CHALMERS

Bench Test of a Satellite Signal Processing Algorithm Based on a Space-qualified Board

Master's Thesis in the Master Degree Program, Communication Engineering

HENG YIN
LU LI

Department of Signals and Systems
Division of Communication Systems
CHALMERS UNIVERSITY OF TECHNOLOGY
Göteborg, Sweden, 2011

“Productivity is never an accident. It is always the result of a commitment to excellence, intelligent planning, and focused effort.”

Paul J. Meyer
American entrepreneur and author

Abstract

In the traditional satellite signal processing algorithm development, the algorithm developer simulates and optimizes the algorithm in MATLAB and then the software department translates the algorithm into the executable software supported by the processor board. To streamline the development procedure, a test bench prototype to conveniently convert the MATLAB algorithm to an executable program and test the algorithm on the target board with LEON2 processor have been be developed.

In this work, the MATLAB add-on product MATLAB Coder was used to automatically convert the MATLAB algorithm to C code which can be built into executable machine code. The algorithm C code converted from MATLAB Coder does not have any configuration for the target board and the real-time operating system RTEMS. In order to supplement these configurations and add the code of time measurement, a bench test program was developed in Eclipse. After the executable machine code was generated, the input data of the algorithm was loaded onto the processor board and the program was executed on board. In the end, the output data of the algorithm was read out and analyzed in MATLAB.

With this test bench MATLAB algorithm, for example FFT, FCM and SIM (the latter two are provided by RUAG AB), can conveniently be tested on the target board. In the end, numerical error of the hardware execution of the algorithm is analyzed and the execution time is measured. According to the result, the fixed length FFT has a higher speed than the general length FFT; the floating point FFT is faster and more accurate than the fixed point FFT. Moreover, the execution time of the basic operations of the FFT algorithm was investigated. Besides the hardware FPU was proved to shorten the execution time compared to the software FPU for those floating-point-intensive algorithms. Finally it was approved that for the FFT-based algorithm, the execution time of its MEX file was proportional with that of its machine code executed on the target hardware.

Keyword: embedded system, LEON2, real-time, test bench, Eclipse, MATLAB Coder, FPU, FFT, fixed point, floating point

Acknowledgment

First of all, we would like to express our deepest appreciation to our supervisors Elin Eklund and Thomas Lindgren. They spend a great deal of precious times on us and offered us so many valuable guidelines to help us complete our thesis work. They have never blamed us when we commit some stupid mistakes; on the contrary, they encouraged us and gave us great confidence. Without their help, this thesis work couldn’t be accomplished. In our heart, they are not only our supervisors, but also are our dearest teachers and kind friends.

We also should show our gratitude to our superman Björn Enoksson who has a broad professional knowledge. Almost every time we come across with tricky problems, he is the one who helped us to crack them. Thanks for him taking so much precious time off from his busy schedule and being so patient to us.

We still would like to thank Magnus Mortberg and Martin Johansson who helped us to configure Eclipse and learn how to use the Subversion. Martin Johansson is also the one who established the location of our project in the Subversion repository and helped us to add an important external repository. Anders Carlström showed great interest on our thesis work and gave us a lot of practical suggestions; Henrik Garbergs helped us to use the COLE Tools. We would especially like to mention our friends Nicklas Sahlbom and Petros Petkakis who gave us great joys in the company.

Thank our examiner Henk Wymeersch who were interested in the proposal of this these work, reviewed our report and examined the presentation. Also thank our advisor Wei Yang who guided our producer of completing the master thesis work and reviewed our report.

In the end, great appreciations to our families, without their strong support both substantially and mentally, we would not have the chance to be in Sweden and finish the two-year study. Thanks our dear friends in Chalmers. Special thanks to our beloved ones, Shenzhou Hou and Quan Ge who accompanied us all the way.

Terminology

Bench Test	A test carried out on a platform before it is released for use, to ensure that it works properly
Byte	8 bits of data
Memory Access	The operation of reading or writing memory
MiB	Megabyte
Test Bench	The platform used to perform a test

Abbreviations

ASIC	Application Specific Integrated Circuit
CDT	C/C++ Development Tooling
COCOS	Computer Core Support ASIC
COLE	COCOS and LEON ASIC
CPU	Central Processing Unit
DCL	Debug Communication Link
E-DSU	Enhanced Debug Support Unit
EEPROM	Electrically Erasable Read Only Memory
FFT	Fast Fourier Transform
FPU	Floating Point Unit
HW	Hardware
IDE	Integrated Development Environment
IO	Input/Output
IU	Integer Unit
LEON	32-bit Microprocessor
LEON2	Second version of LEON processor
LEON2-FT	Single Event Upset Tolerant Version of LEON2 Processor
LS UART	Low Speed Universal Asynchronous Receiver Transmitter
MATLAB	Matrix Laboratory
MEX	MATLAB Executable
PROM	Programmable Read Only Memory
RTEMS	Real-Time Executive for Multiprocessor Systems
SDRAM	Synchronous dynamic random access memory
SPARC	Scalable Processor Architecture
SW	Software
RISC	Reduced instruction set computing
SRAM	Static random-access memory

TABLE OF CONTENTS

1 Introduction

1.1 Background

In space communications, signal processing is no doubt an indispensable and crucial component of the whole satellite communication system. Due to the fact that it demands large amounts of highly reliable and particularly accurate scientific computation, it is of great interest to develop modern satellite signal processing algorithms in a mathematical-oriented programming language, for example MATLAB.

In the final application, the algorithm will be executed on a space-qualified processor board using the LEON2 processor. LEON2 is a 32-bit CPU microprocessor core, based on the SPARC-V8 RISC architecture and instruction set. Any processor only supports machine code, which can be built from C code and cannot be built from MATLAB script. Moreover, the on-board execution of an algorithm is probably to have a different performance than its performance on the standard computer where it is developed. Hence, in a traditional flight application production, engineers simulate their algorithm in MATLAB and someone else codes the MATLAB script into C code which can be built into the executable machine code and test the algorithm on board. From time schedule and cost point of view, it is highly interesting to automatically convert the algorithms defined in MATLAB to C code.

To streamline the procedure of development and production of the satellite signal processing algorithm, it is expected to have a way that straight bench tests the performance of a MATLAB algorithm on the processor board without manual code conversion, so that the MATLAB algorithm under development can be optimized based upon the real feedback from time to time before it is finally coded into software.

1.2 Objective

The goal of this thesis project is to develop a prototype of test bench and apply it to conveniently test a satellite signal processing algorithm on the processor board, in the consideration of the practical situation in RUAG.

1.3 Approach

As described in Section 1.1 Background, different from the more general and low-level programming languages such as C, C++ and Fortran, MATLAB is not supported by the embedded systems such as the target processor board. In order to provide better compatibility for the embedded systems, MathWorks^{TM} has produced an add-on product called MATLAB Coder to convert MATLAB code to C code automatically with its built-in MATLAB compiler. Therefore it is used in our project to automatically convert the MATLAB algorithm to C code.

However, the converted C code does not include any program for the hardware configuration or other necessary configurations. In order to make its corresponding machine code executable on the board, a main program to encapsulate it and complement those essential parts is developed in C language. This ‘main program’ is called the bench test program in this report.

1.4 Scope

In this project, a RUAG provided algorithm under development was applied as the test object. Due to the time limitation, not the entire algorithm was tested. It was divided into several smaller functional components so that the bench test can start from the simpler MATLAB algorithms. The algorithms used in our bench test were FFT, FCM and SIM, in which SIM is the most complex algorithm in the entire algorithm. Both the FCM and SIM are based on FFT calculation.

According to practical industry requirement, the most concerned result of the bench test is the execution time. So in our project, the execution time is the main investigated factor. The other concerned factor is the numeric precision of the on-board computation.

1.5 Organization

This paper begins with a brief introduction of the hardware and software used. It then continues to present the detailed procedure of how this test bench system works, followed by the description, analysis and discussion of the bench test results. The future work is proposed in the end.

2 Processor Board overview

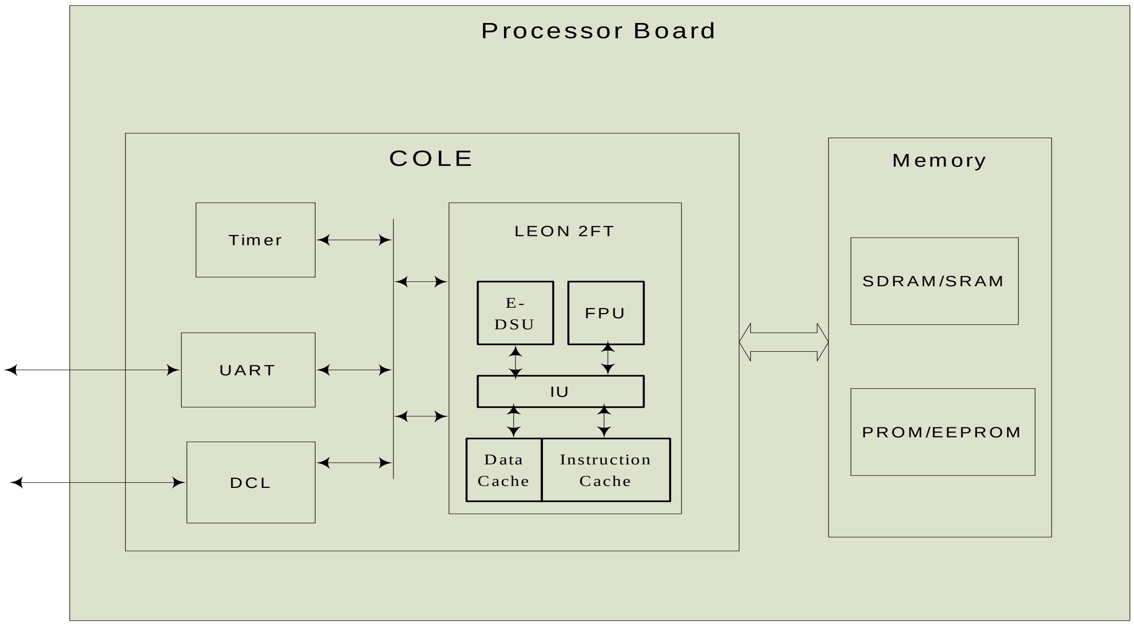


Figure 2-1 Processor board block diagram

The Processor board is equipped with COLE ASIC, acting as a processor and I/O controller. As it is shown in Figure 2-1, the main components used in our project of COLE are the LEON2-FT processor, Memory, the Low Speed Universal Asynchronous Receiver Transmitter (LS UART), the Debug Communication Link (DCL) and a Timer.

