A LABORATORY EXPERIMENT FOR REAL-TIME ECHO CANCELLATION USING A BEAGLEBOARD

Ori Bryt, Asaf Elron, Pavel Lifshits, Tsahee Zidenberg, Yair Moshe, Nimrod Peleg

Signal and Image Processing Laboratory (SIPL)
Department of Electrical Engineering, Technion – Israel Institute of Technology
Technion City, Haifa 32000, Israel
Phone: +(972)-4-8294746, fax: +(972)-4-8292796, e-mail: yair@ee.technion.ac.il
http://sipl.technion.ac.il/

ABSTRACT

Practical experience is an important aspect of the training of every engineer. One way to develop such experience is by hands-on laboratory experimentation which involves cutting-edge technology. In this paper, we present a signal processing lab experiment developed for undergraduate students. Throughout the experiment, students first learn the basics of adaptive filtering and implement an adaptive echo cancellation algorithm using MATLAB. Then, students use a software framework for implementing the same echo cancellation algorithm on a BeagleBoard development platform using both ARM and fixed-point DSP cores of an embedded system-on-chip from Texas Instruments. In a relatively short amount of time (8 hours at the lab and a few additional hours of homework), students learn an important signal processing technique, as well as use complex state-of-the-art DSP hardware to implement it as part of an application running in real-time.

1. INTRODUCTION

Signal processing has many theoretical and practical aspects. While signal processing theory has been taught in academia for decades, teaching the cross-over from theory to practice has always been lacking. Beginning with the theory allows instructors to focus on the underlying concepts of signal processing without the distractions associated with practical implementation. However, engineers must be able to efficiently and cost-effectively implement these concepts in real-world designs [1, 2].

Digital signal processors (DSPs) are optimized to perform typical signal processing tasks, such as filtering, convolution, or FFT. They usually consume less power and are cheaper than general-purpose processors. These properties make DSPs useful in a wide range of applications, especially for real-time, cost-effective embedded systems, such as cellular phones, modems, and disk drivers. The use of DSPs has been incorporated in some universities as part of the signal processing curriculum, usually in the context of a course of one or two semesters [3, 4]. However, allowing one or two semesters for implementation may not leave enough time to gain deep theoretical knowledge; so, giving students a taste of implementation in a shorter time period may be desired. Furthermore, modern embedded systems are using devices with multiple processing units manufactured on a single chip, creating a multi-core system-on-chip (SoC). On such chips, fixed-point DSP cores are usually used to perform specialized DSP operations, while floating-point cores allow the overall system to handle general-purpose tasks. Software development for SoC involves partitioning the application among the various cores. The development environment is usually heterogeneous, with tools that support code generation and debugging on all cores. Obviously, developing for such a platform is more challenging compared to developing for a single-core DSP platform.

In this paper, we present a laboratory experiment that has been developed in the Signal and Image Processing Laboratory (SIPL) at the Department of Electrical Engineering, Technion – Israel Institute of Technology. The experiment is one of about three dozen experiments, 8-hours each, offered to students in our department. Each undergraduate student must select and carry out several of these experiments as part of the obligatory curriculum for a B.Sc. degree. The experiment presented here is designed for undergraduate students who took at least one basic course in signal processing. Some of the students who perform the experiment may have already taken some more advanced signal processing courses, but no knowledge of adaptive filtering is assumed. The experiment is part of our lab's educational activities in the area of embedded signal processing. These activities include experiments and student projects, such as the ones described in [2, 5-8]. One of the main goals of this experiment is to introduce the students to development for an embedded platform; thus, no preliminary knowledge of such a platform is assumed, either. An OMAP3 processor from Texas Instrument [9] is used as an example. Creating a hands-on lab experiment for students, introducing and utilizing SoC architecture, is not an easy task due to the complexity of such a system. By only spending a few hours at the lab, with no assumption of background knowledge beyond a basic signal processing course, the creation of such an experiment becomes very challenging.

