

AC 2007-1131: USING J-DSP AND LABVIEW TO PERFORM UNDERGRADUATE LABS

Andreas Spanias, Arizona State University

Andreas Spanias is Professor in the Department of Electrical Engineering Fulton School of Engineering at Arizona State University (ASU). He is also the Co-Director of the Sensor Signal and Information Processing (SenSIP) center. His research interests are in the areas of adaptive signal processing and speech processing. While at ASU, he has developed and taught courses in DSP, adaptive signal processing, and speech coding. He has also developed and taught continuing education short courses and web courses in digital signal processing and speech coding. Andreas Spanias has been the principal investigator on research contracts from Intel Corporation, Sandia National Labs, Motorola Inc., and Active Noise and Vibration Technologies. He has also consulted with Inter-Tel Communications, Intel Corporation, Motorola, Texas Instruments, DTC, and the Cyprus Institute of Neurology and Genetics. In his work with Intel Corporation he contributed to the development of architectures with signal processing capabilities and received an award from Intel for "leadership and contributions to the development of the Intel 60172 processor architecture" and a corporate award for his support of the Intel research program. He recently published refereed papers in Perceptual Coding of Digital Speech and Audio, Adaptive Beamforming, Genomic Signal Processing, and DSP Java tools. He and his student team developed the NSF funded computer simulation software Java-DSP (J-DSP - ISBN 0-9724984-0-0) which is being used in the ASU DSP courses. He received the 2003 teaching award from the IEEE Phoenix section for the development of J-DSP. Andreas Spanias is associate director of the ASU Arts, Media, and Engineering (AME) program where he heads a program on sound localization for smart stages using microphone arrays. He is involved extensively in IEEE scientific activities. He is member of the DSP Committee of the IEEE Circuits and Systems society, and has served as a member in the technical committee on Statistical Signal and Array Processing of the IEEE Signal Processing society (SPS). He has also served as Associate Editor of the IEEE Transactions on Signal Processing and as General Co-chair of the 1999 International Conference on Acoustics Speech and Signal Processing (ICASSP-99) in Phoenix. He served as the IEEE Signal Processing Vice-President for Conferences and the Chair of the Conference Board. He served as a member of the IEEE Signal Processing Executive Committee and as Associate Editor of the IEEE Signal Processing Letters. He served as a member-at-large of the IEEE SPS Publications Board. He has been Chair of the IEEE Communications and Signal Processing Chapter in Phoenix, and is a member of Eta Kappa Nu, and Sigma Xi. Andreas Spanias is co-recipient of the 2002 IEEE Donald G. Fink paper prize award and he is a Fellow of the IEEE. He served as Distinguished lecturer of the IEEE SPS in 2004 and he received the 2004 IEEE signal processing society award for meritorious scientific service.

Karthikeyan Ramamurthy, Arizona State University

Karthikeyan Ramamurthy is a Masters student in the Department of Electrical Engineering and a student member of the Sensor Signal and Information Processing (SenSIP) center. He worked on the J-DSP project as a programmer of the J-DSP/LabVIEW interface.

Jayaraman Jayaraman , Arizona State University

Jayaraman Jayaraman is a Masters student in the Department of Electrical Engineering and a student member of the Sensor Signal and Information Processing (SenSIP) center. He worked on the J-DSP project as a programmer of the J-DSP/LabVIEW interface.

Mahesh Banavar, Arizona State University

Mahesh Banavar is a Ph.D. student in the Department of Electrical Engineering and a student member of the Sensor Signal and Information Processing (SenSIP) center. He worked on J-DSP

as a programmer and on OFDM systems for his doctoral research.

CHIH-WEI HUANG, Arizona State University

Chih-Wei Huang is a Masters student in the Department of Electrical Engineering and a student member of the Sensor Signal and Information Processing (SenSIP) center. He worked on the J-DSP project as a programmer of audio functions and created a J-DSP hardware interface.

USING J-DSP AND LABVIEW TO PERFORM UNDERGRADUATE LABS

1. Introduction

Java-DSP (J-DSP) is a universally accessible online tool primarily intended for studying concepts in signal processing¹. Extensions to communications², image processing³, speech processing⁴, and controls⁵ have also been previously produced under an NSF EMD program. In addition, J-DSP modules for demos of technology to high schools have also been generated⁶ and interfaces to hardware for real-time DSP have been produced⁷. The software is currently being tested in six different universities⁸ including Arizona State University, University of Texas-Dallas, University of Washington-Bothell, University of Rhode Island, University of Central Florida, and the University of Cyprus. The scripting capabilities embedded in the software enable generation of HTML code⁹ and MATLAB scripts¹⁰.