The LEON2-FT processor includes an Enhanced Debug Support Unit (E-DSU), an Integer Unit (IU), a Floating Point Unit (FPU) and two kinds of caches. The E-DSU provides hardware debugging support such as breakpoints, watch points and trace facilities. It helps COLEmon (see Section 3.3) to remotely debug. The IU implements the integer arithmetic instructions and computes memory addresses for loads and stores. It also maintains the program counters and controls instruction execution for the FPU [7].The registers in IU are 32-bits wild. FPU executes floating-point calculations and implements load and store instructions for the data transfer between the FPU and memory. The floating-point registers in FPU are 32-bits too. If the FPU is disabled, the floating point operations are emulated by software routines, which in called software FPU. However, the processing speed of software FPU is very slow. There are two types of caches in the LEON2-FT processor. One is 32 Kbytes instruction cache used to store instructions. The other is 16 Kbytes data cache for data storage. If the caches are disabled, the processing time will be triple times larger than it in the enabled mode.

The Memory module is connected with the COLE chip through memory interfaces which include error detection and error correction of memory. There are both volatile memory and non-volatile memory on the processor board. As volatile memory, COLE supports SRAM and SDRAM. The memory sizes used are 8MiB for SRAM and 512MiB for SDRAM. During the thesis work, both of these memory types have been used and their performances are examined. For non-volatile memory, PROM and

EEPROM are used for program and data storage, normally. However, in this thesis work, the test programs are only loaded in RAM (SRAM or SDRAM) and executed directly.

The UART and DCL support hardware handshakes, hardware flow control, registers and memory for Debug Communication Link (DCL).

The Timer unit provides real time information, which can be used to measure the processing time.

3 Software Introduction

3.1 MATLAB with MATLAB Coder

MATLAB is the name for an interactive computing environment as well as a high-level programming language, which could be used for algorithm development, data visualization and data analysis. A huge number of built-in functions are provided by MATLAB that can be directly employed, making the programming straightforward and concise.

MATLAB Coder is an add-on product of MATLAB used to convert MATLAB code to C and C++ code. It supports standard MATLAB language features, including matrix operations, subscripting, program control statements and structures [5]. Cooperating with Simulink Coder and Embedded Coder, MATLAB Coder could generate C codes from Simulink models as well.

Besides generating C and C++ code, MATLAB Coder can also produce MATLAB Executable (MEX) files for both fixed-point and floating-point mathematics, which enables the user to verify the generated C or C++ code in MATLAB. MEX-files are dynamically linked subroutines produced from C, C++ or Fortran source code that, when compiled, can be run from within MATLAB in the same way as MATLAB files or built-in functions [6]. As the MEX-file is based on the same C code generated from MATLAB coder, it’s very useful for the developer to check the C code before moving them to another software development environment.

3.2 Eclipse

Eclipse is an integrated software development environment. It provides a free open environment that employs a plug-in concept to broadly extend its functionality. The plug-in concept, on one hand, allows Eclipse to multiple programming languages. As Eclipse was originally used to develop Java, by using its plug-in based mechanism, the C/C++ Development Tooling (CDT) was added to the basic Eclipse framework to provide C and C++ Integrated Development Environment (IDE). One the other hand, it enables Eclipse to work with other third-party plug-ins, such as the application of COLE Tools, which is introduced later.

3.3 COLE Tools

The COLE Tools is a collection of software tools developed in RUAG that designed to aid the development with the COLE system. As described in Section 2, COLE is the main component of the processor board, so the COLE Tools is used as part of the test ben ch in our project, to get access to the processor board. In the COLE Tools collection, COLE Broker, COLEMon, COLE Inspector and Broker Manager are used in this project.

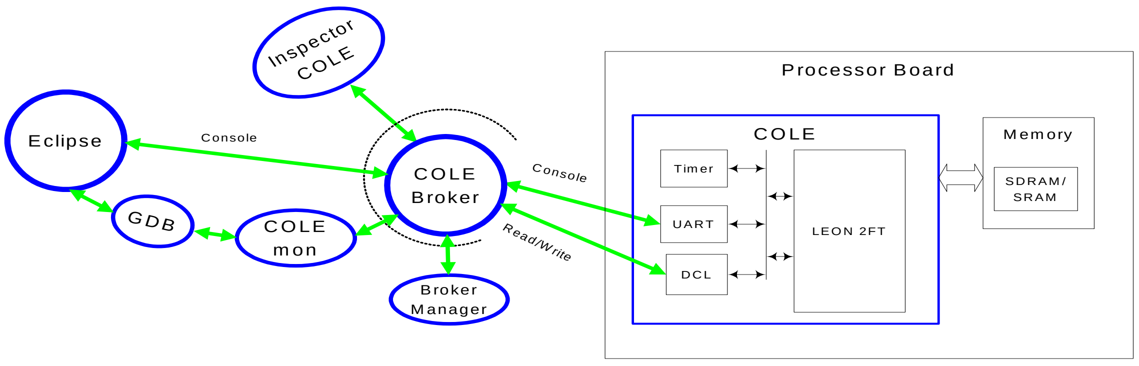


Figure 3-1 The COLE Tools overview [9]

COLE Tools provides a multifunctional development environment, which contains the source level debugging on target, inspection of the memory and registers as well as loading of data or program. As shown in Figure 3-1, the COLE Broker acts like a key anchor for the other COLE Tools to communicate with the processor board. It collects and transmits the commands from the other tools to the COLE and distributes the responses to the waiting tool. Before starting the COLEmon, COLE Launcher or the Inspector COLE, the COLE Broker should be connected to the processor board first. The status of it can be managed by the Broker Manager.

By integrating into the Eclipse framework, the COLE Tools provides an integrated development environment. In Figure 3-1, the Eclipse, GDB, COLEmon together can start or stop debugging and execution of a board-supported program. GDB is the standard GNU debugger. The COLEmon acts like a remote target monitor for remotely debugging. The 'remote' means the system which will run the program is a different type from the one on which the program is developed and built. The Console line in the figure presents the catching of the debug output (from the one of the LS UART port).

For embedded system development, the examination of the registers and memory is very important and useful. Through the Inspector COLE, the real-time register values and memory content can be read out in hexadecimal format. It can also write binary files into the memory and read the memory content out to binary files.

3.4 Real-Time Executive for Multiprocessor Systems (RTEMS)

RTEMS is the short name for a kind of real-time operating system, which provides a set of services for embedded systems. It acts like a bridge to connect two different layers of typical real-time systems. It offers a tool to fit hardware dependencies in the system and also provides a way for application code that accesses them at the same time. It can be grouped to a serial of resource managers, as it is shown in Figure 3-2. They cooperate with each other and provide respective services for real-time systems.

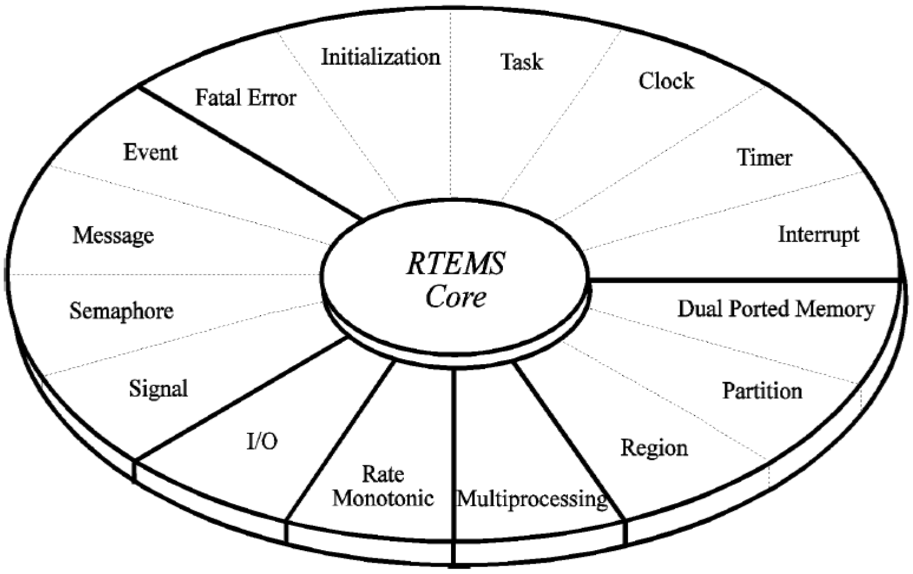


Figure 3-2 The architecture of RTEMS

RTEMS should be configured before using it for any applications. There are several parts that need to be configured: the driver for devices such as clock and console, the length of tick, the maximum of each RTEMS objects (e.g. classic API tasks, semaphores, messages queues and rate monotonic periods) that can be concurrently active, and the initialization tasks. RTEMS provides the rtems/confdefs.h C language header file that contains most of the configuration tables that might be used for applications, so users can just make use of it instead of building these tables by themselves.

In this thesis work, RTEMS can be viewed as a large library that provides a number of supports to the test bench design, for instance, the timer manager in RTEMS could be applied to calculate processing time.

4 Bench Test Implementation

4.1 Composition of the test bench

The test bench in this context refers to the comprehensive platform where the test of a MATLAB algorithm was carried out from a standard computer through a lab computer to the processor board. See Figure 4-1 Composition of the test bench.

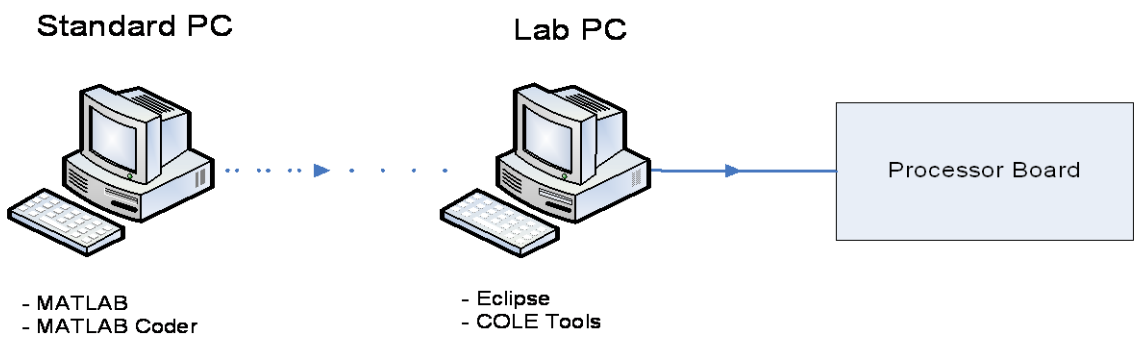


Figure 4-1 Composition of the test bench

The standard computer in the figure above has MATLAB 2011 with MATLAB Coder installed. It can be any standard computer that supports MATLAB. In our project, for accurate comparison, all the bench test result of the algorithm on a standard computer was performed on the same computer with Intel Core Quad CPU, whose clock speed is 3.00 GHZ.

As mentioned in the introduction section, MATLAB is the technical computing software used to develop the satellite signal processing algorithm. A brief introduction of it is in Section 3.1. It is the first software used in the bench testing procedure. In other words, the standard computer is the first platform of the test bench.

For consistency, in the following context, the phrase ‘MATLAB algorithm’ refers to the MATLAB function that implements a certain algorithm. From the test bench’s view, the MATLAB algorithm is the input of the test bench, whose development is supposed to be completed before being fed to the test bench. Hence, in this report there is no detail information about the algorithm development in MATLAB.

