Teaching Digital Signal Processing on Smartphones: A Mobile DSP Laboratory

Nasser Kehtarnavaz and Shane Parris
Department of Electrical Engineering, University of Texas at Dallas, USA
kehtar@utdallas.edu

Abstract

This show and tell provides an interactive demo of a newly developed educational paradigm for teaching applied or real-time digital signal processing courses. It involves the utilization of smartphones to implement digital signal processing algorithms in real-time using ARM processors of smartphones. Such a paradigm is exhibited to be a cost-free mobile laboratory. Attendees will be given the option of downloading one of the experiments as an app and running it on their own smartphones.

1. Introduction and motivation

Applied or real-time digital signal processing courses offered at many universities have greatly enhanced students' learning of signal processing concepts by covering practical aspects of implementing signal processing algorithms. DSP processor boards are often deployed in these courses. A number of textbooks are available discussing how to implement signal processing algorithms on DSP boards, e.g. [1-3]. In this show and tell, we present an alternative hardware platform which students can use right away as it is already in their possession, that being their own smartphones!

Not only do there exist hardware and software costs associated with equipping a teaching laboratory with DSP or other types of boards, in many cases these boards are confined to a specific teaching laboratory location. Taking advantage of the ubiquitous utilization of ARM processors in mobile devices, in particular smartphones, we have developed an alternative approach to teaching applied or real-time DSP courses by enabling students to use their own smartphones to implement digital signal processing algorithms. Changing the hardware platforms that are currently used for applied or real-time signal processing courses to smartphones creates a truly mobile laboratory experience or environment for students. The software development tools for smartphones are free of charge and are well-developed. The total cost of the currently used DSP boards ranges from $100 to $500 per unit when the costs associated with peripherals and software tools needed for these boards are factored in. In contrast, this newly developed teaching paradigm provides clear advantages over the existing approaches in terms of both cost and mobility.

2. Smartphone implementation

In this show and tell, we will demonstrate the algorithms associated with a typical applied or real-time DSP course running on a smartphone. The main challenge in this approach has been the difference between the programming environments on smartphones and C programming, which signal processing students are normally familiar with. We have met this challenge by developing the required software shells to run C codes on smartphones so that the only prerequisite programming knowledge students need to have is C programming. In other words, the programming knowledge expected from students is no different than many existing applied or real-time DSP courses. This newly developed teaching paradigm allows students to implement signal processing algorithms that are written in C on their own smartphones. Also, students get to experiment with real-time and optimization aspects of running C codes efficiently on ARM processors.

2.1 Development environment

To allow C codes to be written and compiled on smartphone targets, we have utilized the following cost-free downloadable development tools: Android Development Tools Bundle (ADT Bundle) [4], Android Native Development Kit (Android NDK) [5] and the ARM DS-5 Community Edition plug-in [6].

The ADT Bundle provides a comprehensive development environment incorporating the Eclipse integrated development environment (IDE), plug-ins, and an emulator. The NDK tool provides the support for incorporating C/C++ codes within the ADT. The ARM DS-5 Community Edition plug-in provides debugging capabilities for C codes.

3. Laboratory experiments

The laboratory experiments that are included in a typical applied or real-time signal processing course include: signal sampling and I/O buffering, FIR filtering, IIR filtering, adaptive filtering, quantization and roundoff errors, fixed-point versus floating-point implementation, DFT/FFT frequency transformation, and optimization techniques to gain computational efficiency. These experiments will be demoed during this show and tell. To provide a better idea of the demos, three of the experiments are described in more details in the next section. Attendees can bring their own smartphones (Android operating system not older than two years) and we will let them run and experiment with one of the experiments as an app on their own smartphones. In essence, this interactive show and tell will present for the first time a smartphone-based laboratory for teaching applied or real-time digital signal processing courses.

4. Representative demos

Three representative laboratory experiments that will be demoed are described in more detail in this section. A video clip of a sample demo can be viewed at http://www.utdallas.edu/~kehtar/ShowTell.mp4.