A new interface has been developed to allow students and users to move between Java-DSP and the National Instruments LabVIEW tool. This interface is made possible using J-DSP Mathscript capabilities and provides an effective way to utilize several functionalities across both visual environments. The motivation for providing this feature is to enable students to access important LabVIEW modules and functions and particularly tap on powerful real-time capabilities of LabVIEW that allow to acquire and process real-time signals. This interface has been tested by students in the Digital Signal Processing laboratory.

2. Generating Mathscript Code from J-DSP Simulation

Mathscript is a text-based scripting language available in LabVIEW that can be executed in the Mathscript window by the LabVIEW runtime engine. A native LabVIEW function, Mathscript node, can be used as well to form and execute a program in block diagram mode. As such most or all J-DSP simulations can be translated to Mathscript code, modularly developed for block execution. Figure 1 illustrates the generation of Mathscript code for a certain J-DSP simulation. The generated code can be obtained by selecting “Export script” feature under the

file menu. The code is then copied and saved as a file and can be subsequently loaded by LabVIEW. Running J-DSP as a signed applet can also eliminate the cut-and-paste process and make this code export seamless. The details of the interface are provided in the next section.

3. Interface between J-DSP and LabVIEW

The interface developed is a LabVIEW model (*JDSP_Labview_Interface .vi*) that takes the Mathscript code generated by J-DSP and simulates the functionality of the original block diagram as designed in J-DSP.

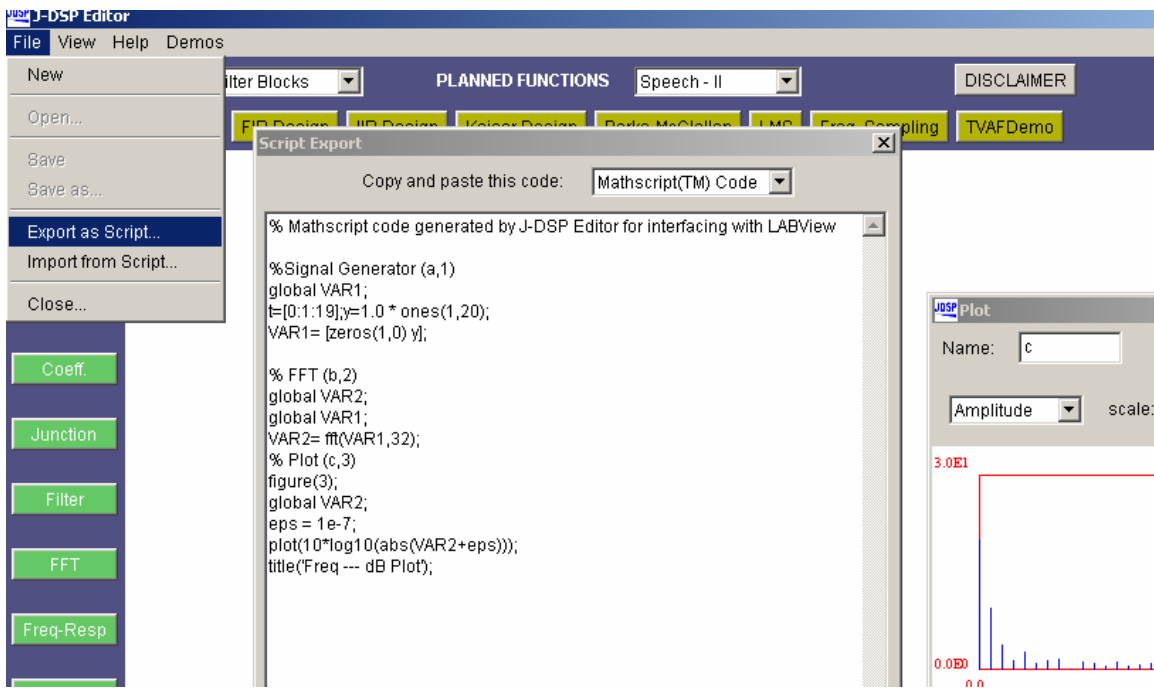


Figure 1. Generating Mathscript code from J-DSP

In addition the interface also generates a pictorial layout of the block diagram in the LabVIEW front panel. This enables the user to visualize the logical flow of the simulation as expressed by J-DSP and allows the user to handle the outputs of intermediate blocks using the native LabVIEW VIs. This has a great advantage in that it enables the user to combine the functionalities of both J-DSP and LabVIEW.

