FFT Library

Yuzhi Zhao

December 19, 2017

1 Revision History

Date	Version	Notes
Date 1	1.0	Notes
Date 2	1.1	Notes

2 Symbols, Abbreviations and Acronyms

symbol	**description**
T	Test
FFT	Fast Fourier Transform
IFFT	Inverse Fast Fourier Transform
CA	Commonality Analysis
IM	Instance Module
MSE	Mean Squared Error
o_i	Output Data
e_i	Expected Output Data

Contents

List of Tables

[If there are no tables, you can comment out the above command —SS]

List of Figures

3 General Information

The following section provides an overview of the Verification and Validation (V & V) Plan for a Fast Fourier Transform (FFT) library.

3.1 Purpose

The main purpose of this document is to describe the verification and validation process that will be used to test a FFT Library. This document is intended to be used as a reference for all future testing and will be used to increase confidence in the software implementation.
This document will be used as a starting point for the verification and validation report. The test cases presented within this document will be executed and the output will be analyzed to determine if the library is implemented correctly.
All reference documentation and actual implementation files can be found at https://github.com/741ProjectFFT/FFT.

3.2 Scope

The whole library includes eight FFT or IFFT calculation functions. All tests should be applied based on this scope.

3.3 Overview of Document

The following sections provides more details about the V&V of a FFT Library. Information about verification tools, automated testing approaches will be stated. And test cases for all system testing and part of unit testing will be provided.

4 Plan

4.1 Software Description

The FFT Library is a library for calculating Discrete Fourier Transform (DFT) more efficiently. Users are responsible to call the right function which takes different parameters and calls different algorithms. The software being tested is 8 different FFT calculation functions.

4.2 Test Team

Yuzhi Zhao

4.3 Automated Testing Approach

A unit testing framework will be implemented in both unit testing and system testing.
All the test cases in test suite will be built in main function which is also a driver to execute this library.
Test coverage analysis will be applied to measure code coverage.
Compiler can do syntax check automatically.

4.4 Verification Tools

This library will be written in C language. All the verification tools below were chosen based on implemented language.

1. Cutest as unit testing framework
2. Xcover as coverage analysis tool

4.5 Non-Testing Based Verification

1. Symbolic Execution
   Because FFT library is based on a mathematical expression. Using Symbolic Execution can trace the path and the result can be compared with mathematical expression directly.
   Symbolic Execution will be tested using CREST, references https://github.com/jburnim/crest.

5 System Test Description

5.1 Tests for Functional Requirements

5.1.1 Calculation Test

Radix-2 Complex Number Calculation Function

1. **T-1:Radix-2 Complex Number FFT Calculation Function**

   Type: Functional, Dynamic, Automated

   Initial State: None

   Input:
   input.txt : Includes all the input datas. The example of input.txt named "test1.txt" is shown in https://github.com/741ProjectFFT/FFT/tree/master/Doc/TestPlan. The floating numbers can be generated by random number generator online. The source can be reached using http://www.meridianoutpost.com/resources/etools/calculators/generator-random-real.php?

   expectedOutput.txt: Includes the output datas using the same input datas but computed by Matlab FFT library. Then expectedOutput.txt named "test1_out.txt" is shown in https://github.com/741ProjectFFT/FFT/tree/master/Doc/TestPlan.

   Output:
   output.txt : Includes the output datas using the input data computed by this FFT library.
   TestResult: pass or not pass. Whether the program passed the test is measured by an admissible error and the Mean Squared Error will be used as the algorithm. The equation is provided below:

$$MSE = \frac{1}{n}\sum_{i=0}^{n-1}(o_i - e_i)^2 \tag{1}$$

   e means expected output.
   o means this library's output.
   i means the index of the output sequence.
   n means the number of output values.
   MSE will be implemented as the measure of the test result in all functional test cases. The specific number of MSE should vary depending on a specific case.

   How test will be performed:
   Automated.

For validation purpose, datas should also be compared with results from normal DFT calculations as well. Do the same test as above but fill the output.txt with results from using DFT library.

