./

Report: N-Point Fast Fourier Transform Calculator

Course Code: <CODE>



Version Number:

Team Members :

Team No:

Module: Model Based System Engineering

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Ver. Rel. No.** | **Release Date** | **Prepared. By** | **Reviewed By** | **Approved By** | **Remarks/Revision Details** |
| 1.0.0 | 09-25-20 | Megha Annie George  SF ID: 105296 |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

**Document History**

# 

# Abstract

“Fast Fourier Transform (FFT) is the most important numerical algorithm of our lifetime”, says Gilbert Strang.

It is a computer algorithm which changes the time domain signal into the frequency domain by quickly performing a discrete Fourier transform (DFT) on the signal. There are myriad uses of FFT namely in engineering, music, science, and mathematics.

For instance, it is used in digital signal processing (DSP) to modify, filter and decode digital audio, video and images. It is the foundation for voice recognition and myriad other pattern recognition and image compression applications.

A live example would be, noise-cancelling headphones using FFT to turn unwanted sounds into simple waves so that inverse signals can be generated to cancel them or FFTs being used to sharpen edges and create effects in static images.

Considering the extreme importance of the FFT algorithm in the industry, this project focusses on the implementation of the Cooley-Tukey FFT algorithm, to convert a digital signal (x) with length (N) from the time domain into a signal in the frequency domain (X). This is commonly known as the N-point radix-2 DIT FFT. The N-point FFT is written in the C programming language and it takes the values of the signal sequence as user inputs along with the value of N. The output would be a discrete Fourier transformed sequence of values corresponding to the user input sequence. In short, the output would be the FFT of the user input.

Contents

[Abstract 2](#_Toc51966252)

[List of Tables 5](#_Toc51966253)

[List of Figures 6](#_Toc51966254)

[1. Objective 7](#_Toc51966255)

[1.1 Problem Statement 7](#_Toc51966256)

[1.2 Description 7](#_Toc51966257)

[2. Fast Fourier Transform 8](#_Toc51966258)

[2.1 Overview 8](#_Toc51966259)

[2.2 Types of FFT 8](#_Toc51966260)

[2.3 Famous FFT Algorithms 8](#_Toc51966261)

[2.4 Applications of FFT 8](#_Toc51966262)

[2.5 Advantages of FFT 9](#_Toc51966263)

[3. Requirements 10](#_Toc51966264)

[3.1 Functional Requirements 10](#_Toc51966265)

[3.2 Non-Functional Requirements 10](#_Toc51966266)

[3.3 Software Requirements 11](#_Toc51966267)

[4. Design 12](#_Toc51966268)

[4.1 Design Constraints 12](#_Toc51966269)

[4.2 High Level Design 12](#_Toc51966270)

[4.2.1 Activity Diagram 12](#_Toc51966271)

[4.2.2 Use Case Diagram 13](#_Toc51966272)

[4.3 Low Level Design 13](#_Toc51966273)

[5. Test Plan 14](#_Toc51966274)

[5.1 Test Strategy 14](#_Toc51966275)

[5.1.1 Test Scope 14](#_Toc51966276)

[5.1.2 Test Type 15](#_Toc51966277)

[5.2 Test Objective 15](#_Toc51966278)

[5.3 Test Criteria 15](#_Toc51966279)

[5.3.1 Suspension Criteria 15](#_Toc51966280)

[5.3.2 Exit Criteria 15](#_Toc51966281)

[5.4 Test Environment 15](#_Toc51966282)

[6. Test Cases 16](#_Toc51966283)

[7. Expected Result 17](#_Toc51966284)

[8. Conclusion 18](#_Toc51966285)

[8.1 Future Scope 18](#_Toc51966286)

[9. Reference 19](#_Toc51966287)

# List of Tables

[Table 1 Functional Requirement Table 10](#_Toc51966415)

[Table 2 Non-Functional Requirement Table 10](#_Toc51966416)

[Table 3 Software Requirement Table 11](#_Toc51966417)

[Table 4 Features to be tested 14](#_Toc51966418)

[Table 5 Features not to be tested 14](#_Toc51966419)

[Table 6 Test Cases 16](#_Toc51966420)

[Table 7 Expected Results 17](#_Toc51966421)

# 

# List of Figures

[Figure 1 Activity Diagram 12](#_Toc51966436)

[Figure 2 Use Case Diagram 13](#_Toc51966437)

# 1. Objective

## Problem Statement

The main aim of this project is to implement an N-point Fast Fourier Transform Calculator in C programming language.