The Mathscript code generated by J-DSP is modular in the sense that it contains the code generated for each block in a sequential manner. This allows the *JDSP_Labview_Interface* to interpret the code by splitting the code and using a subvi (*JDSP_Block.vi*) corresponding to each J-DSP block. This prototype interface hence provides a way to use the intermediate outputs of the block diagram to extend the design. The *JDSP_Block* consists of a Mathscript node which executes the code associated to the block. As illustrated in Figure 2, the *JDSP_Labview_Interface* can be used as a subvi in the model that the user develops, by providing as input, the path of the Mathscript file generated by J-DSP, saved in the local computer. The outputs of each block are available as output terminals of the *JDSP_Labview_Interface*. The design can be extended by using the outputs with native LabVIEW blocks thereby effectively interfacing the features of both applications.

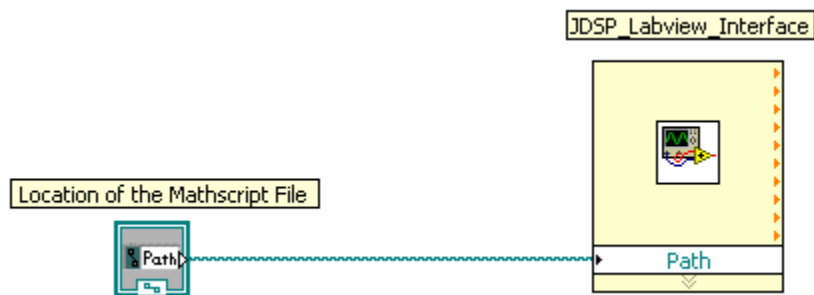


Figure 2. A simple model demonstrating the JDSP_Labview_Interface

Once the model shown in Figure 2 is executed in the LabVIEW environment the simulation is executed. Also by double clicking the *JDSP_Labview_Interface* block, the front panel of the subvi appears where the pictorial layout of the J-DSP block diagram can be seen. The total number of blocks is also displayed in the front panel. An example demonstrating the interface and steps to be followed are explained in the next section.

4. Example of a J-DSP Simulation Interfaced with LabVIEW

A bandpass filter is designed using frequency sampling method in J-DSP. The Mathscript code generated corresponding to this simulation is used with the developed interface. This design is extended in LabVIEW by designing a filter with similar specifications using window method

with native LabVIEW blocks. The input to this extension is taken from output terminals of the *JDSP_Labview_Interface* block. Figure 3 illustrates the creation of a block diagram in J-DSP and Figure 4 shows the pictorial layout generated by the *JDSP_Labview_Interface* block in LabVIEW.