The reason why the MATLAB algorithm can not be operated on the processor board is stated as in Section 1.1 Background. Consequently, there is a need to convert the MATLAB algorithm to C code in order to build the corresponding machine code. Also as explained in Section 1.1, in our project, MATLAB Coder was applied to automatically convert the MATLAB algorithm to C code. MATLAB 2011 on the standard computer integrates with this powerful software MATLAB Coder.

The lab computer is the second platform of the test bench. Actually, it is also a standard computer, specified by being physically connected to the processor board. On base of this physical connection, by using the installed Eclipse and the COLE Tools on it, the bench test program can be loaded and the execution can be started on the processor board.

The COLE Tools is a collection of board support software as described in Section 3.3. It is installed on the lab computer. In our project it is the only interface software that is capable for data transmitting between the lab computer and the processor board, which is shown in the figure above.

Eclipse is an open-source plug-in based IDE. A plug-in in Eclipse is a component that provides a certain type of service within the context of the Eclipse workbench [2]. By using this plug-in based mechanism it is suitable for embedded system cross-development. In our project there are two purposes of using Eclipse. One is to edit, compile and build the bench test program, the other one is together with COLE Tools to load and start execution of the bench test program on the processor board. A short description of why a bench test program is needed and what is its functionality is presented in Section 4.2.

In Figure 4-1, the left side of the lab computer is connected to the standard computer. The connecting line is shown dashed because there is no physical data transmitting between these two computers. The connection means the manual work of copying the MATLAB Coder generated C code to the bench test program in Eclipse.

The final platform of the test bench is the processor board. Three components of the processor board were of concern in our project. One is the memory. There are two sizes of the integrated memories, which are 8MiB and 512MiB respectively. Since the MATLAB algorithm involving large data vectors requires more memory space and vice versa, the 8MiB memory was used to begin with. The question about whether there are different time performances between the two memories was investigated. The other factor of concern is the cache in LEON2. Cache is a terminology used in computer science. It is a small block of memory for temporarily storing some data and instructions that probably to be reused by the following operations. Because of its different fabrication process, the cache is of higher accessing speed than the main memory (SRAM and SDRAM). It saves time by reducing the accesses to the slower main memory. The last factor of concern is the Floating Point Unit (FPU) in LEON2. It is a dedicated hardware to accelerate the speed of floating-point operations. If it is used, it is called using the hardware FPU. If this one is not used, the floating point operations are emulated by software routines. In that case, it is called using software FPU. By using the hardware FPU, the operation of the floating-point-intensive calculation, such as the tested algorithm of this project, is expected to achieve a higher speed than using software FPU.

4.2 Bench test program overview

The bench test program is the frame work around the test algorithm. It is a C program used to encapsulate the C code of a MATLAB algorithm as part of it and it makes the whole C program executable and processing time measurable on the processor board. As presented in Section 1.3, the converted C code lacks the necessary hardware configurations, RTEMS configurations and other program related configurations, without which the corresponding machine code can not be executed successfully. The converted C code for the algorithm also lacks the necessary C code to assess the operation time of the algorithm on board. That is also why a bench test program should be developed. In a word, in order to make the algorithm executable and processing time measurable, a bench test program was developed in Eclipse.

Figure 4-2 illustrates the functionality of the bench test program. In a practical bench testing procedure, the MATLAB Coder converts a MATLAB algorithm into the corresponding C code in terms of a set of C files. Those C files are then copied to the Eclipse project of the bench test that contains the bench test program. After some manual configurations in the bench test program to encapsulate the algorithm C code, those C files are built together with all the other files of the bench test program. As shown in Figure 4-2, the timer is started right before the execution of the algorithm and stopped right after the execution of the algorithm. The execution time of that algorithm is obtained by subtracting the start timer from the stop timer. These manual configurations are further described in Section 4.5.2. If the building is successful, a board-supported executable file will be produced.

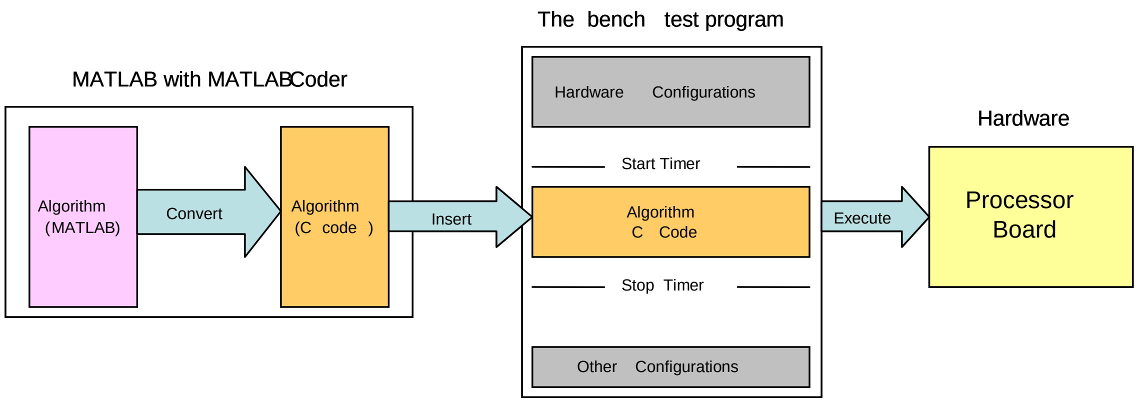


Figure 4-2 Illustration of the functionality of the bench test program

4.3 Frame design of the operation result analysis

In this section, the design of the framework of the operation result analysis will be described. The operation result in our bench test project refers to the execution time and the numerical precision.

As mentioned in Section 1.4, the key index for the algorithm is the execution time of it on the target board. If it takes longer time than the one proposed in the initial specification for that algorithm, the algorithm can not be accepted or should be further optimized. There are three factors that may influence the processing time on board, which are the memory, the cache and the FPU. These three factors are marked with yellow in Figure 4-3. The FPU and the cache can be disabled or enabled manually in the bench test program. If they are enabled, a faster processing time is expected. Therefore the bench test of the same algorithm of different FPUs and cache modes were tested and compared. There are two types of memory, SRAM and SDRAM. Theoretically, it is estimated that there is no difference of the processing time between the two memories. However, it was decided to test and verify it.

The operation time of the MEX algorithm (See Section 3.1) was also measured. It was tested on the standard computer in MATLAB. The result of it was compared with the on-board time of the same algorithm, in order to determine the scale factor between the MEX processing time and the on-board time. If the scalar is fixed, a preliminary evaluation and optimization of the algorithm can be done in MATLAB according to the MEX performance before testing the algorithm on hardware. The working procedure of comparing the MEX operation time and the on-board operation time is illustrated in Figure 4-3.

In addition, besides the total execution time of an algorithm, the memory reading/writing speed was also measured. The time cost of almost every memory related instruction consists of the memory reading/writing time and the CPU calculation time. The CPU calculation time is supposed to be much faster than the memory reading/writing time, which means the memory reading/writing is dominant. Since there are a lot of memory related instructions in the bench test program, it is of worth to investigate how largely the memory reading/writing time will influence the total execution time.

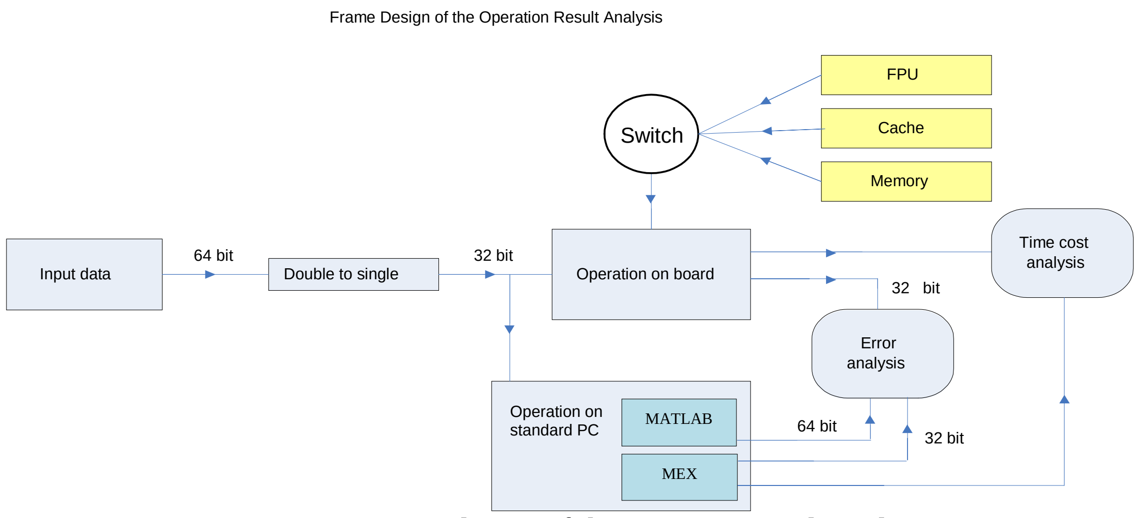


Figure 4-3 Frame design of the operation result analysis

As mentioned in the beginning of this section, the second operation result is the numerical precision. The arithmetic calculations on board are carried out by the processor board’s CPU, LEON2, which is based on the 32-bit SPARC-V8 RISC architecture. 32-bit means the maximum precision of the stored numbers is 32-bit floating point. The standard computer has a maximum precision of 64-bit floating point. The disparity of the maximum precisions is the main affective factor of the numerical precision.

In the bench test, there are three different computation outputs, which are (1) the output of the MATLAB algorithm, (2) the output of the on-board algorithm and (3) the output of the MEX algorithm. Among them, the MATLAB and MEX algorithms were operated on the standard computer. Since the algorithm operated on board and the ones operated on the standard computer are based on the same mathematical algorithm, there should be no difference of the outputs. However, due to the fact that the processor board uses 32-bit architecture, the accuracy of the computation result from the hardware is supposed to be worse than the one from the 64-bit architecture standard computer, although they have the same accuracy of input data. Note that though the MEX algorithm was operated on the standard computer, it was implemented in the way of 32-bit architecture by MATLAB, as shown in the Figure 4-3. So it is estimated to have a similar numerical precision with the on-board algorithm. In our project, the most accurate computation output comes from the MATLAB algorithm calculation, so it was used as a reference for the other computation outputs. The difference between the reference and the other outputs is viewed as error.

The issue of the fixed point and floating point was also investigated. They are different formats used to store and manipulate numbers. Because the floating point format achieves better precision, higher dynamic range and shorter development cycle [3], it more suits the FFT-based algorithm in our case. However, the on-board processing time of it is still a question. Therefore, by using an open-source fixed point FFT C code (See Appendix A), the on-board processing time of the fixed point FFT was assessed and compared to that of the floating point FFT. If the result confirms that the floating point format is much more suitable for our case, the fixed point format will not be considered any further.