## Description

This project focusses on the implementation of the Cooley-Tukey FFT algorithm, to convert a digital signal (x) with length (N) from the time domain into a signal in the frequency domain (X). This is commonly known as the N-point radix-2 DIT FFT. The N-point FFT is written in the C programming language. The total count of samples along with the value of samples would act as user inputs. The user would be able to enter samples of arbitrary length. Both real and complex numbers would be processed with ease. Data samples with even and odd symmetries would also be considered as inputs. And finally, the FFT is computed on the user input data and the same is displayed back to the user.

# 2. Fast Fourier Transform

## Overview

A fast Fourier transform (FFT) is an algorithm that computes the discrete Fourier transform (DFT) of a sequence, or its inverse (IDFT). Fourier analysis converts a signal from its original domain (often time or space) to a representation in the frequency domain and vice versa. The DFT is obtained by decomposing a sequence of values into components of different frequencies. This operation is useful in many fields, but computing it directly from the definition is often too slow to be practical. An FFT rapidly computes such transformations by factorizing the DFT matrix into a product of sparse (mostly zero) factors. As a result, it manages to reduce the complexity of computing the DFT from, which arises if one simply applies the definition of DFT, to, where is the data size. There are many different FFT algorithms based on a wide range of published theories, from simple complex-number arithmetic to group theory and number theory. The best-known FFT algorithms depend upon the factorization of N, but there are FFTs with O(N log N) complexity for all N, even for prime N. Many FFT algorithms only depend on the fact that is an N-th primitive root of unity, and thus can be applied to analogous transforms over any finite field, such as number-theoretic transforms. Since the inverse DFT is the same as the DFT, but with the opposite sign in the exponent and a 1/N factor, any FFT algorithm can easily be adapted for it.

## Types of FFT

* + 1. Discrete Fourier Transform
    2. Inverse Discrete Fourier Transform

## Famous FFT Algorithms

* + 1. Cooley–Tukey FFT algorithm
    2. Prime-factor FFT algorithm
    3. Bruun's FFT algorithm
    4. Rader's FFT algorithm
    5. Bluestein's FFT algorithm
    6. Hexagonal fast Fourier transform

## Applications of FFT

* + 1. Used in digital recording, sampling, additive synthesis and pitch correction software.
    2. Fast large-integer and polynomial multiplication
    3. Efficient matrix-vector multiplication for Toeplitz, circulant and other structured matrices
    4. Filtering algorithms (overlap-add and overlap-save methods)
    5. Fast algorithms for discrete cosine or sine transforms (e.g. fast DCT used for JPEG and MPEG/MP3 encoding and decoding)
    6. Fast Chebyshev approximation
    7. Solving difference equations
    8. Computation of isotopic distributions.

## Advantages of FFT

FFT reduce the complexity of computing the DFT from {\displaystyle O\left(N^{2}\right)}O(N2), which arises if one simply applies the definition of DFT, to {\displaystyle O(N\log N)}O(N log N){\displaystyle N}. The difference in speed can be enormous, especially for long data sets where N may be in the thousands or millions. In the presence of round-off error, many FFT algorithms are much more accurate than evaluating the DFT definition directly or indirectly.

# 3. Requirements

This part of the document focusses on the functional / non-functional and software requirements necessary for the smooth working of the N-point FFT calculator.

## Functional Requirements

Table Functional Requirement Table

|  |  |
| --- | --- |
| Functional Requirement ID | Description |
| FR01 | The program shall be able to compute the FFT for arbitrary lengths of inputs. |
| FR02 | The program shall be able to compute the FFT for real numbers. |
| FR03 | The program shall be able to compute the FFT for complex numbers. |
| FR04 | The program shall be able to compute the FFT for data samples with even symmetry. |
| FR05 | The program shall be able to compute the FFT for data samples with odd symmetry. |

## Non-Functional Requirements

Table Non-Functional Requirement Table

|  |  |
| --- | --- |
| Non-Functional Requirement ID | Description |
| NFR01 | The user should be flagged in-case any mistake occurs in the user input. |
| NFR02 | The output (FFT computed) should be displayed in an unambiguous manner. |

## Software Requirements

Table Software Requirement Table

|  |  |  |
| --- | --- | --- |
| Software Requirement ID | Description | Name |
| SWR01 | Operating System | Windows / Linux / Mac |
| SWR02 | Compiler | MinGW |
| SWR03 | IDE | CodeBlocks |

# 4. Design

This part of the document focusses on the details regarding the design implementation of the N-point FFT calculator.