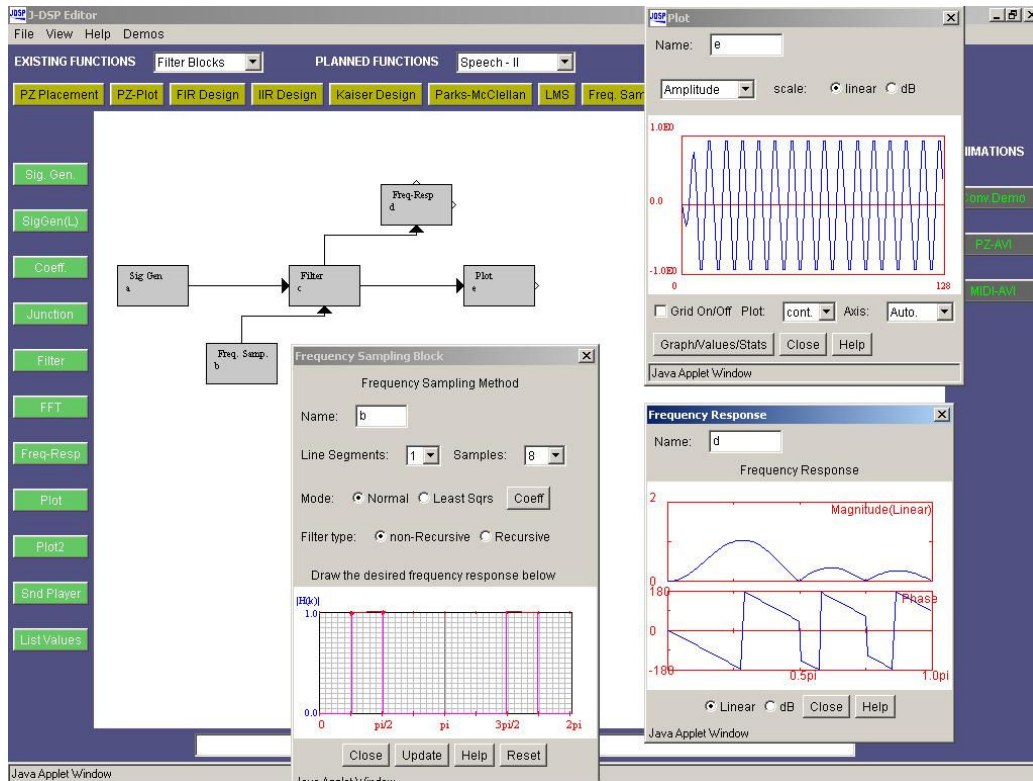


Figure 3. Filter Design using Frequency Sampling in J-DSP

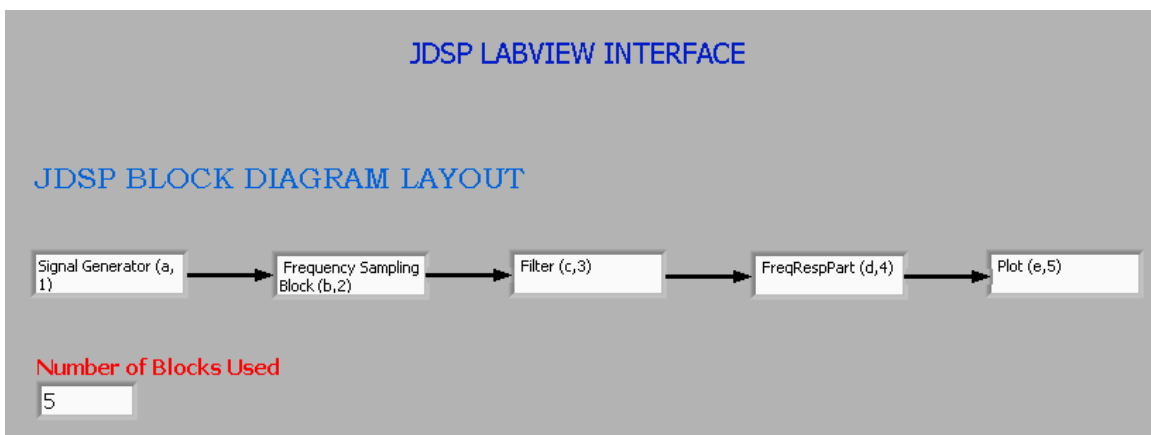


Figure 4. Pictorial Layout Generated by LabVIEW Interface using Mathscript Code

5. Example with Real-time signals

In addition to enabling the user to extend the design in LabVIEW, the interface allows to exploit the real-time processing capabilities of LabVIEW. Real-time inputs can be acquired into LabVIEW and processed by the Mathscript code generated by J-DSP using the interface. Figure 6 (in the next section) illustrates this by using real-time audio inputs with the ‘FFT based compression’ experiment. The user can provide the real-time speech input through the microphone. This experiment can be extended to varied range of real-time signals that LabVIEW supports.

6. Utility in DSP Courses

The interface to LabVIEW was used to demonstrate a real-time signal acquisition, processing, and playback by establishing the program in the J-DSP environment on the Internet and then porting it to LabVIEW that supports real-time signal acquisition. An FFT-based speech compression simulation that helps show the utility of Parseval’s theorem was demonstrated. Other pedagogical byproducts of these experiments across the two platforms include exposition of students to several programming and data flow structures through examples with J-DSP scripts, Mathscript, and LabVIEW VIs.

7. Assessment

FFT based speech compression involves processing the speech signal frame by frame and then transforming it using the FFT. A reduced number of DFT components is then used to reconstruct the signal. In this last “data compression” step, only a specific number of FFT magnitude peaks are selected and used for signal reconstruction. The peak picking implies that maximum signal to noise ratio is maintained which can be shown using the Parseval’s theorem. This experiment combines the functions of both J-DSP and LabVIEW by modularly splitting it across the two environments. Information on FFT-based compressions schemes, such as this simple one, are described in text books^{11,12}.

