DSP Algorithm and Software Development on the iPhone/iPad Platform

by

Shuang Hu

A Thesis Presented in Partial Fulfillment
of the Requirements for the Degree
Master of Science

Approved April 2012 by the
Graduate Supervisory Committee:

Andreas Spanias, Chair
Kostas Tsakalis
Cihan Tepedelenlioglu

ARIZONA STATE UNIVERSITY

May 2012

ABSTRACT

The ease of use of mobile devices and tablets by students has generated a lot of interest in the area of engineering education. By using mobile technologies in signal analysis and applied mathematics, undergraduate-level courses can broaden the scope and effectiveness of technical education in classrooms. The current mobile devices have abundant memory and powerful processors, in addition to providing interactive interfaces. Therefore, these devices can support the implementation of non-trivial signal processing algorithms. Several existing visual programming environments such as Java Digital Signal Processing (J-DSP), are built using the platform-independent infrastructure of Java applets. These enable students to perform signal-processing exercises over the Internet. However, some mobile devices do not support Java applets. Furthermore, mobile simulation environments rely heavily on establishing robust Internet connections with a remote server where the processing is performed.

The interactive Java Digital Signal Processing tool (iJDSP) has been developed as graphical mobile app on iOS devices (iPads, iPhones and iPod touches). In contrast to existing mobile applications, iJDSP has the ability to execute simulations directly on the mobile devices, and is a completely stand-alone application. In addition to a substantial set of signal processing algorithms, iJDSP has a highly interactive graphical interface where block diagrams can be constructed using a simple drag-n-drop procedure. Functions such as visualization of the convolution operation, and an interface to wireless sensors have been developed. The convolution module animates the process of the continuous and discrete convolution operations, including time-shift and integration, so that users can observe and learn, intuitively. The current set of DSP functions in the application enables students to perform simulation exercises on continuous and discrete convolution, z-transform, filter design and the fast Fourier transform.

The interface to wireless sensors in iJDSP allows users to import data from wireless sensor networks, and use the rich suite of functions in iJDSP for data processing. This allows users to perform operations such as localization, activity detection and data fusion. The exercises and the iJDSP application were evaluated by senior-level students at Arizona State University (ASU), and the results of those assessments are analyzed and reported in this thesis.

ACKNOWLEDGEMENTS

First, I would like to express my sincere gratitude to my advisor, Professor Andreas Spanias, for his invaluable guidance and support. His patience, enthusiasm and vision helped me achieve more than I ever thought possible.

Next, I would like to express my appreciation to Professors Kostas Tsakalis and Cihan Tepedelenlioglu for being my supervisory committee. Your advices will benefit a lot in my academic life.

I would also like to express my deep gratitude to Jinru Liu who built this app, and taught me the basics of iOS programming.

Thanks to my colleagues and friends who gave me invaluable assistance during my work and life. Special thanks to Mahesh Banavar, Xue Zhang, Jayaraman Thiagarajan, Karthikeyan Natesan Ramamurthy, Alex Fink, Henry Braun, Prasanna Sattigeri, Mohit Shah, Suhas Ranganath, Akhilesh Thyagaturu, Girish Kalyanasundaram, and Deepta Rajan. Thank you all for assisting me to test and develop the software.

This project supported in part by the NSF jdsp phase 3 grant and the SenSIP center and NCSS IUCRC. Thanks Sprint Communications and Debbie Vogel, Sprint SW Manager for providing devices to our project.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Figure Page

Figure Page

Chapter 1

INTRODUCTION

The use of software applications with interaction capabilities presents huge potential for instructors to stimulate students interest, in addition to enhancing traditional classroom teaching. By incorporating innovative components such as animations, real-time demonstrations, audio and video features, those software applications can complement the teaching materials. The importance of interactivity in knowledge acquisition and cognitive skills development has been emphasized by educators [1, 2]. In [3], the authors compared several web applications, and observed that education tools typically require a higher degree of interactivity in order to enhance learning potential. Most of the current popular interactive software applications [4–7] are characterized by rich user interaction and a friendly visual programming interface to fully engage the user for an efficient learning experience. Furthermore, web-based educational tools are an integral part of modern day classroom and distance learning frameworks.

Employing software tools in STEM (Science, Technology, Engineering and Mathematics) edcuation has facililated improved student interest in engineering courses. In particular, it has impacted the delivery of advanced courses such as Digital Signal Processing (DSP), by allowing students to demonstrate and learn the application of mathematical principles in the real world. Several illustrative simulations and programming enviroments have been developed over the last decade. In order to enable students to perform DSP labs over the Internet, the authors in [8] developed J-DSP, a visual programming environment. J-DSP was designed as a zero footprint, standalone Java applet that can run directly on a browser. Several interactive laboratories have been developed and assessed in undergraduate courses.

In recent years, the advent of mobile devices has provided a new and powerful platform for students and distance learners. From 2007, when the first iPhone introduced, the traditional personal computer (PC) market has been affected by the huge growth in the use of the smartphones and tablet PCs. Statistics show that the amount of Android and iOS units shipped in 2011 is already larger than that of traditional computers units [9]. In the current smartphone market in United States, iPhone operation system (iOS) from Apple and Android operation system from Google occupies the major share of the market [10]. Both Apple and Google have developed their online distribution systems, making it convenient for mobile users to download and update applications published by third-party developers. This motivates a large number of applications for iOS and Android devices. A recent survey indicates that, there are about 550,000 apps for iPhone, iPad and iPod touch users in 123 countries worldwide, and Apple's App Store has reached more than 25 billion downloads [11].

The large market penetration of mobile devices opens up avenues for using mobile phone technologies in STEM education. Considerable research has been undertaken to understand the effectiveness of smartphones and tablets in a classroom setting. In this thesis, we present a graphical-programming application, *iJDSP*, to allow simulation of signal processing algorithms on mobile devices. This application features a simulation environment and a palette of DSP functions, which will allow students to perform laboratories using iOS devices.

1.1 J-DSP Programming Environment

J-DSP was developed as a standalone Java applet which allows users to run programs directly on a browser [12]. The primary objective of the J-DSP software was to provide undergraduate students and distance learners hands on experience with DSP concepts and to simplify the understanding of complex mathematical procedures [13]. All signal manipulation functions appear in J-DSP as *graphical blocks* that are brought into the simulation environment by a drag-n-drop process. Signal and data flow is established by simply linking the blocks. Each block contains a dialog window using which the parameters of the function can be updated dynamically. The students can plot outputs at various stages of a simulation and thereby understand the signal flow in the implementation. Figure **??** illustrates the J-DSP interface for integrated web-based learning that can access all the required components such as lecture slides, course notes, audio/video content, links to websites, animations and software modules. In addition to signal processing, JDSP has been successfully extended to multidisciplinary areas such as digital communications, control systems, genomic signal processing [13], earth systems signal processing [14], RF power amplifiers [15], and sensor networks [16] (Figure 1.2).

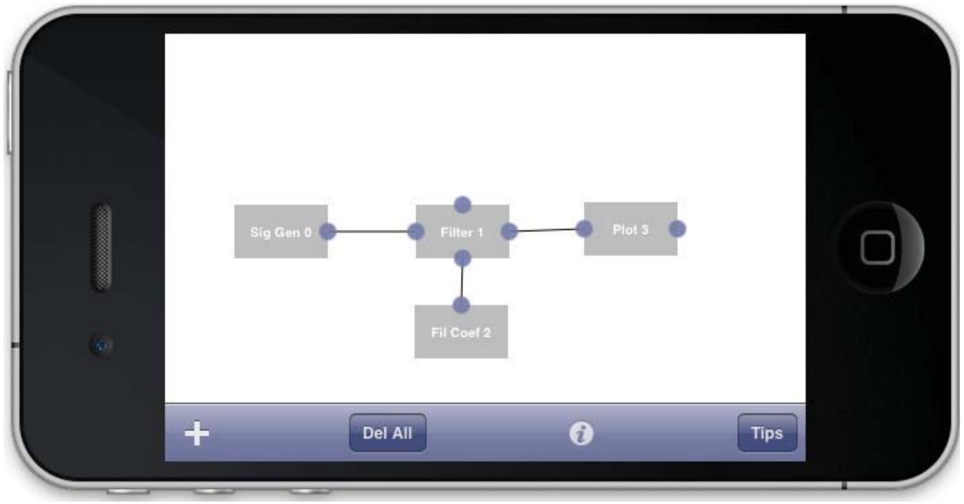


Figure 1.1: The iJDSP Simulation Environment.

1.2 DSP Education on iOS Platforms

The large multi-touch screens and the computing capabilities of mobile devices such as smartphones and tablet PCs make educational applications more attractive to students and fosters student learning. In a recent study on iPad adoption and use in the classrooms, it was found that smartphones and tablet PCs provide better performance in terms of mobility, engagement and collaboration [17]. Studies on the adoption of mobile devices in education indicate that mobile tools have remarkable advantages in teaching and learning a broad range of subject areas [18–20].

Examples of educational tools on mobile platforms include Star Walk from Vito Technology Inc. for astronomy [21], the HP 12c Financial Calculator for business [22], and Spectrogram for analysis of music signals [23]. For signal processing simulations, one of the most popular applications is MATLAB Mobile, which performs simulations through a traditional command line interface [24]. It provides a lightweight mobile version of MATLAB that runs on a remote computer via an internet connection. However, this application does not support a sophisticated graphic user interface (GUI) and relies heavily on a stable internet connection. Therefore, MATLAB Mobile may not be the most interactive, practical or portable platform for educational purposes. As a standalone mobile application, iJDSP not only provides a substantial set of signal processing algorithms, but also provides an intuitive interactive GUI that engages users and supports their learning and understanding. Furthermore, no internet access is required for running this application. iJDSP is designed exclusively for iOS-based devices such as iPhone and iPad using Apple's Xcode IDE and iOS SDK, and is meant as an extension of JDSP onto mobile platforms. As a mobile version of the JDSP editor [25], iJDSP has similar color scheme and operation mode in order to make the application consistent. The main interface is shown in Figure **??**. A rich suite of DSP functions is available in iJDSP including fast fourier transform (FFT), filter design, pole-zero placement, frequency response, sound recorder and player, and several others. Additionally, a graphical interface that allows users to import and process data harvested by wireless sensors has been developed in iJDSP.

1.3 Contributions

The underlying architecture of the iJDSP application is designed and optimized for the iOS platform. There are several challenges involved in the migration of JDSP to an iPhone operation system (iOS). First, the Java language, which was used to design J-DSP, is not supported by the iOS system. Native applications on iPhone/iPad platform are typically developed using Objective-C, C and C++. In order to improve the software performance, we employed hybrid programming by combining C with the Objective-C language. Second, the user interface of JDSP is not suitable for mobile users because of the smaller screen size. Hence, we needed to redesign the user interface to fit the screen.

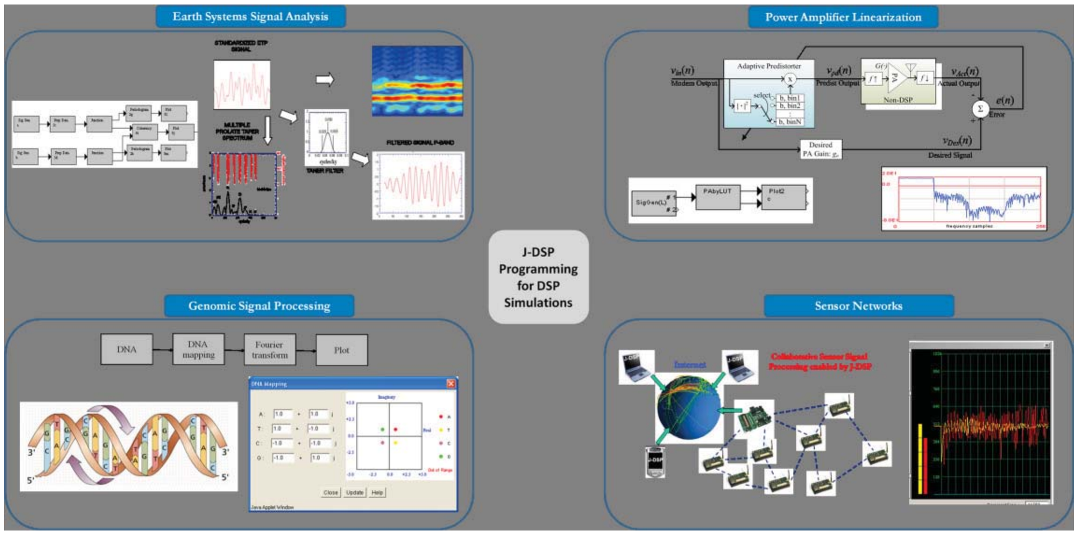

Earth Systems Signal Analysis
Power Amplifier Linearization
J-DSP Programming for DSP Simulations
Genomic Signal Processing
Sensor Networks
DNA
DNA mapping
Fourier transform
Plot

Figure 1.2: Multidisciplinary Extensions of the J-DSP Programming Environment.

In addition, constraints on processors and memory need to be taken into account during the application design. The processor of iPhone 4S is clocked at 800MHz, and the iPad 2's is clocked at 1GHz. The amount of random-access memory (RAM) on iPhone 4S and iPad 2 is 512 MB [26, 27]. Compared to computers, the calculation capability is limited on iPhone and iPad platforms. Especially during real-time calculations, those limitations even induce laggy interfaces and errors. This thesis hence details the efforts in developing the iJDSP application. The user interface of iJDSP is exclusively designed for mobile devices, and assessed by students. 95 percent of users got used to the environment within 10 minutes.

1.3.1 iJDSP Software Development

DSP algorithms typically involve complicated computations, which is not intuitive for students to understand. Hence we have developed some animated demonstrations and functions that are helpful to describe those concepts. We have developed a set of DSP functions in iJDSP, that are necessary for performing laboratory exercises in a senior-level DSP course: sampling, convolution, z-transform, FFT and filter design functions. The set of functions developed as part of this thesis include the user-defined signal generator, junction, windowing, convolution, animated convolution demo, FIR design (windowing, kaiser, the Parks-McClellan), IIR design, and up/down sampling. Furrthermore, extensive testing and debugging of the entire software has been carried out. The implementation and the features of these functions will be described in detail in this thesis. Furthermore, an extensive set of laboratory exercises has been developed for use in a senior level DSP course. The software and associated exercises have been fully assessed by students ranging from children to graduates through a series of workshops and open house activities. Based on the user feedback, we have improved the software considerably by optimizing the user interface and incorporating additional features. The DSP exercises and a detailed analysis of the assessment results are also included in the thesis.

1.3.2 Hardware Interface to Wireless Sensor Networks

An important limitation of several software applications is the lack of an interface to work with hardware devices and platforms. Working with inexpensive hardware can provide students a rich experience in working with real-time data. Associated paradigms such as data fusion and real-time spectrum estimation are very impressive in describing the theory along with practical applications of signal processing. The challenge in building such a hardware interface is to enable real-time computations. The capability of processor on mobile devices is limited, while real-time calculations typically consumes system resources. As a result, any algorithm that is implemented must be optimized to maximize efficiency. The hardware component chosen to expose students to real-time processing is a Crossbow mote [28] which is widely used in several wireless sensor network applications (Figure 1.3).

We have developed a novel interface on iJDSP to interface with the sensor motes to communicate and acquire real-time data. This interface has been designed using Objective-C/C and nesC with a user-friendly GUI. As shown in Figure 1.4, iJDSP communicates with the wireless sensor through a centralized base station via Wifi. This communication is built under a local area network (LAN) which is established by

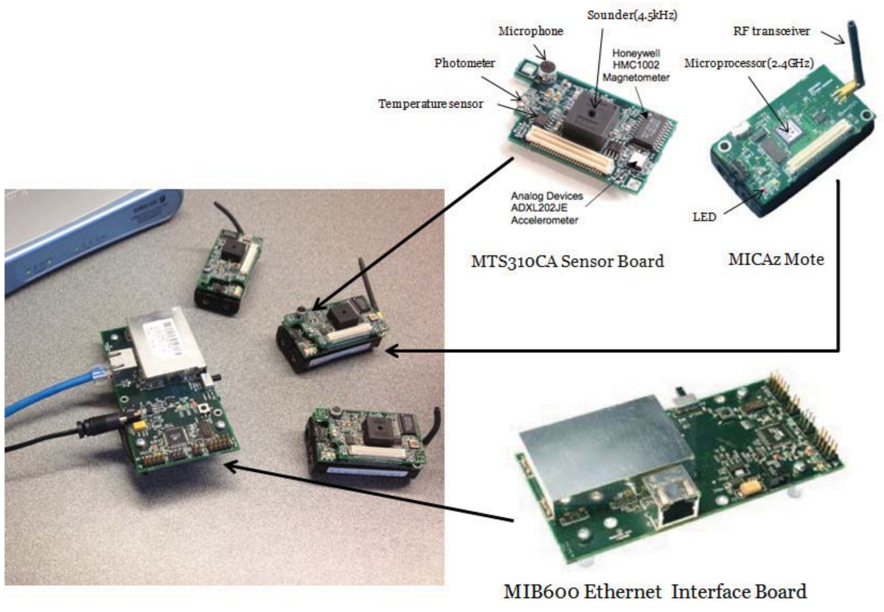


Figure 1.3: Hardware Platform from Crossbow.

a wireless router. The base station then communicates to the sensor nodes through ZigBee [29], a wireless communication protocol developed for low-cost and low-power systems.

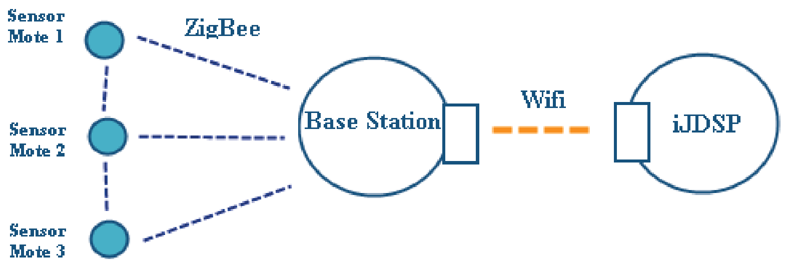


Figure 1.4: iJDSP Interface with WSN.

Targeted applications areas of such an interface with wireless sensor networks include research topics in communication, collaborative signal processing, and power-aware implementations [16].

1.4 Organization of the Thesis

This thesis describes development of convolution, FIR filter design, IIR filter design as well as software to control a sensor network using an iPhone/iPad. This software infrastructure is developed in iJDSP to support sensor network experiments and research on iPhone/iPad. The proposed interface is demonstrated using a real time spectral estimation experiment for acoustic signals on iPhone/iPad. We also developed an extensive set of DSP laboratory exercises. In the whole development process, the software has constantly been tested and debugged. The maintenance procedure includes a series of testing and assessments to gather crash reports as well as feedback in terms of user experience.

The rest of the thesis is organized as follows. Chapter 2 introduces the background of iJDSP, and describes the user interface. Furthermore, the system design pattern and the view architecture used in iJDSP are discussed. Chapter 3 describes DSP algorithms implemented in iJDSP and presents the implementation detials of the functions. A description of the associated laboratory exercises is provided in Chapter 4 that includes continuous/ discrete convolution, frequency response, filter design and FFT. The hardware interface of iJDSP with wireless sensor networks is described in detail in Chapter 5. This chapter details the development from the aspects of hardware platforms, socket programming, graphical interface design as well as building real-time experiments. In Chapter 6, the pedagogy adopted in evaluating the iJDSP application and the analysis of assessment results are presented. Finally, the extensions of iJDSP to other platforms, interdisciplinary areas, and further work along with concluding remarks are presented in Chapter 7.

Chapter 2

OVERVIEW OF iJDSP ARCHITECTURE

In this chapter, we will first introduce the iJDSP environment, and navigate the user interface. The iJDSP is developed using Xcode IDE, an integrated development environment developed by Apple Inc. and the iOS SDK, a software development kit for Apple's mobile operating system. Then a description of the system architecture of iJDSP will be given including the Model-View-Controller paradigm, inheritance, and delegation mechanism.

2.1 iJDSP Environment and User Interface

As a native Cocoa Touch application [30], the iJDSP environment is compatible with all iOS devices that running iOS later than 3.2. To improve performance, we choose hybrid programming using C and Objective-C. All algorithms in iJDSP are developed using standard C in order to achieve high calculation speed compared to Objective-C and C++ [31]. Objective-C is primary used in building the user interface(UI) on iOS and extends the C language with more object-oriented features such as class and messaging [32]. Two separate libraries are used in iJDSP to deal with custom drawing; Quartz 2D, which is part of the Core Graphics framework [33] and CorePlot, an open source 2D plot framework [34]. iJDSP is a completely standalone application that can directly run on mobile devices without the need for a Internet connection. iJDSP also provides features such as multi-touch gestures, animations and audio processing using the build-in microphone on iOS devices.

The user interface of iJDSP consists of multiple layers of views. The architecture of the views is shown in Figure 2.1. The views of iJDSP include: (a) a navigation bar (hidden in main view) for navigating multiple view controllers; (b) a main canvas; (c) DSP block diagrams; (d) a tool bar for displaying controls; (e) bar button items. As basic components, the navigation bar and the main canvas are placed at the lowest layers. The navigation bar is the view of *UINavigationController*, a control managing

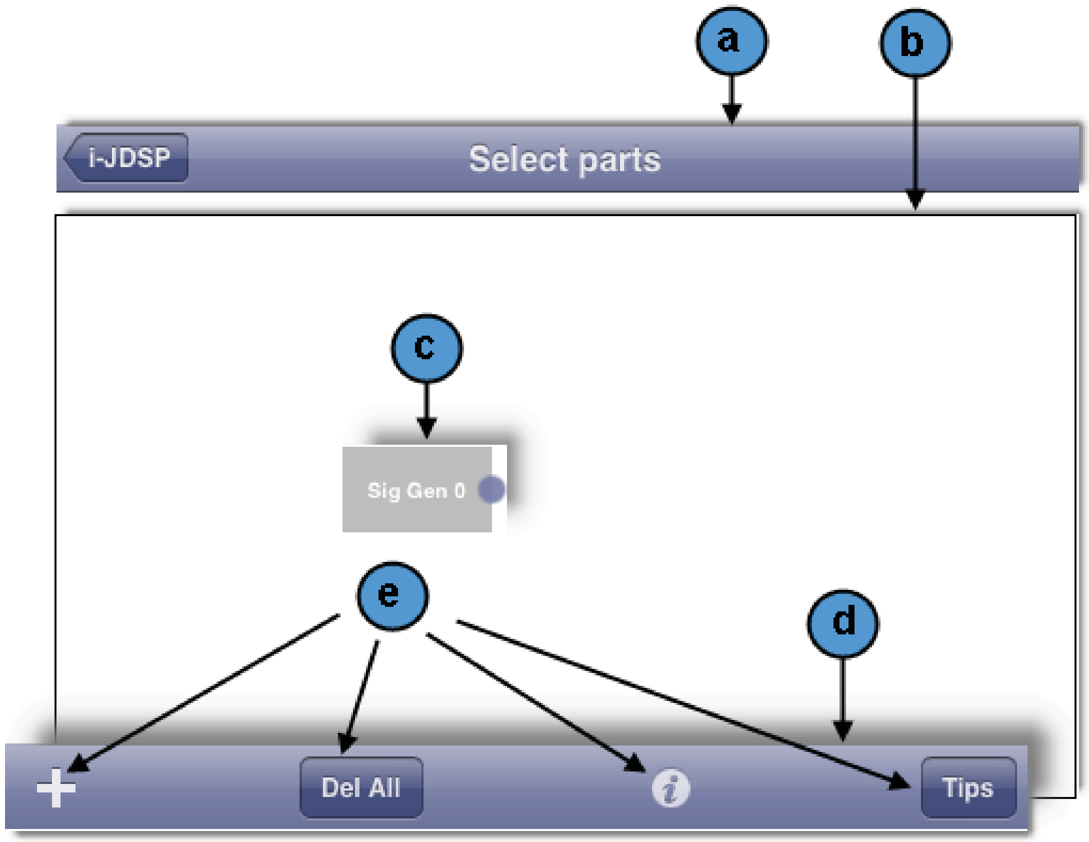


Figure 2.1: Architecture of Views in iJDSP.

the navigation of hierarchical content. The main canvas is the workspace where users perform simulations. It is also capable of handling user gestures shown in Table 2.1. Upon these layers, a sequence of graphical modules encapsulating DSP functions are then added as child views (subviews) of the main canvas. The front view is the tool bar consists of four bar button items: an add button, a trash button, an information button, and a tip button. The function of each bar button item will be introduced in the next content. The following part will show how to navigate iJDSP with a simple Filter example. After pressing the plus-like button on the bottom bar, a set of functions appears, as shown in Figure 2.2. Then we select a block named *Signal Generator*, and press the “Add” button on the top upper corner. A light grey rectangular block named “Sig Gen 0” is now shown on the main canvas. Similarly, add a *Filter* block, a *Filter Coeff*

Table 2.1: User Gesture Recognition.

Gesture	Operation
Double tap on a block	Open a block to see its property dialogue
Long hold on a block	Delete the block
Single tap on a pin	Make a connection between blocks
Single tap on a connection	Delete a connection between blocks
Hold and drag on a block	Move blocks with your fingers
Swipe down/up on the main canvas	Hide/show the bottom bar

block and a *Plot* block on the main canvas. Each block is very easy to move by placing a finger on it and dragging it to a new location. To delete a block, hold for several seconds on the corresponding block. A pop-up menu appears. Select "Delete". To connect blocks, tap once inside the blue dot on the right side of the *Signal Generator* block, then tap once the blue dot on the left side of the *Filter* block. A black line will be drawn. The connection in iJDSP is made in the direction of the signal flow (i.e. signal transmits from an output of a block to an input of another block), otherwise you will fail making connections and an alert is shown. The block diagram of this filter example is shown in Figure 2.3.

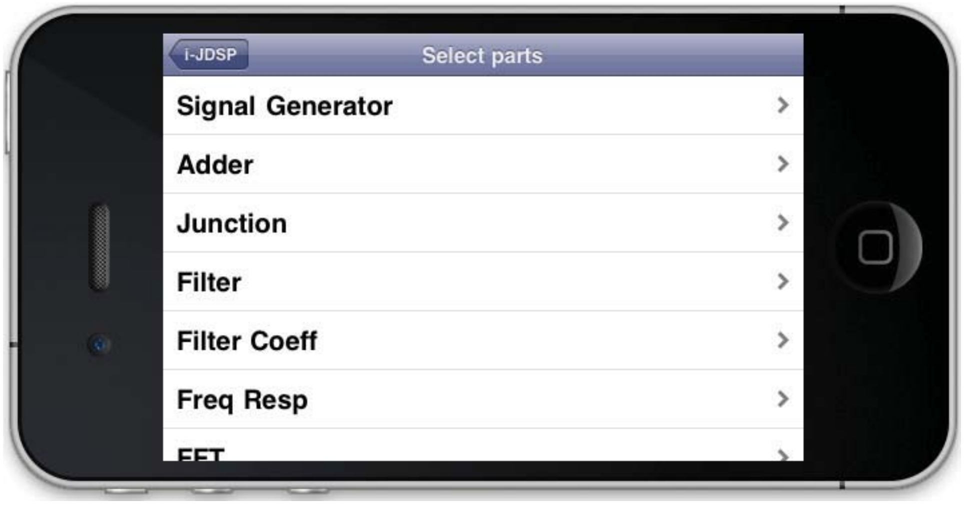


Figure 2.2: The List of Functions Available in iJDSP.