2. **T-2:Radix-2 Complex Number IFFT Calculation Function**

   Type: Functional, Dynamic, Automated

   Initial State: None

   Input:
   input.txt : Includes all the input datas. The example of input.txt named "test2.txt" is shown in https://github.com/741ProjectFFT/FFT/tree/master/Doc/TestPlan.

   expectedOutput.txt: Includes the output datas using the same input datas but computed by Matlab FFT library. Then expectedOutput.txt named "test2_out.txt" is shown in https://github.com/741ProjectFFT/FFT/tree/master/Doc/TestPlan.

   Output:
   output.txt : Includes the output datas using the input data computed by this IFFT library.
   TestResult: pass or not pass. Whether the program passed the test is measured by an admissible error and the algorithm is same as it in T-1.
   How test will be performed:
   Same as above.

Radix-2 Real Number Calculation Function

1. **T-3:Radix-2 Real Number FFT Calculation Function**

   Type: Functional, Dynamic, Automated

   Initial State: None

   Input:
   input.txt : Includes all the input datas. The example of input.txt named "test3.txt" is shown in https://github.com/741ProjectFFT/FFT/tree/master/Doc/TestPlan.

expectedOutput.txt: Includes the output datas using the same input datas but computed by Matlab FFT library. Then expectedOutput.txt named "test3_out.txt" is shown in https://github.com/741ProjectFFT/FFT/tree/master/Doc/TestPlan.

Output:
output.txt : Includes the output datas using the input data computed by this FFT library.
TestResult: pass or not pass. Whether the program passed the test is measured by an admissible error and the algorithm is same as it in T-1.

How test will be performed:
Same as above.

2. **T-4:Radix-2 Real Number IFFT Calculation Function**

   Type: Functional, Dynamic, Automated

   Initial State: None

   Input:
   input.txt : Includes all the input datas. The example of input.txt named "test4.txt" is shown in https://github.com/741ProjectFFT/FFT/tree/master/Doc/TestPlan.

   expectedOutput.txt: Includes the output datas using the same input datas but computed by Matlab FFT library. Then expectedOutput.txt named "test4_out.txt" is shown in https://github.com/741ProjectFFT/FFT/tree/master/Doc/TestPlan.

   Output:
   output.txt : Includes the output datas using the input data computed by this IFFT library.
   TestResult: pass or not pass. Whether the program passed the test is measured by an admissible error and the algorithm is same as it in T-1.

   How test will be performed:
   Same as above.

Radix-3 Complex Number Calculation Function

1. **T-5:Radix-3 Complex Number FFT Calculation Function**

   Type: Functional, Dynamic, Automated

   Initial State: None

   Input:
   input.txt: Includes all the input datas. The example of input.txt named "test5.txt" is shown in https://github.com/741ProjectFFT/FFT/tree/master/Doc/TestPlan.
   expectedOutput.txt: Includes the output datas using the same input datas but computed by Matlab FFT library. Then expectedOutput.txt named "test5_out.txt" is shown in https://github.com/741ProjectFFT/FFT/tree/master/Doc/TestPlan.

   Output:
   output.txt : Includes the output datas using the input data computed by this FFT library.
   TestResult: pass or not pass. Whether the program passed the test is measured by an admissible error and the algorithm is same as it in T-1.

   How test will be performed:
   Same as above.

2. **T-6:Radix-3 Complex Number IFFT Calculation Function**

   Type: Functional, Dynamic, Automated

   Initial State: None

   Input:
   input.txt: Includes all the input datas. The example of input.txt named "test6.txt" is shown in https://github.com/741ProjectFFT/FFT/tree/master/Doc/TestPlan.
   expectedOutput.txt: Includes the output datas using the same input datas but computed by Matlab FFT library. Then expectedOutput.txt named "test6_out.txt" is shown in https://github.com/741ProjectFFT/FFT/tree/master/Doc/TestPlan.