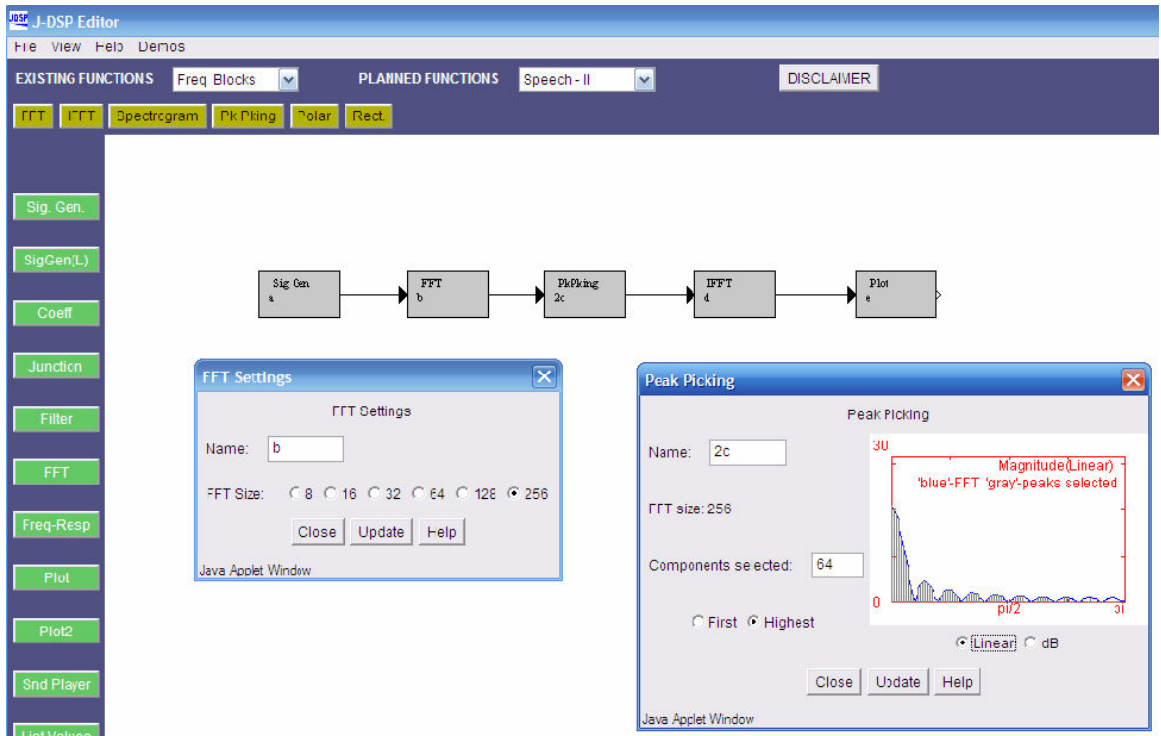


Figure 6. FFT Compression – Basic Block Diagram in J-DSP

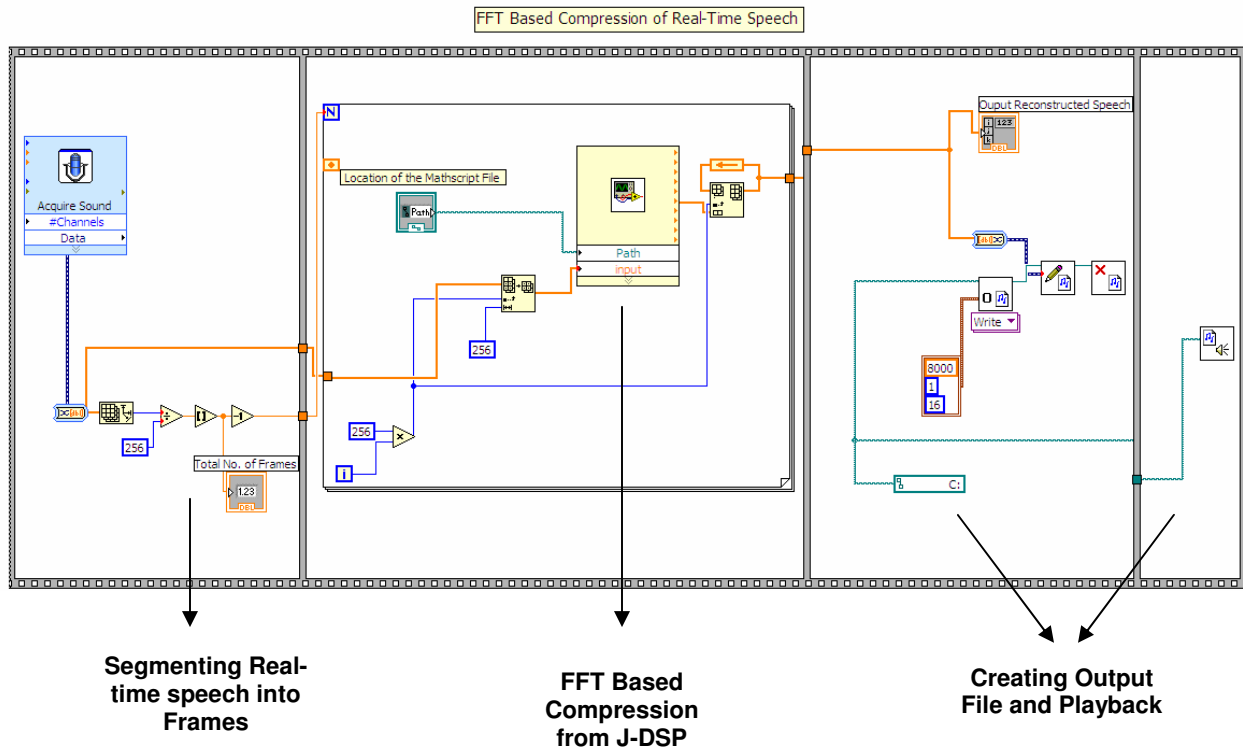


Figure 7. FFT Compression – LabVIEW Model with Real-Time Input and Playback

This experiment uses LabVIEW to acquire real-time speech and split the signal into frames of size 256 samples. The frames are then processed by the *JDSP_Labview_Interface* block, which executes the Mathscript code generated by J-DSP for the design shown in Figure 5. The processing uses the Peak Picking block in J-DSP, which selects the highest components or first few components from the DFT coefficients based on the choice made. The reconstructed frames after peak picking can then be handled in LabVIEW to create output files with playback features. The model created in LabVIEW is illustrated in Figure 6.

An assessment quiz was administered before (pre-quiz) and after (post-quiz) the hands-on laboratory exercise. Some of the questions posed are itemized below:

1. \mathbf{S} is the frequency domain vector representation of the speech signal vector s . If \mathbf{S} consists of N components, which one of the following approaches, would result in better speech quality ($n < N$)
 - a) Picking the n largest components of \mathbf{S}
 - b) Picking the first n components of \mathbf{S}