4.1 Real-time filtering demo

This demo involves processing a frame of signal samples captured by the smartphone microphone. Figure 1 illustrates a snapshot of the smartphone running screen where processing times of frames are displayed. The frame length can be altered by the user through a graphical-user-interface (GUI) settings menu, shown in Figure 2. The sampling rate can also be altered depending on the sampling rates permitted by the A/D converter of the smartphone. In the smartphone model to be demoed, the sampling rate can be altered from 8 kHz to 48 kHz. Thus, any processing of one frame of data needs to be done in less than $N*dt$ sec in order to achieve a real-time throughput, where N denotes the frame length and dt the sampling time interval. For example, for a sampling rate of 8 kHz and a frame length of 256, the processing needs to be completed within 32msec in order for all the frames to get processed without any frames getting skipped.

A low-pass FIR filter together with a user specified delay are considered to act as the signal processing algorithm running on the ARM processor of the smartphone. The delay can be changed by the user through the GUI settings menu, adding additional processing time to the low-pass filtering time. By increasing the sampling frequency or lowering the sampling time interval, data frames will get skipped and hence a real-time throughput cannot be met. Besides skipped frames noted on the GUI, one can hear both the original signal and the filtered signal through the speaker of the smartphone and notice the distortion caused by increasing the sampling frequency due to the real-time demand. Distortion can also be experienced by increasing the processing time delay. This demo provides the option of reading data from an audio file or receiving audio data from the microphone, as well as writing data to an audio file or outputting audio data through the speaker.

4.2 Quantization effect demo

In this demo, the quantization effect is exhibited. The demo involves running an FIR filter on the smartphone using fixed-point arithmetic. 16 bits are used to quantize the double precision floating-point filter coefficients generated by filter design packages. Due to quantization, the frequency response of the filter gets affected. The quantization word length can be adjusted in the settings menu and the deviation of the frequency response magnitude can be observed in a pop-up graph, shown in Figure 3. The settings menu allows the user to alter the quantization bits to examine the deviation of the frequency response from the frequency response of the floating-point implementation. In addition, due to quantization, overflows may occur depending on the number of coefficients. This demo shows how scaling can be used to overcome overflows by scaling down input samples and scaling back up output samples generated by the filter.

4.3 Adaptive filter demo

This demo exhibits adaptive filtering. An adaptive FIR filter based on the least mean squares (LMS) coefficient update algorithm is implemented to match the output of an IIR filter. Figure 4 illustrates the error between the output of the adaptive FIR filter and the IIR filter for a square wave input signal displayed on the smartphone screen. Over time the error between the two outputs diminishes towards zero.

The user can experiment with the rate of convergence by altering the adaptive filter order through the settings menu without needing to recompile the code. As the filter order is increased, it can be observed that the convergence rate also increases. The drawback of increasing the filter order, which is an increase in the processing time, can also be observed. The demo allows establishing a tradeoff between convergence rate and real-time throughput.

5. Conclusion

This show and tell exhibits how smartphones can be used as the target platform for teaching applied or real-time digital signal processing courses. This alternative approach to the currently utilized DSP boards is cost-free and provides a truly mobile laboratory experience for students. A textbook is being written by the authors discussing the details of this alternative approach which is planned to be completed by the year end.

6. References

[1] N. Kehtarnavaz, *Real-time Digital Signal Processing Based on the TMS320C6000*, Elsevier, 2005.

[2] T. Welch, C. Wright, and M. Murrow, *Real-Time Digital Signal Processing from MATLAB to C with the TMS320C6x DSPs*, CRC Press, 2011.

[3] S. Kuo and B. Lee, *Real-Time Digital Signal Processors: Implementations, Applications and Experiments with the TMS320C55x,* Wiley, 2001.