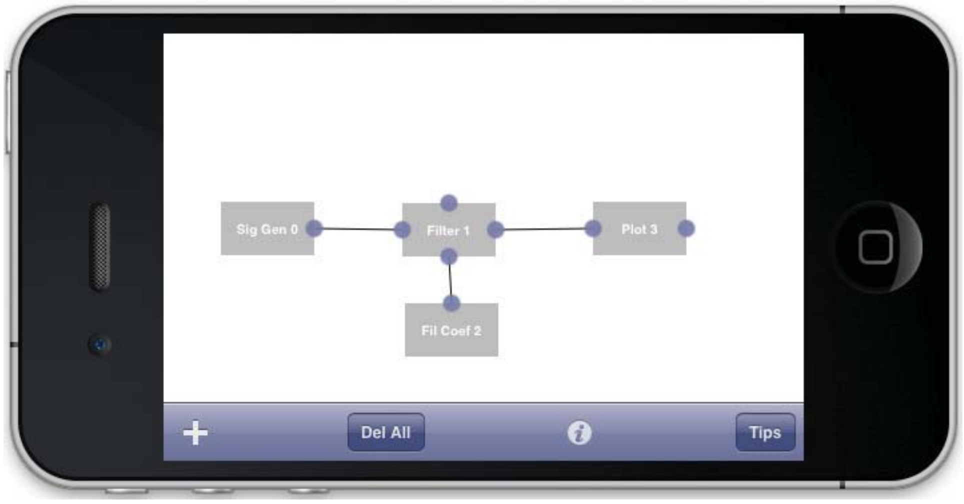
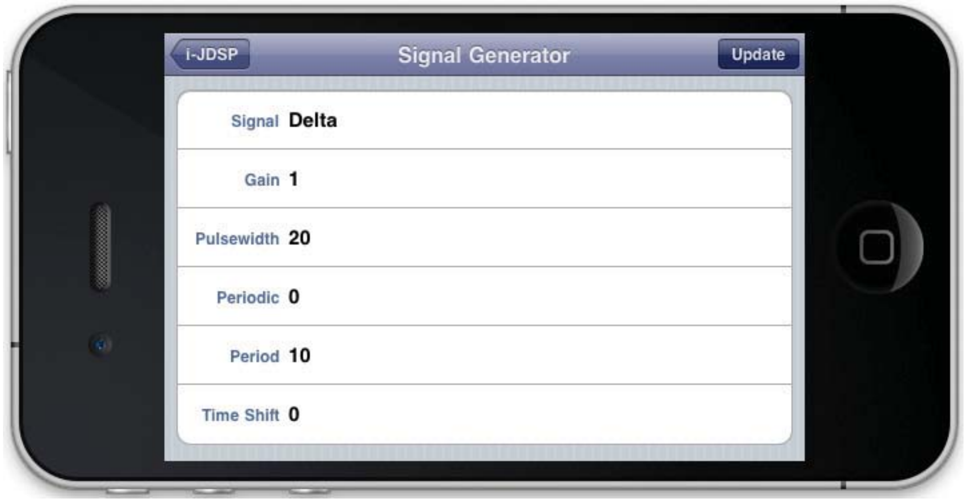


Figure 2.3: An Example of a IIR Filter Simulation.

In what follows, methods to provide parameters for the various blocks are described. Double tap on the *Sig Gen* block. A list that details all parameters to define a signal such as signal type, gain and pulse width will appear as shown in Figure 2.4.



Signal	Delta
Gain	1
Pulsewidth	20
Periodic	0
Period	10
Time Shift	0

Figure 2.4: The Parameters in the Signal Generator Block.

Change the signal type to Delta. Then double tap on the *Filter Coeff* block, and enter the following filter coefficients: $b_0 = 1, a_0 = 1, a_1 = -0.9$. Double tapping on the *Plot* block will show the impulse response of a first-order causal IIR filter defined by the difference equation $y(n) = x(n) + 0.9y(n-1)$ along with the transfer functions defined as follows,

$$H(z) = \frac{1}{1-0.9z^{-1}}, \quad |z| < 0.9 \tag{2.1}$$

The closed-form expression for the impulse response is $h(n) = 0.9^n u(n)$ and is shown in Figure 2.5.

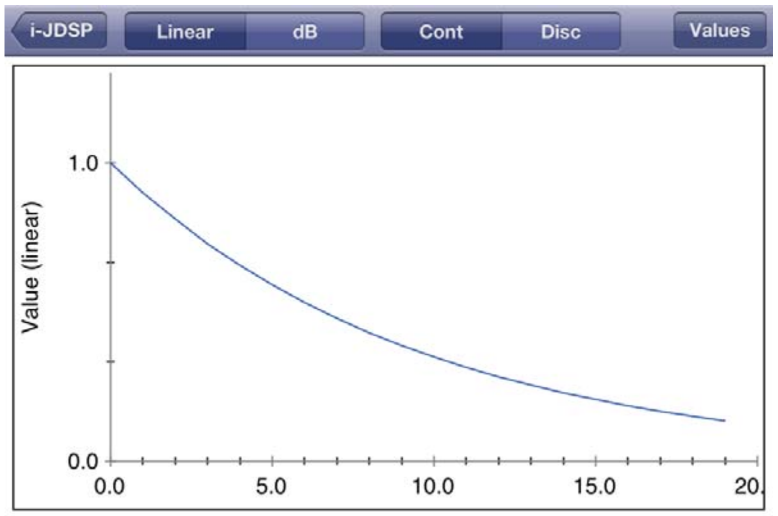


Figure 2.5: Impulse Response in the Example.

The process of the iJDSP application is illustrated in Figure 2.6. Once iJDSP detects a touch event, it will first decide which step should be processed according to the type of the touch. After completing corresponding operation, iJDSP will go back to start status to wait detection. Activities such as Establishing/ deleting a connection, and updating parameters will make re-execution of all the blocks on the data flow. Once completing reloading data, it will go back to start status as well.

2.2 System Structure

This part explains the system structure in iJDSP from three aspects: Model-View-Control model, inheritance, and delegation mechanism.

2.2.1 The Model-View-Controller Paradigm

As a GUI-based application, the design of iJDSP is guided by a concept called Model-View-Controller (MVC), which aims to logically organize the code and ensure maxi-

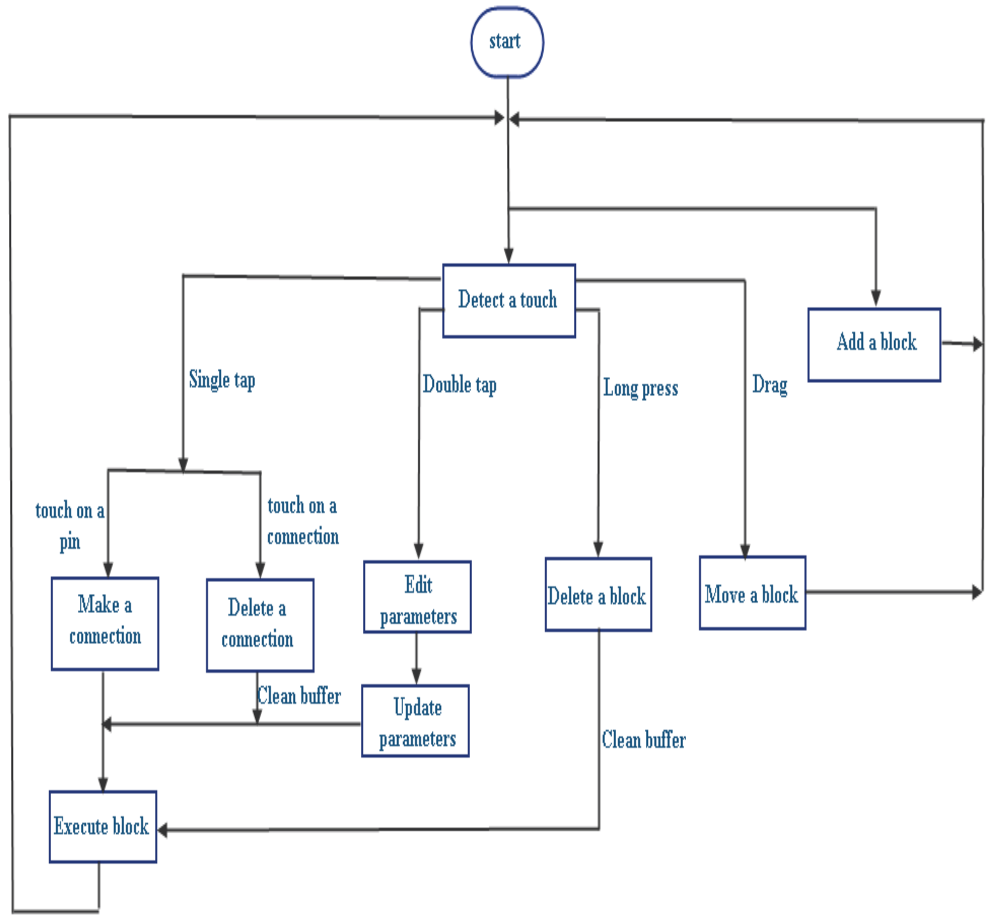


Figure 2.6: Flow Chart of iJDSP.

mum reusability of the code [35]. As shown in Figure 2.7, iJDSP divides source code into three parts. Model is the category that consists of all the classes that implement DSP algorithms. Those DSP algorithms are inherited from a root class called *NSObject*. View is made up of all graphs and other elements that user can see and interact with. All of view objects are inherited from a basic class *UIView*. Controller is to bind the model and view together, and manage communication between them. Therefore the contents of views can be updated according to user actions. Those controller objects are inherited from *UIViewController* class. The inheritance pattern will be explained later.

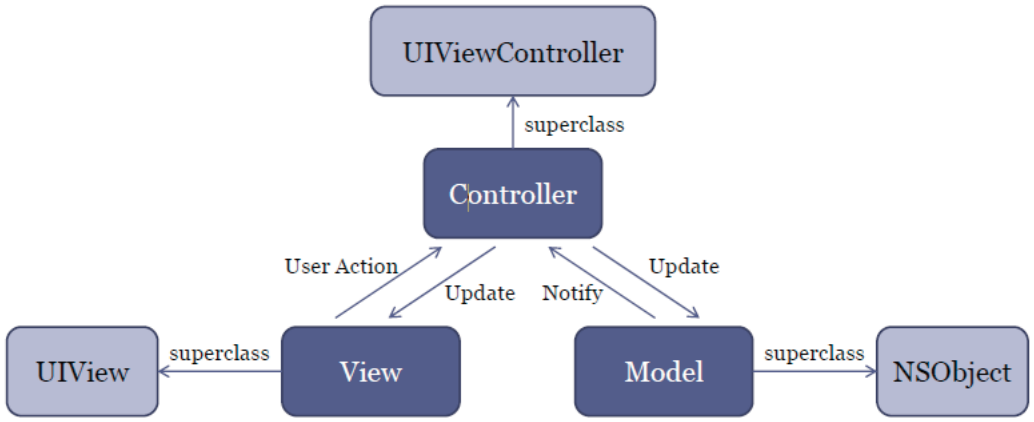


Figure 2.7: MVC Model of iJDSP.

2.2.2 Inheritance

Inheritance is very important to object-oriented programming. The class diagram in Figure 2.8 describes an inheritance pattern in iJDSP. As a root class in the foundation framework, NSObject defines some essential methods of an object such as *alloc*, *init* and *dealloc* to deal with the memory management [36]. *Alloc* and *init* methods are used to allocate memory of an object and declare its ownership. Since Objective-C does not support garbage collection as Java, we need call *dealloc* to destroy the object if we no longer need it. Part is a subclass of NSObject. Part declares a set of general variables about a block such as *partName*, *parents* and *children*. It also defines common methods including *updateBlock*, *executeBlock*, and *setDatawithIndex*. In Figure 2.8, the *FIRFilter* block inherits those variables and methods from Part. Besides, it defines private variables such as ftype which refers to the type of filter and private methods such as *FilterDesign* which implements algorithms used in FIR filter design.

Each block has *parents* and *children*. Either of them is an array consisting of three pointers which point to its preceding blocks or following blocks. Figure 2.9 shows the view of a part including its three output pins (Pin3, Pin4, Pin5) and three

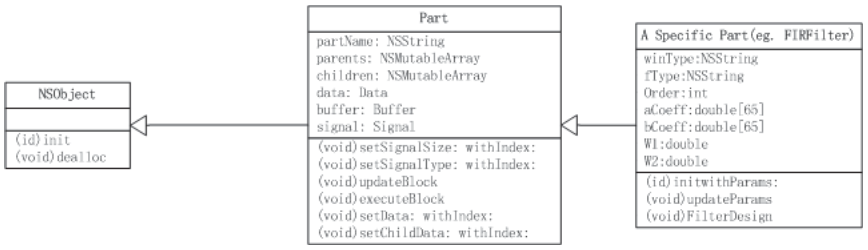


Figure 2.8: Class Diagram of Inheritance in iJDSP.

Table 2.2: The Mapping Relationship between the Pointers and the Pins.

Pin number	Pointers
0	parent[0]
1	parent[1]
2	parent[2]
3	children[2]
4	children[0]
5	children[1]

input pins (Pin0, Pin1, Pin2). The corresponding relation between those pointers and pins is shown in Table 2.2. Any specific function such as SigGen, FFT or FIRFilter is a subclass of Part. The private variables and methods are defined and implemented in their own classes.

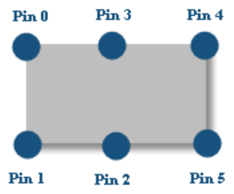


Figure 2.9: The View of a Part.

2.2.3 *Delegation Mechanism*

Delegation is another important mechanism applied in iJDSP. Some classes have limited common behaviors shared with each other. In order to communicate between those classes, therefore, we describe that relationship with a delegation mechanism [37]. A delegation mechanism applied in iJDSP is illustrated in Figure 2.10. When the user clicks the save button after editing the parameter, the object *AttributeEditor* sends a message, *saveforKey*, to its delegate *PartDetailViewController*. *PartDetailViewController* will save that value and show it on the list of parameters. Similarly, once the user clicks the "Add button, message *partAdded* will be sent by the *PartDetailViewController* object to its delegate *PartsTableViewController*. To implement the request of *partAdded*, the *PartsTableViewController* object asks its delegate *RootViewContorller* to draw the corresponding part on the main canvas.

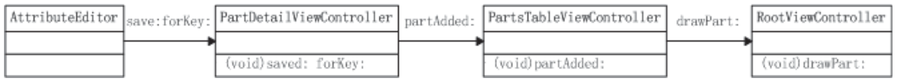


Figure 2.10: Class Diagram of Delegation in iJDSP.

2.2.4 View Hierarchy and Event Handling

As a view-based application, iJDSP needs to deal with multiple views and handle different events on them. The view hierarchy is illustrated in Figure xx. The view of navigation controller (a) and main canvas (b) are located at the bottom. Above the main canvas, DSP function diagrams (c) are added as subviews of the main canvas. At the most top is the tool bar and its bar buttons.

This view architecture not only determines the order of views displayed on the screen, but also the order to handle user actions. The responder chain are defined in Cocoa's event-handling machinery is used to handle different user actions. Those touch events in iJDSP are already listed in Table 2.1 in previous context. Take making a connection for example. As shown in Figure 2.12, once user single taps on the pin, a straight line will be drawn on the canvas.

Pin view is the first responder on the chain to handle this touch event. Pin view does not have corresponding method, then the event is passed along to its viewcontroller or superview. In this case, this event is delivered to its parent view called part

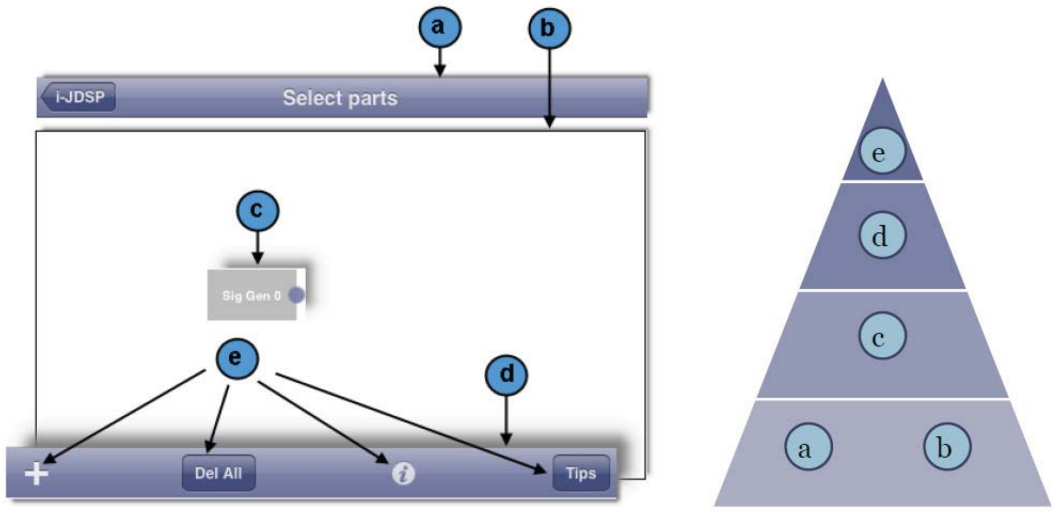


Figure 2.11: View Hierarchy in iJDSP.

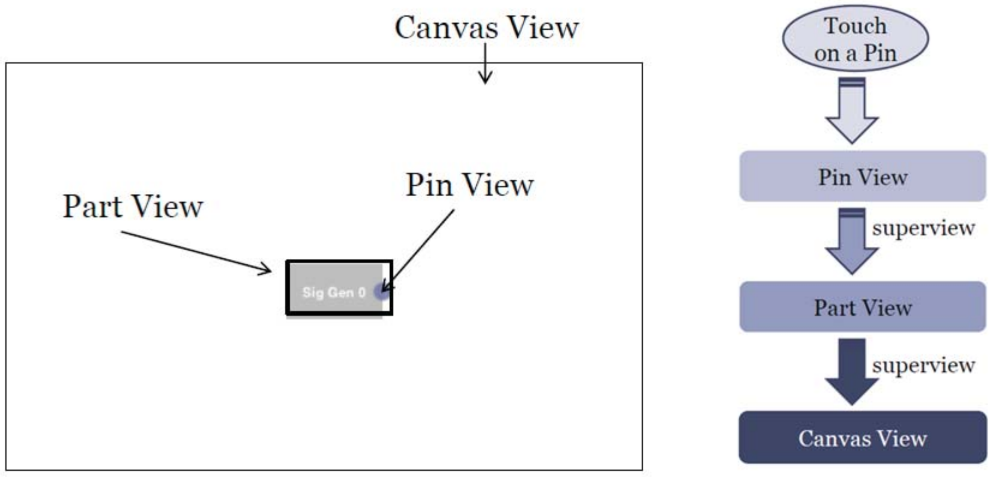


Figure 2.12: Example of Responder Chain in iJDSP.

view. After examining a series of objects on the chain, this event finally arrives in main canvas's frame, and drawing method is called. If there is no corresponding processing methods found in the chain, the event will be discarded in the end.

Chapter 3

ALGORITHMS AND IMPLEMENTATIONS

iJDSP has a rich suite of signal processing functions dealing with fundamental issues in digital signal processing courses including signal representations in discrete-time domain and z-domain, transform analysis of linear-time-invariant (LTI) system, techniques in digital filter design, multirate signal processing and spectral analysis.

Functions in iJDSP include signal generators, arithmetic operations, and basic DSP functionality.

- Basic operations: *Signal Generator, Junction, Adder, Plot, Convolution*.
- Filtering: *Filter, Filter Coeff, PZ Plot, PZ2Coef*.
- Spectral analysis: *Freq Resp, Up Sampling, Down Sampling, FFT(IFFT), Window, Peak Picking*.
- Audio processing: By using build-in microphone in iPhone/ iPad, iJDSP has capability to deal with real sound. Those functions are *MIDI, DTMF, Sound Recorded* and *Sound Player*.

3.1 Demo

Through animation and visual display of concepts, students are easy to accept knowledge that are not obvious from traditionally static explanations from textbooks or blackboard. There are three demos in iJDSP, *frequency response demo*, *PZ placement*, and *convolution demo*. The *Frequency Response Demo* block shows three samples of filters representing low-pass filter , high-pass filter and band-pass filter respectively. The *PZ Placement* block explains the relationship between the frequency response of the filter and the positions of poles and zeros. By dragging poles and zeros in the pz-plane shown in the left part of Figure 3.1, we can see corresponding frequency response in real time.

The *Convolution Demo* block graphically visualizes the convolution process involving two signals both for the continuous-time and discrete-time signals. Figure

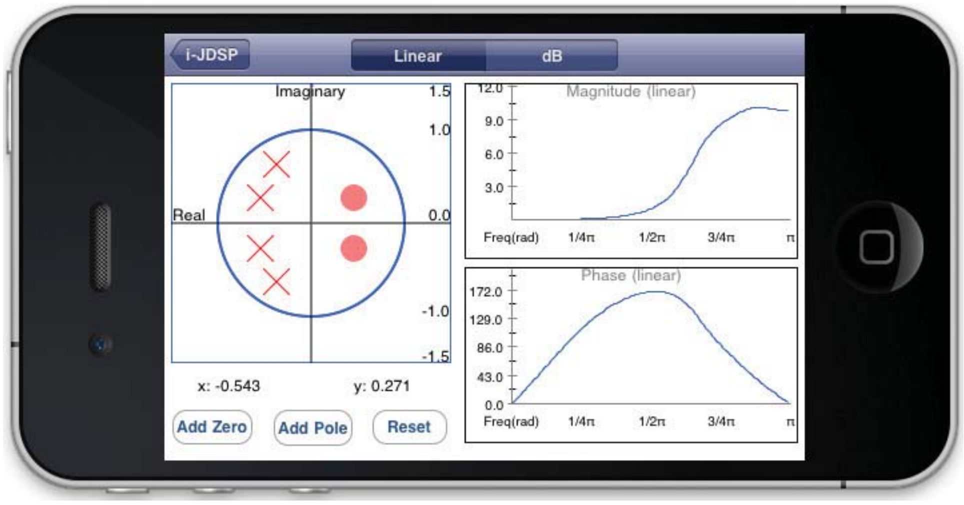
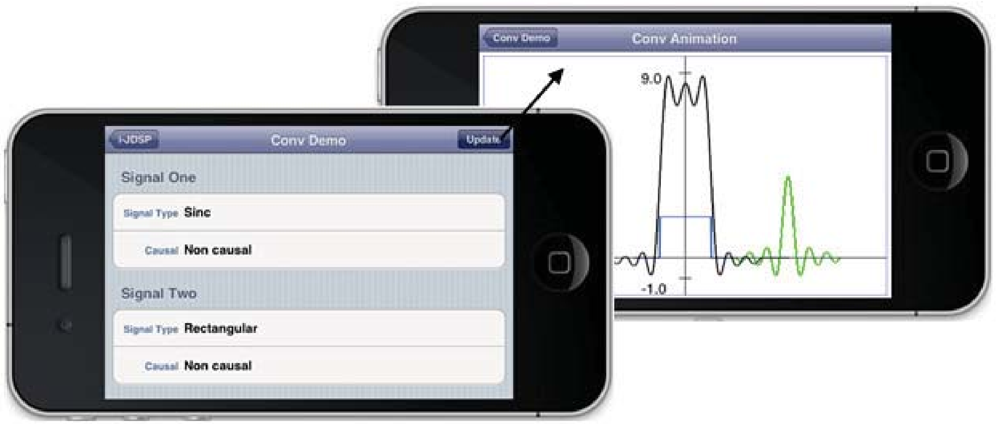


Figure 3.1: Pole-zero and Frequency Response Computation in the PZ Placement Block.

3.2 and Figure 3.3 show the GUI design of the *Convolution Demo* block. Other than providing a set of pre-defined sequences, discrete convolution allows users create their own signals by holding and dragging each sample to the desired amplitude. Therefore one more view is added after choosing signal types as shown in Figure 3.3.



Signal One

Signal Type: Sinc

Causal: Non causal

Signal Two

Signal Type: Rectangular

Causal: Non causal

Figure 3.2: GUI Design of Continuous Convolution.

The continuous-time convolution is defined as followed,

$$y(t) = \int_{-\infty}^{\infty} x_1(\tau)x_2(t-\tau)\,\mathrm{d}\tau = x_1(t) * x_2(t) \tag{3.1}$$

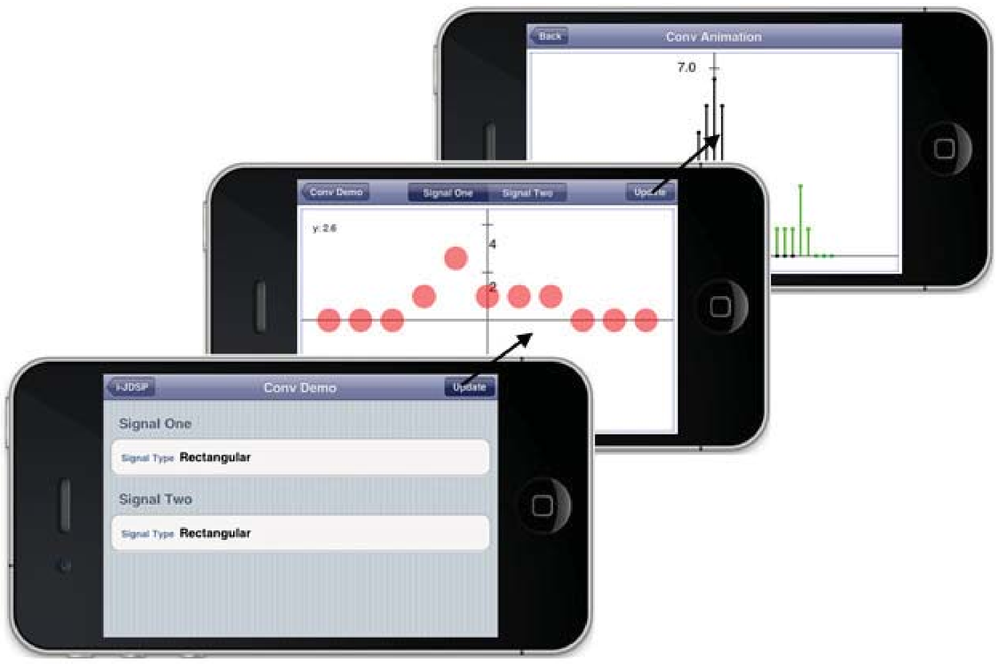


Figure 3.3: GUI Design of Discrete Convolution.

The asterisk between $x_1(t)$ and $x_2(t)$ is the convolution operation [38]. In the iJDSP, the continuous-time signal is approximated by 121-points of sequence.

The discrete-time convolution is defined by the sum of two sequences,

$$y[n] = \sum_{k=-\infty}^{\infty} x_1[k]x_2[n-k] = x_1[n] * x_2[n] \tag{3.2}$$

If L_1 is the length of the sequence $x_1[n]$, L_2 is the length of the sequence $x_2[n]$. The length of sequence $y[n]$ equals to $L_1 + L_2 - 1$. The logic design in the convolution demo is illustrated in the Figure 3.4. *ConvSigGen* and *DiscConvSigGen* are used by *ConvDemo* to generate instances of input signal.

The graphic convolution design is illustrated in Figure 3.5. *ConvDemoDetailViewController* call *DiscConvPlotViewController* for discrete-time convolution and *ConvPlotViewController* for continuous-time convolutoin. They get signals generated from *ConvDemo*and set the interval time using *NSTimer*. Hense *draw()* will be called by the timer at the scheduled time to refresh the graph. The animation of convolution is then graphically shown on the screen. An exercise using convolution demo was de-

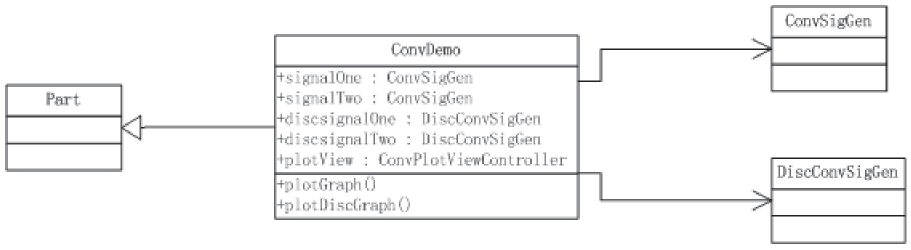


Figure 3.4: The UML Diagram for the Convolution Demo Block.