2. In question 1, if \mathbf{S} contained 256 FFT components, which one of the following would result in best speech quality
 - a) Selecting the first 16 components
 - b) Selecting the highest 16 components
 - c) Selecting the highest 64 components
 - d) Selecting the first 64 components
3. The FFT Peak-picking analysis-synthesis process can be used to compress speech signals
 - a) True
 - b) False
4. The magnitude of the N -point FFT of the speech signal is defined as $S(0), \dots, S(N-1)$. It can be shown that the magnitude spectrum is
 - a) Even-Symmetric about the point $N/2$
 - b) Odd-symmetric about the point $N/2$
 - c) Even-symmetric about the point $N/2+1$
 - d) Odd-symmetric about the point $N/2+1$
5. \mathbf{S} is the N -point FFT of the signal vector s . Which of the two maximizes the SNR of the reconstructed signal ?
 - a) Selecting the first n components of \mathbf{S}
 - b) Selecting the highest n components of \mathbf{S}
6. Which of the two methods for selecting the reduced set of FFT components of the signal produces a low pass filtering effect?

- a) Selecting the highest n components
- b) Selecting the first n components

7. SNR maximization in FFT component selection for speech analysis-synthesis can be shown using the Parseval's theorem

- a) True
- b) False

In addition to the above questions an interview was carried with the participating students.