4.4 Design of the test content and the order

This section introduces what are tested in the bench test and the order. As mentioned in Section 1.4, for developing and investigating the bench test in a steady way, the test objectives are from simple to complex, as the order below:

- First of all, the memory reading/writing speed is tested. After that, the execution time of some basic arithmetic operations are tested, which are multiplication, division and cosine calculation.

- In the next step, FFT is tested. It is the most fundamental constitution of the tested algorithm. In the bench test of FFT, the on-board execution time of the FFT with hardware FPU, FFT with software FPU, fixed length FFT, general length FFT, floating point FFT and fixed point FFT of the increasing input data sizes are investigated. The numerical precision of the fixed point and floating point FFT is analyzed. Meanwhile, the execution time of the MEX FFT is tested and compared with the on-board FFT.

- Afterwards, the second most complex algorithm of the target algorithm, called FCM, is tested.

- Finally, the most complex algorithm, called SIM, is tested. Similarly, the execution time and numerical precision of MEX FCM and SIM are tested and compared with the on-board FCM and SIM.

4.5 Bench test workflow in detail

The whole bench test workflow is illustrated in Figure 4-4. The first step is to generate the C code of the target algorithm and prepare the associating input data in MATLAB. Section 4.5.1 introduces how the code was converted and the input data was prepared. The next step is to encapsulate the generated C code of the algorithm into the bench test program. Suppose that the generated C files have already been copied to the Eclipse project of the bench test, Section 4.5.2 will discuss the settings of some speed affecting test parameters. The following step is to load the input data and the file of bench test program onto the processor board as well as to start the execution. A short description is given in Section 4.5.3. Finally, the output of the on-board operation of the algorithm shall be read out from the memory, which is introduced in Section 4.5.4.

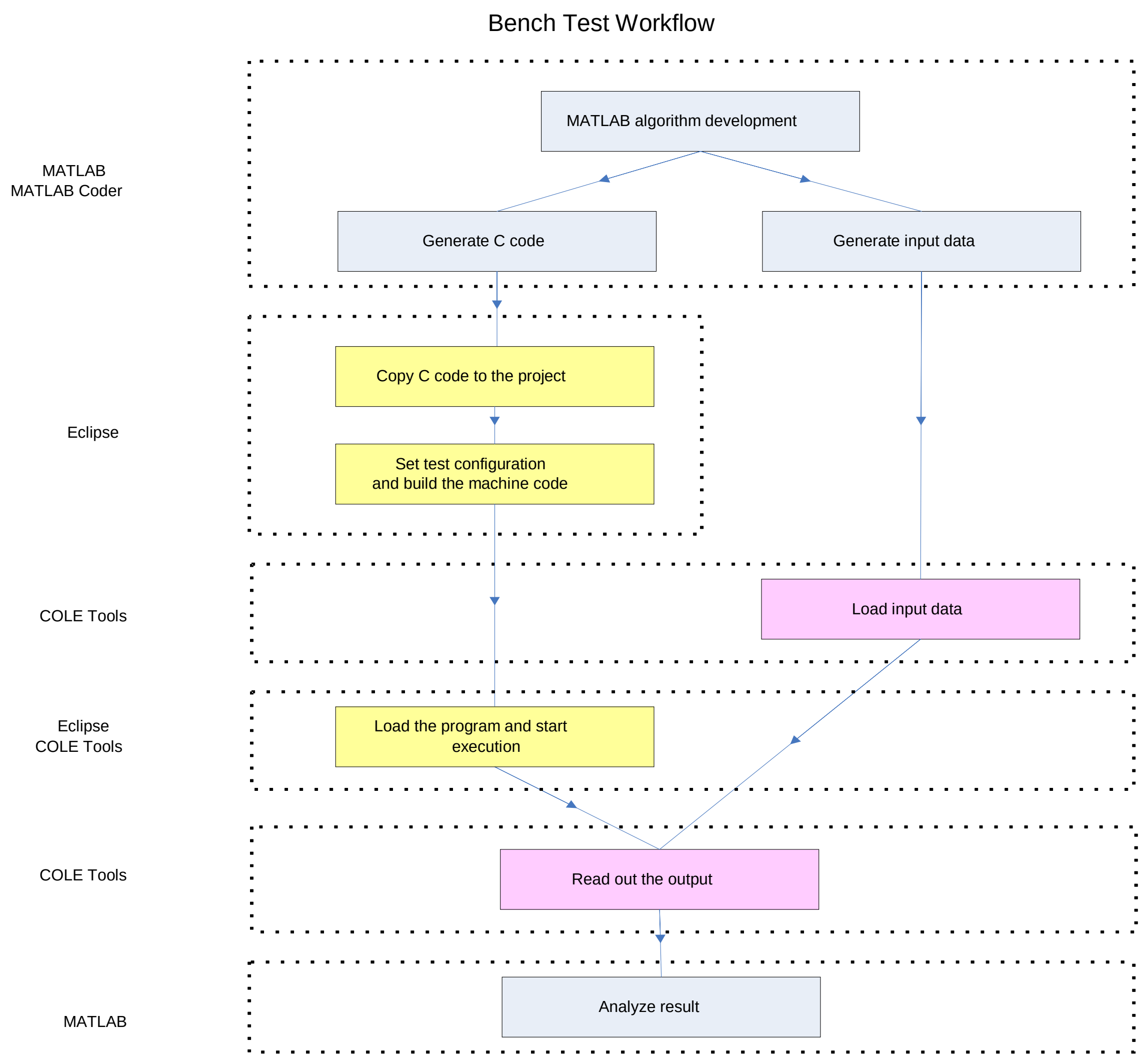
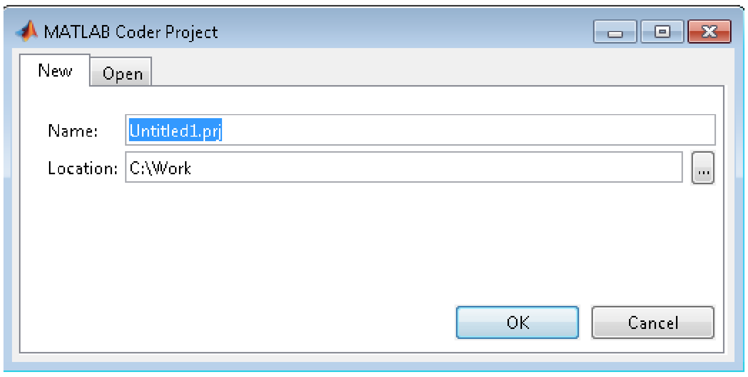


Figure 4-4 Workflow of the bench test

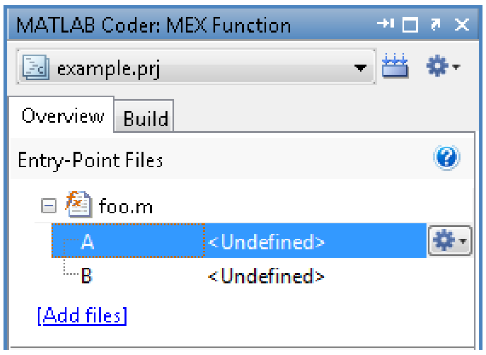
4.5.1 C code and input data generation

The integrated software product MATLAB Coder was used to convert the target algorithms into the C code. MATLAB Coder provides two ways to compile a MATLAB function to C code. One way is to create a new MATLAB Coder project and configure the necessary settings in a Graphical User Interface (GUI) dialog box. This method was mainly used when we investigated the functionalities of MATLAB Coder or debugging the compilation. In the GUI window, the format for the input data, the global variables, the hardware configuration and the compiler configuration can be set manually, see Figure 4-5 and Figure 4-6 for example. The advantage of this method is that its intuitive interface helps the user to quickly locate a certain configuration option. The disadvantage of it is when the target algorithm is changed; it will cause a lot of re-clicking to re-set the format of input data and global variables again.



Name: Untitled1.prj
Location: C:\Work

Figure 4-5 Create a MATLAB Coder project [7]



foo.m
A <Undefined>
B <Undefined>

Figure 4-6 An example of setting input data [8]

The other way is to write a normal MATLAB script which includes the command-lines for all the essential settings. Each configuration option in the GUI dialog box has a corresponding command. This method was afterwards preferred in our project because the compile script of an algorithm could be reused to a great extent. Comparing with the GUI method, this one reduces a lot of messy re-clicking work.

As described earlier, MATLAB Coder can not only convert a MATLAB function to C code for an executable file, but also to an MEX file. The C code for an execution file was used on the target hardware and the MEX file was operated on the standard computer, usually the C code in the context refers to the C code to run on board.

The input of the algorithm was preferred not to be hard coded in the algorithm C code for lightening the C code and having the input data flexible. So the input data was generated separately from the code compilation procedure. Consequently, in the third step, the generated input data was loaded on board separately from the program.

4.5.2 **Test configurations in Eclipse**

As mentioned before, the reason why a bench test program is needed is because the original C code of the algorithm lacks the code for some necessary configurations and the time measurement. Among the lacking code, the time measurement code, the RTEMS configuration code and some other program related code has been hard coded in the initial development of the bench test program, so there is no need to add any manual adjustment for those part of code. In this section the discussion of the test configuration only focuses on the configurations for some time sensitive items.

There are three time sensitive items to be configured in the bench test program, which are the FPU, the cache and the memory. The operation time performance of using hardware FPU should be compared with that of using software FPU on the same algorithm. By assigning ‘yes’ or ‘no’ to the variable called ‘FPU’ in the make file of the bench test program, the FPU can be set yes (hardware) or no (software). Also the different operation time performances due to different cache modes should be investigated. The switch of cache mode is to comment or uncomment two instruction lines in the bench test program. Considering that the memory is potentially time sensitive, the operation time of different memories is planned to be measured and evaluated in case. To switch between the different memories, the hardware of the memory should be changed and some instructions in the bench test program should be commented or uncommented. In order to perform those different scenarios, the three items was configured properly according to that specific scenario.

4.5.3 **Load input data and start execution**

After the step in Section 4.5.2, the bench test program was built into an executable file. After that, the prepared input data was loaded to the memory of the processor board through the COLE Tools. Finally, the executable file of the bench test could be loaded on-board and its execution was started from Eclipse.

4.5.4 **Read out the result and analyze**

In our project, the execution time was displayed on the screen of the lab computer and the numerical computation output of the algorithm on board was read out from the memory through COLE Tools. The output is saved in binary format which is supported by MATLAB, which means it can be analyzed in MATLAB like any normal output. The bench test result and analysis is presented in the next section.

5 Bench Test Result and Analysis

5.1 Introduction

In this bench test, firstly the processing time of simple memory reading/writing, multiplication, division and cosine were tested. Then, it moved onto standard algorithm inside the algorithm, FFT. Afterwards, the majority part of the algorithm—FCM and SIM would be investigated. Finally, the relationship between MEX operations and on-board operations was tested.