[4] http://developer.android.com/sdk/index.html

[5] http://developer.android.com/tools/sdk/ndk/index.html

[6] http://ds.arm.com/ds-5-community-edition/

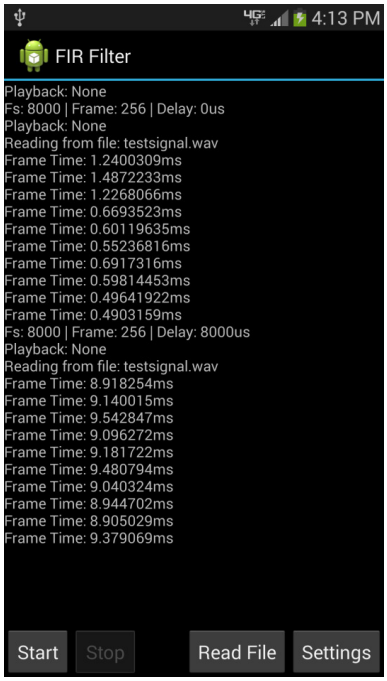

Figure 1. Real-time filtering demo: running screen

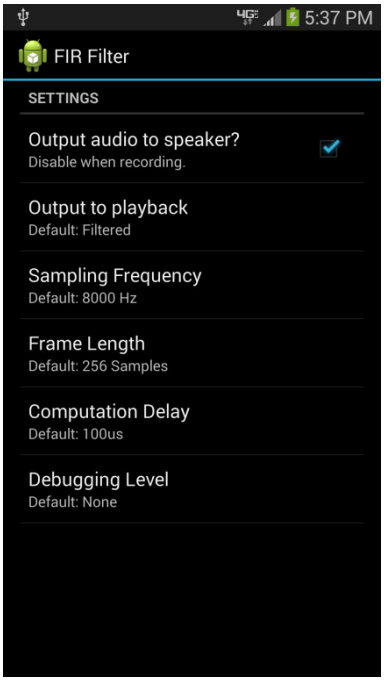

Figure 2. Real-time filtering demo: settings interface

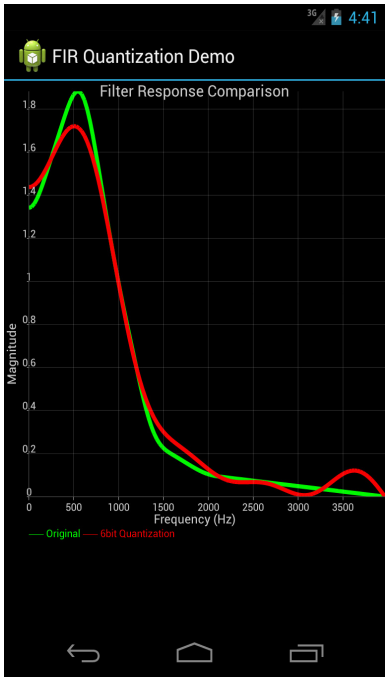

Figure 3. Quantization demo: frequency response graph

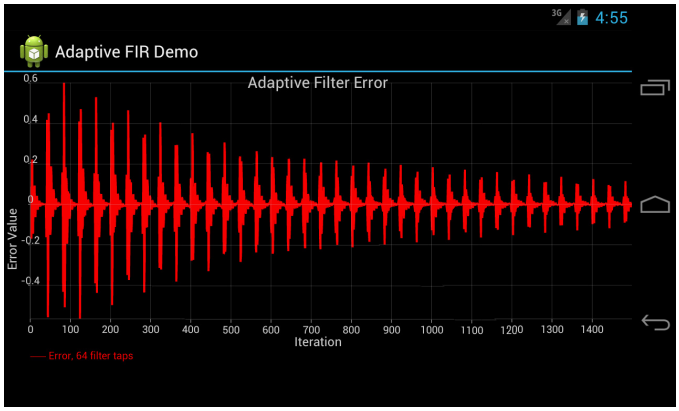

Figure 4. Adaptive filter demo: error versus iterations