We conducted a preliminary assessment using a small group of undergraduate and graduate students. Students were given the quiz before and after working with these two environments. There was evidence of improvement in the understanding of this FFT application after forming the simulation on J-DSP and conducting the real time experiment on LabVIEW. When students were interviewed they noted on the average that combining features of J-DSP and LabVIEW was useful in terms of utilizing the real time capability of LabVIEW and the simplicity of organizing the DSP simulation on J-DSP. Our assessment above was preliminary with only a few students. We will be conducting additional assessment in the DSP class in the spring and fall semesters of 2007 and will have additional detailed quantitative assessment at the conference presentation.

8. Conclusion

A J-DSP interface to LabVIEW was described. The interface uses the LabVIEW Mathscript code and the J-DSP scripting capability. The scripting functionality of J-DSP provides an intuitive way to combine the features of J-DSP with advanced real time processing capabilities of LabVIEW. A preliminary assessment was conducted using pre- and post-quizzes and student interviews. Students noted that combining features of J-DSP and LabVIEW was useful in terms of utilizing the real time capability of LabVIEW and the simplicity of organizing the DSP simulation with J-DSP. More detailed assessment results will be reported at the conference.

Acknowledgement

Portions of this work have been supported by the NSF EMD CCLI Program DUE 0443137 and the ASU SenSIP Center.

Bibliography

- [1] Spanias, A.; Atti, V., "Interactive online undergraduate laboratories using J-DSP," *IEEE Transactions on Education*, vol. 48, no. 4, pp. 735- 749, Nov. 2005.
- [2] Youngwook Ko; Duman, T.M.; Spanias, A., "On-line laboratory for communication systems using J-DSP," *Proc. FIE 2003*, vol. 1, pp. T3E-13- T3E-18, Denver, Nov. 2003.
- [3] Yasin, M.; Karam, L.J.; Spanias, A., "On-line laboratories for image and two-dimensional signal processing," *IEEE FIE 2003*, pp. T3E-19- T3E-22 Vol.1, Denver, Nov. 2003.
- [4] Atti, V.; Spanias, A., "On-line simulation modules for teaching speech and audio compression techniques," *Proc. FIE 2003*, vol. 1, Denver, Nov. 2003.
- [5] Thrasyvoulou, T.; Tsakalis, K.; Spanias, A., "J-DSP-C, a control systems simulation environment: labs and assessment," *Proc. IEEE FIE 2003*, pp. T4E_11- T4E_16, Denver, Nov. 2003.
- [6] Spanias, A.; Thrasyvoulou, T.; Yu Song; Panayiotou, C., "Using J-DSP to introduce communications and multimedia technologies to high schools," *IEEE FIE 2003*, vol. 2, pp. F3A_22- F3A_27, Denver, Nov. 2003.
- [7] A. Spanias, V. Berisha, H. Kwon, C. Huang, A. Natarajan and R. Ferzli, "Using the Java-DSP Real-Time Hardware Interface in Undergraduate Classes," *IEEE FIE 2006*, Session M4D, San Diego, Oct. 2006.
- [8] A. Spanias, V. Atti, R. Chilumula, S. Haag, A. Papandreou-Suppappola, C. Tepedelenlioglu, J. Zhang, F. Boudreaux-Bartels, M. Stiber, T. Kasparis, P. Loizou, "WIP - Multi-University Development and Dissemination of Online Laboratories in Probability Theory, Signals and Systems, and Multimedia Computing," *IEEE Proc. FIE-2005*, Oct. 19-22, Indianapolis 2005.
- [9] A. Spanias, and F. Bizuneh, "Development of new functions and scripting capabilities in Java-DSP for easy creation and seamless integration of animated DSP simulations in web courses," *IEEE ICASSP '01*, vol. 5, 7-11, pp. 2717-2720, Salt Lake City, May 2001.
- [10] A. Spanias, C. Panayiotou, T. Thrasyvoulou, and V. Atti, "MATLAB Interface with Java Software", *Proceedings of the 2004 American Society for Engineering Education Annual Conference & Exposition*, Salt Lake City, June 2004.
- [11] A. Spanias, T. Painter, V. Atti, *Audio Signal Processing and Coding*, Hardcover 464 pages, ISBN: 0-471-79147-4, Wiley, New Jersey, February 2007.
- [12] Andreas Spanias, *Digital Signal Processing; An Interactive Approach*, Textbook with JAVA-DSP computer exercises, ISBN 978-1-4243-2524-5, Tempe, February 2007.