5.2 Basic operations

The processing time of some basic operations – memory reading/writing, multiplication, division and cosine is shown in Table 5-1. As all the operations are the basic building blocks in the algorithm, they could be used to check the processing time of all the sub-algorithms (FFT, FCM and SIM). For example, it is used in Section 5.3.1.1 for the evaluation of the time cost of FFT.

Table 5-1 Comprehensive comparison of the processing time of basic operations

(SRAM & Hardware FPU)

Operation	Vector length of A or B or C	Time cost (ms)	Complexness
Read/Write	65536	14	complex
Addition(A+B)	65536	65	complex
Multiplication(A.*B)	65536	94	complex
Division(A./B)	65536	299	complex
Cos(C)/Sin(C)	65536	276	real

The input for each operation is floating point random values.

5.3 FFT

In this section, floating point FFT and fixed point FFT are discussed. The processing time and error are analyzed for both cases. All the tests in this section are based on SRAM.

5.3.1 Floating point FFT

The C code for floating point FFT is generated from MATLAB. There are two kinds of C code implementations. The first one is called fixed length FFT, which means for different size of input, there will be a specific C-code file for that length. The second one is a C-code file that can adapt up to a maximum size of input, which is called general length FFT. The processing time of both implementations for FFT is demonstrated in Section 5.3.1.1. Note that, the FFT is not optimized. It is directly generated by MATLAB Coder.

5.3.1.1 **Processing time of floating point FFT**

The input signal in this section is a complex signal with unity amplitude and constant frequency:

$$x(n) = A \cdot e^{\frac{j2\pi n}{F_s} \cdot F_0} \qquad (5\text{-}1)$$

x(n) is the input data. n = 0: N-1, where N is the length of input data. A is the amplitude of input data (A=1 in this case); Fs is the sampling frequency; F_0 is the frequency of the input signal (F_0 =100 in this case). According to the Nyquist Criterion, Fs should be larger than the twice of F_0. Because N is always a power of 2 in our project, if Fs is also a power of 2 and smaller than N, the input data will have integral number of time periods.

As mentioned in Section 4.5.3, there are three parameters in the test environment that need to be considered. The processing time regarding to different kinds of FPU and memories are analyzed in Table 5-2 and Table 5-3, respectively.

Table 5-2 Processing time of fixed length FFT regarding to different FPU (ms) (Fs=1023)

Condition / Length of Input	Software FPU	Hardware FPU
16384	4062	223
32768	8731	476
65536	18587	1010
131072	39409	2135

From this table, one can observe that the HW FPU is about 18 times faster than SW FPU. For that reason, only HW FPU is used in the following tests.

Table 5-3 Processing time of general length FFT for different memories (ms) (Fs=1023)

Condition / Length of Input	SRAM	SDRAM
16384	283	284
32768	594	598
65536	1245	1252
131072	2608	2620

As it can be seen in this table, the processing time operating on SRAM and SDRAM is almost the same. That means the type of memory won’t influence processing time, so that when selecting memories, the processing time is not necessary to be considered.

The processing time for fixed length and general length FFT is given in

Table 5-4.

Table 5-4 Processing time for fixed length and general length FFT (ms) Fs=1023 periods

Condition / Length of Input	Fixed Length FFT	General Length FFT
1024	9	12
16384	223	283
32768	476	594
65536	1010	1245
131072	2135	2608

As shown in this table, the processing time of general length FFT is 25% longer than the fixed length FFT. The reason might be that, general length FFT adds some initial operations around vector size check. Another reason probably is general length FFT add more memory copy operations which cost time too. According to the test result, the operating time of copying a 64K static complex vector to another memory address is 116ms, which is relatively long.

A model used to estimate FFT processing time is

$$T = B \cdot N \cdot \log_2(N) / F \qquad (5\text{-}2)$$

Where B is an estimated constant, N is the input vector size, F is the system frequency of the processor board (64MHz), and T is the processing time. From the processing time of FFT with 16384 bit input data length in

Table 5-4, B can be computed. For fixed length FFT, B = 50.7; for general length FFT, B =64.3; Since the general length FFT is more flexible than fixed length FFT, only general length FFT is used in the following test.

With the computed B, the processing time of general length FFT is estimated in Figure 5-1 , shown in red line. The blue line shows the measured processing time.

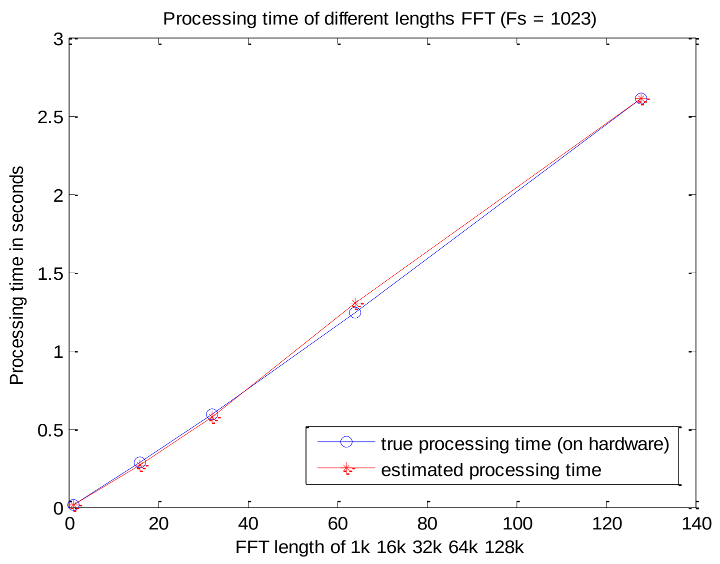


Figure 5-1 Processing time for general length FFT (Fs= 1023 periods)

As observed in Figure 5-1, the estimated 'T' from model (5-2) matches the tested processing time from processor board.

When Fs is the power of 2 (e.g. 1024), the processing time will be different from when Fs is not a power of 2 (e.g. 1023). Figure 5-1 shows the processing time of general length FFT using the different input signal with Fs = 1024. With the model (5-2), the estimated B = 85.5, illustrated in red line.

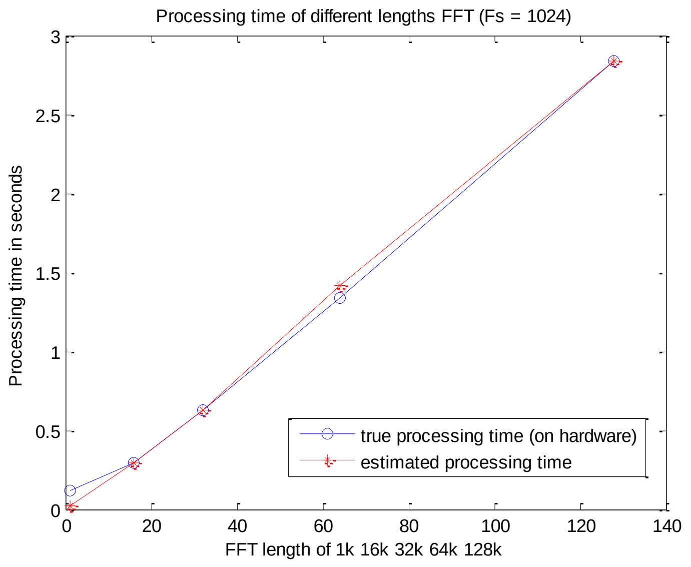


Figure 5-2 Processing time for general length FFT (Fs = 1024 periods)

It can be seen that the processing time in Figure 5-2 is higher than the processing time in Figure 5-1, which means the input signal with Fs=1023 is processed faster than it with Fs=1024. In addition, when *Fs* is not a power of 2, its processing time is equivalent to the case that the input signal is random noise:

$$X = A(randn(1,N) + j \cdot randn(1,N)) \qquad (5\text{-}3)$$

In other words, the processing time with random input signal is faster than the case that the input signal has integral number of time periods. One possible reason for those results is that when the input signal has integral number of waves, it will result in a peak in its frequency spectrum. All the other non-peak points are very small numbers. The FPU might take more time when calculate the small value data, which causes a longer processing time for the input signal with integral number of waves. However, for the time limitation, this has not been investigated further.

The FFT calculation flow can be expressed as Figure 5-3. For N-point FFT, there are $\log_2 N$ stages of decimation, where each stage involves N/2 butterflies arithmetic.

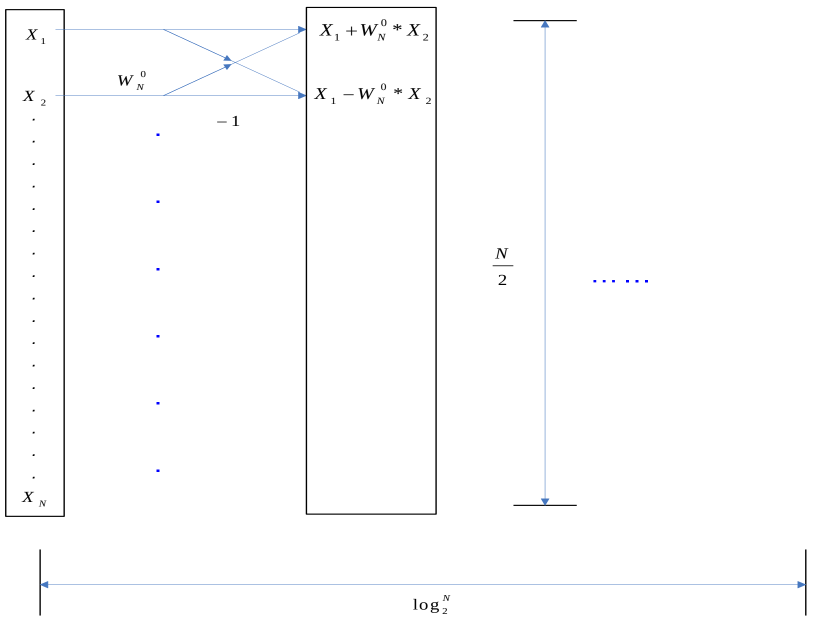


Figure 5-3 FFT calculation flow

As it can be seen in Figure 5-3, each butterfly arithmetic has one ‘pure complex multiplication’ and two ‘pure complex additions/subtraction’. In the embedded programming point of view, the procedure of a multiplication operation consists of two parts. One part is to access the memory to read the two multiplicative operands and write the product. The other part is the CPU calculation to multiply the two operands, which is called ‘pure multiplication’ in this report. Simultaneously, the ‘pure addition’ means the CPU calculation for adding the two operands to compute the summation.

The N-point FFT calculation needs $\frac{N}{2}\log_2 N$ pure complex multiplications, $N\log_2 N$ pure complex additions and $N\log_2 N$ read and write operations (shown in Table 5-5).

Table 5-5 Number of operations to compute an N-point complex FFT

Operations	Number
pure complex multiplication	$\frac{N}{2}\log_2 N$
pure complex addition	$N\log_2 N$
read(complex)	$N\log_2 N$
write(complex)	$N\log_2 N$

The processing time for FFT operation can be estimated by the number of operations in Table 5-5 and computed with the data in Table 5-1.