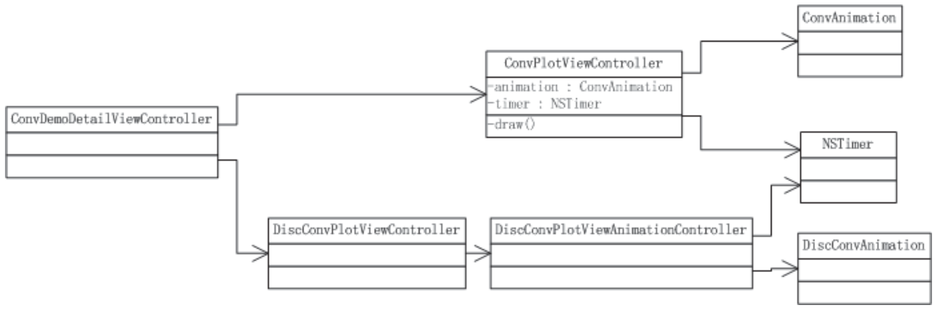

ConvDemoDetailViewController
ConvPlotViewController
-animation : ConvAnimation
-timer : NSTimer
-draw()
ConvAnimation
NSTimer
DiscConvPlotViewController
DiscConvPlotViewAnimationController
DiscConvAnimation

Figure 3.5: The UML Diagram for the Graphically Animation in the Convolution Demo Block.

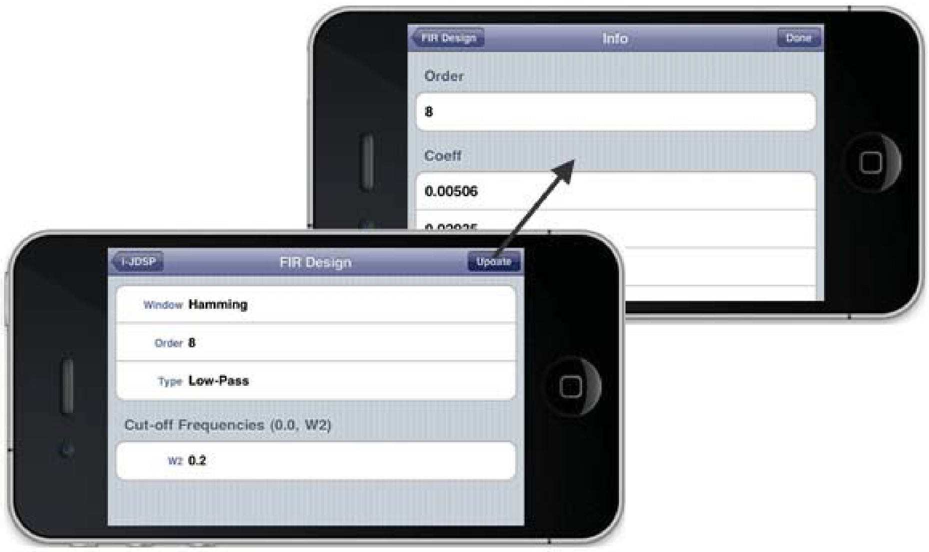
signed and assessed by students from senior-level DSP course. The assessment results are presented in later chapter, and the exercise is given in Appendix.

3.2 FIR Filter Design

iJDSP supports multiple FIR filter design modules including the *FIR Design* block, the *Kaiser* block and the *Parks-McClellan* block. Algorithms used in these FIR filter design modules obey the constraint that the phase of the frequency response is linear [7]. Based on this constraint, two design techniques, the window method and optimal minmax method (Parks-McClellan algorithm) are developed.

3.2.1 Windowing Method

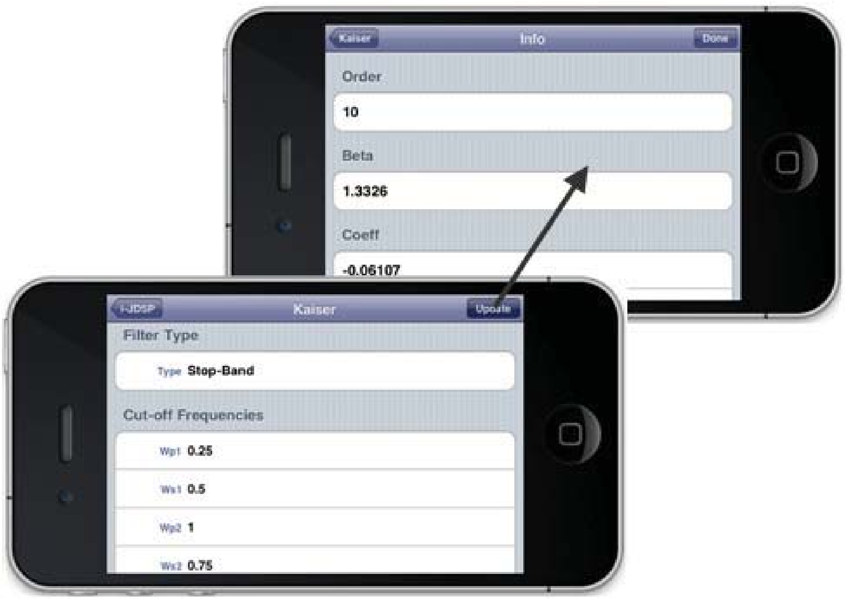
As shown in Figure 3.6, the *FIR Design* block has four user entries defining window type, order, filter type and cut-off frequencies. After tapping the “Update” button, the filter coefficients are calculated to best fit the specified parameters. The Kaiser design shown in 3.7 provide three entries including filter type, cut-off frequencies and tolerances. The filter order, beta of the Kaiser window and the coefficients are calculated to fit those specifications by multiplying the Fourier series of the ideal filter with a Kaiser window. In the following content, algorithms are described in detail.



Order
8
Coeff
0.00506

Window Hamming
Order 8
Type Low-Pass
Cut-off Frequencies (0.0, W2)
W2 0.2

Figure 3.6: GUI Design of FIR Filter Design Module based on Windowing Method.



Order
10
Beta
1.3326
Coeff
-0.06107

Filter Type
Type Stop-Band
Cut-off Frequencies
Wp1 0.25
Ws1 0.5
Wp2 1
Ws2 0.75

Figure 3.7: GUI Design of FIR Filter Design Module based on Kaiser Design.

A causal M order FIR filter can be obtained by truncating an ideal non-causal impulse response of the IIR system with a finite length window *w(n)*,

$$h_{FIR}(n) = w(n)h_d(n - M/2), 0 \leq n \leq M \tag{3.3}$$

The desired filter is designed using the FS method according to the specifications on the frequency-selective filter. Typically there are four parameters used to determine a discrete-time filter, the passband cutoff frequency W_p, the stopband cutoff frequency W_s, the passband gain variance *PB(dB)* and the stopband gain variance *SB(dB)*. FIR filter design in iJDSP provides multiple types of windows. The equations of windows are defined in Equation 3.4 to Equation 3.9, the corresponding window is plotted in Figure 3.8 to Figure 3.13 [39].

- *Rectangular*

$$w(n) = \begin{cases} 1, & 0 \leq n \leq M \\ 0, & otherwise \end{cases} \tag{3.4}$$

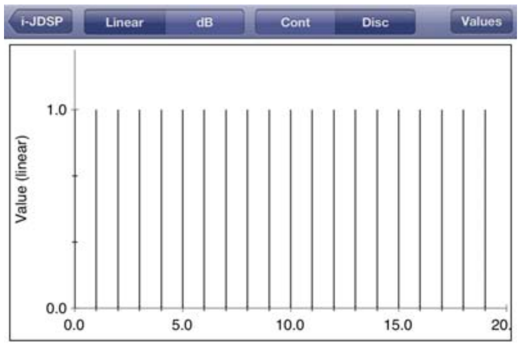


Figure 3.8: Rectangular Window.

- *Bartlett(Triangular)*

$$w(n) = \begin{cases} 2n/M, & 0 \leq n \leq M/2 \\ 2 - 2n/M, & M/2 < n \leq M \\ 0, & otherwise \end{cases} \tag{3.5}$$

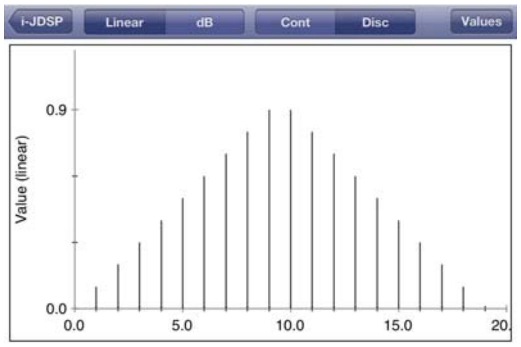


Figure 3.9: Rectangular Window.

- *Hanning*

$$w(n) = \begin{cases} 0.5 - 0.5cos(2\pi n/M), & 0 \leq n \leq M \\ 0, & otherwise \end{cases} \tag{3.6}$$

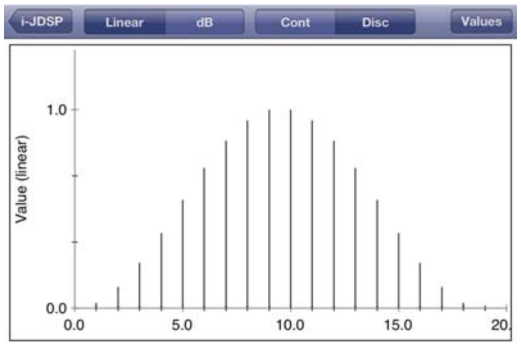


Figure 3.10: Hanning Window.

- *Hamming*

$$w(n) = \begin{cases} 0.54 - 0.46cos(2\pi n/M), & 0 \leq n \leq M \\ 0, & otherwise \end{cases} \tag{3.7}$$

- *Blackman*

$$w(n) = \begin{cases} 0.42 - 0.5cos(2\pi n/M) + 0.08cos(4\pi n/N), & 0 \leq n \leq M \\ 0, & otherwise \end{cases} \tag{3.8}$$

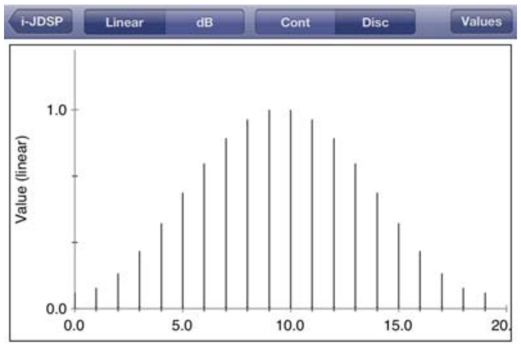


Figure 3.11: Hamming Window.

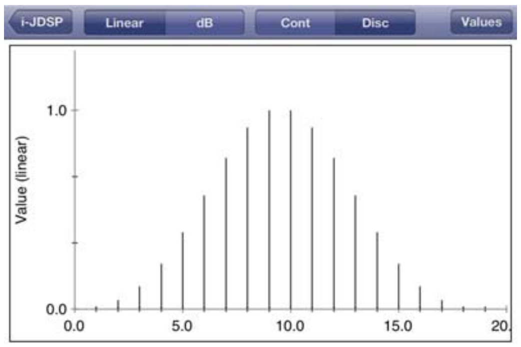


Figure 3.12: Blackman Window.

- *Kaiser*

$$w(n) = \begin{cases} \frac{I_0(M,\beta)}{I_0(\beta)}, & 0 \le n \le M \\ 0, & otherwise \end{cases} \tag{3.9}$$

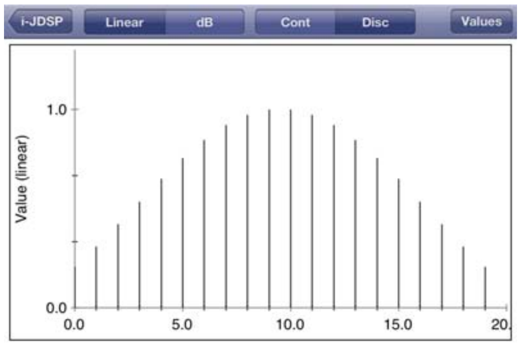


Figure 3.13: Kaiser Window with Beta = 3.

Table 3.1: Frequency-Domain Characteristics of Windows.

(L+1)-point Window	Mainlobe Width	Strongest Sidelobe Level (dB)
Rectangular	$4\pi/L+1$	-13
Bartlett	$8\pi/L$	-25
Hamming	$8\pi/L$	-41
Hanning	$8\pi/L$	-31

The relationship between windows beamwidth and sidelobe level is shown in Table 3.1. Compared to the rectangular window, tapered windows such as the bartlett window have lower sidelobe characteristics but wider mainlobe.

In the Kaiser function, I_0 represents the zeroth order modified Bessel function. Parameter M represents the length of the window. Parameter β controls the shape of the Kaiser window. It is a rectangular window when β equals to 0. As β increases, the varying degree of the window becomes larger [38]. This relationship is illustrated in Figure 3.14.

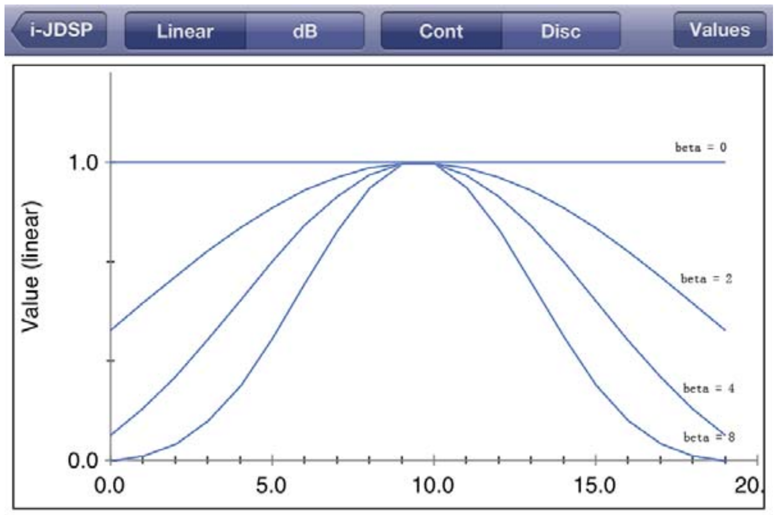


Figure 3.14: The Kaiser Window Changes with Parameter Beta.

The order of Kaiser filter design is given by,

$$M = \frac{A - 7.95}{2.285 \times 2\pi \times \Delta W} \tag{3.10}$$

where the difference between two cutoff frequencies ΔW is defined as $\Delta W = |W_p - W_s|$.

The parameter β in Eq. (3.9) is defined as,

$$w(n) = \begin{cases} 0.1102(A-8.7), & A > 50 \\ 0.5842(A-21)^{0.4} + 0.07886(A-21), & 21 \leq A \leq 50 \\ 0, & A < 21 \end{cases} \tag{3.11}$$

A(dB) is defined to be the smaller one between the passband gain variance *PB* and the stopband gain variance *SB* [38].

The block diagram in FIR filter design is shown in Figure 3.15. *FIRFilter* and *KaiserFilter* are inherited from superclass *Part*. They define parameters interfacing with views such as window type, filter type, order and coefficients. *FIRDesign* class implements FIR design algorithms. The instance of *FIRFilter* or *KaiserFilter* uses methods in *FIRDesign* to calculate corresponding coefficients.

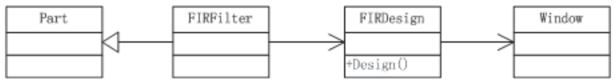


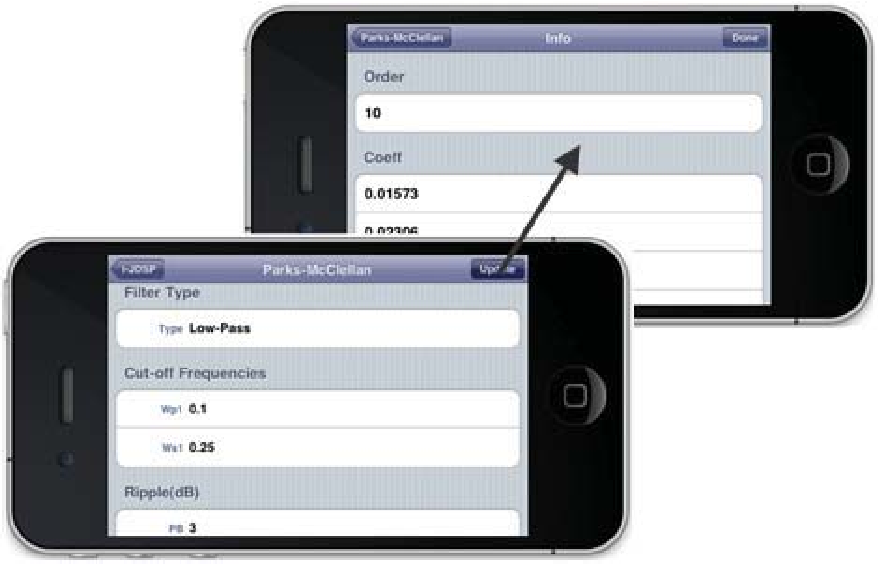
Figure 3.15: The UML Diagram for the FIR Filter Design.

3.2.2 The Parks-McClellan Filter Design

The GUI interface of the *Parks-McClellan* block is shown in Figure 3.16. Three entries, filter type, cut-off frequencies and tolerances, are defined, and then filter coefficients are calculated using the Min-Max algorithm. The frequency response is shown in Figure 3.17.

In order to minimize the maximum difference between the impulse response of the designed filter and the ideal filter for a certain filter order M, Parks-McClellan algorithm is proposed by minimizing the following expression,

$$\max_{\Omega \in \Theta} |P(e^{j\Omega})(H_d(e^{j\Omega}) - |H(e^{j\Omega})|)| \tag{3.12}$$



Order
10
Coeff
0.01573
0.02306
Filter Type
Type Low-Pass
Cut-off Frequencies
Wp1 0.1
Ws1 0.25
Ripple(dB)
PB 3

Figure 3.16: GUI design of FIR Filter Design Module based on Parks-McClellan Algorithm.

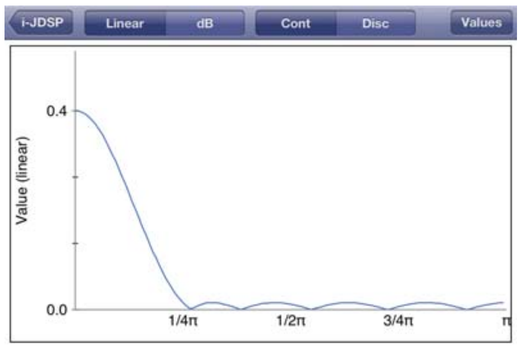
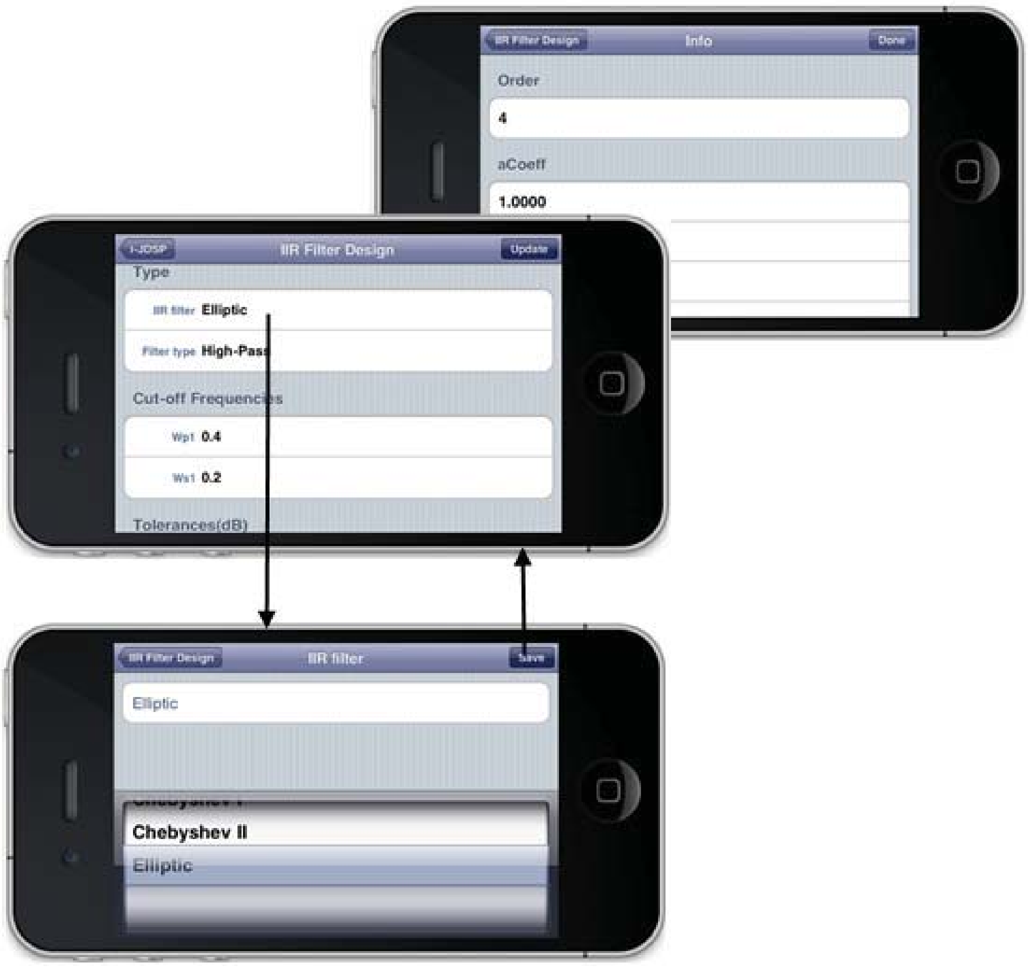


Figure 3.17: Frequency Response of the Parks-McClellan FIR Filter.

3.3 IIR Filter Design

IIR filter design is developed upon the approximation of the analog filter using bilinear transformation. There are four types of IIR filter approximation methods provided in the *IIR Design* block, Buttorworth, Chebyshev I, Chebyshev II, and Elliptic [38, 39] developed in iJDSP. Figure 3.18 shows the user interface of IIR filter design in iJDSP. There are four entries including IIR type, filter type, cut-off frequencies and tolerances provided in this interface. The frequency response of Elliptic IIR filter is shown in Figure 3.19.



Order
4
aCoeff
1.0000

Type
IIR filter Elliptic
Filter type High-Pass
Cut-off Frequencies
Wp1 0.4
Ws1 0.2
Tolerances(dB)

Elliptic

Figure 3.18: GUI Design of IIR Filter Design Module based on IIR Analog Approximation.

The specifications of IIR filter are determined by normalized passband cutoff frequency W_p, normalized stopband cutoff frequency W_s, gain at passband edge PB, and gain at stopband edge SB. A typical design pattern of IIR filter is described here: (1) determine the filters order according to the given parameters; (2) calculate the desired cutoff frequencies of the filter; (3) find stable s-domain poles; (4) calculate z-domain poles and zeros; (5) apply transformations to obtain the desired filter from the prototype low-pass filter.

The forms of M-order analog lowpass filter with a desired cutoff frequency w_c and a ripple tolerance β are given by,

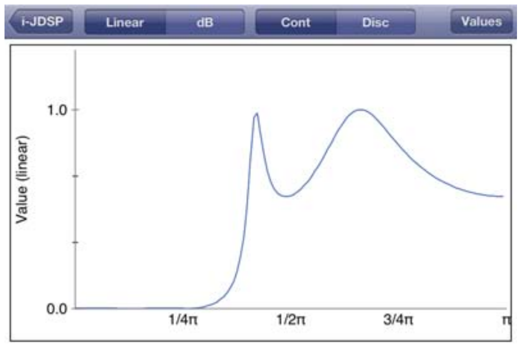


Figure 3.19: Frequency Response of the Elliptic IIR Filter.

- *Butterworth*

$$|H_a(w)|^2 = \frac{1}{1+(\frac{w}{w_c})^{2M}} \tag{3.13}$$

- *Chebyshev type I*

$$|H_a(w)|^2 = \frac{1}{1+(\eta^2 T_M^2 \frac{w}{w_c})} \tag{3.14}$$

- *Chebyshev type II*

$$|H_a(w)|^2 = \frac{1}{1+(\eta^2 T_M \frac{w_p}{w})}^{-1} \tag{3.15}$$

  The function $T_M(w)$ in Eq.(3.14) and Eq.(3.15) is the Mth-order Chebyshev polynomial defined by,

$$T_M(w) = cos(Mcos^{-1}w) \tag{3.16}$$

- *Elliptic*

$$|H_a(w)|^2 = \frac{1}{1+(\eta^2 U_M^2 \frac{w_p}{w})} \tag{3.17}$$

The function of $U_M(w)$ is a Jacobian elliptic function [39], also called the Rational Normalized Function [40]. Elliptic filter can use the smallest order to meet the same specifications.

The mapping from analog domain frequency ω to digital domain frequency W is indicated by,

$$\omega = tan\frac{W}{2} \tag{3.18}$$

Then we could find the cutoff frequency w_c and s-domain poles p_k. By applying bilinear transform, z-domain poles can be determined from the stable s-domain poles,

$$s = \frac{z-1}{z+1} \tag{3.19}$$

After getting a prototype lowpass digital filter, a digital-to-digital transformation needs to be applied in order to get desired filter. Given a prototype lowpass filter $H_LP(w)$, the desired filter is,

$$H_d(z) = H_{LP}(v^{-1}), \quad v^{-1} = F(z^{-1}) = \pm \prod_{k=1}^{M} \frac{z^{-1} - a_k}{1 - a_k z^{-1}} \tag{3.20}$$

The implementation of IIR filter design is described by class diagram shown in Figure 3.20. Four algorithms used in *IIRFilter* are implemented in *Butterworth*, *Chebyshev*, *Chebyshev2*, and *Elliptic* respectively. Expect butterworth, the other three algorithms need to use matrix operations defined in *UtilityFunction*.

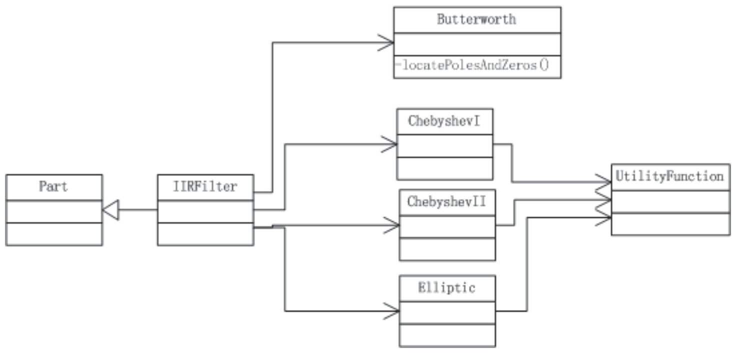


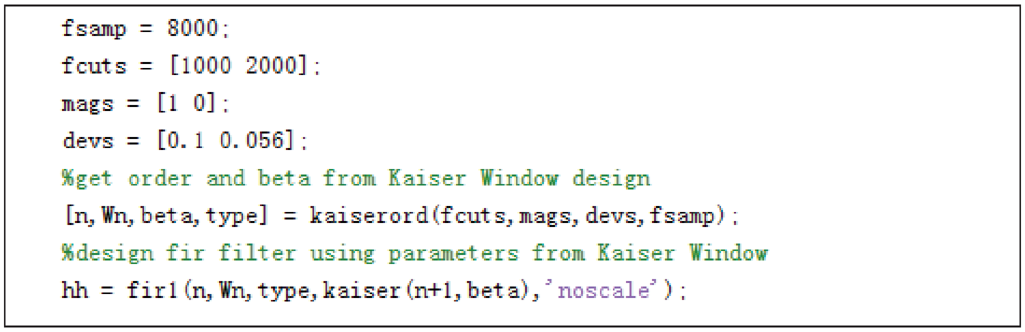
Figure 3.20: The UML Diagram of IIR Filter Design.