The experiment consists of two sessions of 4-hours each, with equal time for homework. The experiment is performed in pairs. We have installed four experiment stations; so, currently, 8 students perform the experiment at the same time and are guided by one instructor. Before each session, students are required to complete a preparatory

report. Questions that should be answered in this report guide the reading of the lab booklet, emphasize important concepts, and ensure that students arrive at the lab with the proper background. After completing each lab session, students have to submit a final report in which they analyze their results and draw conclusions. The students' grades are based on the two preparatory reports and two final reports, as well as on the lab instructor's evaluation.

The remainder of this paper is organized as follows. In Section 2, we describe the first part of the experiment, where students learn the basics of adaptive filtering using MATLAB. In Section 3, we describe the second part of the experiment. In this part, students implement an adaptive filter in C on a host computer and port their implementation to an embedded platform. We summarize and draw conclusions in Section 4.

2. ADAPTIVE ECHO CANCELLATION

Adaptive filtering is a technique that estimates an unknown linear system based on the sequential analysis of input and reference signals. Adaptive filters are useful in cases where system parameters are slowly changing, and where a filter should automatically adapt to this change. They can be used in many applications, such as echo cancellation, channel equalization, and active noise-control. At every time instance, the adaptive filtering algorithm maintains a current estimate of the linear system, while applying it to a reference signal $x[n]$ and examining the output $d[n] - y[n]$, as depicted in Figure 1. The primary input and reference signals, $d[n]$ and $x[n]$, respectively, are assumed to be partially correlated. A well-known archetype of adaptive filtering algorithms is the least mean squares (LMS) algorithm.

The LMS algorithm arises as a stochastic gradient descent approach to minimizing

$$E\{(d[n] - y[n])^2\} \tag{1}$$

assuming that $x[n]$ and $d[n]$ are random processes. The vector of adaptive filter coefficients $\widehat{\boldsymbol{w}}_{\boldsymbol{n}}$ takes the form

$$\widehat{\boldsymbol{w}}_{\boldsymbol{n}} = \widehat{\boldsymbol{w}}_{\boldsymbol{n-1}} + \mu\,(d[n] - y[n])\,\boldsymbol{x_n} \tag{2}$$

where $\boldsymbol{x_n} = [\,x[n] \quad \cdots \quad x[n-N+1]\,]^T$ and μ is the step size of the underlying stochastic gradient descent. Assume some arbitrary initial condition $\widehat{\boldsymbol{w}}_{\boldsymbol{0}}$. Further details regarding LMS and adaptive filtering can be found in [10].

In this experiment, students deal with the well-known acoustic echo cancellation problem [11]. We simulate a device containing a speaker and a microphone, positioned inside a medium-sized reverberant room. The device's microphone picks up a signal $d[n]$ containing the voice of two persons speaking simultaneously, where one is located somewhere in the room (s_1), and the other is on the other side of the phone line, speaking through the device's speaker (s_2). We denote by $h_1[n]$ and $h_2[n]$ the Room Impulse Response vectors (RIRs) modifying the signals $s_1[n]$ and $s_2[n]$, respectively. The aim, as presented to the students, is to extract the speech of the person located in the room from the microphone signal $d[n]$, assuming the signal $x[n]$ emanating from the device's speaker is known. Formally,

$$\begin{aligned} x[n] &= s_2[n] \\ d[n] &= s_1[n] * h_1[n] + s_2[n] * h_2[n] \end{aligned} \tag{3}$$

and the aim is to produce the signal $s_1 * h_1$ as $d[n] - y[n]$. An adaptive filtering approach suits this problem well, as the signal from the device's speaker is slightly modified before being picked up by the microphone due to the room reflection and absorption properties. We model the microphone signal as the primary input signal $d[n]$ and the device's speaker signal as the reference signal $x[n]$, and attempt to estimate h_2 for correct subtraction of $s_2 * h_2$ from the input signal $d[n]$. One should note that in this setup, the RIR does not change over time; thus, there is no real need for adaptivity, but rather for online operation of the algorithmic solution.