Take 65536-point FFT for example. Suppose the processing time for FFT is T_{fft}, and the processing time of multiplication, addition, read and write is T_{multi}, T_{add}, T_{read} and T_{write}, respectively.

$$T_{fft} = \frac{N}{2}\log_2 N \cdot T_{multi} + N\log_2 N \cdot T_{add} + N\log_2 N \cdot T_{read} + N\log_2 N \cdot T_{write} \qquad (5\text{-}4)$$

, where N=65536.

As it is written in Table 5-1, the processing time of complex multiplication, addition, read and write for 65536 point vector is 94ms, 65, 14ms and 14ms, respectively. Each complex multiplication includes one times of pure complex multiplication, two times of reading and one time of writing. So that the processing time of pure complex multiplication for 65536 point vector is 94-2*14-1*14=52ms. Analogously, the processing time of pure complex addition is 65-2*14-1*14=23ms. (Shown in Table 5-6)

Table 5-6 The processing time of each operation to compute an 65536-point complex FFT

Operations	Size of input	Processing time(ms)
pure complex multiplication	65536	52
pure complex addition	65536	23
Read/write(complex)	65536	14

With the Equation (5-4) and data above, T_{fft} can be computed:

$$T_{fft} = \frac{1}{2}\log_2 N \cdot 52 + \log_2 N \cdot 23 + \log_2 N \cdot 14 + \log_2 N \cdot 14 = 1232(ms)$$

The percentage of each operation in the view of processing time can be calculated. (Shown in)

Table 5-7)

Table 5-7 The processing time percentage of each operation in complex FFT

Operations	Percentage
pure complex multiplication	34%
pure complex addition	30%
Read and write(complex)	36%

As the value of *T_fft* is based on the data in Table 5-1, where the input data in Table 5-1 is random data, the estimated processing time for FFT is for random input data. Since when *Fs* is not a power of 2, the processing time is equivalent to the case that the input data is random noise, the value in Table 5-4 can be used to compare with the estimated processing time. The measured processing time for general length FFT with 65536 point input in Table 5-4 is 1245ms. As can be seen, it has a good match with the estimated *T_fft*.

5.3.1.2 Error analysis of floating point FFT

For concision, there is no need to analyze the calculation error for every test and every algorithm. Hence only FFT error analysis is displayed in detail in this report. The input signals is like the Equation (5-1), where Fs=1024, A=1; N is 1K, 16K, 32K, 64K and 128K.

In further analysis of the calculation error, the normalized standard deviation error and maximum error are calculated in the following way,

1) Suppose error = absolute value of (hardware result-MATLAB result);
2) Δ=exclude the peak value of error;
3) Normalized standard deviation error (which is called ‘normalized Std error’ in short) = std(Δ)/max(truth data);
4) Normalized max error = max (Δ)/max(truth data).

Figure 5-4 is the FFT output of the on-board FFT calclulation. Figure 5-5 is the error between the hardware result and the MATLAB result (Δ). Figure 5-6 and Figure5-7 are the error analysis in the form of normalized Std error and normalized max error.

Figure 5-4 improves the calculation result from the processor board is the same as the result from MATLAB. The relationship between the peak values and their corresponding input data length can be represented like the equation below:

$$Peak(N \cdot a) = a \cdot Peak(N) \qquad (5\text{-}5)$$

, where ‘N’ is the length of input data, ‘a’ is an constant.

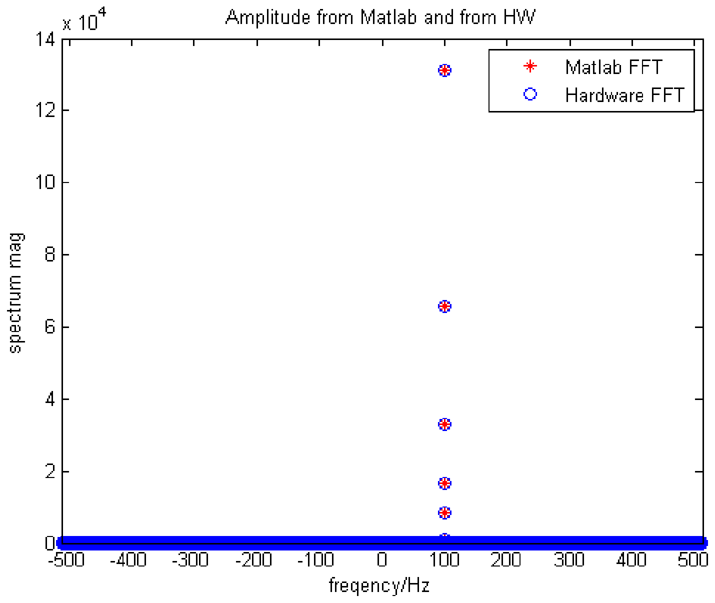


Figure 5-4 FFT calculation result from processor board

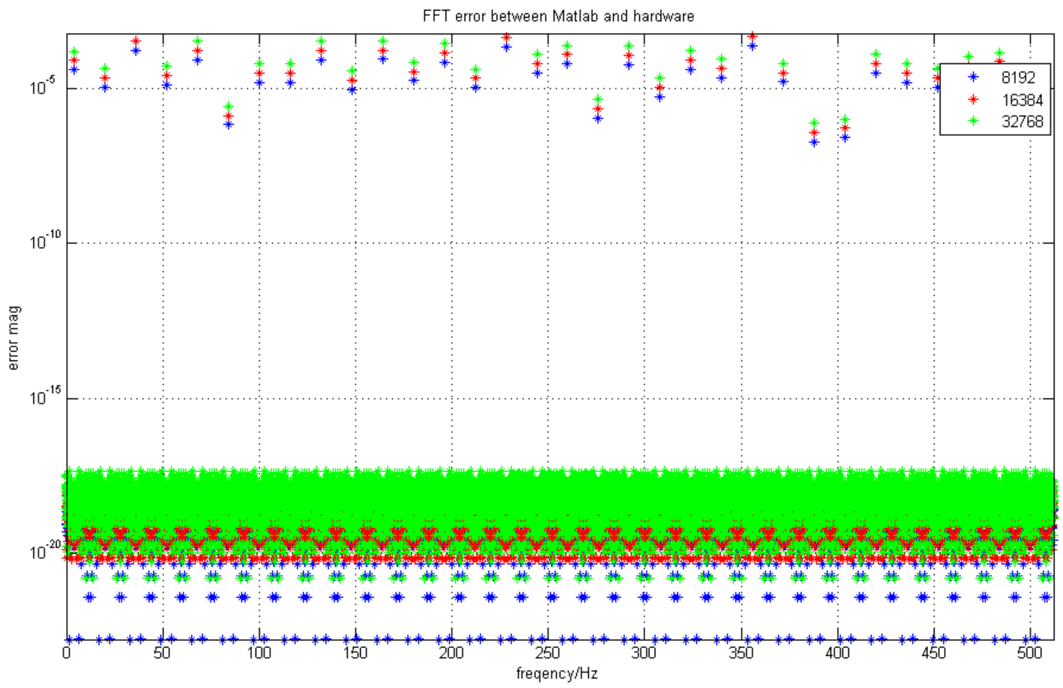


Figure 5-5 Error between the result from hardware and the MATLAB result

The plot in Figure 5-5 is the error ‘Δ’ with different input data length. As can be seen, ‘Δ’ increase when the length of input data increase, and they have the same scale factor.

Table 5-8 illustrates the Std error and max error of floating point FFT with different lengths. The relationship of the error and the length of input data can be represented by the following equation:

$$Std\_error(N \cdot a) = \sqrt{a} \cdot Std\_error(N) \tag{5-6}$$

$$Max\_error(N \cdot a) = a \cdot Max\_error(N) \tag{5-7}$$

, where ‘N’ is the length of input data; ‘a’ is a constant.

Table 5-8 Error of floating point FFT with different lengths

Size	Std error	Max error
1k	0.027e-004	0.379e-004
8k	0.077e-004	0.303e-005
16k	0.109e-004	0.606e-005
32k	0.154e-004	0.121e-006
64k	0.218e-004	0.242e-006
128k	0.308e-004	0.485e-006

With Equation (5-5) and Equation (5-6), one can compute the relation between normalized Std error and the input data length like the following equation:

$$\begin{aligned} & Normalized\_Std\_error(N \cdot a) \\ & = Std\_error(N \cdot a) / \Delta(N \cdot a) \\ & = \frac{1}{\sqrt{a}} \cdot Std\_error(N) / \Delta(N) \\ & = \frac{1}{\sqrt{a}} \cdot Normalized\_Std\_error(N) \end{aligned} \tag{5-8}$$

, where ‘N’ is the length of input data; ‘a’ is a constant.

The same as normalized Std error, the normalized max error could be represented as the equation below:

$$Normalized\_\max\_error(N \cdot a) = Normalized\_\max\_error(N) \tag{5-9}$$

The expectation of normalized Std error and normalized max error can be proved in Figure 5-6 and Figure5-7.

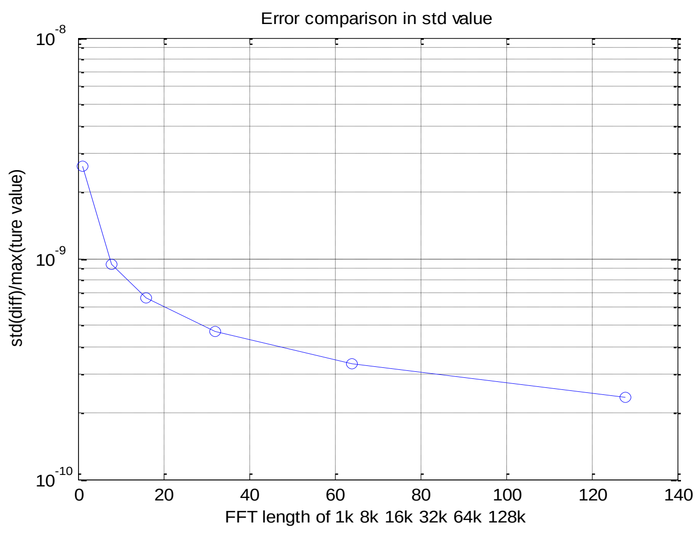


Figure 5-6 FFT error analysis in normalized Std error

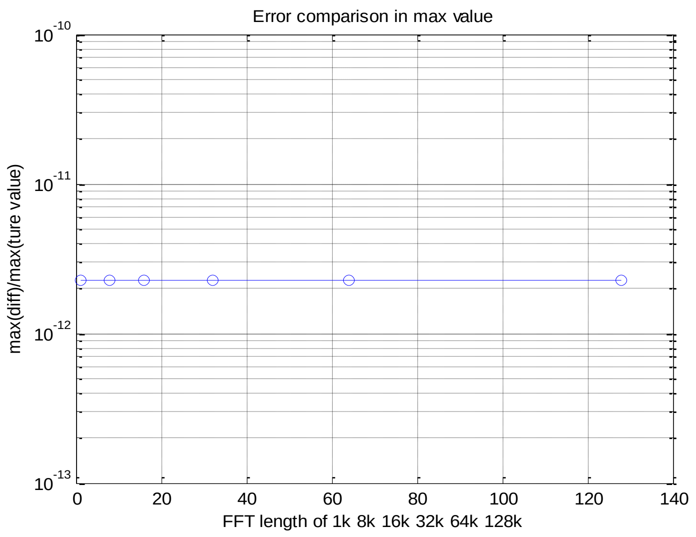


Figure5-7 FFT error analysis in normalized max error

To be clearer, the value of the normalized floating point FFT errors is shown in Table 5-9.