## Design Constraints

* + 1. The FFT is only suitable for periodic signals.
    2. The sampled signal segment must contain a whole number of periods.

## High Level Design

### Activity Diagram

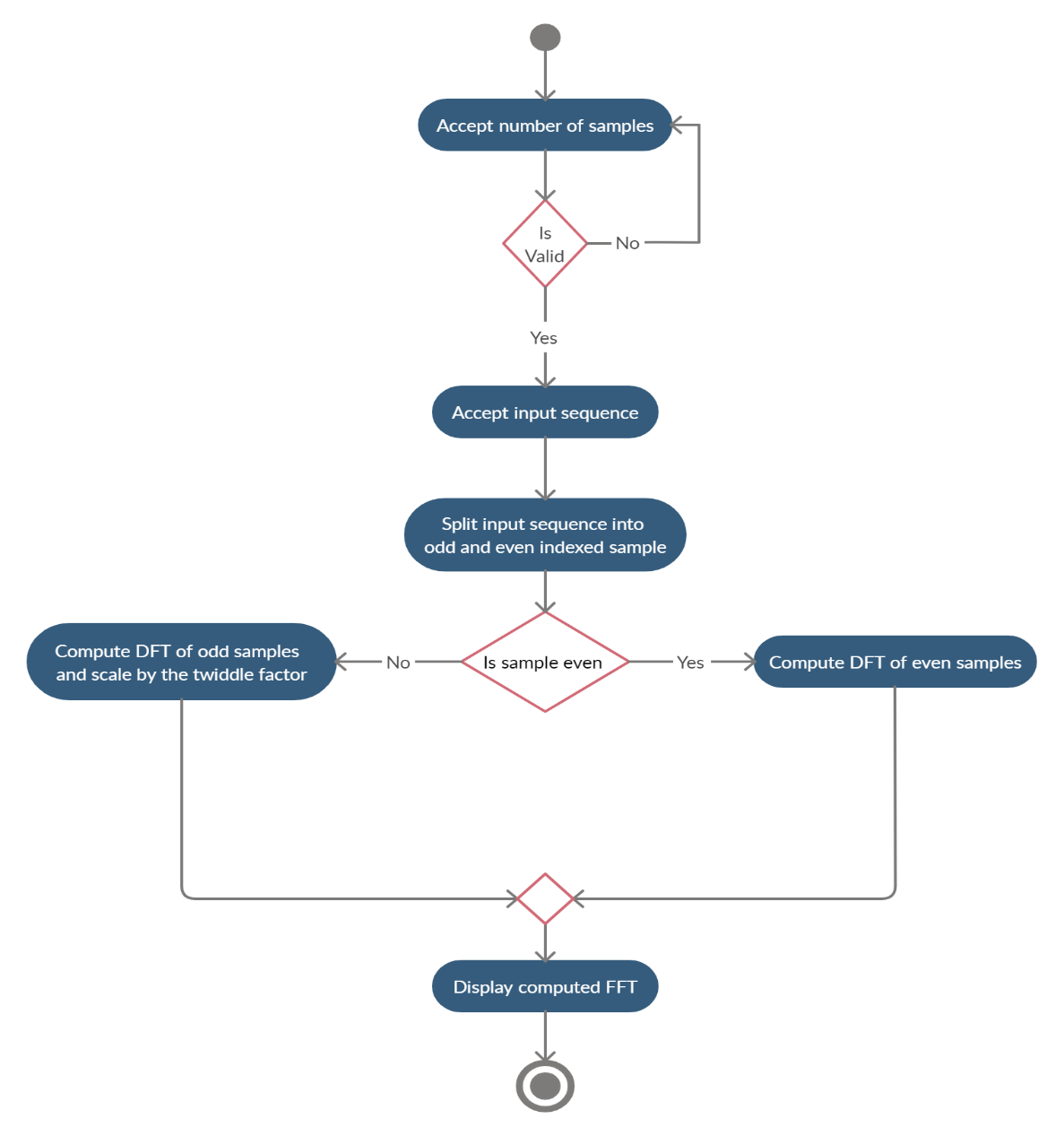


Figure Activity Diagram

### Use Case Diagram

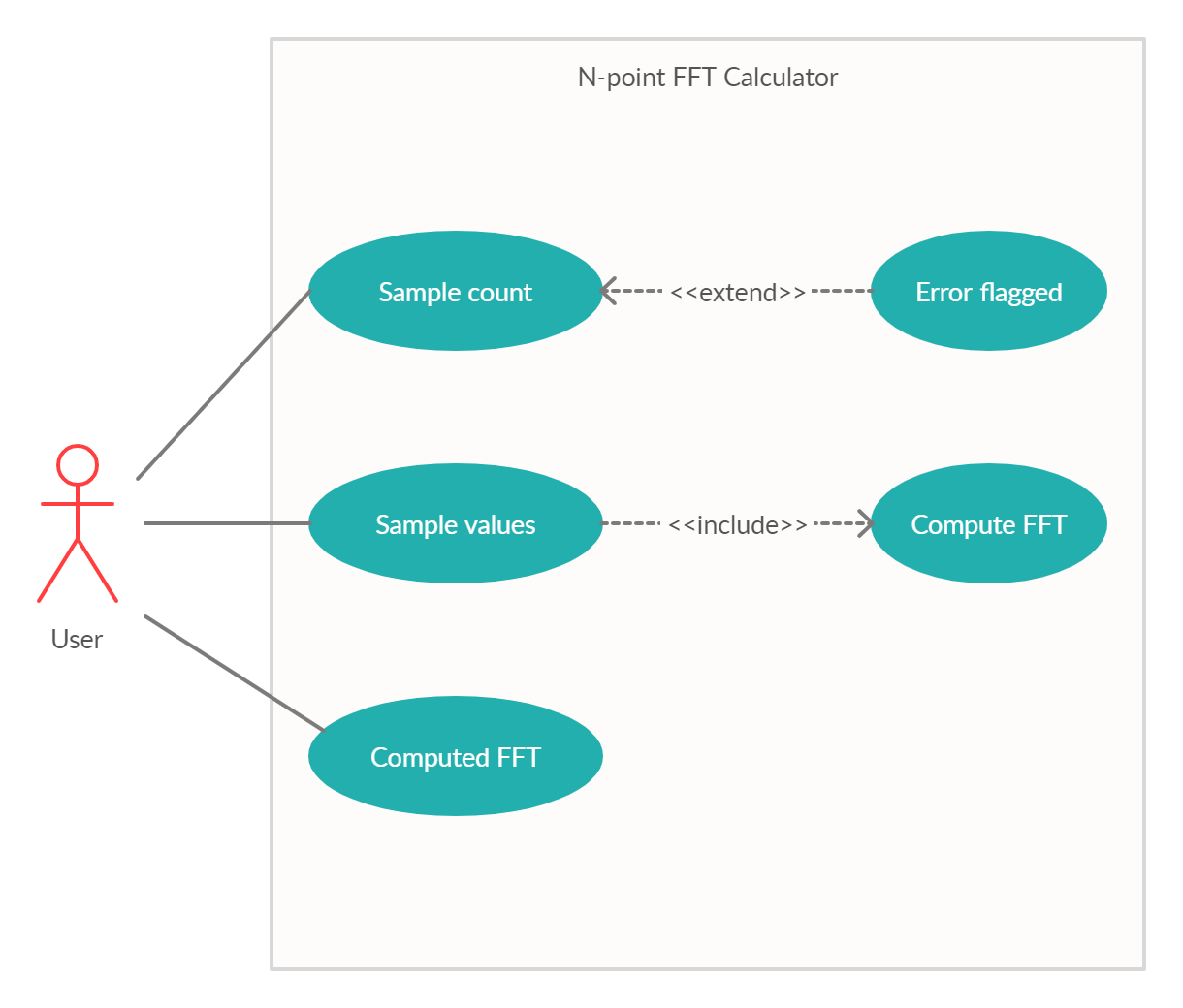


Figure Use Case Diagram

## Low Level Design

The FFT program uses a standard three-loop structure for the main FFT computation. The outer loop steps through the stages, the middle loop steps through "flights" (butterflies with the same twiddle factor from each short-length DFT at each stage), and the inner loop steps through the individual butterflies. This ordering minimizes the number of fetches or computations of the twiddle-factor values. Since the bit-reverse of a bit-reversed index is the original index, bit-reversal can be performed fairly simply by swapping pairs of data.

