % 2 + 2][0] \* sin(angle \* k));  }  }  void FFT\_8\_Point(int N, float input[SIZE][2], float output[SIZE][2]) {  float G[4][2], H[4][2], temp[SIZE][2];  int k;  float angle = 2 \* M\_PI / N;  // Split input into two 4-point sequences  for (k = 0; k < 4; k++) {  G[k][0] = input[2 \* k][0];  G[k][1] = input[2 \* k][1];  H[k][0] = input[2 \* k + 1][0];  H[k][1] = input[2 \* k + 1][1];  }  // Perform 4-point FFT on both sequences  FFT\_4\_Point(4, G, G);  FFT\_4\_Point(4, H, H);  // Combine the results  for (k = 0; k < 4; k++) {  output[k][0] = G[k][0] + (H[k][0] \* cos(angle \* k) + H[k][1] \* sin(angle \* k));  output[k][1] = G[k][1] + (H[k][1] \* cos(angle \* k) - H[k][0] \* sin(angle \* k));  output[k + 4][0] = G[k][0] + (H[k][0] \* cos(angle \* (k + 4)) + H[k][1] \* sin(angle \* (k + 4)));  output[k + 4][1] = G[k][1] + (H[k][1] \* cos(angle \* (k + 4)) - H[k][0] \* sin(angle \* (k + 4)));  }  }  void Inverse\_FFT(int N, float input[SIZE][2], float output[SIZE][2]) {  int i;  float temp[SIZE][2];  // Take the conjugate of the input  for (i = 0; i < N; i++) {  temp[i][0] = input[i][0];  temp[i][1] = -input[i][1];  }  // Perform FFT on the conjugate  if (N == 4) {  FFT\_4\_Point(N, temp, output);  } else {  FFT\_8\_Point(N, temp, output);  }  // Take the conjugate again and divide by N  for (i = 0; i < N; i++) {  output[i][0] = output[i][0] / N;  output[i][1] = -output[i][1] / N;  }  } |
| **RESULT:**  **CASE 1 : FOUR POINT FFT AND IFFT**    **CASE 2 : EIGHT POINT FFT AND IFFT** | |
| **CONCLUSION:** | **1. Computational Efficiency in DFT**  **For N=4N :**   * **Total Real Multiplications:** 4×4^2 = 64 * **Total Real Additions:** 4×4^2−2×4=64−8=56   **For N=8N :**   * **Total Real Multiplications:** 4×82=256 * **Total Real Additions:** 4×8^2−2×8=256−16=240   **2. Computational Efficiency in FFT**  **For N=4:**   * **Total Real Multiplications:** 2×4×log2(4) =2×4×2 =16 * **Total Real Additions:** 3×4×log2(4) =3×4×2 =24   **For N=8 :**   * **Total Real Multiplications:** 2×8×log2(8) =2×8×3 =48 * **Total Real Additions:** 3×8×log2(8) =3×8×3 =72   **3. FFT Performance**  **The FFT produces fast results due to:**   * **Less Computations:** The FFT significantly reduces the number of multiplications and additions compared to the DFT. For N=4N and N=8, the reduction in operations is substantial.   The FFT dramatically reduces computational complexity compared to the DFT. For small N like 4 and 8, the difference is clear and substantial. As N increases, the efficiency of the FFT becomes even more pronounced, making it the preferred choice for larger datasets. The FFT achieves this efficiency through a reduction in the total number of computations and the potential for parallel processing, which enhances performance in practical applications. |