Table 5-9 Normalized error of floating point FFT with different lengths

Size	Normalized Std error	Normalized Max error
1k	2.6234e-009	3.6976e-008
8k	9.3823e-010	3.6976e-008
16k	6.6397e-010	3.6976e-008
32k	4.6969e-010	3.6976e-008
64k	3.3218e-010	3.6976e-008
128k	2.3491e-010	3.6976e-008

5.3.2 Fixed point FFT

As described in Section 4.3, the comparison between fixed point and floating point FFT is worth to investigate to see which format is the best for our case.

In this section, the test result of fixed point FFT was illustrated in order to compare to floating point FFT. In Section 5.3.2.1, the processing time of 16-bit FFT and 32-bit FFT was tested. Section 5.3.2.2 is to investigate the error of fixed point FFT. In the fixed point FFT, integer numbers are used to represent -1.0 to +1.0, accordingly, the larger range of integer values, the better accuracy it will have. It's easy to observe that 16bit integer is more susceptible to common numeric computational inaccuracies than 32bit integer. In the consideration of the high requirement of the computation accuracy, the 16-bit FFT will not be involved in Section 5.3.2.2.

5.3.2.1 Processing time of fixed point FFT

The input signal in this section is a complex signal with constant frequency, like the Equation (5-1), where Fs=1024.

Figure 5-8 and Table 5-10 shows the processing time of different input data length for both 16bit FFT and 32bit FFT, where $A=2^b$, b=15.

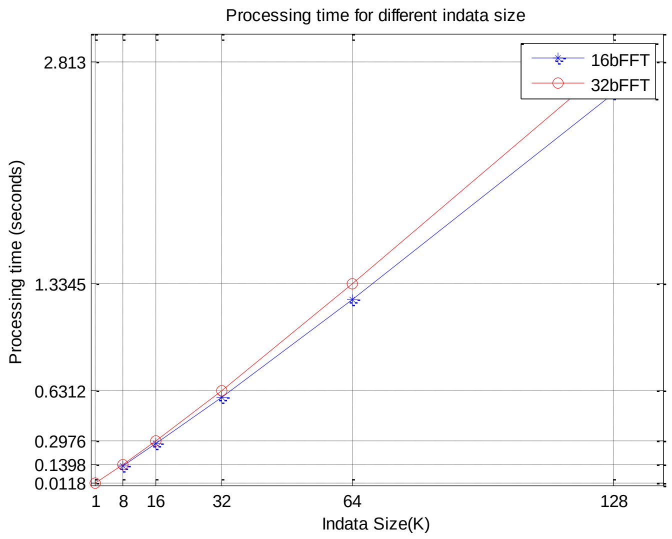


Figure 5-8 Processing time of fixed point FFT with different lengths

Table 5-10 Processing time (ms) of fixed point FFT with different lengths

Size	16bit FFT	32bit FFT
16384	274	297
32768	582	631
65536	1233	1334
131072	2604	2812

As can be seen in Figure 5-8 and Table 5-10, 32-bit fixed point FFT is a little bit slower than 16-bit fixed point FFT. However, as 32bit FFT performs better in accuracy of the computation, 32bit FFT is more preferred to use.

By applying the Equation (5-2), it can be obtained that, for 16-bit FFT, B is 80 and for 32-bit FFT, B is 87. As it is computed in Section 5.3.1.1, for fixed length floating point FFT operating in hardware FPU with Fs equals to 1023, B = 50.7; For general length FFT with the same condition, B = 64.3. As a result, comparing floating point FFT, fixed point FFT is slower. There is no advantage of using fixed point FFT in the processing time point of view.

5.3.2.2 **Error analysis of fixed point FFT**

Two parameters will influence the calculation error on the processor board. One is the error due to different input data lengths of fixed point FFT, the other one is the error due to input data amplitudes. The influence of both parameters was tested on hardware and the investigation is given below.

Figure 5-9 and Table 5-11 illustrate the normalized Std error and max error of fixed point FFT with different lengths. The input data is a complex signal with constant frequency, as it is in Equation (5-1), where A= 2^{30}, Fs=1024.

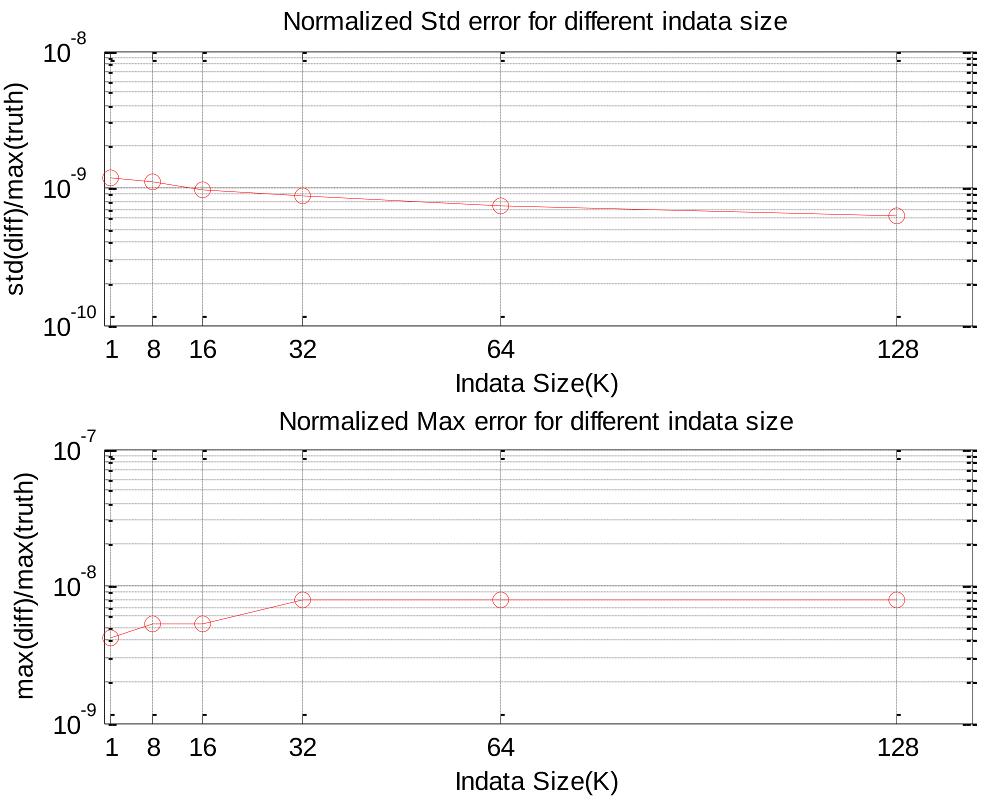


Figure 5-9 : Normalized error of fixed point FFT with different lengths

Table 5-11 Normalized error of fixed point FFT of different lengths

Size	Normalized Std error	Normalized Max error
1024	1.179e-009	4.165e-009
8192	1.090e-009	5.268e-009
16384	9.829e-010	5.268e-009
32768	8.758e-010	7.903e-009
65536	7.475e-010	7.903e-009
131072	6.302e-010	7.903e-009

The figure and table above indicate the normalized Std error decrease with the increasing of the input data size. Meanwhile, the normalized max error is almost the same due to different size of input data.

The dependency to input data amplitude on the error rate is shown in Figure 5-10 and Table 5-12 Normalized error of different input data amplitudes. The input signal is a complex signal with constant frequency, as it is in Equation (5-1), where $A = 2^b$, b=30; Fs=1024; N=8192.

As the range of 32bit integer is from − 2,147,483,648 to 2,147,483,647, there might be overflow during the FFT calculation if the size of input vector is too big, so a scale factor is applied in every FFT iteration (divided by two). However, the scale vector influences the accuracy of result. On the other hand, the amplitude of input vector influences the error, too. If the amplitude is large, the contrast between 'Δ' and the maximum value of FFT result will be very striking. In that case, with the increase of the amplitude, the normalized Std error and max error will decrease. The test result can be observed from Figure 5-10 and Table 5-12.

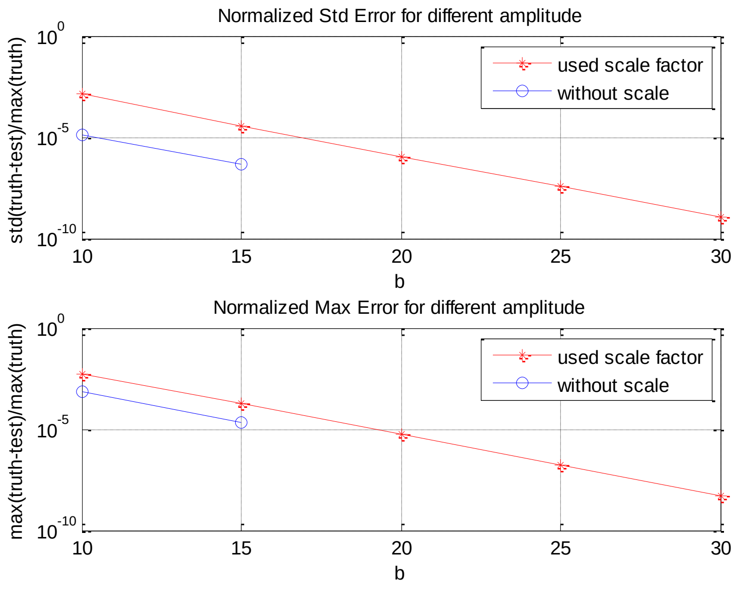


Figure 5-10 Normalized error comparison of different input data amplitudes (The number in x axis is the log2 value of the amplitude.)

Table 5-12 Normalized error of different input data amplitudes

Amplitude	Std error		Max error	
	Without scale	With scale	Without scale	With scale
2^10	1.381e-005	0.001	6.741e-004	0.006
2^15	4.376e-007	3.545e-005	2.107e-005	1.779e-004
2^20		1.135e-006		5.395e-006
2^25		3.491e-008		1.686e-007
2^30		1.090e-009		5.268e-009

As expected, for the same input vector, without scale factor will result in smaller normalized Std error and max error. Meanwhile, for input vectors with the same length and different amplitude, the Std error and max error is reduced when the amplitude rise.

Comparing Table 5-11 and Table 5-12 with Table 5-9, the normalized error of fixed point FFT is much larger than floating point FFT when the amplitude of input data is less than 2^30. However, the amplitude of input data in Table 5-9 is 1, so that the accuracy of floating point FFT is much better than fixed point FFT.