3.4 Verification using MATLAB

MATLAB is used to verify results from i-JDSP. We will examine the performance of filter design blocks. For FIR filter, we take Kaiser as an example, using MATLAB and iJDSP respectively to design a lowpass FIR filter with following specifications,

$$\begin{aligned} 0.9 \le |H(e^{j\Omega})| \le 1.1, \quad 0 \le \Omega \le 0.25\pi \\ |H(e^{j\Omega})| \le 0.056, \quad 0.5\pi \le \Omega \le \pi \end{aligned} \tag{3.21}$$

Corresponding MATLAB code is given in Figure 3.21, and the performance comparisons between MATLAB and iJDSP in terms of Pole-zero plane and frequency response are shown in Table 3.2.



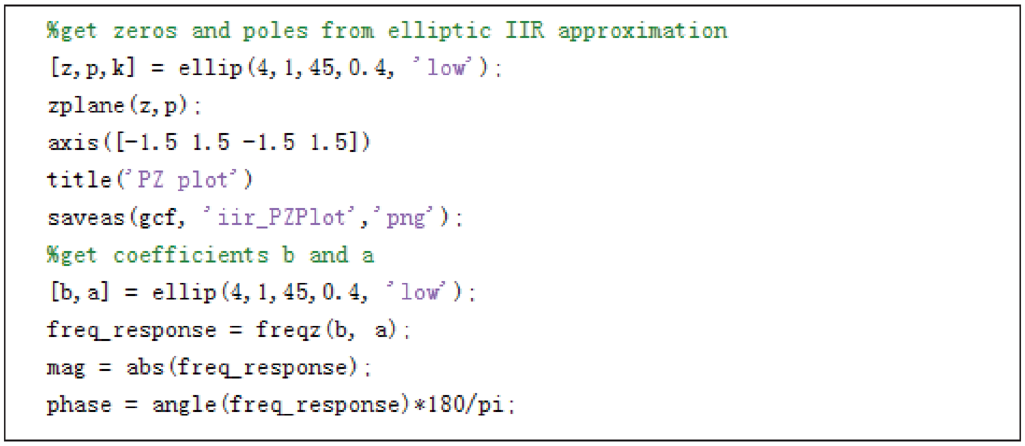
```
fsamp = 8000;
fcuts = [1000 2000];
mags = [1 0];
devs = [0.1 0.056];
%get order and beta from Kaiser Window design
[n,Wn,beta,type] = kaiserord(fcuts,mags,devs,fsamp);
%design fir filter using parameters from Kaiser Window
hh = fir1(n,Wn,type,kaiser(n+1,beta),'noscale');
```

Figure 3.21: Kaiser MATLAB code.

Take an elliptic IIR filter design as another example. Given a lowpass filter with following specifications,

- Passband Cutoff frequency: 0.4π; Stopband Cutoff frequency: 0.6π;
- Tolerance in passband: $1dB$; Tolerance in stopband: $45dB$.

MATLAB code is as well given in Figure 3.22. The performance comparisons are shown in Table 3.3.



```
%get zeros and poles from elliptic IIR approximation
[z,p,k] = ellip(4,1,45,0.4, 'low');
zplane(z,p);
axis([-1.5 1.5 -1.5 1.5])
title('PZ plot')
saveas(gcf, 'iir_PZPlot','png');
%get coefficients b and a
[b,a] = ellip(4,1,45,0.4, 'low');
freq_response = freqz(b, a);
mag = abs(freq_response);
phase = angle(freq_response)*180/pi;
```

Figure 3.22: Elliptic IIR MATLAB Code.

Table 3.2: Verify Kaiser Function in iJDSP using MATLAB.

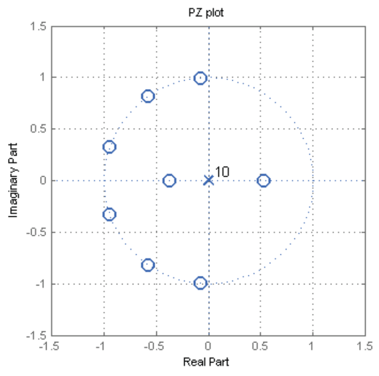
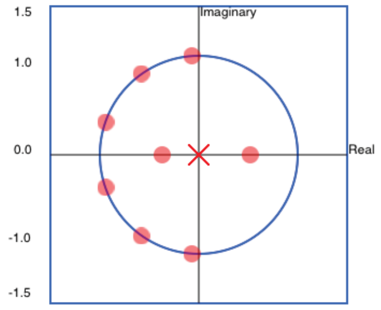
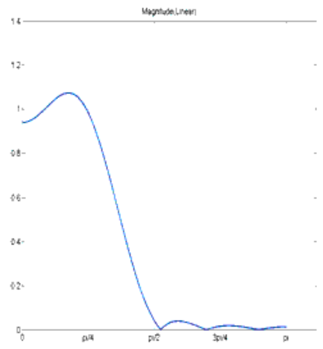
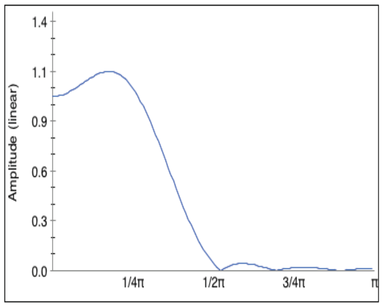
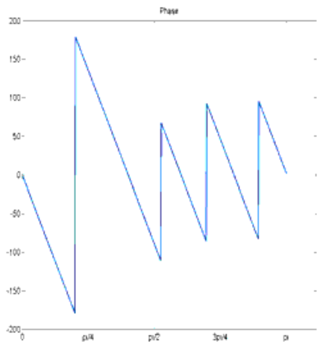
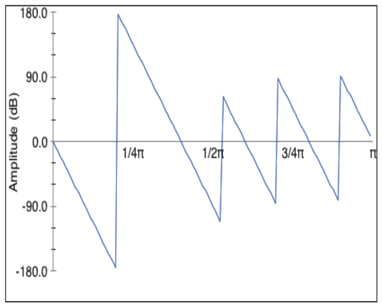
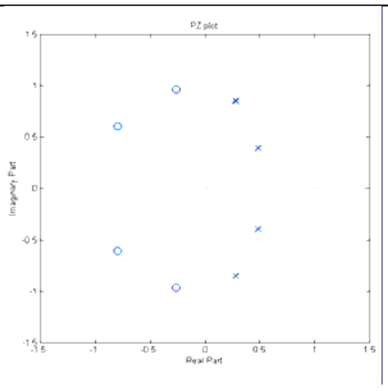
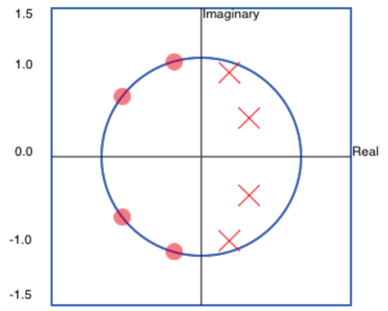
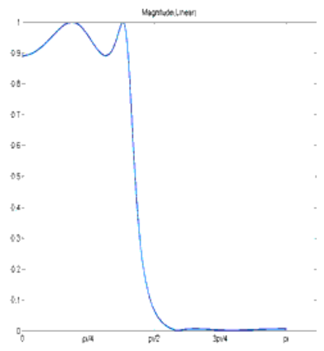
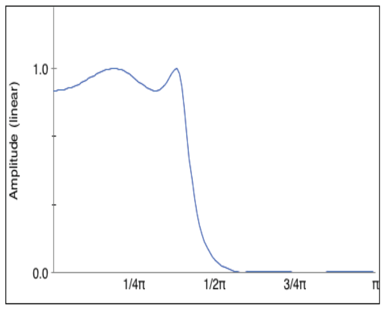
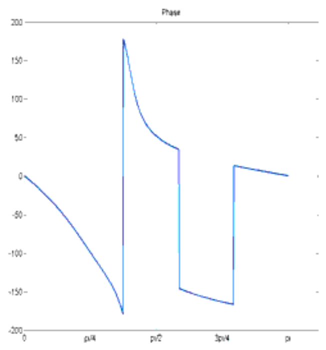
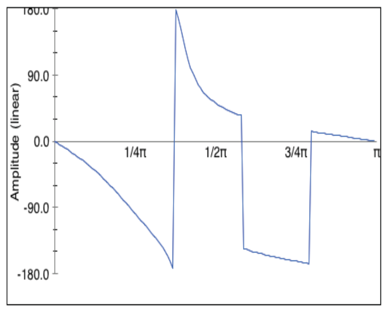
Experiment	**MATLAB**	**iJDSP**
PZ Plot		
Freq. Resp. Magnitude		
Freq. Resp. Phase		

Table 3.3: Verify Elliptic Algorithm in iJDSP using MATLAB.

Experiment	**MATLAB**	**iJDSP**
PZ Plot		
Freq. Resp. Magnitude		
Freq. Resp. Phase		

Chapter 4

EXERCISES

The object of this exercise is to provide hands-on experiences on iJDSP that help students get familiar with the environment and learn how to establish DSP simulations on this platform. This exercise is developed according to JDSP online laboratory exercises [8, 12]. It consists of four parts covering continuous/discrete convolution, z-transform, frequency response of LTI system, poles and zeros on the z-plane , filter designs, and Fast Fourier Transform. The laboratory exercises are typically performed after the students are introduced basic concepts of related filed in their lecture sessions.

4.1 Continuous/ Discrete Convolution

This exercise introduces the concepts of continuous and discrete time convolution and enables students to visualize the demonstrations in iJDSP. To begin with students are asked to analytically compute impulse responses obtained by convolving two signals. Then the *Convolution Demo* block in iJDSP is used to demonstrate the effect of causality in continuous convolution. For a given set of input signals, the *Convolution Demo* block animates the convolution procedure and plots the resulting impulse response on the same plot screen. Figure 4.1 illustrates an example of convolving a sinc signal and a rectangular signal. The black plot in Figure 4.1 is the convolution result. The students are asked to make observations on the convolution results in the following cases:

1. Causal rectangular and non causal rectangular.

2. Causal rectangular and causal rectangular.

3. Non-causal rectangular and non causal sinc.

The *Convolution Demo* block also provides the discrete-time convolution. In this part of the exercises, the students compare the results obtained by sampling two continuous signals and performing discrete convolution, with the result obtained by sampling the

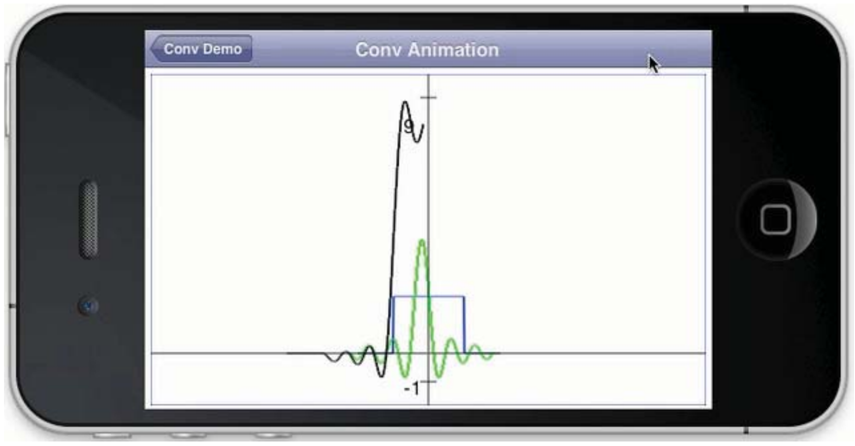


Figure 4.1: Continous-time Convolution.

convolution result obtained by convolving the two continuous signals directly. Furthermore, students are asked to sample the impulse response of a system and the given input signal. Use these sampled signals to perform a discrete time convolution, and compare with the result of the continuous-time convolution of the analog signals. Figure 4.2 shows the simulation between two discrete signals.

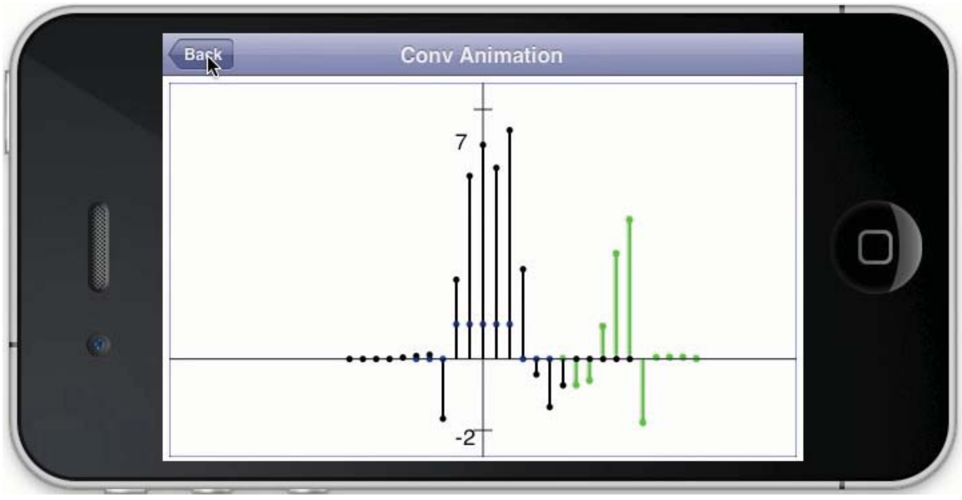


Figure 4.2: Discrete-time Convolution.

4.2 Frequency Response of LTI system

The exercise aims to help the students understand the concept of linear-time-invariant (LTI) system and z-domain representations of the impulse response of a LTI system, as well as the relationship between the pole-zero plot and the frequency response.

First of all, the students are asked to perform three typical digital filters with given transfer functions.

- $$H_1(z) = \frac{1 - 0.45z^{-1} + 0.55z^{-2}}{1 - 1.7z^{-1} + 0.6z^{-2}} \tag{4.1}$$

- $$H_2(z) = 0.06 + 0.0876z^{-1} - 0.3z^{-2} + 0.375z^{-3} - 0.3z^{-4} + 0.0876z^{-5} + 0.06z^{-6} \tag{4.2}$$

- $$H_3(z) = \frac{-0.8 + 1.62z^{-1} - 1.8z^{-2} + z^{-3}}{1 - 1.8z^{-1} + 1.62z^{-2} - 0.8z^{-3}} \tag{4.3}$$

$H_1(z)$ represents an unstable LTI system. $H_2(z)$ is a high-order FIR filter. $H_3(z)$ is a all-pass filter. Those three functions are used to varify *rootFinder* algorithm which is applied to find poles and zeros. Besides, by examining poles and zeros of each system, students can learn how the poles and zeros take effect on system's stability. They also can differ filter types by performing simulations. Establish the block diagram shown in Figure 4.3.

Type the corresponding filter coefficients in the *Filter Coeff* block. To see the poles and zeros in the z-plane, double-tap the *PZ Plot* block. To see the impulse response of the system, set "Delta" signal in the *Sig Gen* block, and double tap on the *Plot* block to see the simulation result.

The poles and zeros of $H_1(z)$ is shown in Figure 4.4.

To perform a LTI system in iJDSP, the students need to determine the impulse response $h(n)$in discrete-time domain, and the coefficients a and b from the z-transform

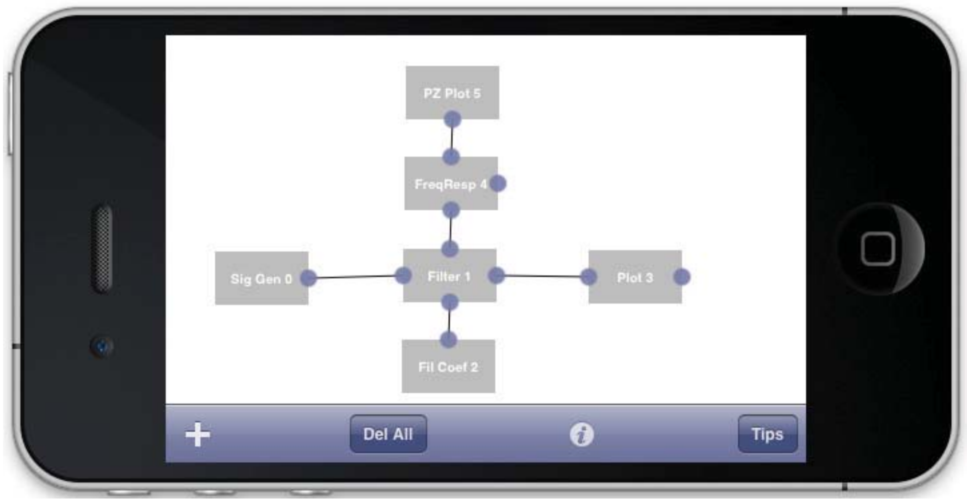
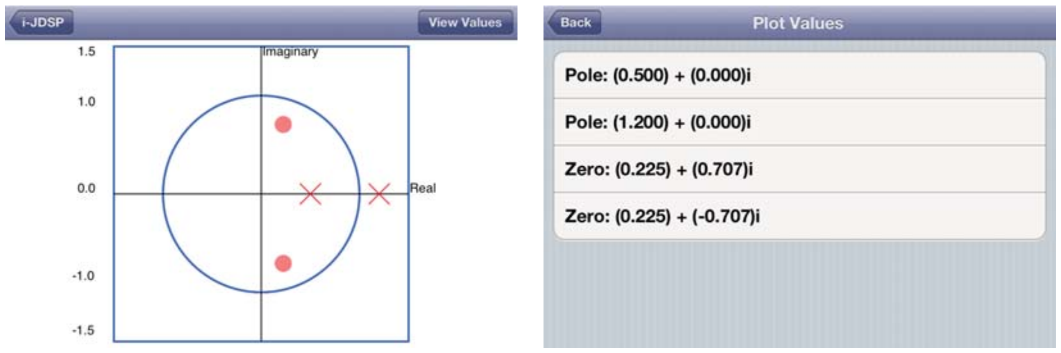


Figure 4.3: Block Diagram for Performing Frequency Response of LTI System.



Pole: (0.500) + (0.000)i
Pole: (1.200) + (0.000)i
Zero: (0.225) + (0.707)i
Zero: (0.225) + (-0.707)i

Figure 4.4: Poles and Zeros of $H_1(z)$.

of the system's impulse response. The impulse response $h_1(n)$ corresponding to $H_1(z)$ is shown in Figure 4.5.

Figure 4.5 indicates the impulse response of $H_1(z)$ is not absolutely summable. Meanwhile, student can observe that one of poles in Figure 4.4 is outside the unit circle. Therefore, student can conclude that the causal filter $H_1(z)$ is not BIBO unstable due to the ROC of $H_1(z)$ doesn't include the unit circle.

Similarly, the poles and zeros of $H_2(z)$ and $H_3(z)$ are shown in Figure 4.6 and Figure 4.7.

The frequency response of the all-pass filter $H_3(z)$ is shown in Figure 4.8.

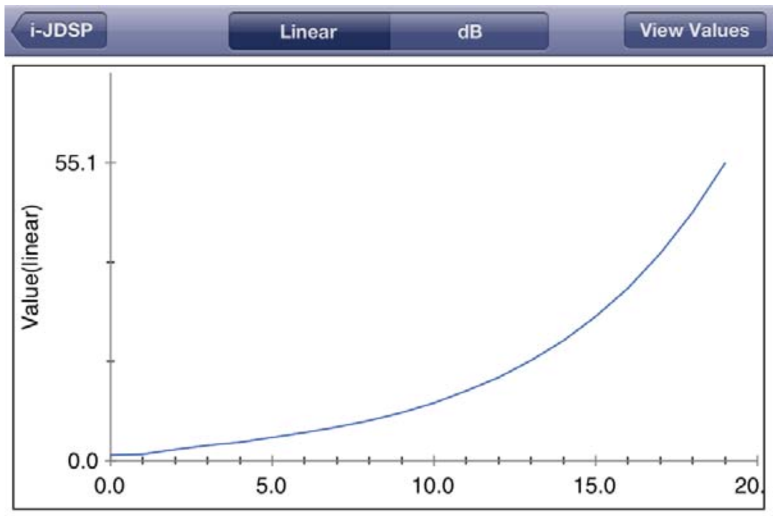
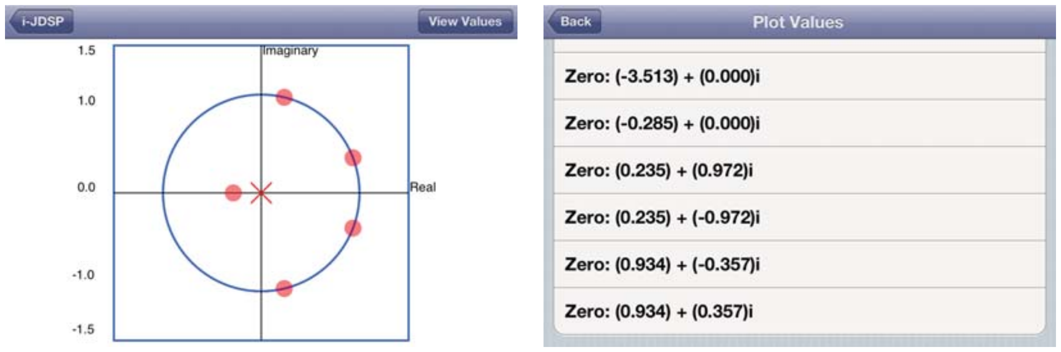


Figure 4.5: Impulse Response $h_1(n)$.



Zero: (-3.513) + (0.000)i

Zero: (-0.285) + (0.000)i

Zero: (0.235) + (0.972)i

Zero: (0.235) + (-0.972)i

Zero: (0.934) + (-0.357)i

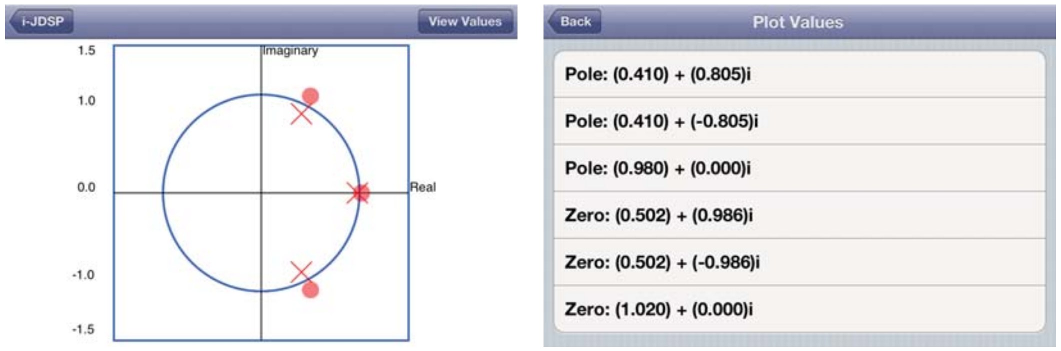
Zero: (0.934) + (0.357)i

Figure 4.6: Poles and Zeros of $H_2(z)$.

The second problem aims to learn the face that the LTI systems in cascade or parallel configurations can be combined into a single system [38]. Students are asked to perform a cascaded system using two filters, and then reconstruct the system using only one filter. The cascaded system is illustrated in Figure 4.9, and the single system is shown in Figure 4.11. The corresponding impulse response is shown in Figure. The cascaded system is given in the form of the convolution of two causal sequences shown in Eq. (4.4)

$$y(n) = \sum_{k=0}^{n} a^k b^{n-k} \tag{4.4}$$

According to Figure 4.10 and Figure 4.12, students can verify that the impulse



Pole: (0.410) + (0.805)i
Pole: (0.410) + (-0.805)i
Pole: (0.980) + (0.000)i
Zero: (0.502) + (0.986)i
Zero: (0.502) + (-0.986)i
Zero: (1.020) + (0.000)i

Figure 4.7: Poles and Zeros of $H_3(z)$.

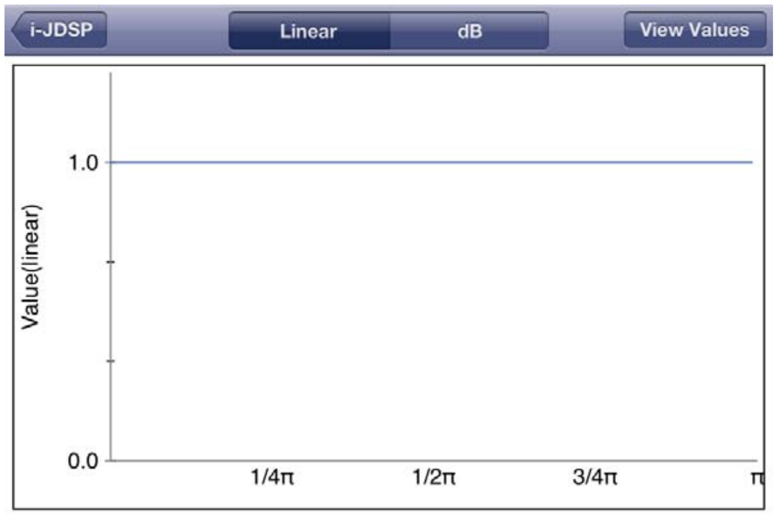


Figure 4.8: Frequency Response of $H_3(z)$.

responses of the two systems are equal.

The third problem related to LTI system is to design a filter by placing poles and zeros in the z-plane. The *PZ Placement* block enables the user to place poles and zeros graphically in the z-plane and analyze the corresponding frequency response in real time. Poles and zeros are added as conjugate pairs, and no more than 5 pairs can be entered. Graphical manipulation of poles and zeros is achieved through “Add Pole, “Add Zero and “Reset buttons on the left part of the screen. Users can place and move poles and zeros pairs. As you move the poles and zeros, the frequency response will be immediately updated. Adjust the location of the poles and zeros until the desired response is obtained. Note that poles near the unit circle give rise to spectral peaks

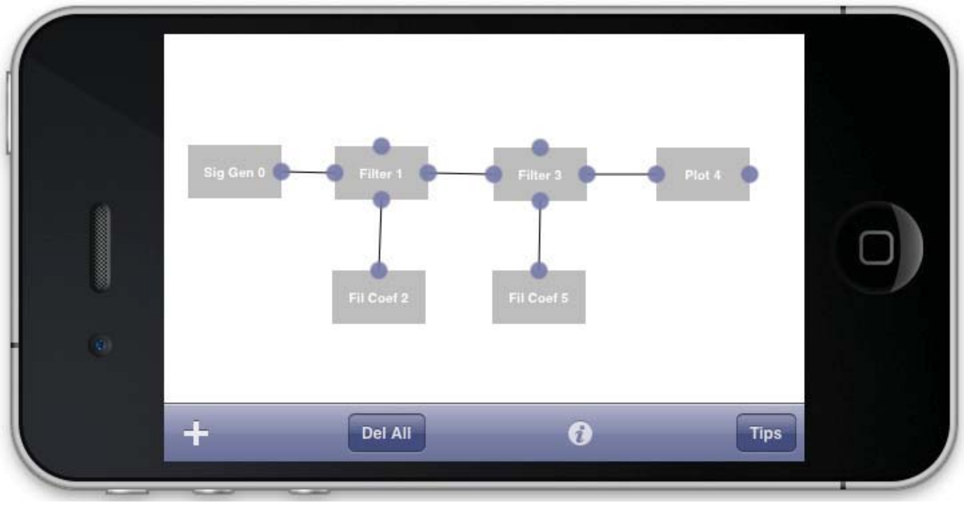


Figure 4.9: Block Diagram of the Cascaded System.