Output:
output.txt : Includes the output datas using the input data computed by this FFT library.
TestResult: pass or not pass. Whether the program passed the test is measured by an admissible error and the algorithm is same as it in T-1.

How test will be performed:
Same as above.

Radix-3 Real Number Calculation Function

1. **T-7:Radix-3 Real Number FFT Calculation Function**

   Type: Functional, Dynamic, Automated

   Initial State: None

   Input:
   input.txt: Includes all the input datas. The example of input.txt named "test7.txt" is shown in https://github.com/741ProjectFFT/FFT/tree/master/Doc/TestPlan.
   expectedOutput.txt: Includes the output datas using the same input datas but computed by Matlab FFT library. Then expectedOutput.txt named "test7_out.txt" is shown in https://github.com/741ProjectFFT/FFT/tree/master/Doc/TestPlan.

   Output:
   output.txt : Includes the output datas using the input data computed by this FFT library.
   TestResult: pass or not pass. Whether the program passed the test is measured by an admissible error and the algorithm is same as it in T-1.

   How test will be performed:
   Same as above.

2. **T-8:Radix-3 Real Number IFFT Calculation Function**

   Type: Functional, Dynamic, Automated

   Initial State: None

Input:
input.txt: Includes all the input datas. The example of input.txt named "test1.txt" is shown in https://github.com/741ProjectFFT/FFT/tree/master/Doc/TestPlan.
expectedOutput.txt: Includes the output datas using the same input datas but computed by Matlab FFT library. Then expectedOutput.txt named "test1_out.txt" is shown in https://github.com/741ProjectFFT/FFT/tree/master/Doc/TestPlan.

Output:
output.txt : Includes the output datas using the input data computed by this FFT library.
TestResult: pass or not pass. Whether the program passed the test is measured by an admissible error and the algorithm is same as it in T-1.

How test will be performed:
Same as above.

[You did not include the tests suggested by Alex and Isobel in class. All of your tests rely on parallel testing with a comparison to Matlab. That is fine, but if you make an error, those tests won't help much with tracking it down. You should also include tests where you know the solution theoretically. Generate your initial data with a sine function and then verify that you get the expected frequency back. A more complex test would be like that given on the Wikipedia page for FFT in the figure at the top of the page (https://en.wikipedia.org/wiki/Fast_Fourier_transform) —SS]

5.2 Tests for Nonfunctional Requirements

5.2.1 Speed Comparation Test

1. **T-9:Compare Calculation Speed with FFT Library in Matlab**

   Type: Dynamic, automated, Manual

   Initial State: None

   Input: speed_test1.txt, speed_test2.txt, speed_test3.txt, speed_test4.txt. speed_test5.txt, speed_test6.txt

Output: Time1, Time2, Time3, Time4, Time5, Time6 corresponding to the different input files with different number of inputs. And get the same information from using Matlab doing calculation. Call them MTime1, MTime2, MTime3, MTime4, MTime5, MTime6.

How test will be performed:
Compare each pair of time from this FFT library and Matlab FFT library. See how much difference with them.

5.2.2 Loading Library Test

1. **T-10:Under Win X86, Mac OS platform**

   Type: Functional, Dynamic, Manual

   Initial State: None

   Input: input.txt(can be chosen from any input.txt above mentioned and call the corresponding function.) to an C Language compiler.

   Output: output.txt

   How test will be performed: Manual
   If the test case is run successfully, it means that this library can be loaded successfully.

2. **T-11:Different Compilers Under The Same Platform**

   Type: Functional, Dynamic, Manual

   Initial State: None

   Input: input.txt(can be chosen from any input.txt above mentioned and call the corresponding function.) to different compilers including different versions and different languages.

   Output: output.txt

   How test will be performed: Manual
   If the test case is run successfully, it means that this library can be loaded successfully.

5.3 Traceability Between Test Cases and Requirements

Since CA reference https://github.com/741ProjectFFT/FFT/tree/master/Doc/SRS does not include requirements part, the Test Cases will be relevant

to IM.
T-1, T-2, T-3, T-4 all relevant to IM1 in CA.
T-5, T-6, T-7, T-8 all relevant to IM2 in CA.