The first part of the experiment is performed solely on a host computer, and its main stages are: introduction to echo cancellation, estimation using LMS, and estimation using partially fixed-point implementation of LMS. Due to the feedback present in adaptive filters, they are sensitive to numerical impreciseness. Such numerical impreciseness may be easily introduced by inadvertent fixed-point implementation. Thus, it is didactically interesting to investigate fixed-point implementation as early as possible in the experiment. In the introduction stage, students are presented with the above problem and are encouraged to analyze it and solve it using naïve measures, such as simple subtraction of the reference signal $x[n]$ from the primary input signal $d[n]$. Such an attempt will fail due to above-mentioned room reverberation, embodied in h_2. Following the failure of the naïve measures, the students implement a basic version of the LMS algorithm and examine its performance for the problem at hand. They are guided to test and find a step size that leads to good performance of the algorithm for this specific scenario and set of signals. Then, the students force the implementation to use only values feasible under a fixed-point regime for the LMS parameters and intermediates, while the mathematical operations are still carried out in floating-point. Since we expect that having students perform the mathematical operations in fixed-point would be time consuming beyond the scope of the experiment, we intentionally avoid that. To the effect of using fixed-point values for parameters and intermediates, students are instructed to use MATLAB's fixed-point toolbox. The students are asked to find suitable parameters for the fixed-point representation of the algorithm parameters and intermediates, i.e., word size and fraction length, bringing into consideration both the stability and accuracy of the resulting implementation, as well as its expected computational cost. To this effect, they are instructed to observe histograms of the values of the intermediates which arise when running LMS with the

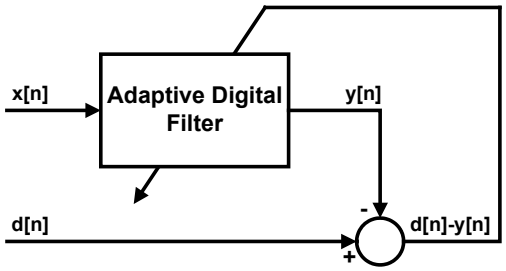

Figure 1: Adaptive filtering structure.

given signals and chosen parameters. In forming this experiment, we chose a basic version of the LMS algorithm for several reasons. Most importantly, this algorithm is a classical and an important archetype in adaptive filtering, and serves as a necessary basis for more complex adaptive filtering algorithms. Nevertheless, it is quite simple and rather intuitive, while the vast majority of the target students are unfamiliar with it.

Even the most basic version of LMS has two design parameters: the length of the estimated impulse response N, as well as the step size μ. Examining the effect of these parameters on the behavior of the algorithm is of educational value. However, in order to remain within the experiment's time frame, we chose to examine only the effect of the step size μ, as we believe understanding its importance and effect has the highest educational value among the two. The step size μ has a critical effect on the convergence and performance of the LMS algorithm. Moreover, it is the more difficult parameter to choose. The students are instructed in the choosing of an appropriate value for the step size μ. At first, the students witness divergence of LMS, given an inappropriate step size μ. Then, we explain to them that μ should be inversely proportional to the variance of the input signal σ_d^2, as well as to the length of the estimated impulse response N, i.e.

$$\mu = \mu_0 \frac{1}{N\sigma_d^2} \qquad (4)$$

where μ_0 is an appropriate constant. Thus, given the variance of the input signal σ_d^2, the problem of finding a good value for μ is transformed into one of finding a good value for μ_0. Given a range of values for μ_0, students are instructed to perform a coarse grid search within that range, choosing a value for μ_0 according to the resulting performance, as reflected by performance evaluation measures.