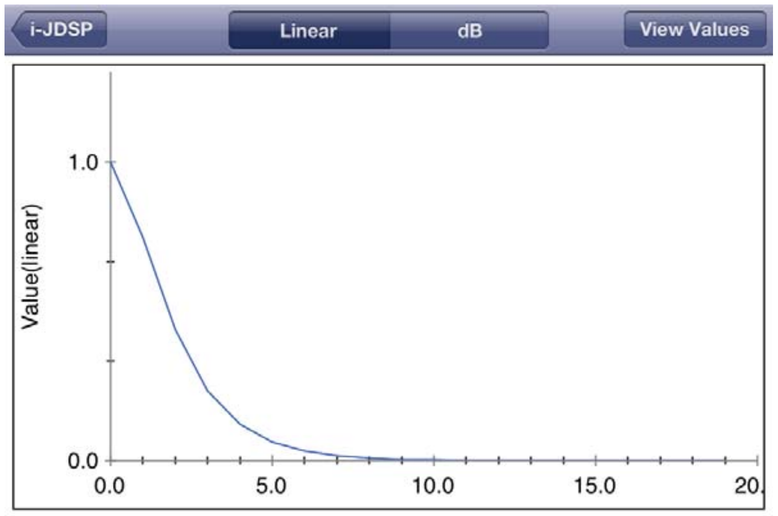


Figure 4.10: Impulse Response of the Cascaded System.

and zeros near the unit circle create spectral valleys in the magnitude of the frequency response. Students are asked to design a low-pass filter using two sets of zero pairs and one pole pair with roughly approximate cutoff frequency of $\Omega_c = \pi/4$. One example of the design can be done as shown in Figure 4.13.

4.3 Filter Design

The exercise includes three problems which aim to study the knowledge of window, compare different design methods for FIR filter and IIR filter. There are four associated filter design blocks, *Kaiser*, *FIR Design*, *Parks-McClellan* and *IIR Design*, can be used

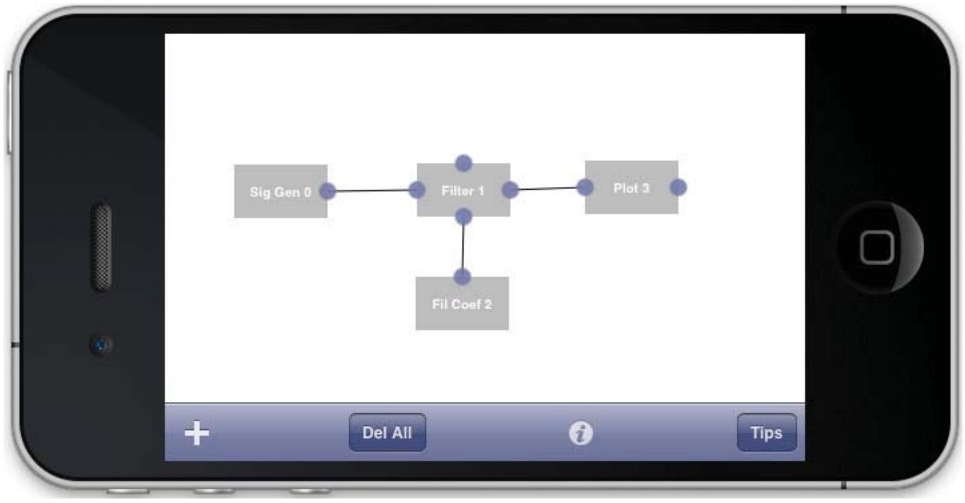


Figure 4.11: Block Diagram of the Single System.

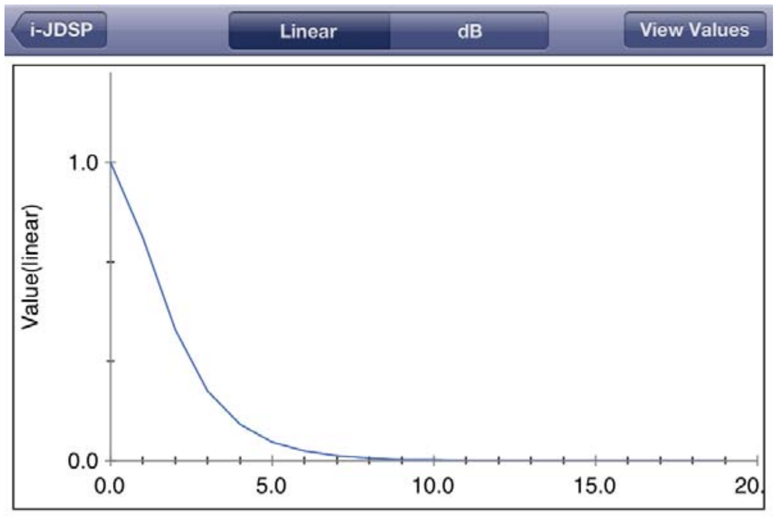


Figure 4.12: Impulse Response of the Single System.

in iJDSP.

The purpose of the first question is to verify the property that tapered windows have better behaved sidelobs and hence better behaved ripple effect relative to the rectangular window.

Students are asked to use a *Sig Gen* block, a *Window* block, and an *FFT* block to design a causal lowpass filter from the ideal impulse response $h_d(n) = 0.2sinc(\frac{\pi n}{5})$, i.e.,

$$h_{FIR}(n) = w(n)h_d(n - L/2), \quad 0 \le n \le L \tag{4.5}$$

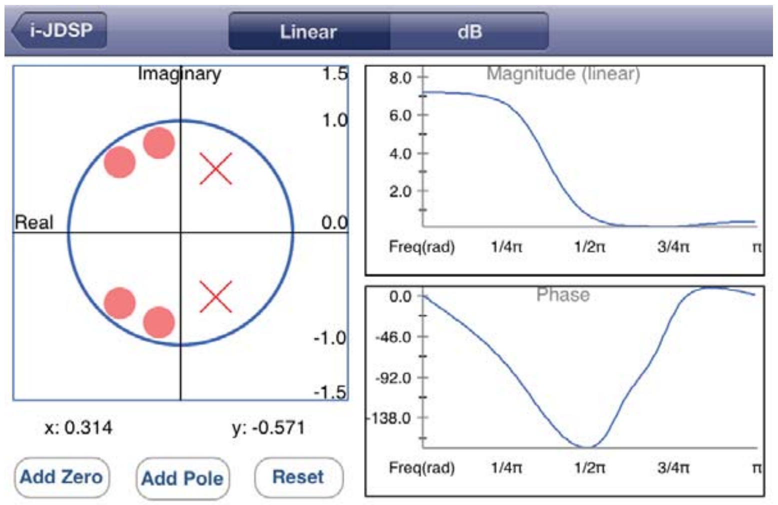


Figure 4.13: Low-pass Filter Design Using the PZ Placement Block.

Table 4.1: Settings in the Sig Gen block.

Signal Type	Amplitude	Pulsewidth	Periodic	Period	Time Shift	Freq.
Sinc	**0.2**	**120**	**NO**	**10**	**30**	**0.2**

The corresponding setting in the *Sig Gen* block is given in 4.1

The iJDSP block diagram should look like Figure 4.14. Take 256-point FFT of the impulse response. Set the size of window to be 60. Tapered windows have better behaved sidelobes and hence better behaved ripple effect relative to the rectangular window. Students can verify this property by checking FFT magnitude of Rectangular window and Bartlett (Triangular) window respectively. The simulation result is shown in Figure 4.15.

We also ask students to compare performance of FIR design by using windowing method and the Parks-McClellan algorithm. The *Parks-McClellan* block and the *Kaiser* block is used to design FIR filters with following specifications,

- Filter Type: low-pass filter
- Cut-off frequencies:

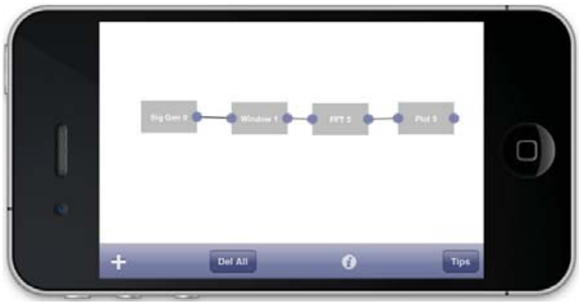
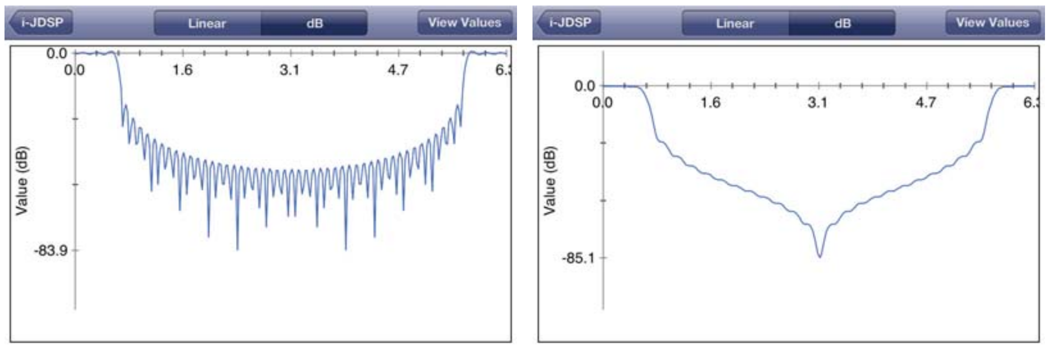


Figure 4.14: Block Diagram for iJDSP Prob2.1.



(a) Rectangular (b) Bartlett

Figure 4.15: FFT Magnitude of a FIR Filter Using Windowing Method.

Passband cutoff frequency $W_{p1} = 0.25\pi$

Stopband cutoff frequency $W_{s1} = 0.4\pi$

- Tolerances:

  $PB = 1dB$

  $SB = 40dB$

A sample of the block diagram is shown in Figure 4.16.

In order to compare those two methods, students are asked to check frequency response and order of each design. The comparison is given in Figure 4.17 and Figure 4.18. They also can check the filter's order in corresponding blocks, and find that Parks-McClellan algorithm has lower order to achieve the same specifications.

The last task in filter design is to design an IIR filter with the following specifications,

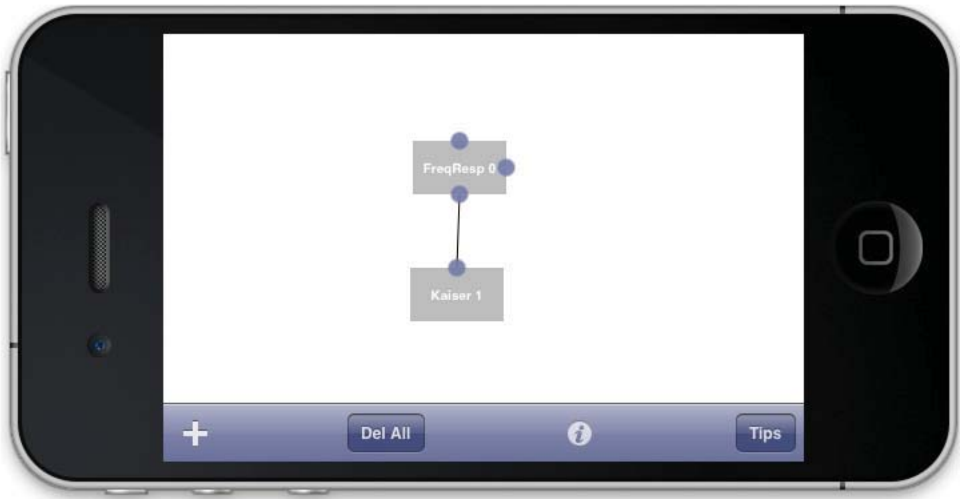
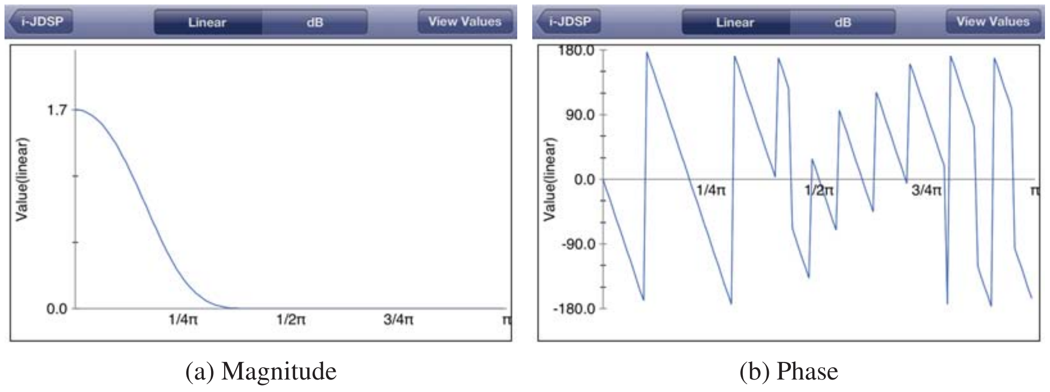


Figure 4.16: Block Diagram for iJDSP Prob2.2.



(a) Magnitude (b) Phase

Figure 4.17: Frequency Response of the Park-McClellan Filter.

- Filter Type: Low-pass

- Cut-off frequencies:

  Passband cutoff frequency $W_{p1} = 0.4\pi$

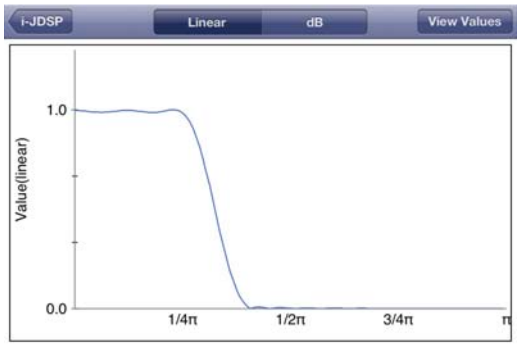
  Stopband cutoff frequency $W_{s1} = 0.6\pi$

- Tolerances:

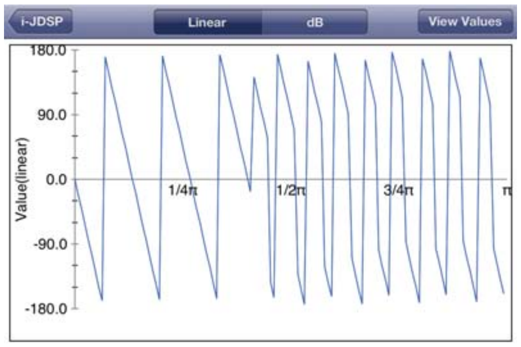
  $PB = 1dB$

  $SB = 45dB$

The design can be done using the *IIR Design* block in iJDSP. Connect the output



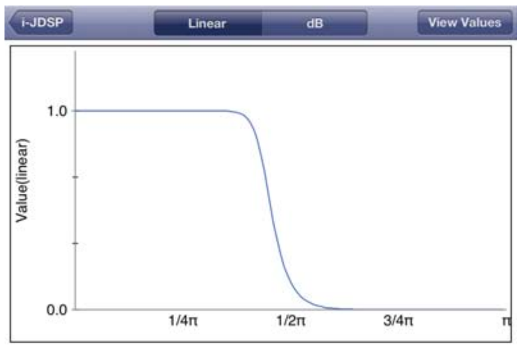
(a) Magnitude



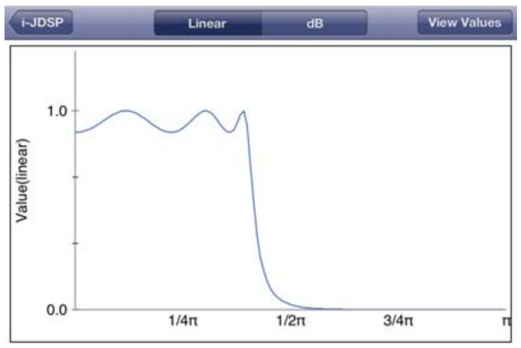
(b) Phase

Figure 4.18: Frequency Response of the Kaiser Filter.

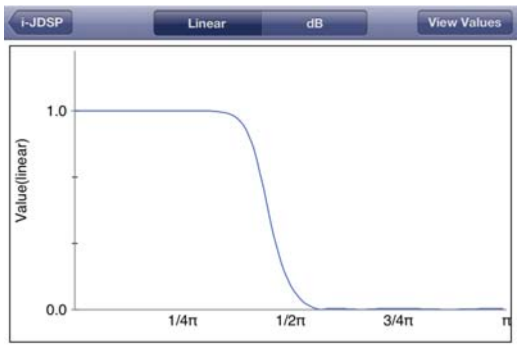
of the IIR block to the Freq Resp block to plot the frequency response. Compare the frequency response of the IIR filter using the four design techniques respectively, Butterworth, Chebyshev I, Chebyshev II, and Elliptic. The comparison is shown in Figure 4.19.



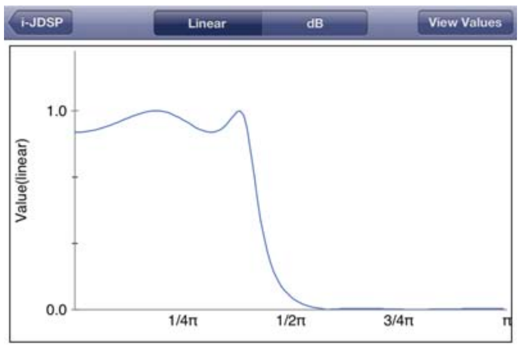
(a) Butterworth



(b) Chebyshev I



(c) Chebyshev II

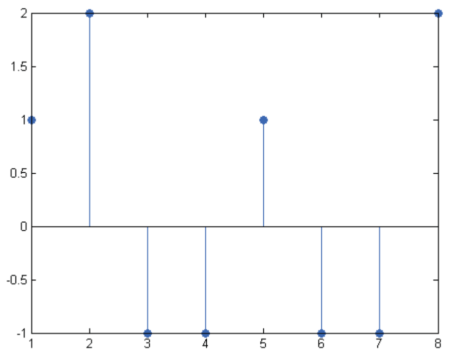


(d) Elliptic

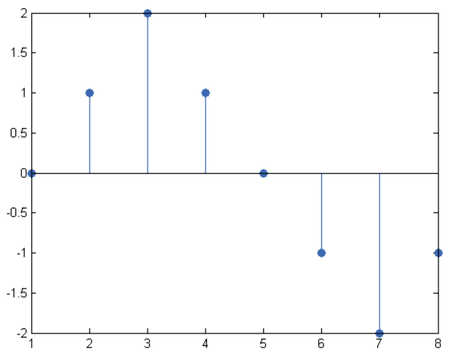
Figure 4.19: FFT Magnitudes of IIR filters.

4.4 FFT

In the laboratory of the fast Fourier transform (FFT), different fundamental issues of the FFT are examined. Problem 1 studies the symmetries in the time domain and frequency domain, and describe how these symmetries affect the FFT spectrum [38]. Students are asked to set up the two signals as shown in Figure 4.20 using the block diagram in Figure 4.21. $x_1[n]$ in Figure 4.20a is an even symmetric signal, and $x_2[n]$ in Figure 4.20b is an odd symmetric signal. They are used to examine the imaginary part and real part of the FFT spectrum.



(a) Signal 1



(b) Signal 2

Figure 4.20: The Signals Generated in the Sig Gen Block.

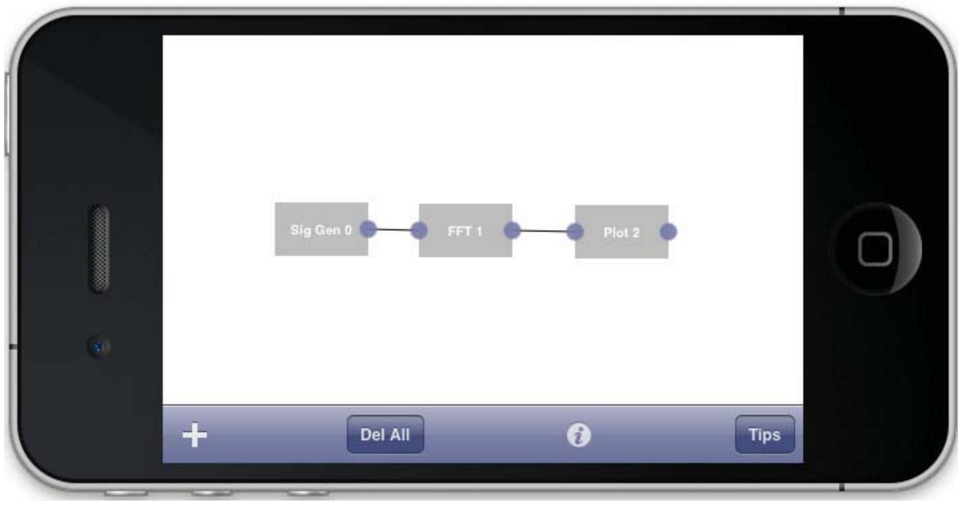
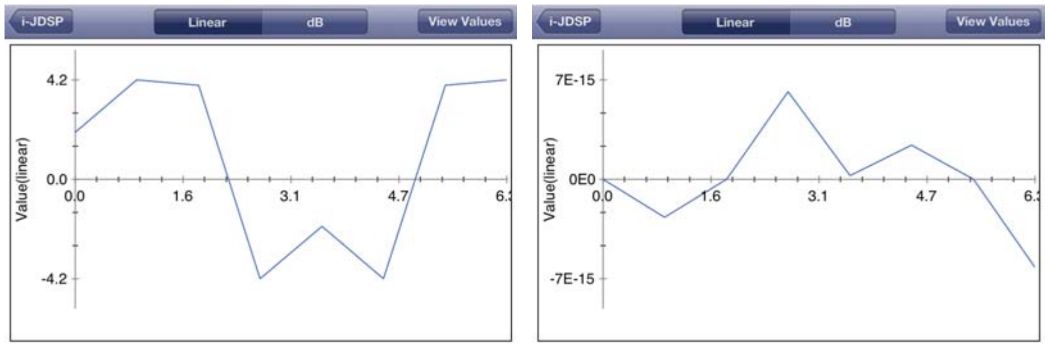
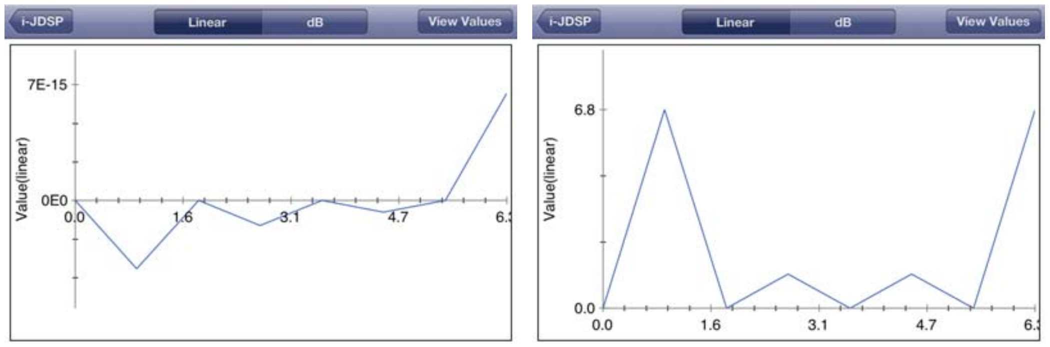


Figure 4.21: Block Diagram for iJDSP Prob3.1.



(a) Real part (b) Imaginary part

Figure 4.22: FFT Spectrum of $x_1(n)$.



(a) Real part (b) Imaginary part

Figure 4.23: FFT Spectrum of $x_2(n)$.

Problem 2 deals with the issue of resolution and spectral leakage of the FFT. To begin with students are asked to generate a signal $x[n] = 0.4sin(0.25\pi n) + sin(0.22\pi n)$ with the length of 128 samples using two *Sig Gen* blocks and one *Adder* block shwon in Figure 4.24.

Use a rectangular window that has length 128 samples to truncate the signal. Then take 128-point FFT after truncating the signal. In order to verify the property that the effects of loss of resolution and spectral leakage are controlled by the shape and length of the window, the students are asked to redo the problem by changing window's type and length respectivily. The FFT magnitude with 128-point rectangular window is shown in Figure 4.25. They need to change the length of the rectangular window to be 64 samples, and take 128-point FFT. The corresponding FFT magnitude is shown as Figure 4.26. Then students are asked to change the window type to Hanning, and set

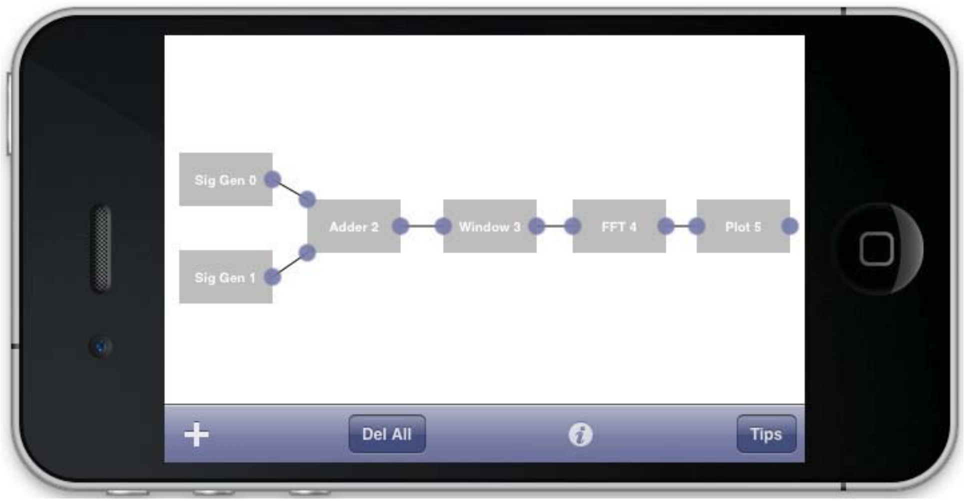


Figure 4.24: Block Diagram for iJDSP Prob3.2.

the window length to be 128. The FFT magnitude is shown in Figure 4.27. It is easy to find that the rectangular window can resolve the closed-spaced frequency components better than hanning window with the same length, and the length of window can affect FFT spectrum as well.

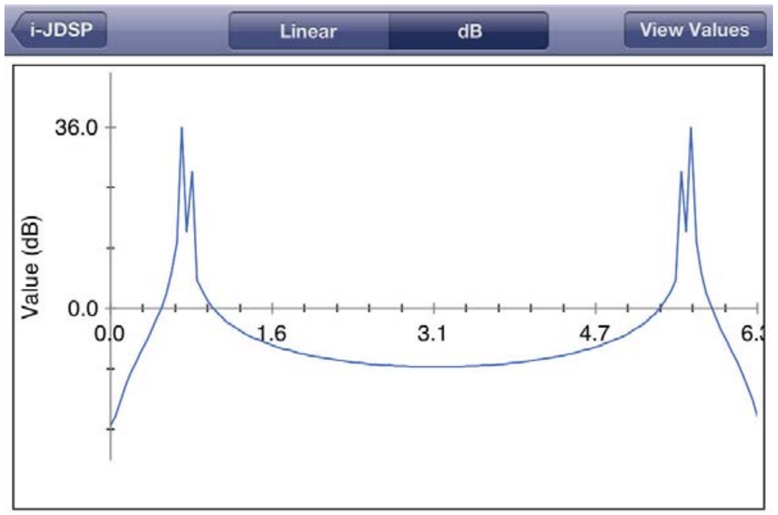


Figure 4.25: FFT Magnitude of 128-point Rectangular Window.