# 5. Test Plan

This part of the document focusses on planning the test for the N-point FFT calculator.

## Test Strategy

The main aim of the test strategy is to check whether the C program on N-point FFT calculator produce error free results.

### Test Scope

#### Features to be Tested

Table Features to be tested

|  |  |
| --- | --- |
| Feature ID | Description |
| FW01 | Input samples can be of arbitrary lengths. |
| FW02 | Inputs samples can be real or complex valued. |
| FW03 | Input samples can be of even symmetry or odd symmetry. |

#### Features not to be Tested

Table Features not to be tested

|  |  |
| --- | --- |
| Feature ID | Description |
| FW04 | The FFT output should be displaced in a neat and unambiguous manner. |
| FW05 | Inputs shall be valid. |

#### Assumptions

1. The sampled signal segment is repeated periodically for an infinite period of time
2. The length of the input data sequence should be a power of 2.

### Test Type

1. Unit Testing: To test the smallest piece of verifiable software in the application.
2. System Test: Conducted on a complete, integrated system to evaluate the systems compliance with its specified requirements.

## Test Objective

The objective of the test is to find as many software defects as possible, thus to ensure that the software under test is bug free before release. Here, the program should be able to compute an error free FFT output sequence for any arbitrary length of the user input, for real / complex input data and for even or odd datasets.

## Test Criteria

### Suspension Criteria

If 40% of test cases failed, testing is suspended until the previous cases are fixed.

### Exit Criteria

Run rate - 100%  
Pass rate - greater than or equal to 90%.

## Test Environment

The test is carried out using CUnit – A unit test framework.

# 6. Test Cases

Table Test Cases

|  |  |  |  |
| --- | --- | --- | --- |
| Test Case ID | Test Scenario | Test Steps | Test Data |
| TC01 | Test whether the FFT is computed for arbitrary length of inputs. | 1.Enter the no. of samples.  2. Enter the samples. | 1. 8  2.0, j, 0, 1, 0, j, 0, 1 |
| 1. Enter a different no. of samples.  2. Enter the samples again. | 1. 2  2. 1, 1 |
| TC02 | Test whether the FFT is computed for real numbers. | 1. Enter the no. of samples.  2. Enter real samples. | 1. 4  2. 1, -1, 1, -1 |
| TC03 | Test whether the FFT is computed for complex numbers. | 1. Enter the no. of samples.  2. Enter complex samples. | 1. 4  2. j, 0, -j, 0 |
| TC04 | Test whether the FFT is computed for data samples with even symmetry. | 1. Enter the no. of samples.  2. Enter even data samples. | 1. 4  2. -1+1j, 0, 1+j, 0 |
| TC05 | Test whether the FFT is computed for data samples with odd symmetry. | 1. Enter the no. of samples.  2. Enter odd data samples. | 1. 4  2. -1-1j, 0, 1+j, 0 |

# 7. Expected Result

Table Expected Results

|  |  |
| --- | --- |
| Test Case ID | Expected Results |
| TC01 | FFT should be computed for different lengths of data samples. |
| TC02 | FFT should be computed for real data samples. |
| TC03 | FFT should be computed for complex data samples. |
| TC04 | FFT should be computed for data samples with even symmetry. |
| TC05 | FFT should be computed for data samples with odd symmetry. |

# 8. Conclusion

The radix-2 Decimation-In-Time fast Fourier transform written in C programming language is a high-level mathematical algorithm of supreme importance. It is capable of computing the FFT for different kinds of data samples, namely, real and complex data samples, odd and even symmetry data samples etc.

## Future Scope

This C code for FFT computation can be used in the foreseeable future in works like denoising signals, solving complex PDEs, audio/ image/ video compression techniques etc.

# 9. Reference

* + 1. Saidi, Ali. "Decimation-in-time-frequency FFT algorithm." *Proceedings of ICASSP'94. IEEE International Conference on Acoustics, Speech and Signal Processing*. Vol. 3. IEEE, 1994.
    2. Qadeer, Shaik, Mohammed Zafar Ali Khan, and Syed Abdul Sattar. "A Radix-2 DIT FFT with reduced arithmetic complexity." *2014 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*. IEEE, 2014.
    3. Meyer, Raimund. "Error analysis and comparison of FFT implementation structures." *International Conference on Acoustics, Speech, and Signal Processing,*. IEEE, 1989.

### 