6 Unit Testing Plan

1. **T-12: TestgetComplexNumber**

   Type: Functional, Dynamic

   Initial State: None

   Input: 1.1, 2.2

   Output: check the return value called 'a' in ComplexNum struct, a.real = 1.1, a.img = 2.2

   How test will be performed: Automated Unit Test

2. **T-13:TestfakeImageTerm**

   Type: Functional, Dynamic

   Initial State: None

   Input: 10 as size

   Output: array = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0}

   How test will be performed: Automated Unit Test

3. **T-14: TestComplex_Multi**

   Type: Functional, Dynamic

   Initial State: None

   Input: com1.real = 5.3, com1.img = 5.3, com2.real = 2.0, com2.img = 2.0

   Output: output.real = 0, output.img = 21.2

   How test will be performed: Automated Unit Test

4. **T-15: Testload_data_real**

   Type: Functional, Dynamic

Initial State: None

Input: filename = "input_Real.txt" with numbers 321, 438, 128, 75, 94, 237, 85, 190, 235, 169, 42, 173, 376, 468, 3, 289, 149, 267, 458, 404, 291, 66, 415, 387, 64, 366, 372, 2, 205, 297, 329, 117 in it.
double* pRealNumber;
long lInputSize = 0;

Output: pRealNumber = {321, 438, 128, 75, 94, 237, 85, 190, 235, 169, 42, 173, 376, 468, 3, 289, 149, 267, 458, 404, 291, 66, 415, 387, 64, 366, 372, 2, 205, 297, 329, 117}

How test will be performed: Automated Unit Test

5. **T-16:Testload_data_complex**

   Type: Functional, Dynamic

   Initial State: None

   Input: filename = "input_Real.txt" with numbers 321, 438, 128, 75, 94, 237, 85, 190, 235, 169, 42, 173, 376, 468, 3, 289, 149, 267, 458, 404, 291, 66, 415, 387, 64, 366, 372, 2, 205, 297, 329, 117 as real number set, 262, 410, 339, 131, 206, 380, 99, 285, 18, 349, 228, 71, 463, 452, 56, 358, 275, 422, 485, 144, 219, 392, 112, 431, 163, 495, 240, 217, 476, 465, 369, 371 as image number set.
   double* pRealNumber;
   double* pImageNumber;
   long lInputSize = 0;

   Output: pRealNumber = {321, 438, 128, 75, 94, 237, 85, 190, 235, 169, 42, 173, 376, 468, 3, 289, 149, 267, 458, 404, 291, 66, 415, 387, 64, 366, 372, 2, 205, 297, 329, 117}
   pImageNumber = {321, 438, 128, 75, 94, 237, 85, 190, 235, 169, 42, 173, 376, 468, 3, 289, 149, 267, 458, 404, 291, 66, 415, 387, 64, 366, 372, 2, 205, 297, 329, 117}

   How test will be performed: Automated Unit Test

6. **T-17:TesttwiddleN**

   Type: Functional, Dynamic

   Initial State: None

Input:
k = 1
N = 4
Output: output = 0 - 1i

How test will be performed: Automated Unit Test

7. **T-18:TesttwiddleP**

   Type: Functional, Dynamic

   Initial State: None

   Input:
   k = 1
   N = 4
   Output: output = 0 + 1i

   How test will be performed: Automated Unit Test

8. **T-19:TestComplex_Plus**

   Type: Functional, Dynamic

   Initial State: None

   Input: input1 = 5.3 + 5.3i, input2 = 2.0 + 2.0i

   Output:output = 7.3 + 7.3i

   How test will be performed: Automated Unit Test

9. **T-20:TestComplex_Minus**

   Type: Functional, Dynamic

   Initial State: None

   Input: input1 = 5.3 + 5.3i, input2 = 2.0 + 2.0i

   Output:output = 3.3 + 3.3i

   How test will be performed: Automated Unit Test