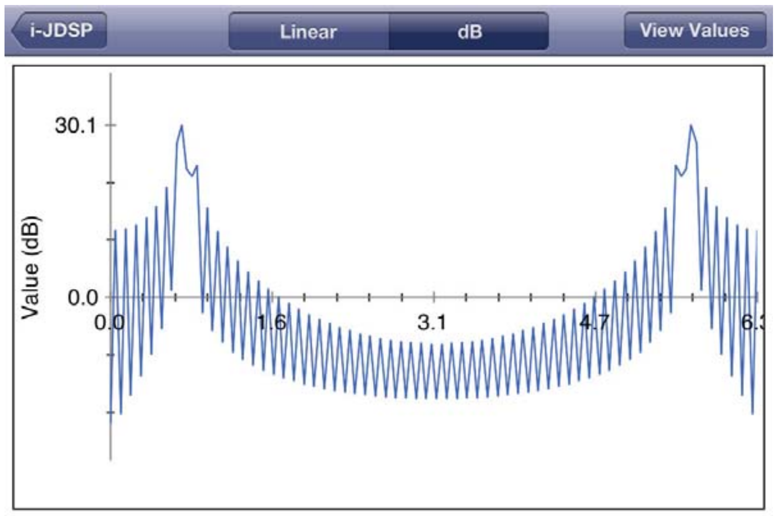


Figure 4.26: FFT Magnitude of 64-point Rectangular Window.

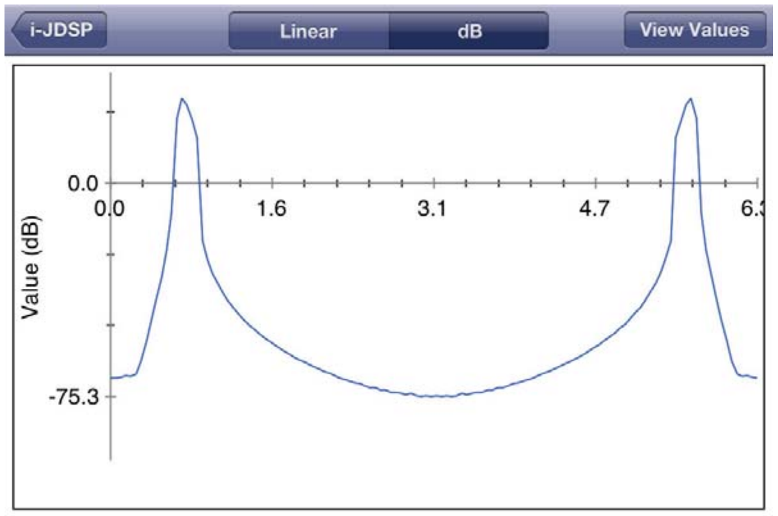


Figure 4.27: FFT Magnitude of 128-point Hanning Window.

Chapter 5

IJDSP HARDWARE INTERFACE

In this chapter, we will describe the development of a hardware interface that enables iJDSP to communicate with wireless sensors. With the use of this interface, we can perform DSP algorithms in iJDSP by receiving real-time acoustic signals such as sound and light intensity. Furthermore, some interesting applications are able to be implemented on iOS platforms.

5.1 Overview of Wireless Sensor Networks

Wireless sensor networks have been an active research field for several years. Due to their low power consumption and small size, WSNs are applied in diverse areas. For example, by measuring parameters such as: temperature, humidity and acceleration, WSNs can monitor volcano activities [41], manage smart building systems [42], or provide health care [43]. In addition, acoustic sensors can be applied in relative researches such as speech analysis, acoustic scene characterization, localization and target tracking [44–47].

The wireless sensor networks we used is based on non-cooperative topology. In general, there are two kinds of topology in terms of the wireless sensor networks, non-cooperative WSN and cooperative WSN. Non-cooperative WSN consists of several independent nodes and centralized base stations. The ability to process information is limited by the simple structures of independent nodes. If the central nodes fail, the entire network will collapse. Cooperative WSN is a distributed control system including self-organized nodes, which takes burden off the central node, and hence reduces bandwidth consumption. Compared to non-cooperative WSN, cooperative WSN improves the reliability of the entire network robustness but the expenditure is increased as well [48].

The hardware platform is provided by Crossbow Technology [49]. The central node is the MIB600 (Figure 5.3) programmable board, a base station which has an

Ethernet network interface and a embedded full TCP/IP protocol module. The independent node consists of a MICAz microprocessor board (Figure 5.1) and a MTS310 sensor board (Figure 5.2). The MICAz microprocessor board has a 2.4 GHz embedded microprocessor. The maximum supportable data rate is 250kps. There is a standard 51-pin expansion connector on MICAz which is used to connect to the MTS310 sensor board [50]. The MTS310 sensor board embeds six components including a microphone, a sounder (or buzzer), a photometer, a thermistor, a 2-axis accelerometer and a 2-axis magnetometer [51]. The applications for wireless sensor networks are developed using TinyOS, an open source embedded operating system developed for low power embedded system programming, and nesC, a dialect of the C language used on TinyOS [45].

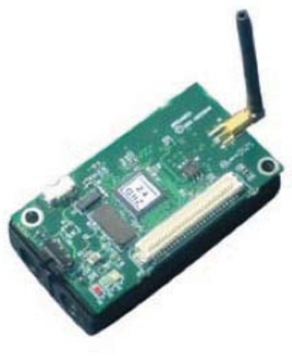


Figure 5.1: MICAz.

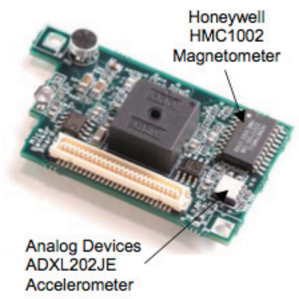


Figure 5.2: MTS310CA Sensor Board.

Figure 5.3: MIB600 Ethernet Interface Board.

5.2 Wireless Interfacing iJDSP with the Sensor Motes

The wireless interface in iJDSP is developed by using TCP socket in Server/Client model. Before iJDSP communicates with wireless sensor networks, iOS devices need to get access to the local area network (LAN) which is established by the router first. In Figure 5.4, MIB600 is assigned a static IP address 129.219.25.71 by PC-based software, DeviceInstaller of Lantronix. The routers IP is set to 129.219.25.72. They are ensured to work in the same network.

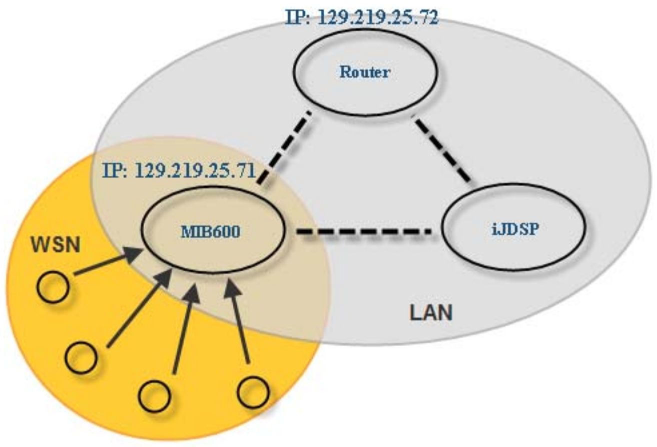


Figure 5.4: Local Area Network for iJDSP and the Sensors.

After setting the IP addresses, the TCP/IP connection needs to be established using socket programming. The communication protocol based on the Client/Server

TCP socket is illustrated in Figure 5.5. MIB600 waits for a connection request from iJDSP. Once the connection is established successfully, a confirmation message will be sent back to iJDSP by the server. Then iJDSP is able to write and read data seamlessly.

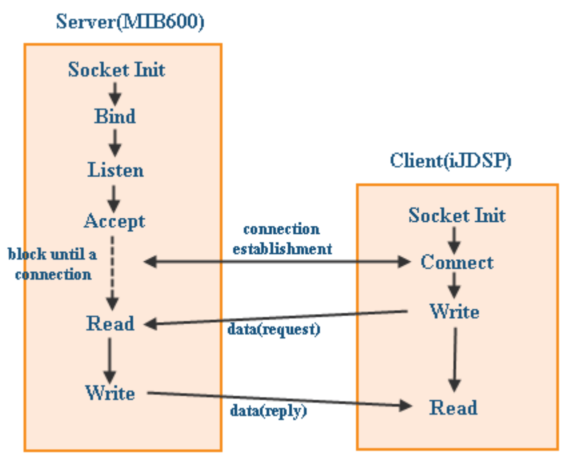


Figure 5.5: TCP Socket Protocols.

5.2.1 Interface Design

Figure 5.6 shows graphical interface design of the *Sensor* block. It consists of three panels: (a) Buffering panel; (b) Control panel; (c) Real-time plot panel. The buffering panel enables users to select buffer size from 32 to 256 samples. The buffer length determines the size of a frame processed with other DSP functions. Once tap the “Update” button, the corresponding buffer length is saved and passed to the next viewcontroller.

The control panel has five buttons enable connection/disconnection along with on/off commands to the LED, buzzer, photo sensor, and microphone. Once a button clicked, the corresponding command is send to the WSN. The commands are shown in Table 5.1.

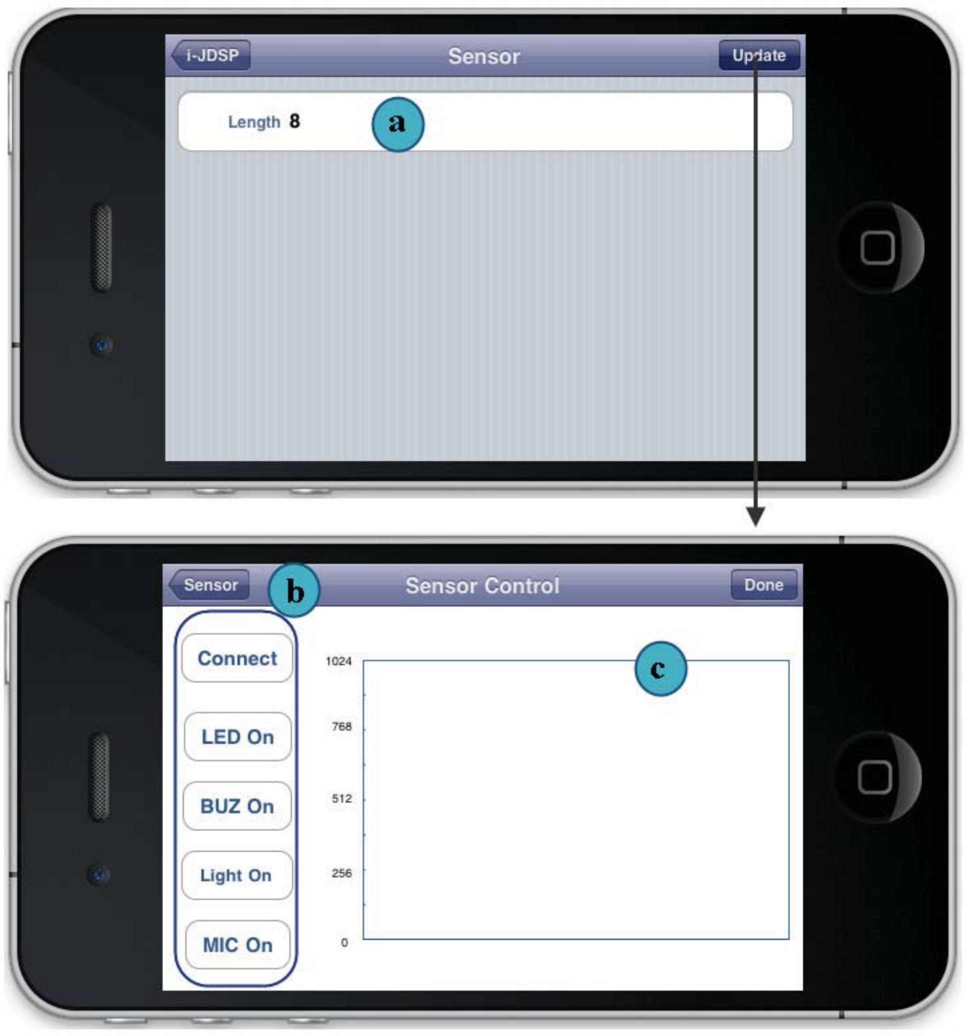


Figure 5.6: GUI Design in iJDSP to Interface WSN.

The workflow is illustrated in Figure 5.7. In iJDSP, we use AsyncSocket, a TCP/IP socket networking library to establish TCP connection [52]. The instance of Asyncsocket is globally accessible after initiating so that the connection keeps alive during the whole process. Once users click the “Connect” button, iJDSP will create a socket, and send a request, *connectToHost : onPort : error :*, to MIB600. After the socket is connected and ready for reading and writing, it will inform the client by calling the message *onSocket : didConnectToHost : port :*. The host address is the 129.Otherwise an error message will be returned to the delegate, and tell it to recon-

Table 5.1: Command Content.

Command Description	Value
LED ON	1
LED OFF	2
BUZZER ON	3
BUZZER OFF	4
LIGHT ON	5
LIGHT OFF	6
MIC ON	9
MIC OFF	10

nect. Once the connection successfully established, reading and writing operations are enabled between iJDSP and the MIB600 Ethernet gateway.

To disconnect this TCP connection, the message *EndComm* will be sent to terminate all writing and reading processes, and flush buffers on those motes. Then disconnect the socket by implementing *disconnectAfterWriting*. This message ensures that the socket will be disconnected after completing sending *EndComm* to motes.

5.2.2 Writing Operation

iJDSP sends commands to sensors by calling *SendPacketWithNumber : andID :*. The packet architecture is given in Table 5.2. *SendPacketWithNumber : andID :* attaches additional information to the raw data including two synchronization bytes which are used to note the start and end of a packet from the data stream, a type identifier, a CRC checking, destination address, message handler ID, Group ID and Node ID [45]. The packet will be sent to MIB600 by calling *writeData : withTimeout : tag :*. In the reading process, iJDSP keeps acquiring data from MIB600 by calling *onSocket : readDataToLength : withTimeout : tag :*.

5.2.3 Reading Operation

iJDSP keeps acquiring data from MIB600 by calling *onSocket : readDataToLength : withTimeout : tag :*. The parameter timeout is set to -1 in order to keep the reading oper-

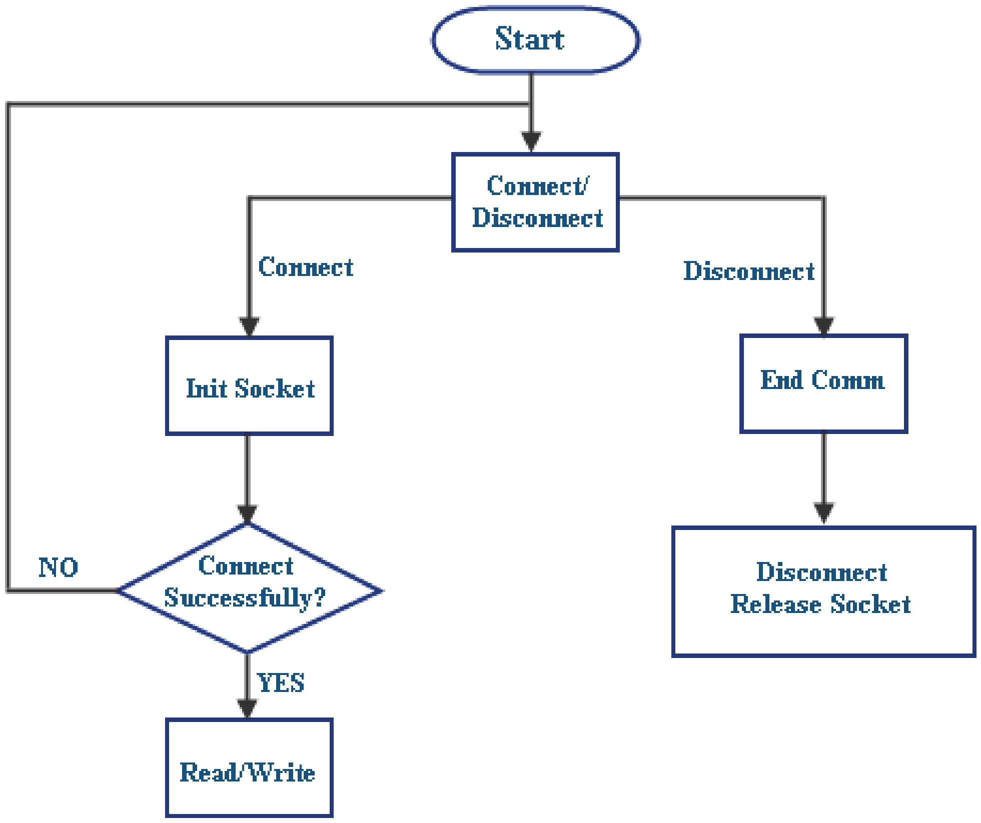


Figure 5.7: Flowchart of establishing a TCP connection in iJDSP.

ation alive during the whole process. Once a reading operation is completed within the allotted time, the socket will implement *onSocket : didReadData : withTag :* method to handle the received data. Figure 5.8 illustrates the flowchart of reading operation. Received data is saved in a buffer with the length determined by the buffering panel. Then iJDSP analyses the data packet and get ten samples of raw data. Those raw data is saved in pData and plotted.

5.3 iJDSP Real-time Exercises Using the Sensor Motes

iJDSP enables real-time signal processing experiments using wireless sensors. The frequencies of tones are measured by using FFT as shown in Figure 5.9. The signals in Figure 5.10 is a real-time recording and plotting of a persons voice in a room. The sound intensity is measured by a 10-bit analog-to-digital converter (ADC), hence the

Table 5.2: Packet Content.

Index	Variable Name	Description	Byte Value
0	SYNC_ BYTE	Identify the start of the packet	0x7E
1	P_PACKET_ACK	ACK required. Receiver will reply 0x40 in the prefix byte	65
2	seqNo	-	13
3 - 4	sData[0], sData[1]	Destination address	0xFF, 0xFF
5	sData[2]	Message ID	8
6	sData[3]	Group ID	125
7	-	-	93
8	sData[4]	Length	11
9	sData[5]	Command	1 byte
10	sData[6]	Node ID	1 byte
11-12	ttt, ttt1	CRC bytes	2 bytes
13	SYNC_BYTE	Identify the end of the packet	0x7E

sensor data ranges from 0 to 1024. Peaks shown in Figure 5.10 reflect the person's voice, and ripples show the environmental noise. After importing by the *Sensor* block, the sensor data is then frame-by-frame processed with the *FFT* block. The FFT magnitude of the incoming sensor data is shown in Figure 5.11. Both frame length and FFT size in this example are set to be 256. Other than frequency response, other problems such as spectral leakage, and advanced spectral analysis can be introduced to students by importing sensor data to other DSP functions in iJDSP.

5.4 Improvement

The hardware interface can be improved from the following aspects. Firstly, to expand the work area so that users are able to remotely control those motes and acquire sensor data from them, we can replace the wireless router by using a server. The active work area will not be constrained in a limited space (about 2 or 3 rooms that a LAN can reach). By establishing sockets in the server shown in Figure 5.12, we can upload and download sensor data from servers. Secondly, iJDSP can support multi-user collaboration. So far, the socket has been developed in a single thread model. Therefore

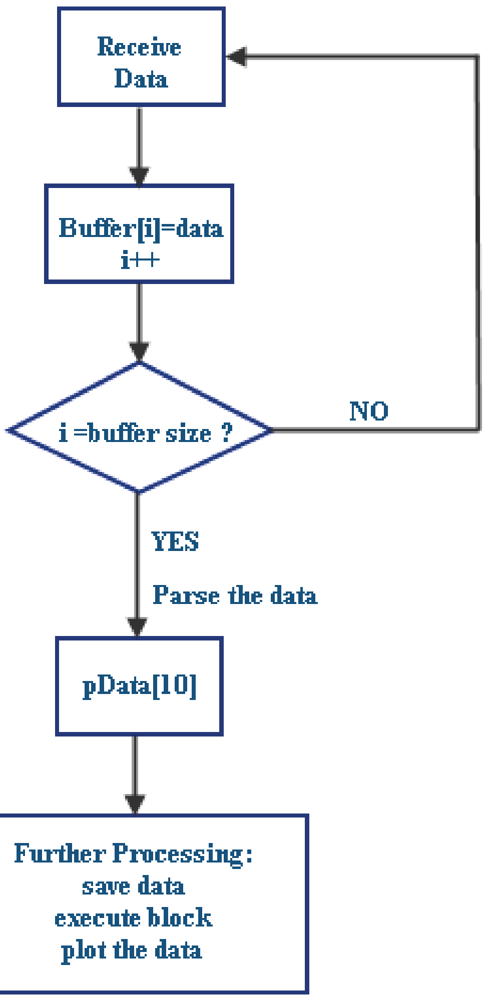


Figure 5.8: Flowchart of reading operation in iJDSP.

MIB600 can only process one connection per time. We can use multi-thread socket programming so that multiple connections can be processed simultaneously. In addition, an increase in the sampling rate can also be achieved. The maximum sampling rate of MICAz is 400Hz due to the limitation of Zigbee RF module in MICAz. However, the sampling rate of MICA2 can reach upto 4kHz, which is more suitable for acoustic research such as sound tracking [47].

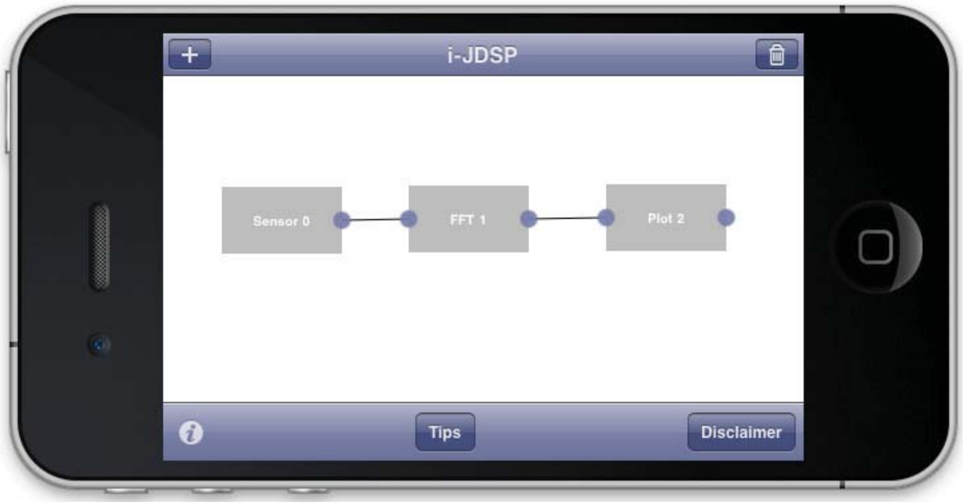


Figure 5.9: Block Diagram for Frequency Analysis of Sensor Data.

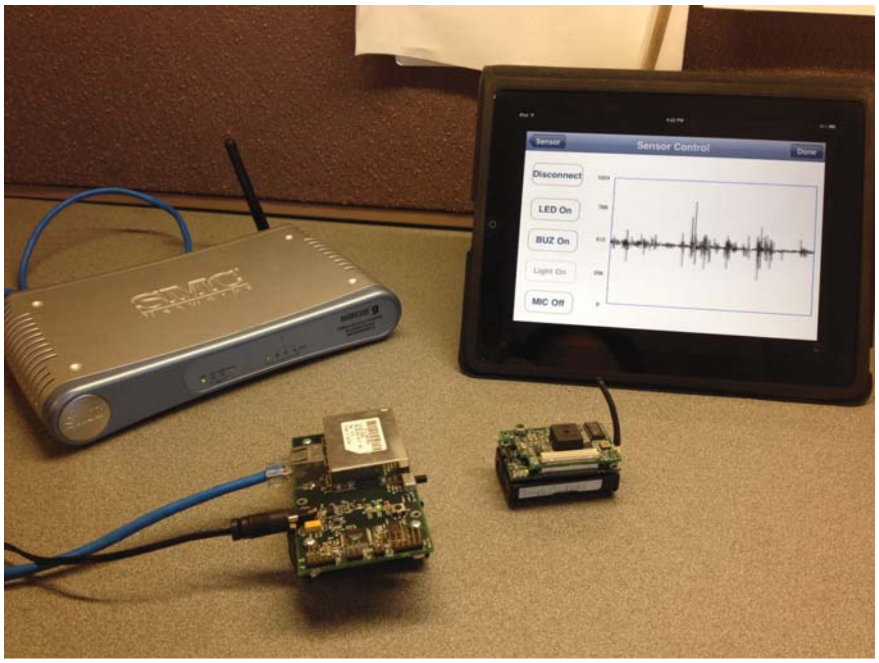


Figure 5.10: Real-time Plots of Incoming Sensor Data.

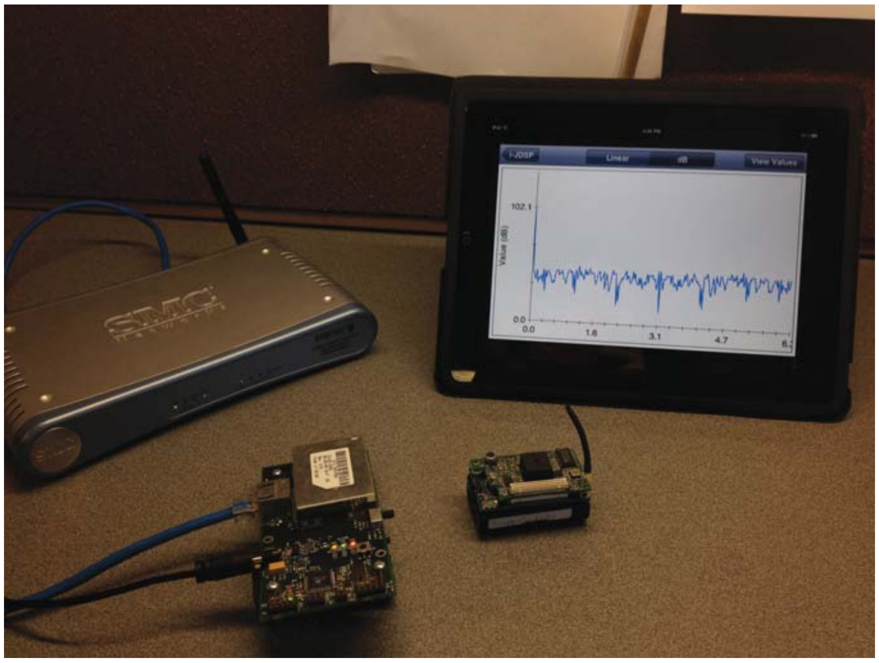


Figure 5.11: FFT Magnitude of a Microphone Sensor Signal.

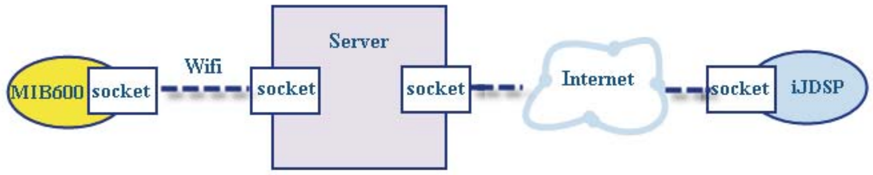


Figure 5.12: Use a Server to Access Sensor Data Remotely.

Chapter 6

ASSESSMENTS

The set of laboratory exercise described in Chapter 4 was evaluated by students at Arizona State University (ASU) during fall 2011 and spring 2012. In fall 2011, iJDSP was first introduced to EEE407 students. They evaluated the *Convolution Demo* block and completed associated exercises using iJDSP. In spring 2012, we organized a workshop involving senior level undergraduate students and graduate students to evaluate the entire software as well as exercises.