We utilize two performance evaluation measures. One is qualitative where the students are asked to listen to the output and evaluate its perceived intelligibility and quality. The other performance measure is quantitative where, given a ground-truth signal $s_1 * h_1$, we define

$$R[n] = \sum_{k=1}^{n} \tau^{n-1} \frac{(s_1[k] * h_1[k])^2}{(s_2[k] * h_2[k] - y[k])^2} \qquad (5)$$

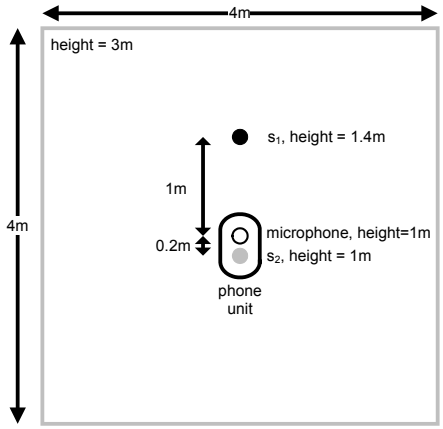

Figure 2: Room setup simulated in the experiment.

where τ is an exponential forgetting factor. At every time instant n, $R[n]$ gives an assessment of the SNR up to that time instant, with the past weighted exponentially to reduce its importance. The weighting allows for this performance measure to signify the convergence of the examined algorithm.

For the two signals, $s_1[n]$ and $s_2[n]$, we use recordings of two men, with a sample depth of 16 bits. The two signals are downsampled to 11025 Hz and trimmed to 30 seconds. The two RIRs, h_1 and h_2, both of length 4096 samples, are generated according to the setup depicted in Figure 2, using publicly available software [12]. In choosing these scenario parameters, we aim to balance five considerations, with various trade-offs existing among them:

- Evident convergence of the online algorithm.
- Rate of computations on the target platform.
- Acceptable MATLAB running times (even for naïve implementations).
- Adequate precision for representing speech signals.
- Realistic room modeling.

We chose a 16-bit sample depth so that the signals contain only a small quantization noise, as it might hinder the performance of the echo cancellation. The room arrangement, including physical size, speaker, and microphone locations and reverberation time, were chosen as to reflect a typical office and conference call setting.

3. EMBEDDED IMPLEMENTATION

After completing the first part of the experiment, the students have gained some practical experience with LMS using MATLAB, including fixed-point implementation. In the second part of the experiment, the students implement an adaptive filter in C on a host computer. Then, they implement the same adaptive filter on an embedded platform. During the implementation, the students are introduced to the different aspects of embedded development, such as embedded operating systems, heterogeneous multi-core processors, real-time constraints, and optimization.

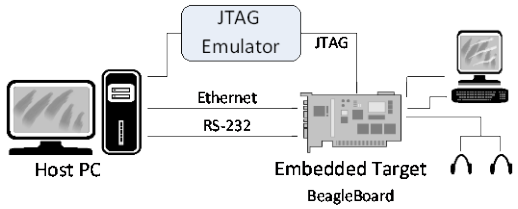

Figure 3: Hardware layout of the embedded part of the experiment.

The growing popularity of mobile devices, such as smartphones and tablets, sets some new requirements for embedded hardware and software. Such devices are often used for varied applications that involve different types of media and that are computationally intensive. Heterogeneous multi-core SoC processors are a natural choice for this kind of devices. The OMAP3530 processor from Texas Instrument [9] is a very popular example of such a processor. The BeagleBoard [13], which is an OMAP3530-based development module, makes the perfect choice for the laboratory target platform, mainly because of its low cost, availability, state-of-the-art OMAP3 processor, sufficient peripheral access, and the existence of a large and active development community. The growing popularity of embedded Linux and its free licensing makes it the obvious choice of a target operating system.

Figure 3 depicts the hardware configuration used in the embedded part of the experiment. A BeagleBoard (embedded target) is connected to a host computer using a dedicated Ethernet connection, a serial RS-232 connection, and a JTAG emulator. The Ethernet connection is the main connection channel between the host and target, while the JTAG connection is used for debugging the DSP core of the target. The serial connection is used by the experiment instructor for initial setup and maintenance. We equip the BeagleBoard with a small keyboard and connect it to a monitor. This is done in order to help the students understand the ability of the embedded platform to constitute a standalone product. To allow a pair of students to cooperate on the experiment easily, without disturbing others at the lab, we also provide two pairs of headphones with a signal splitter. The general-purpose processor (GPP) ARM core of the embedded platform is running embedded Linux, and the host computer is also running Linux. There are two main reasons for this choice. First, since the destination platform is running embedded Linux, and since for many students this is their first encounter with the Linux operating system, getting to know it on the host computer makes for a good kick start. Second, some development and debugging tools are currently only available for Linux.