With the result of processing time and normalized error for floating point FFT and fixed point FFT, one can draw the conclusion that there is no advantage of using fixed point FFT comparing to floating point FFT.

5.4 FCM and SIM —processing time

The most time consuming part of FCM algorithm is FFT, thus FCM can be viewed as the combination of several FFTs. As it is shown in
Figure 5-11, the processing time of FCM regarding to different input data length have the same trend as the one in FFT.

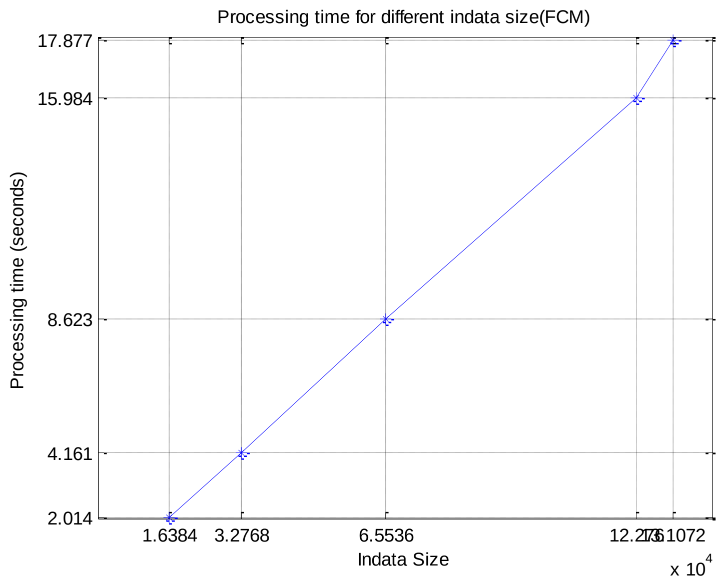


Figure 5-11 Processing time of FCM regarding to different input data length (SDRAM)

Figure 5-11 shows that with the increasing of input data size, the processing time of FCM increases in scale. From Equation 5-2, the estimated constant B = 562.

The time-consuming calculations of SIM could be separated to two parts. One is FFT, whose processing time will enhance with the increasing of input data length. The other part is the combination of phase calculation, upsampling etc, which is constant due to different input data length. The model used to estimate SIM processing time is

$$T = C + B \cdot N \cdot \log_2(N) / F \tag{5-10}$$

Where N is the input vector size, F is the system frequency of the processor board (64MHz), B is an estimated constant, C is the constant processing time for all size of input data and T is the processing time. The processing time of SIM with different input data lengths is shown in Figure 5-12.

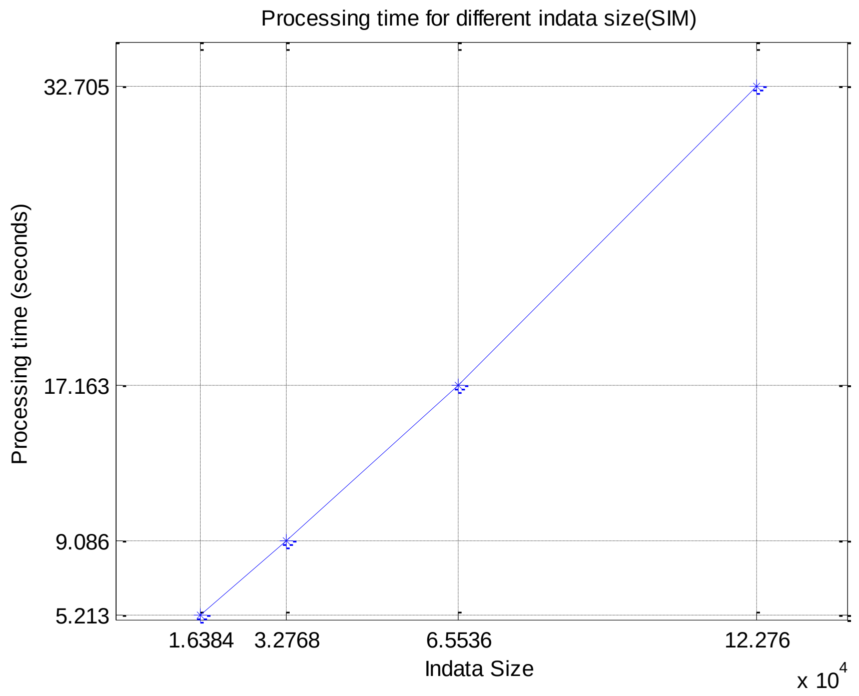


Figure 5-12 Processing time of SIM with different input data lengths (SDRAM)

With the value of first and second point in Figure 5-12, the parameters in Equation 5-3 can be computed: C = 1802 ms; B = 952.

5.5 Comparison between MEX-file computation time and on-board computation time

As it is written in Section 4.3, MEX-file is a very excellent tool for algorithm developing. In this section, the relationship between MEX computation time and on-board computation time is described.

The scale factor in Table 5-13 and Table 5-15 is the On-board computation time divided by MEX computation time, which indicates the relationship between on-board computation time and MEX computation time. The MEX files runs on a computer with Intel Core Quad CPU, whose clock speed is 3.00 GHZ.

Table 5-13 Compare the processing time of different functions between MEX-file and hardware

Function name	Size of data	MEX computation time (ms) (On PC 2592)	HW computation time (ms)	Scale factor	Complexness of input data
X+Y	65536	2.7	65	24	complex
fftcorr(X,Y)	65536	124	3736	30	complex
abs(X)	65536	4.3	132	31	complex
angle(X)	65536	5.4	792	146	complex
exp(1i*P)	65536	6.6	111	17	real
polyfit(t,P,1)	65536	7.3	627	80	real

In Table 5-14,the processing time of some basic MATLAB functions was tested. As can be seen, the scale factor is diverse for different functions.

Table 5-15 Compare the processing time of different algorithms between MEX-file and hardware

Function name	Size of data	MEX computation time (ms) (On PC 2592)	On-board computation time (ms)	Scale factor	Number of equivalent FFT (measured)
FFT	32768	22	598	27	1
FFT	65536	46	1252	27	1
FFT	122760		2343		1
FCM	32768	135	4161	31	7
FCM	65536	281	8623	31	7
FCM	122760		15984		7
SIM	32768	334	9086	27	15
SIM	65536	668	17163	27	14
SIM	122760		32705		14

As it is shown in Table 5-15, because FCM and SIM are based on FFT, they have the similar scale factor, around 30. For the same algorithm, the scale factor is the same regarding to different length of input data.

In conclusion, the relationship between MEX operation time and on-board operation time is fixed for the same functions. Different functions result with different scale factor. The scale factors for algorithms depend on what functions is included in each algorithm.

Another observation from the scale factor is that the processer board is more efficient per CPU clock cycle. The clock speed of the computer that MEX files run on is 3GHz. The clock speed of the processor board is 64MHz. The scale between the CPU clock speed of the computer and processor board (3GHz/64MHz=47) is larger than the scale factor (about 30), so that for one clock cycle, the processor board can manage more operations than the computer.

6 Conclusion

6.1 Different parameters that influence the test results

When testing a specific algorithm, there are four factors that vary from different test cases. These factors are the memory types (SRAM and SDRAM), cache modes, FPU types (software FPU and hardware FPU) and input data. The processing time on-board is the same no matter use which kind of memory. The caches modes and FPU types have a significant influence on the processing time. The test bench executed with caches enabled results with three times faster than the caches disabled case. Meanwhile, if the test bench runs with hardware FPU, the processing time is ten times shorter than it runs with software FPU. In conclusion, to get the fastest execution, one has to enable the caches and use hardware FPU. The input values affect the processing time, too. When the input data is an integral number of sine waves, the processing time is longer than the case when input is not an integral number of sine waves or random signal(like Equation 5-3). This case has not been investigated further.

The data formats in algorithms also influent the test results. Comparing with the fixed point format, the floating point format achieves better accuracy and faster execution speed with hardware FPU. As a result, the floating point format is more preferred to use than the fix point format.

6.2 The effectiveness of MEX files

As it is proved in Section 5.5 , the processing time of MEX files has a good match with the on-board processing time. In that case, when the algorithm developers test their algorithms, they can firstly convert these algorithms to MEX files and run the MEX files in MATLAB. Then, the algorithms can be evaluated and optimized in MATLAB instead of doing every step on hardware.

7 Future work

As mentioned in the beginning, for the time limitation, not the entire algorithm under development was tested. Only two thirds of the algorithm was tested and analyzed, which implies that the total execution time of the algorithm on the processor board is still unknown. One of the future tasks is to finish the bench test of the complete algorithm. The challenge for this future task is the capability of the automatic conversion from MATLAB code to C code provided by the MATLAB Coder, since the new MATLAB functions applied in the rest of the algorithm might not be supported by the MATLAB Coder. If the bench test of the entire algorithm is successful, it would be interesting to use the real signal as input data instead of the simulated input data from MATLAB.

The other future task of interest is a deeper investigation of why the input values affect the execution time. As Section 5.3.1.1 described, when the input signal is an integral number of sine waves, its execution time is longer than the input signal is not an integral number of sine waves or the input signal is random values. We preliminary think that the reason might be related to how the hardware FPU works. To investigate it, first of all, the execution time of the two kinds input signal when using software FPU should be measured. If the execution time of the two scenarios is the same, the reason affecting a different execution time is surely the way how hardware FPU works in.

This bench test is only a prototype. But it indicates a possibility of accelerating and saving the labour in traditional development procedure of satellite signal processing. In traditional development procedure, the algorithm should be coded into software by the software department before it can be tested on the processor board. For the developer, this waiting time is a kind of block. By using this bench test prototype, the block time could be largely decreased, which helps the developer to obtain the on-board test result quickly after he thinks of some new idea. Meanwhile, there is no need for the labour work form the software department in the preliminary developing stage any more.

In addition, as a further development of the bench test program, it could be worthy to try to develop a GUI of the program.

Reference

[1] Wikipedia, Eclipse (software)
http://en.wikipedia.org/wiki/Eclipse_%28software%29

[2] Brian Handley, Embedded Cross-Development with Eclipse,
http://www.macraigor.com/downloads/Macraigor_with_Eclipse.pdf

[3] Steven.W.Smith, Ph.D, The Scientist and Engineer’s Guide to Digital Signal Processing ©1997-1998, Chapter 28,
http://www.dspguide.com/ch28/4.htm

[4] On-Line Applications Research Corporation ©1988-2007, RTEMS C User’s Guide Edition 4.8.0, for RTEMS 4.8.0, 14 February 2008

[5] MATLAB Coder product description
http://www.mathworks.se/products/matlab-coder/description1.html

[6] Product Support, MEX-files Guide
http://www.mathworks.se/support/tech-notes/1600/1605.html#intro

[7] SPARC International, Inc. The SPARC Architecture Manual Version 8 ©1991, 1992