Through assessments, we would like to gather an overall subjective opinion about the application on aesthetics and usability. In addition, we could identify the impact of employing mobile devices to perform DSP simulations. The assessments could determine whether the iJDSP platform was interactive and interesting for students to learn DSP. The pedagogy adopted for use of iJDSP includes the following, (a) lecture on the pertinent signal processing concepts, (b) a pre-lab on the concepts involved in the laboratory exercise, (c) a simulation exercise using iJDSP, (d) post-lab to test student understanding of the concepts. After exercises, students will evaluate the exercises as well as the iJDSP application. The evaluation questionnaire includes general DSP concepts and concept-specific questions in terms of user experience.

6.1 Convolution Assessment

This assessment examined concepts of continuous/discrete-time convolution using the *Convolution Demo* block in iJDSP by 36 undergraduates from EEE407 of Fall 2011. Most of students liked the animation to convolve two signals, and felt it helpful in understanding the related concepts. Over 75 percent of students would recommend this application to their colleagues, and they also would like to see the software in interdisciplinary areas. These results are tabulated in Table 6.1 and Table 6.2.

Table 6.1: Statistics Based on the General Assessment. Total Number of Students = 36.

Evaluation Questions	Strongly Agree (%)	Agree (%)	Neutral (%)	Disagree (%)	Strongly Disagree (%)
The contents of this iJDSP exercise improved your understanding of the concepts of convolution.	27.8	58.3	13.9	0	0
Performing this iJDSP exercise improved your understanding of causality	13.9	63.9	16.6	2.8	2.8
Performing this exercise on the iPhone/iPad improved your understanding of discrete-time convolution	25	47.3	19.4	8.3	0
If a student colleague is having difficulty understanding convolution, would you recommend the iJDSP convolution demo?	(*Yes*) 69.5	(*No*) 8.3	(*Maybe*) 22.2	–	–
Is it more beneficial to use iJDSP to learn graphical convolution when compared to other computer-based tools, such as MATLAB or LabView?	(*Yes*) 69.5	(*No*) 19.4	(*Maybe*) 11.1	–	–

6.2 DSP Workshop Assessment

DSP workshop was organized at ASU on March 8th-9th, 2012. 19 undergraduate students from EEE407 class and 15 graduate students from School of Electrical, Computer and Energy Engineering (ECEE) in ASU attended this workshop. During the workshop, they were asked to complete three laboratories covering z-transform, filter design, FFT and sound related functions (MIDI, DTMF). The completed set of assessments is given in Appendix. Figure 6.1 shows students from the workshop doing exercises by using iJDSP on iPad and iPhone. Assessments results from undergraduate students were concluded in Table 6.3 and Table 6.4, and graduate students' assessments

Table 6.2: Statistics Based on the General Assessment. Total Number of Students = 36.

Evaluation Questions	Strongly Agree (%)	Agree (%)	Neutral (%)	Disagree (%)	Strongly Disagree (%)
What do you think about the number of steps taken to set up the continuous-time signal convolution simulation in iJDSP?	(*Many*) 5.6	(*Right*) 75	(*Few*) 19.4	–	–
What do you think about the number of steps taken to set up specific values for a user-defined signal convolution in iJDSP?	(*Many*) 5.6	(*Right*) 72.2	(*Few*) 22.2	–	–
Would you like to recommend iJDSP to your colleagues?	(*Yes*) 75	(*No*) 8.3	(*Maybe*) 16.7	–	–
It is more convenient to perform exercises and simulations on an iPhone/iPad than on a PC/Laptop?	27.8	25	27.8	19.4	0
We should extend this app to cover topics in other courses, for example, physics or mathematics.	33.3	41.7	16.7	5.6	2.7

results were tabulated in Table 6.5 and Table 6.6.

(a) Students from EEE407

(b) Students from SenSIP Center

Figure 6.1: The iJDSP Workshop in March 2012.

Evaluations show that the use of iJDSP appeals to students by intuitive and interactive user interface. More than 66.7 percent of users get used to the environment

Table 6.3: Statistics Based on the General Assessment from Undergraduates in EEE407. Total Number of Students = 19.

Evaluation Questions	Strongly Agree (%)	Agree (%)	Neutral (%)	Disagree (%)	Strongly Disagree (%)
Performing this exercise, you learned the concept of cascaded and parallel configuration of systems	21.1	31.6	31.6	15.7	0
Do you now understand more clearly the relationship of the frequency response with the poles and zeros?	89.5	10.5	0	0	0
The contents of this exercise helped you understand the concepts of FIR and IIR filter design.	36.9	52.6	10.5	0	0
After the lab, you know which of the IIR filters have ripple characteristic in both stopband and passband.	47.4	31.6	15.8	0	5.2

within 5 minutes. In addition, exercises on the touch screen are interactive and interesting. Students are engaged more in exercises and impressed by performing simulations using their fingers.

Compared to JDSP, iJDSP is more convenient to users. As a standalone mobile application, iJDSP takes less time to be loaded from operation system, and it is more convenient to manipulate diagrams instead of using mouse and keyboard. Furthermore, those hands-on devices enable users to use iJDSP anywhere and anytime.

iJDSP performance on iPhone and iPad are compared, and the size of screen is the most important factor affecting the user experiences. iPad has larger workspace to manipulate diagrams and more easier to enter parameters. For iPhone users, it is more convenient to carry the device. However, the current software is developed for iPhone screen, and iPad uses iJDSP in zooming mode. Therefore, those users suggested that we can develop an high-definition version on iPad exclusively.

Table 6.4: Statistics Based on the Concept-specific Assessment from Undergraduates in EEE407. Total Number of Students = 19.

Evaluation Questions	Strongly Agree (%)	Agree (%)	Neutral (%)	Disagree (%)	Strongly Disagree (%)
How long did it take to get used to the simulation environment on iJDSP?	t <5min 73.7	5min< t <10min 21.1	10min< t <20min 5.2	20min< t <30min 0	t >30min 0
Does the graphic user interface of iJDSP appeal to you?	26.4	63.2	5.2	5.2	0
It is easy to set up the lab simulations.	68.4	31.6	0	0	0
You feel comfortable performing simulations with the size of the screen.	31.6	36.8	10.5	21.1	0
Did you feel comfortable with the processing speed of the device for all the exercises?	73.7	26.3	0	0	0

Table 6.5: Statistics Based on the General Assessment from Graduate Students in School of ECEE at ASU. Total Number of Students = 15.

Evaluation Questions	Strongly Agree (%)	Agree (%)	Neutral (%)	Disagree (%)	Strongly Disagree (%)
Performing this exercise, you learned the concept of cascaded and parallel configuration of systems	53.4	33.3	13.3	0	0
Do you now understand more clearly the relationship of the frequency response with the poles and zeros?	100.0	0	0	0	0
The contents of this exercise helped you understand the concepts of FIR and IIR filter design.	40.0	46.7	13.3	0	0
After the lab, you know which of the IIR filters have ripple characteristic in both stopband and passband.	46.7	46.7	6.6	0	0
The contents of this exercise helped you understand the introductory spectral analysis concepts of the Fast Fourier Transform.	46.7	40.0	13.3	0	0

Table 6.6: Statistics Based on the Concept-specific Assessment from Graduate Students in School of ECEE at ASU. Total Number of Students = 15.

Evaluation Questions	Strongly Agree (%)	Agree (%)	Neutral (%)	Disagree (%)	Strongly Disagree (%)
How long did it take to get used to the simulation environment on iJDSP?	t <5min 60.0	5min< t <10min 20.0	10min< t <20min 6.7	20min< t <30min 6.7	t >30min 0
Does the graphic user interface of iJDSP appeal to you?	40.0	53.3	6.7	0	0
It is easy to set up the lab simulations.	53.3	46.7	0	0	0
You feel comfortable performing simulations with the size of the screen.	40.0	40.0	13.3	6.7	0
Did you feel comfortable with the processing speed of the device for all the exercises?	80.0	20.0	0	0	0

Chapter 7

CONCLUSIONS

This thesis describes the development of iJDSP, a highly interactive application that can be used to perform DSP simulations on mobile phones and tablets. The application has been developed as a native Cocoa Touch application and has a user-friendly programming environment. iJDSP has a rich suite of signal processing functions such as signal generators, animated convolution demo, digital filters, pole-zero and frequency response computations, FIR and IIR filter design algorithms, an FFT, and plot functions. The interface is highly intuitive and the block diagrams can be constructed using a simple drag-n-drop procedure.

iJDSP also has a hardware interface that works with sensor networks, a system that consists of several independent nodes and a centralized base station that can harvest environmental signals from distributed sensors. The wireless interface is developed based on a TCP socket in Server/Client model, and AsyncSocket, a TCP/IP socket networking library, is used to implement the socket. iJDSP can communicate with the sensor motes by initiating a TCP connection through a local area network established by a wireless router. iJDSP wireless sensor interface enables collaborative sensor signal processing. The proposed interface has the following features: (a) wireless connection between sensors and iPad, (b) GUI for the motes on iPad (c) control panel is used to control the individual sensor motes from the mobile devices, (d) capability to acquire inputs from multiple sensors: photometer, microphone, thermometer and accelerometer. (e) real-time plots of data acquired through the sensors. (f) sensor data can be processed frame-by-frame with DSP functions in iJDSP.

iJDSP was evaluated using assessments shown in Chapter 6. Most users found that the multitouch experience to be impressive and pointed out that the ease of using iJDSP is the most attractive feature for students. Students asserted that the intuitive and interactive environment of iJDSP helped them to learn DSP concepts visually and stimulated their learning interests. Besides, iJDSP is faster to load, and easier to manipulate diagrams compared to JDSP. Students indicated that they would recommend this application to their colleagues. In addition, they expect a high definition version of iJDSP exclusively developed for iPad platform.

7.1 Future Work

Enhancing iJDSP with more advanced functions related to image and audio processing can make the application comprehensive. Furthermore, iJDSP can be extended to interdisciplinary areas by developing versions of the application to perform simulations pertinent to concepts in biology, communications, control systems, and image processing.

In terms of wireless sensor networks, more applications can be developed by using the hardware interface we described in this thesis. Targeted applications include environmental monitoring, security, source localization, tracking and motion detection. The local network established by routers can be replaced by the Internet so that users will be able to access and process sensor data remotely. They also can share those data with others via the Internet. Other than WSNs, this wireless interface can be extended to other hardware platforms such as Wii motes, Xbox, and Kinect Controller to provide a better user experience. An interactive GUI could be provided to work with the LEGO Mindstorm and similar hardware systems.

REFERENCES

[1] R. Sims, “Interactivity: A forgotten art?” *Computers in Human Behavior*.

[2] S.Vosniadou, “How children learn,” 2001.

[3] C. Chou, “Interactivity and interactive functions in web-based learning systems: A technical framework for designers,” *British Journal of Educational Techonology*, vol. 34, pp. 265–279, 2003.

[4] D. Millard and G. Burnham, “Innovative interactive media for electrical engineering education,” in *IEEE FIE Conference*, vol. 3, 2001.

[5] “The infinity project: Engineering education for todays classroom,” http://www.infinity-project.org/infinity/.

[6] D.J.Brown, M.Covington, and M.L.Swafford, “Mallard: An educational tool for digital signal processing,” in *Asilomar conference on SSC*, no. 1, 1996, pp. 231–235.

[7] M.J.Jackson, D.I.Laurenson, and B.Mulgrew, “Developing and evaluating java-based educational tools,” in *IEEE International Symposium on Engineering Education: Innovations in Teaching, Learning and Assessment*, 2001, p. 26/1 26/6.

[8] A. Spanias and V. Atti, “Interactive online undergraduate laboratories using J-DSP,” in *IEEE Transactions on Education*, vol. 48, November 2005.

[9] “Will android and ios take over the pc market?” http://i-stuff.org/will-android-and-ios-take-over-the-pc-market/.

[10] “U.s. smartphone market: Whos the most wanted?” http://blog.nielsen.com/nielsenwire/?p=27418.

[11] “Apples app store downloads top 25 billion,” http://www.apple.com/pr/library/2012/03/05Apples-App-Store-Downloads-Top-25-Billion.html.

[12] A.Clausen, A.Spanias, A. Xavier, and M. Tampi, “A java signal analysis tool for signal processing experiments,” in *ICASSP*, vol. 3, May 1988, pp. 1849–1852.

[13] A. Spanias, N. Chakravarthy, Y. Song, and L. Iasemidis, “Teaching genomics and bioinformatics to undergraduates using J-DSP,” in *Proceedings of ASEE Annual Conference and Exposition*, June 2004.

[14] K. Ramamurthy, A. Spanias, L. Hinnov, and J. Ogg, “On the use of J-DSP in earth systems,” in *Proceedings of ASEE Annual Conference and Exposition*, Pittsburgh, PA, June 2008, p. 4 pages.

[15] R. Santucci, T. Gupta, M. Shah, and A. Spanias, “Advanced functions of Java-DSP for use in electrical and computer engineering courses,” in *Proceedings of ASEE Annual Conference and Exposition*, June 2010.

[16] H. Kwon, V. Berisha, V. Atti, and A. Spanias, “Experiments with sensor motes and Java-DSP,” in *IEEE Transactions on Education*, vol. 52, no. 2, May 2009, pp. 257–262.

[17] S. Henderson and J. Yeow, “ipad in education: A case study of ipad adoption and use in a primary school,” in *Hawaii International Conference on System Sciences*, 2012.

[18] G. Engel, “Using mobile technology to empower student learning,” in *27th Annual Conference on Distance Teaching and Learning*, 2011.

[19] N. Ostashewski and D. Reid, “ipod, iphone, and now ipad: The evolution of multimedia access in a mobile teaching context,” in *Proceedings of World Conference on Educational Multimedia, Hypermedia and Telecommunications*, 2010, pp. 2862–2864.

[20] N. Ostashewski, D. Reid, and M. Ostashewski, “Mobile teaching and learning technologies: Ukrainian dance instruction in canada,” in *IADIS Mobile Learning*, February 2009, pp. 2862–2864.

[21] “Star walk,” http://vitotechnology.com/star-walk.html.

[22] “hp12c,” http://www8.hp.com/us/en/products/smart-phones-handhelds-calculators/mobile-apps/app_details.html?app=tcm:245-799200&platform=tcm:245-799129.

[23] “Spectrogram,” http://spectrogramapp.com/.

[24] “Matlab mobile,” http://www.mathworks.com/mobile/.

[25] “The J-DSP Web Page,” http://jdsp.asu.edu.

[26] “iphone 4s,” http://en.wikipedia.org/wiki/IPhone_4S.

[27] “ipad2,” http://en.wikipedia.org/wiki/IPad_2.

[28] “Berkely tinyos porject,” http://webs.cs.berkeley.edu/tos.

[29] “Zigbee alliance,” http://zigbee.org.

[30] Apple Inc., “Cocoa frameworks,” http://developer.apple.com/technologies/mac/cocoa.html.

[31] Akten. M, “Nsarray vs. c array performance,” http://memo.tv/nsarray_vs_c_array_performance_comparison.

[32] Mark, Dalrymplt, and K. Scott, *Learn Objective-C on the Mac*. Apress, 2011.

[33] Apple Inc., “Quartz 2d programming guide,” https://developer.apple.com/library/mac/#documentation/graphicsimaging/conceptual/drawingwithquartz2d/Introduction/Introduction.html.

[34] "Cocoa plotting framework for mac os x and ios," http://code.google.com/p/core-plot/.

[35] Dave Mark and Jeff LaMarche, *Beginning iPhone Development: Exploring the iPhone SDK.* Apress, 2008.

[36] Apple Inc., "The objective-c programming language: Objects, classes, and messaging," http://developer.apple.com/library/ios/#documentation/cocoa/conceptual/objectivec/Chapters/ocObjectsClasses.html, Oct 2011.

[37] ——, "Cocoa fundamentals guide: Communicating with objects," http://developer.apple.com/library/mac/#documentation/Cocoa/Conceptual/CocoaFundamentals/CommunicatingWithObjects/CommunicateWithObjects.html, Dec 2010.

[38] A. Spanias, *Digital Signal Processing: An Interactive Approach.* Lulu, 2007.

[39] A. V. Oppenheim, R. W. Schafer, and J. R. Buck, *Discrete-time Signal Processing, second edition.* Prentice Hall, 1999.

[40] A. D. Pourlarikas, *The Handbook of Formulas and Tables for Signal Processing*. CRC Handbook in Corporation with IEEE Press, 1998.

[41] G. W. Allen, K. Lorincz, and M. Welsh, "Deploying a wireless sensor network on an active volcano." IEEE Internet Computing, March/April 2006.

[42] I. Ituen and G. Sohn, "The environment application of wireless sensor networks," *International Journal of Contents*, no. 4, 2007.

[43] J. Chen, K. Kwong, D. Change, J. Luk, and R. Bajcsy, "Wearable sensors for reliable fall detection." IEEE-EMBS'05, 2005, pp. 3551–3554.

[44] H. Kwon, V. Berisha, and A. Spanias, "Real-time sensing and acoustic scene characterization for security application." IEEE, 2008.

[45] H. M. Kwon, "Acoustic characterization in wireless sensor networks," Ph.D. dissertation, Arizona State University, December 2009.

[46] T. Ajdler, I. Kozintsev, R. Lienhart, and M. Vetterli, "Acoustic source localization in distributed sensor networks," in *Conference Record of the Thirty-Eighth Asilomar Conference on Signals, Systems and Computers*, vol. 2, November 2004, pp. 1328–32.

[47] A. Swain, "Characterization of acoustic sensor motes for target tracking in wireless sensor networks," Master's thesis, Arizona State University, December 2006.

[48] H. Qi, S. Iyengar, and K. Chakrabarty, "Distributed sensor networks - a review of recent research," *Journal of The Franklin Institute*, pp. 655–668, 2001.

[49] "Crossbow techonology inc." http://www.xbow.com.

[50] *MICAz datasheet*, Crossbow Technology Inc.

[51] C. T. Inc., *MTS/MDA Sensor Board Users Manual*, June 2007.

[52] “Cocoa asyncsocket,” http://github.com/robbiehanson/CocoaAsyncSocket.

APPENDIX A

ESTABLISH A TCP CONNECTION IN IJDSP

This chapter gives a template of the client program to establish a TCP connection between iJDSP and the sensor motes. We use AsyncSocket, a TCP/IP socket networking library to efficiently handle data in the communication process including connecting/disconnecting and reading/writing.

A.1 Create an instance of AsyncSocket

After users tap on the “Connect” button, an instance of AsyncSocket is created. At the same time, we set its delegate to be the viewController. The delegate will respond to certain operations or errors sent by the socket. If the connection is established successfully (this decision is made in the delegate method *(void)onSocket:(AsyncSocket *)sock didConnectToHost:(NSString *)host port:(UInt16)port*), the text shown on the “Connect” button will be replaced by “Disconnect”. If users want to disconnect, tap on the ”Disconnect” button. A message *EndComm* will be sent to the server, and the socket *Client* is disconnected after completing writing this message.

```
//establish a TCP/IP connection
- (IBAction)ConnectBtnClicked:(id)sender
{
        if ([Connect.currentTitle isEqualToString:@"Connect"]) {
                Connect.enabled = NO;
                Connect.alpha = 0.5;
                D_cn = 0;
                if (client == nil) {
                        client = [[AsyncSocket alloc] initWithDelegate
                            :self];
                        [client setConnectTimeout:2]; //set timeout to
                            be 2 sec for connection
                }


        }
        else {
                [Connect setTitle:@"Connect" forState:
                    UIControlStateNormal];
                // complete all write operations before disconnecting
                [self EndComm];
                [client disconnectAfterWriting];
                client = nil;
        }
}
```

A.2 Implement AsyncSocket delegate methods

The delegate will complete the following methods:

- connect:

  (void)onSocket: didConnectToHost: port:

- disconnect when errors occur:

  (void)onSocket: willDisconnectWithError:

- disconnect:

  (void)onSocketDidDisconnect:

- read:

  (void)onSocket: didReadData: withTag:

```
- (void)onSocket:(AsyncSocket *)sock didConnectToHost:(NSString *)
    host port:(UInt16)port{
        [client readDataToLength:sizeof(UInt8) withTimeout:-1 tag:0];
        Connect.enabled = YES;
        Connect.alpha = 1;
        [Connect setTitle:@"Disconnect" forState:UIControlStateNormal
            ];
}

//disconnect TCP/IP connection when errors occur
```

```
- (void)onSocket:(AsyncSocket *)sock willDisconnectWithError:(NSError
    *)err
{
        NSString *msg = @"Connection request times out, please retry";
        [self showMessage:msg];
        [msg release];
}

//disconnect TCP/IP connection
- (void)onSocketDidDisconnect:(AsyncSocket *)sock
{
        //Disconnect socket
        [client setDelegate:nil];
        [client release];
        client = nil;
}

//read data from server

- (void)onSocket:(AsyncSocket *)sock didReadData:(NSData *)data
    withTag:(long)tag
{
        //convert bytes to int
        UInt8 tmp1 = * (UInt8 *)([data bytes]);
        D_cn = D_cn + 1;
        [self DisplayPacketWithData:tmp1];
        [client readDataToLength:sizeof(UInt8) withTimeout:-1 tag:0];
}
```

A.3 Send packets to server

```
- (void)SendPacketWithNumber:(int)c andID:(int)i
{
        int crc;
        int escapePtr;
        seqNo = seqNo + 1;
        int ttt, ttt1;
        int tmp;

        if (seqNo > 256) {
                seqNo = 14;
        }

        sData[0] = (Byte)255; //dest.
        sData[1] = (Byte)255; //dest.
        sData[2] = (Byte)8;   //message id
        sData[3] = (Byte)125; //group id
        sData[4] = (Byte)11;
        sData[5] = (Byte)c;
        sData[6] = (Byte)i;

        escapePtr = 0;
        crc = 0;
        ttt = 0;
        ttt1 = 0;

            //add CRC checking
        escaped[escapePtr++] = (Byte)SYNC_BYTE;
        crc = calcByte(crc, P_PACKET_ACK);

        escaped[escapePtr++] = (Byte)P_PACKET_ACK;
        crc = calcByte(crc, seqNo);
        escaped[escapePtr++] = (Byte)seqNo;

        for (int i = 0; i < 7; i++) {//7 is the length of sData[],
            calculate CRC for each byte in sData
                crc = calcByte(crc, sData[i]);
            escaped[escapePtr++] = sData[i];
        }

        ttt = (crc & 0xff);
        escaped[escapePtr++] = (Byte)ttt;

        ttt1 = (crc >> 8);
        escaped[escapePtr++] = (Byte)ttt1;

        escaped[escapePtr++] = (Byte)SYNC_BYTE;

        for (int i = 13; i > 7; i--) {//escaped[14]
            escaped[i] = escaped[i-1];
        }

        escaped[7] = 93;

        @try {
            for (int i = 0; i < 14; i++) {
                tmp = escaped[i];
            }

            //Send to host, write data in bytes
            NSData *data = [NSData dataWithBytes:escaped length:
                sizeof(escaped)];
            [client writeData:data withTimeout:-1 tag:1];
        }

        @catch (NSException * e) {

        }
        @finally {

        }

}
```

A.4 Display packets

We need to get the raw data from the received packet first. Then those data will be sent to the graph to be plotted in real time.

```
- (void)DisplayPacketWithData:(int)p
{
        int N_id = 0;
        int N_samples = 0;
        buffer[D_cn - 1] = p;

        if (p == 126) {
                if (Pstart)
                        Pstart = FALSE;
                else
                        Pstart = TRUE;
        }

        if (Pstart) {
                count = count + 1;
        }

        else if (!Pstart)
        {
                if (count == PacketSize - 1) {
                        for (int i = 0; i < PacketSize; i++) {
                                packet[i] = buffer[i];
                        }

                        N_id = packet[7] - packet[8] * 256;
```

```
                    N_samples = (packet[9] + packet[10] * 256)/2;
                    for (int i = 0; i < DataSize; i++) {
                            pDataa[i] = packet[13 + i * 2] + packet
                                [14 + i * 2] * 256;
                    }

                    // N_id refers to the id of nodes, here is
                        NODE1

                    [self UpdateBufferWithData:pData];

                    //send data to graph, and update data to plot
                    if (start) {
                            for (int i = 0; i < DataSize; i++) {
                                    [self setData:pData[i] toGraph
                                        :graph];
                            }
                    }

                    D_cn = 0;
                    count = 0;
            }
            else if ((buffer[1] == 66) & (count > 35))
            {
                    D_cn = 0;
                    count = 0;
            }
            else if (buffer[1] == 64)
            {
                    D_cn = 0;
                    count = 0;
            }
            else {
                    D_cn = 0;
                    count = 0;
                }

        }

}

- (void)setData:(int)plotData toGraph:(PlotGraph*)plotgraph
{
        // set the incoming data to the end of the array
        // update the array
        for (int i = 0; i < 999; i++) {
                currentArray[i] = currentArray[i + 1];
        }
        currentArray[999] = plotData;

            // update graph
        [graph setDataToPlot:currentArray];
        [graph setNeedsDisplay];


}
```

APPENDIX B

ASSESSMENTS

B.1 Convolution Exercise

B.1.1 Objectives

The first lab will cover both the continuous-time convolution and the discrete-time convolution concept. After performing this lab, you should be able to analytically calculate the impulse response and output of a system given an input signal. You should also be able to visualize the process of discrete and continuous time convolution.

B.1.2 Introduction

To help you visualize these concepts, we have provided the convolution demo block in iJDSP that will perform the animated convolution of two signals of your choice. To establish the convolution demo block, tap the "+" button on the left corner of the navigation bar, and select the Conv Demo. By double tapping the Conv Demo box, you can choose to operate either continuous-time convolution or discrete-time convolution. Once this is selected, you can define the signals to be convolved.

B.1.3 Part 1: Continuous Time Convolution

Problem 1.1.

a. Given a circuit shown in Figure B.1. Find analytically the impulse response $h(t)$ of the system where $R = 1000\Omega$ and $C = 1000\mu F$.

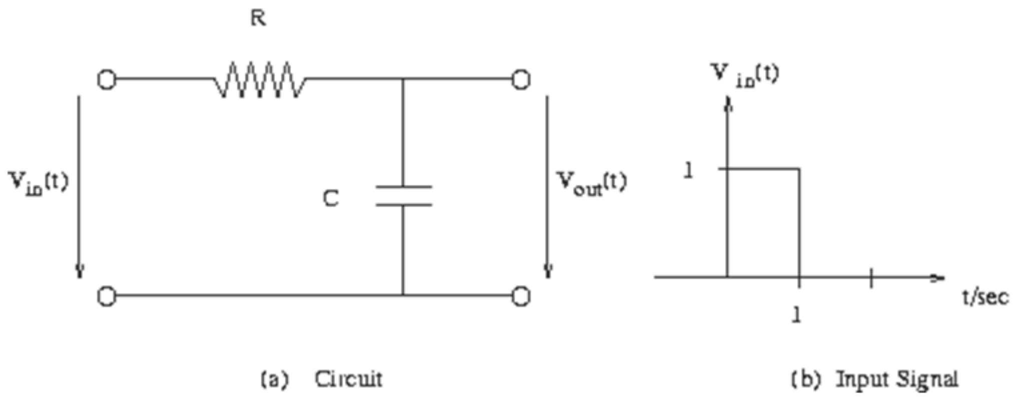


Figure B.1: Circuit Given for Prob 1.1.

b. We obtain $V_{out}(t)$ by convolving the impulse response and the given input signal. Find the analytical expression for $V_{out}(t)$.

Problem 1.2. In this part, we want to investigate how convolution affects causality. A signal will be considered causal if it is zero for $t < 0$ and non-causal otherwise.