Students who perform the experiment are faced not only with the need to grasp embedded implementation concepts, but must also deal with new tools. There are many complex development tools due to the heterogeneous nature of SoC platforms. To ease coping with this challenge, we have built a unified development framework. We achieved this by creating a development environment that produces all

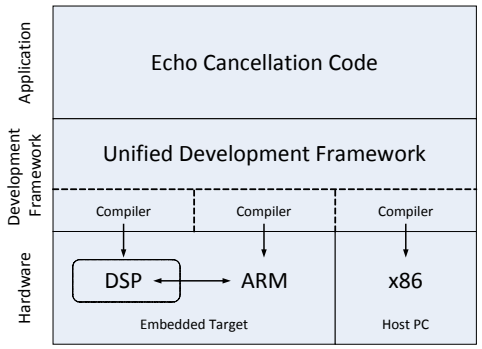

Figure 4: The unified development framework created for this experiment helps students software development by creating an abstraction over different hardware targets.

executables during the practical phase of the laboratory, whether the destination platform is the host computer, the embedded ARM core, or the embedded ARM core outsourcing the computationally intensive tasks to the DSP core. In all three configurations, the students use the same integrated development environment (Code Composer Studio [14]) and implement the same functionality in order to perform acoustic echo cancellation. Figure 4 depicts this idea. In addition to the unified development framework, we supply a skeleton of a multimedia application, including means for input and output, memory management, file access, inter-processor communication, etc. The students implement a given function, while the rest of the application is already implemented. This way, students do not have to spend time on technical details and can focus on the algorithm, the concept of fixed-point representation, and the unique DSP architecture.

An alternative approach to the skeleton application and unified development environment that we supply, and that we don't use in this experiment, is using Simulink's Embedded Coder [15]. The Embedded Coder tool allows the user to design an algorithm using Simulink, and to automatically generate a prototype application that runs on an embedded processor. This approach is easy to implement in a short amount of time. However, it has three main drawbacks compared with the approach we use in the experiment. First, the resulting embedded code is complex and cannot be edited by a beginner. Second, porting to the embedded platform is automatic, so the user is not exposed to key embedded concepts, development environments and tools. Third, this approach is not common in the industry; so, using it in the experiment will neither prepare the students to practical work today, nor in the near future.

During the embedded part of the experiment, we use Linux input and output redirection capabilities to switch signal sources and destinations seamlessly. We switch the input between a file (a signal prepared in advance for the laboratory) and a microphone, and the output between a file (for playing the signal in order to check that the result sounds as expected) and the analog audio output of the BeagleBoard (in order to hear if the implementation meets real-time constraints). At first, the students are instructed to

compile their code for the ARM core. Students are surprised to find that, although the application performs correctly, real-time constraints are not met, meaning that the audio reproduced in real-time is discontinuous. From this point on, students are instructed how to evolve the implementation to meet real-time constraints. The application displays the processing rate, so that the students get to see a numeric indication of the improvement in running time. The students are presented with a simplified version of a typical DSP optimization process, as described in [16]. The students are guided to apply the optimization process until real-time constraints are met. Optimization is performed through several stages:

- Buffering – A buffer is an area of memory that is reserved to temporarily store input data. As input is acquired sample-by-sample, and in order to minimize system overheads, input samples are loaded into a buffer and processed together. The students are introduced to this idea and have to discuss the tradeoffs of using different buffer sizes.
- Outsourcing computationally intensive tasks to the DSP core
- Transforming the code to be fixed-point – After outsourcing part of the code from the ARM to DSP the core, the students transform this part from floating-point to fixed-point in order to support the native numerical representation of the DSP. In this stage, the scaling factors that were found during first session of the experiment are used.
- Correct usage of compiler flags, keywords, and pragma (implementation-dependent) directives – The students are guided to use different compiler options and keywords for parallel utilization of the DSP functional units. High throughput is achieved by usage of the DSP pipeline. The students analyze compiler feedback and resulting performance to get insights on the architecture of the DSP core.

In all parts of the experiment, special emphasis is placed on teaching the students correct embedded development habits, such as frequent compilation, testing, and debugging. General engineering methodologies and considerations are also presented and practiced.

4. CONCLUSION

In this paper, we have described a novel laboratory experiment for undergraduate students with only basic background in signal processing, and none in embedded systems. During the experiment, the students are exposed to the basics of adaptive filtering, and to aspects of embedded signal processing implementation. We use the BeagleBoard development board as a target platform. This board contains an OMAP3 system-on-chip, with ARM and DSP cores. The students are able to use such a complex system in a very short time (only 8 hours in class) due to a software framework that was created especially for this purpose. In order to encourage similar experiments in other universities, we will supply the full code and implementation solutions for the experiment upon an instructor's request.

ACKNOWLEDGMENT

We would like to thank Prof. David Malah, head of SIPL, and Prof. Israel Cohen for their helpful advice and comments. Creating the experiment described in this paper was supported, in part, by Texas Instruments.

REFERENCES

[1] C. Wicks, "Lessons Learned: Teaching Real-time Signal Processing [DSP Education]," *IEEE Signal Processing Magazine,* vol. 26, pp. 181-185, 2009.

[2] N. Peleg, *et al.*, "Benefits of DSP Extra-curricular Activities: A Look at the Texas Instruments DSP and Analog Challenge," in *IEEE Intl. Conf. on Acoustics, Speech, and Signal Processing (ICASSP)*, Orlando, 2002.

[3] M. Klostermann, *et al.*, "OMAP 3 based Signal Processing for Biomedical Engineering Teaching," in *17th European Signal Processing Conf. (EUSIPCO)*, Glasgow, pp. 495-499, 2009.

[4] R. Chassaing and D. Reay, *Digital Signal Processing and applications with the TMS320C6713 and TMS320C6416 DSK, 2nd ed.*: Wiley, 2008.

[5] T. Mizrahi, *et al.*, "Real-Time Implementation for Digital Watermarking in Audio Signals Using Perceptual Masking," in *3rd European DSP Education and Research Conf.*, Paris, 2000.

[6] Y. Kuszpet, *et al.*, "Implementations of H.264/AVC Baseline Decoder on Different Digital Signal Processors," in *16th Picture Coding Symposium (PCS)*, Lisbon, 2007.

[7] R. Giryes, *et al.*, "Embedded System for 3D Shape Reconstruction," in *3rd European DSP Education and Research Symposium (EDERS)*, Tel-Aviv, pp. 265-272, 2008.

[8] E. Roichman, *et al.*, "Real-Time Pedestrian Detection and Tracking," in *European DSP Education and Research Symposium (EDERS)*, Tel-Aviv, pp. 281-288, 2008.

[9] *OMAP3530/25 Applications Processor (SPRS507F)*. http://www.ti.com/product/omap3530: Texas Instruments, 2008.

[10] S. O. Haykin, *Adaptive Filter Theory* 4th ed.: Prentice Hall, 2001.

[11] J. Benesty, *et al.*, *Advances in Network and Acoustic Echo Cancellation*: Springer, 2001.

[12] E. A. P. Habets, *Room Impulse Response Generator*. http://home.tiscali.nl/ehabets/rir_generator.html, 2010.

[13] BeagleBoard. http://beagleboard.org/.

[14] Code Composer Studio IDE v5. http://www.ti.com/tool/ccstudio.

[15] Embedded Coder - User Guide (R2012a), MathWorks. http://www.mathworks.com/products/embedded-coder/.

[16] P. Yin, *Introduction to TMS320C6000 DSP Optimization (SPRABF2)*: Texas Instruments 2011.