Be sure to take screenshots of your results so you can submit them later. Double tap the Conv Demo block in i-JDSP, and choose Continuous Conv to perform the convolution of each of the following pairs of signals and then determine whether the resulting signal is causal or non-causal. Again be sure to save your screenshots.

Perform convolutions of the following 3 pairs of signals using the Continuous Conv function of Conv Demo block in i-JDSP.

a. causal rectangular and non causal rectangular

b. causal rectangular and causal rectangular

c. causal rectangular and non causal sinc

B.1.4 Part 2: Discrete Time Convolution

The Conv Demo block in i-JDSP can also perform discrete time convolution. Double tap the Conv Demo box, choose Discrete Conv. To create your own signal, you can choose type for each signal first. Tap "Update" on the right corner of the navigation bar and change the value of each sample by holding and dragging it up or down with your finger. The value of y-axis for each sample is shown on the top-left. The values change in steps of 0.033 units.

Problem 2.1. Perform convolutions of the following 3 pairs of signals using the Dicrete Conv function of Conv Demo block in i-JDSP.

a. Convolve a non causal rectangular and non causal sinc.

b. Convolve the following two signals shown in Figure B.2.

c. Convolve the following two signals shown in Figure B.3.

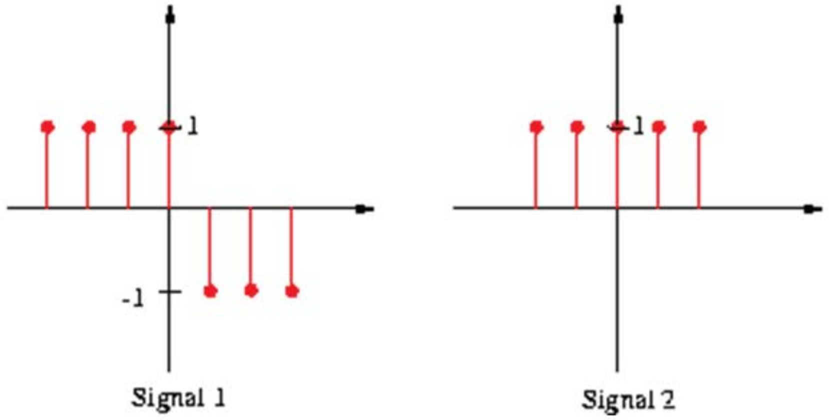


Figure B.2: Discrete Signals to be Convolved.

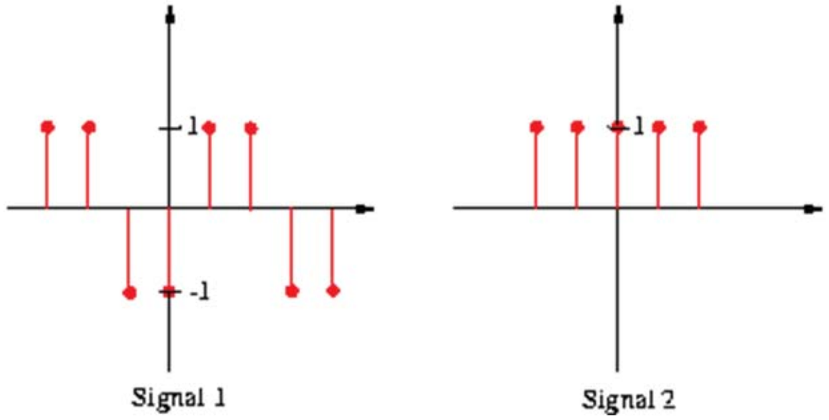


Figure B.3: Discrete Signals to be Convolved.

Problem 2.2. In part 1, problem 1.1 of this lab you found the impulse response of a system. Now, we want you to sample that impulse response and the given input signal. Use these sampled signals to perform a discrete time convolution. You can use the Conv Demo block in i-JDSP and set samples which are smaller than 0.1 to zero.

Is the waveform which results from the discrete-time convolution the same as (or similar to) what would be obtained by sampling the result of the continuous-time convolution of the analog signals? Does the sampling rate make a difference?

For sampling rate, use:

a. 1 s

b. 250 ms

B.2 Workshop Exercise

B.2.1 Objectives

The objective of this exercise is to provide hands-on experiences on iJDSP. It consists of three parts covering frequency response of LTI systems, pole/zero locations with the frequency response, filter designs, and Fast Fourier Transform.

B.2.2 Introduction

All the DSP functions are represented in iJDSP as graphical blocks that are also capable of handling user gestures.

- Double tap: Open a block to see its property dialog.
- Long hold: Delete a specific block.
- Single tap on a pin: Make a connection between blocks.
- Hold and drag on a block: move blocks with your fingers.
- Swipe down/up: hide/show the bottom bar.
- Tap "+" button: Add blocks.
- Tap trash button: Delete all the blocks at one time.

The z-transform of the impulse response of a LTI system can be written in the following form:

$$H(z) = \frac{\sum\limits_{l=0}^{10} b_l z^{-l}}{1 + \sum\limits_{m=1}^{10} a_m z^{-m}} \tag{B.1}$$

This is also known as the transfer function of the system. The a_m's and the b_l's are the filter coefficients of the system with a_0 always being equal to one.

This part will explain how iJDSP works through a simple example. Open the iJDSP application on your device. When the welcome page appears press “Start”. Press the “+” button on the left corner of the navigation bar. Then select Signal Generator, and a list of parameters about the signal will appear. Tap the “Add” button on the right corner of the navigation bar to use the default settings for the block. Similarly, add a Filter block, a Filter Coeff block and a Plot block on your main canvas. You can then move the established block by placing your finger over it and dragging it to a new location. To delete a block, place your finger over the block and hold for 3 seconds. When the pop-up menu appears, select “Delete”. To link blocks, tap once inside the blue dot on the right side of the signal generator box, then tap once the blue dot on the left side of the filter box. A line will be drawn. Please make the connections in the direction of the signal flow, otherwise you will get an alert or fail on making connections. Now, connect the Filter Coeff block and the Plot block to the Filter block so that your screen looks like Figure B.4.

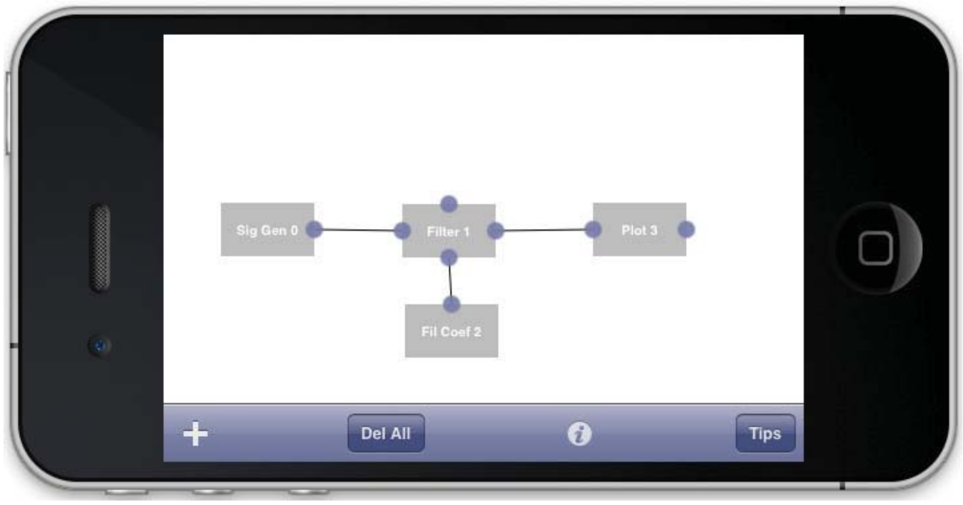


Figure B.4: iJDSP Example of a Filter Simulation.

Double tap on the Filter Coeff block, and type in the following filter coefficients: $b_0 = 1$, $a_0 = 1$, $a_1 = -0.8$. Note that to save your current settings, you need to tap the “Update” button on the upper right corner. Similarly, double tap on the Sig Gen block.

Change the signal type to "Delta", and then "Update" the block. Double tap on the Plot block, and choose "Amplitude". You will see the impulse response of a first-order causal IIR filter defined by the difference equation $y(n) = x(n) + 0.8y(n-1)$. The closed-form expression for the impulse response is $h(n) = 0.8^n u(n)$. In the Plot block, a list of specific values is shown by tapping "View Values" button.

B.2.3 Part 1: Frequency Response of LTI System

Problem 1.1. Given the following transfer functions,

- $$H_1(z) = \frac{1 - 0.45z^{-1} + 0.55z^{-2}}{1 - 1.7z^{-1} + 0.6z^{-2}} \tag{B.2}$$

- $$H_2(z) = 0.06 + 0.0876z^{-1} - 0.3z^{-2} + 0.375z^{-3} - 0.3z^{-4} + 0.0876z^{-5} + 0.06z^{-6} \tag{B.3}$$

- $$H_3(z) = \frac{-0.8 + 1.62z^{-1} - 1.8z^{-2} + z^{-3}}{1 - 1.8z^{-1} + 1.62z^{-2} - 0.8z^{-3}} \tag{B.4}$$

Establish the block diagram shown in Figure B.5: Type the corresponding filter coefficients in the Filter Coeff block. To see the poles and zeros in the z-plane, double-tap the PZ Plot block. To see the impulse response of the system, set "Delta" signal in the Sig Gen block, and double tap on the Plot block to see the simulation result.

- Specify the values of the poles and zeros of $H_1(z)$.
- Sketch the impulse response $h_1(n)$ corresponding to $H_1(z)$.
- Place the poles and zeros in the z-plane for $H_2(z)$ and $H_3(z)$.
- Sketch the frequency response magnitude (linear scale) of $H_3(z)$. What is the type of this filter (High-pass, Low-pass or All-pass filter)?

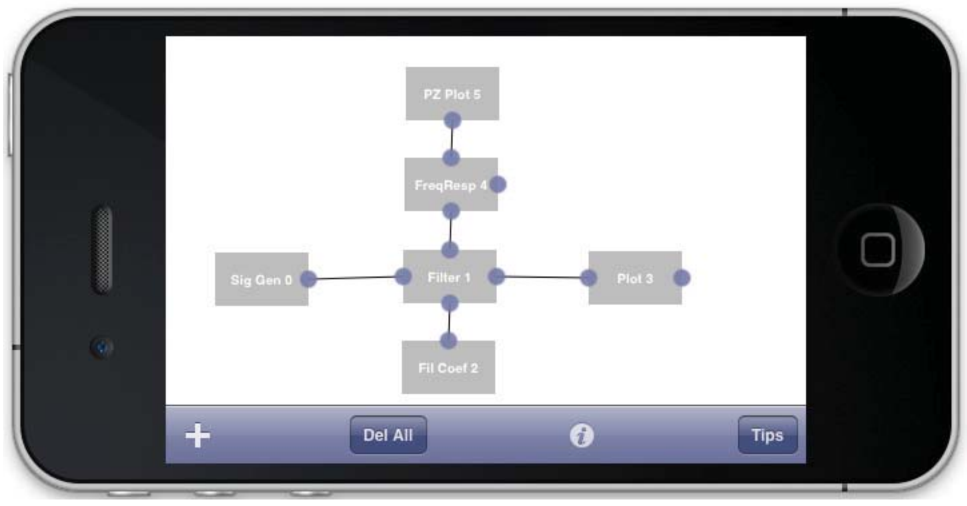


Figure B.5: Block Diagram for iJDSP Prob. 1.1.

Did any of the simulations have errors or the program fail to execute (crash)? Please indicate in details so that we can reproduce the error.

Problem 1.2. Consider the following sequence, which is the convolution of two causal sequences.

$$y(n) = \sum_{k=0}^{n} a^k b^{n-k} \tag{B.5}$$

(a) Implement the cascaded system using one Sig Gen block, two Filter blocks, two Filter Coeff blocks, and one Plot block for $a = 0.5$ and $b = 0.25$.

- Save the screenshot of the impulse response of the system.
- Save the screenshot of the block diagram of the system.

(b) Now create a system using only one filter block such that the impulse response is also . Again use $a = 0.5$ and $b = 0.25$. Verify that the impulse response of this system is the same as the impulse response of the system in the previous part.

Did any of the simulations have errors or the program fail to execute (crash)? Please indicate in details so that we can reproduce the error.

Problem 1.3. For this problem, you will design a filter by placing poles and zeros in the z-plane. The PZ Placement block enables the user to place poles and zeros

Table B.1: Settings in the Sig Gen Block.

Sig type	Amplitude	Pulsewidth	Periodic	Period	Time shift	Freq
Sinc	0.2	120	NO	10	30	0.2

graphically in the z-plane and analyze the corresponding frequency response in real time. Poles and zeros are added as conjugate pairs, and no more than 5 pairs can be entered. Graphical manipulation of poles and zeros is achieved through "Add Pole", "Add Zero" and "Reset" buttons on the left part of the screen. Users can place and move poles and zeros pairs. As you move the poles and zeros, the frequency response will be immediately updated. Adjust the location of the poles and zeros until the desired response is obtained. Note that poles near the unit circle give rise to spectral peaks and zeros near the unit circle create spectral valleys in the magnitude of the frequency response. For this problem, keep all plots in dB scale.

Design a lowpass filter using with approximate cutoff frequency of $\Omega_c = \pi/4$. Use two sets of zero pairs and one pole pair. This can be a rough approximation.

- Save the screenshot of your design using the PZ Placement block.

Did any of the simulations have errors or the program fail to execute (crash)? Please indicate in details so that we can reproduce the error.

B.2.4 Part 2: Filter Design

Problem 2.1. We will use a Sig Gen block, a Window block, and an FFT block to design a causal lowpass filter from the ideal impulse response $h_d(n) = 0.2sinc(\frac{\pi n}{5})$, i.e.,

$$h_{FIR}(n) = w(n)h_d(n-L/2), \quad 0 \le n \le L \tag{B.6}$$

Set parameters in the Sig Gen block as shown in Table B.1 :

Take 256-point FFT of the impulse response. Set the size of window to be 60. Tapered windows have better behaved sidelobes and hence better behaved ripple effect relative to the rectangular window. Verify this property by checking FFT magnitude of Rectangular window and Bartlett (Triangular) window respectively.

The iJDSP block diagram should look like Figure B.6.

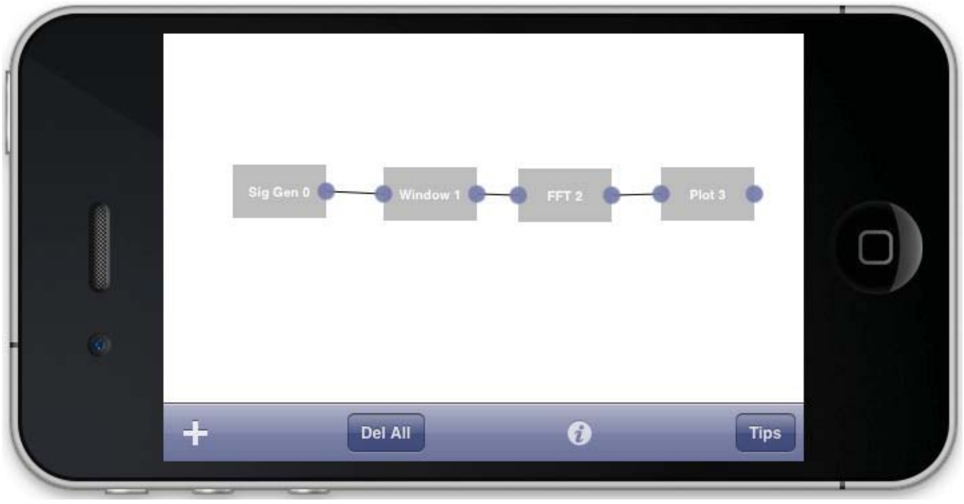


Figure B.6: Block Diagram for iJDSP Prob. 2.1.

Did any of the simulations have errors or the program fail to execute (crash)? Please indicate in details so that we can reproduce the error.

Problem 2.2. Use the Parks-McClellan block and the Kaiser block to design FIR filters with the following specifications:

Filter Type: low-pass filter

Cut-off frequencies:

Passband (W_p) cutoff frequency: $W_{p1} = 0.25\pi$

Stopband (W_s) cutoff frequency: $W_{s1} = 0.4\pi$

Tolerances: $PB = 1dB$; $SB = 40dB$.

A sample of the block diagram is shown in Figure B.7.

- Sketch the magnitude and phase plots of the Parks-McClellan in dB scale (using the Freq. Resp. block).
- Sketch the magnitude and phase plots of the Kaiser in dB scale (using the Freq.

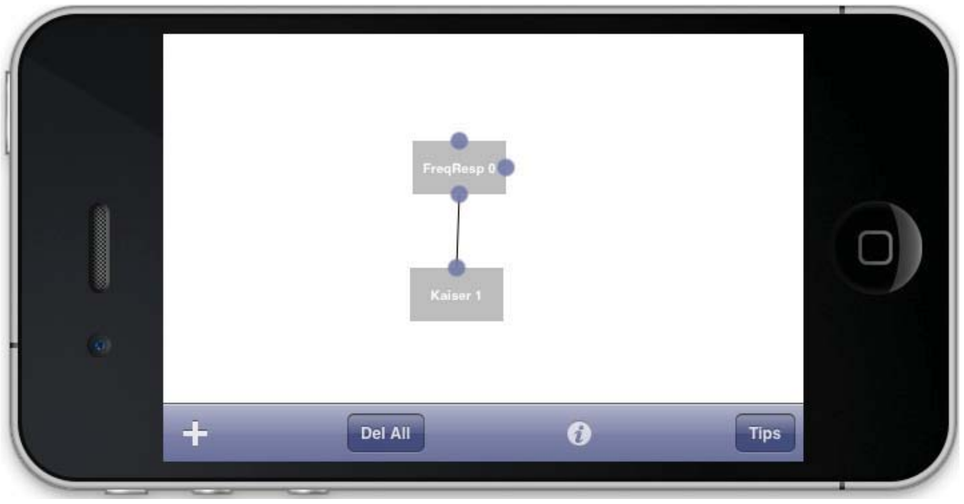


Figure B.7: Block Diagram for iJDSP Prob. 2.2.

Resp. block).

- Which of two methods has the lower order? (To see the order of the filter, tap the "Update" button in the corresponding filter design block.)

Did any of the simulations have errors or the program fail to execute (crash)? Please indicate in details so that we can reproduce the error.

Problem 2.3. Design an IIR Filter with the following specifications,

IIR filter: Butterworth

Filter type: Low-pass

Cutoff frequencies: $W_{p1} = 0.4\pi$, $W_{s1} = 0.6\pi$

Tolerance in passband: $PB = 1.0dB$

Tolerance (rejection) in stopband: $SB = 45.0dB$.

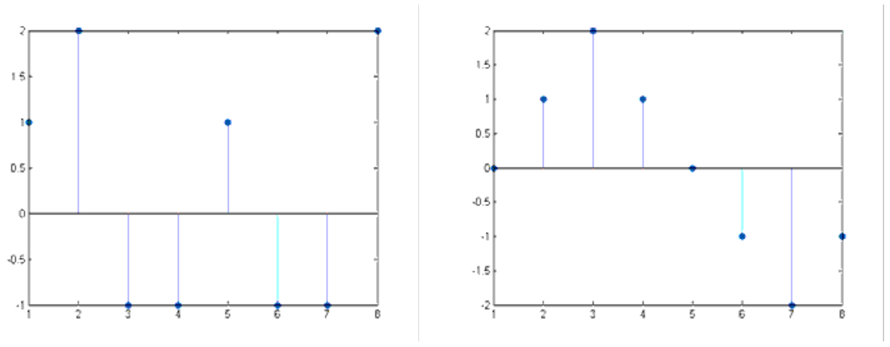
The design can be done using the IIR Design block under the list of functions in iJDSP. Connect the output of the IIR block to the Freq Resp block to plot the frequency response. Compare the frequency response of the IIR filter using four design techniques respectively. (Change IIR filter to Butterworth, Chebyshev I, Chebyshev II, and Elliptic).

- Note down the order of each design. Which one will give the lowest order?
- Sketch the frequency response magnitudes in linear scale. (Butterworth, Chebyshev I, Chebyshev II, Elliptic)
- Is the phase response linear?

Did any of the simulations have errors or the program fail to execute (crash)? Please indicate in details so that we can reproduce the error.

B.2.5 Part 3: FFT

Problem 3.1. Set up the two signals as shown in Figure B.8 using the block diagram in Figure B.9. Examine the FFTs of the following two signals. Choose user-defined signal type in the Sig Gen block (Pulsewidth = 8). Take 8-point FFT to the output of the Sig Gen, then plot the signal (Choose "Real" to see the real part of the signal, and "Imag" to the imaginary part. Tap "View Values" in the Plot block to see the exact values).



(a) Signal 1 (b) Signal 2

Figure B.8: The User-defined Signals.

- Tabulate and compare the real and imaginary parts of the two signals above (linear scale). Keep three digits after the decimal point.

Did any of the simulations have errors or the program fail to execute (crash)? Please indicate in details so that we can reproduce the error.

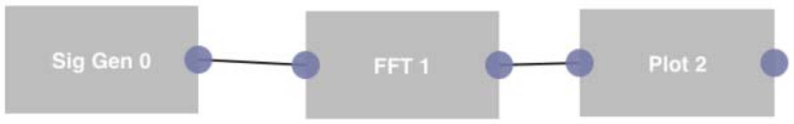


Figure B.9: Block Diagram for iJDSP Prob. 3.1.

Problem 3.2. Generate a signal $x(n) = 0.4sin(0.25\pi n) + sin(0.22\pi n)$ with the length of 128 samples using two Sig Gen blocks and one Adder block. Use a rectangular window that has length 128 samples to truncate the signal. Take 128-point FFT after truncating the signal.

- Double tap the Plot block to see the FFT magnitude in dB scale.
- Set the length of the rectangular window to be 64 samples. Still take 128-point FFT. Check whether it takes effect on the FFT magnitude.
- Change the window to a Hanning window that has length 128 samples, and check the FFT magnitude again. Is the signal resolved precisely?

Did any of the simulations have errors or the program fail to execute (crash)? Please indicate in details so that we can reproduce the error.

B.3 Workshop Technique Questions

These questions are used in pre- and post- labs. They are designed to examine whether the understanding of signal processing is improved after performing simulations on iJDSP. **Question 1.** Given the transfer function for a causal system, which of the statements is CORRECT?

$$H_1(z) = \frac{1}{(1-1.2z^{-1})(1-0.5z^{-1})} \tag{B.7}$$

a. $H_1(z)$ is unstable.

b. The impulse response of $H_1(z)$ is absolutely summable.

c. $H_1(z)$ has two poles and one zero.

d. The ROC of $H_1(z)$ is $|z| < 0.5$.

Question 2. The given cascade system shown in Figure B.10 is equivalent to

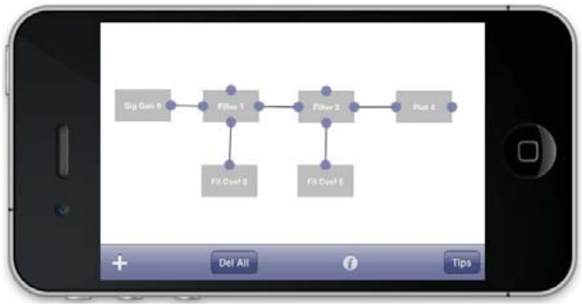


Figure B.10: Block Diagram for Question 2.

a. 1 system, whose impulse response is the convolution of the impulse responses of the 2 systems.

b. 1 system, whose impulse response is the sum of the impulse responses of the 2 systems.

c. 1 system, whose impulse response is the product of the impulse responses of the 2 systems.

Question 3. As the poles of a causal and stable system move away from the unit circle,

a. the peaks in the frequency response become sharper.

b. the peaks in the frequency response become less sharp.

c. there is no change in the frequency response.

Question 4. Which of the two windows in general has better (lower) sidelobe characteristics?

a. Rectangular

b. Triangular

Question 5. Which of the following IIR filters is monotonic in both stopband and passband, in terms of the frequency response magnitude in linear scale?

a. The Butterworth filter

b. The Chebyshev type I filter

c. The Chebyshev type II filter

d. The Elliptic filter

Question 6. IIR filters have linear phase.

a. True

b. False

Question 7. Which signal in Figure B.11 has an FFT that is purely imaginary (real part equals to zero)?

a. $x_1[n]$

b. $x_2[n]$

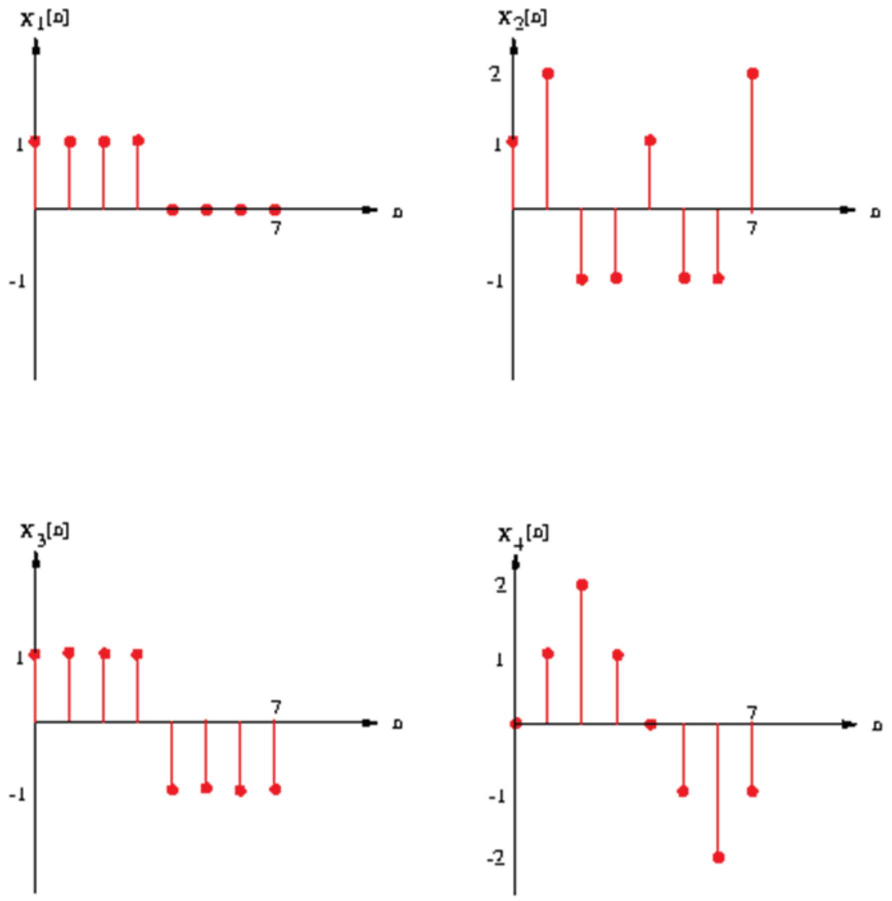


Figure B.11: Signals for Question 7.

c. $x_3[n]$

d. $x_4[n]$

Question 8. Which of the following statements is CORRECT?

a. For a real-valued signal which is odd symmetric around the point k = N/2 (N is the size of FFT), the imaginary part of FFT is zero.

b. FFT components of a real-valued signal are conjugated symmetric.

c. The FFT of a discrete-time signal is continuous in frequency.

Question 9. Which window can better resolve two sine waves which are closely-spaced in frequency (all windows have the same length)?

a. Rectangular

b. Hanning

Question 10. Which of the following statements is INCORRECT?

a. The rectangular window has a narrow mainlobe but prominent sidelobes.

b. Increasing the FFT size improves the ability of the FFT to resolve closely-spaced frequency components.

c. The effects of loss of resolution and spectral leakage are controlled only by the shape of the window.