TEACHING DIGITAL SIGNAL PROCESSING WITH STANFORD'S LAB-IN-A-BOX

Fernando A. Mujica, William J. Esposito, Alex Gonzalez, Charles R. Qi,
Chris Vassos, Maisy Wieman, Reggie Wilcox, Gregory T. A. Kovacs, and Ronald W. Schafer

Department of Electrical Engineering
Stanford University
Stanford, CA 94305-9505

ABSTRACT

This paper describes our efforts to include a hands-on component in the teaching of core concepts of digital signal processing. The basis of our approach was the low-cost and open-source "Stanford Lab in a Box." This system, with its easy to use Arduino-like programming interface allowed students to see how fundamental DSP concepts such as digital filters, FFT, and multi-rate processing can be implemented in real time on a fixed-point processor. The paper describes how the Lab in a Box was used to provide a new dimension to the teaching of DSP.

Index Terms—real-time DSP, filters, FFT, C++

1. INTRODUCTION

Traditional DSP courses at both the undergraduate and graduate levels often focus on the basic theory and the mathematical representation of algorithms based on that theory. This is not surprising, since actual computational implementation of the algorithms can involve an additional layer of knowledge of programming languages and arcane debugging systems. For this reason, most DSP courses today rely on MATLAB[1] or other high level programming languages to provide a taste of practical application of the theory. While this approach resembles the prototyping phase of real DSP system implementations, the final implementations often use heterogeneous compute elements including programmable processors, configurable hardware accelerators (HWA) and/or hardcoded hardware blocks. The idea behind the Lab in a Box DSP Shield is to take students one step closer to real-time applications with a minimum of overhead for learning new programming systems. As described in Esposito, et al. [1], the Lab in a Box DSP Shield is based on a Texas Instruments TMS320C5535 processor and an AIC3204 audio codec. It is designed to stand alone or to operate as a "shield" connected to an Arduino board. The programming language is C++, which is known by most engineering graduate students, and the interface to the board is through an Arduino-like open source interface called Energia [2]. The Energia/Arduino application programming interface (API) paradigm offers easy to use function calls to access the various hardware components. In addition, the Energia environment includes TI's C55 DSP library of functions for common signal processing operations [3]. Our hypothesis was that this simplified programming paradigm would make it relatively easy for most students to experience the satisfaction of seeing mathematical concepts turn into real experiences. A key feature of the approach based on the DSP Shield is that no physical laboratory facilities are required. Students' laptop computers are all that are needed to program and control the DSP Shield.

[1] MATLAB is a registered trademark of Mathworks, Inc.

2. CLASS CONTENT

The Stanford course, EE 264, "Digital Signal Processing" is aimed at mature undergraduates and first-year graduate students. It assumes a previous undergraduate introduction to the basic concepts of DSP, including discrete convolution, difference equations, sampling and the discrete-time Fourier transform. This class digs deeper into many of the important details that are useful knowledge for applications of DSP technology. Table 1 shows the topical outline of the 10-week-long course.

Table 1: Class outline

Label	Topic	classes
a	Introduction and review	1
b	Discrete-time random signals	2
c	Sampling, reconstruction, D-T filtering of C-T signals, and multi-rate systems	3
d	Quantization and oversampling in A-to-D conversion	1
e	Properties of LTI systems	1
f	Quantization in fixed-point implementations of filters	3
g	Digital filter design	1
h	Discrete Fourier Transform and FFT	1
i	Spectrum analysis using the DFT	3
j	Parametric signal modeling	2

These are all topics that arise in practical applications and are therefore prime targets for enhanced learning with an associated lab based on a fixed-point embedded processor. In particular, topics c, d, f, g, h and i can be supported by the addition of a hands-on lab/project component.

3. DSP IMPLEMENTATIONS

A canonical DSP System for processing analog signals is shown in Figure 1. It comprises an Analog-to-Digital converter, a DSP System and a Digital-to-Analog converter.

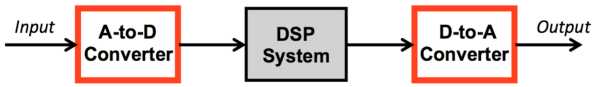

Figure 1: Canonical DSP system.

In most DSP courses the generic DSP System block above is typically expanded with a functional block diagram of the particular algorithm at hand, but rarely are implementation details exposed. In most practical systems however, the actual implementation might include programmable processors, configurable HWA, and even parameterized analog components. In this paper we describe a set of laboratory exercises using the DSP Shield that expose students to HWAs, programmable processors and such related concepts as flow control background processing, and interrupt service routines. The hardware partitioning is as depicted in Figure 2.

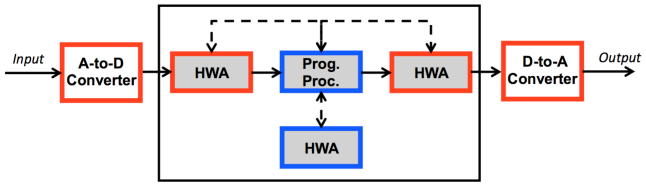

Figure 2: DSP System partitioning.

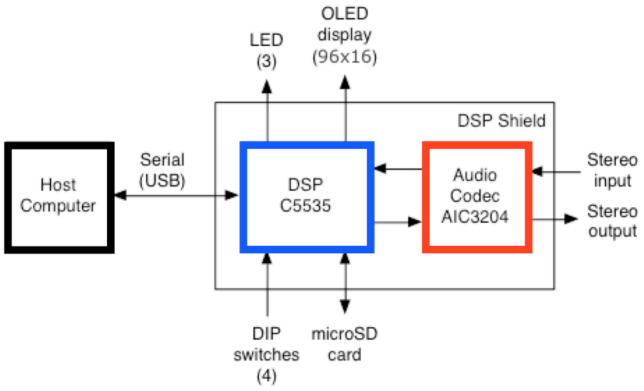

Figure 3: DSP Shield components.

The audio codec in the DSP Shield offers not only stereo A-to-D and D-to-A converters but also a set of configurable HWA that implement various types of FIR and IIR filters. The programmable processor also includes an FFT HWA. The mapping between the various implementation components and the integrated circuits in the DSP Shield is illustrated in Figure 3. This figure also shows the serial interface between the DSP Shield and the host computer that is used for programming, debugging and as input/output (I/O). It is noted that this interface is optional and that both the Arduino and the DSP Shield, once programmed, can operate untethered from a computer.

4. LAB STRUCTURE

We piloted the lab part of the class as an optional one extra credit hour to the regular Stanford Digital Signal Processing class in the Winter 2015 academic term. The lab structure comprised 6 pre-defined labs, which were assigned one per week, and a 4-week long open-ended project. In the following subsections we describe the objective of each of the labs and the project.

4.0. Lab 0: Introduction and HW/SW Setup

Instructions to install the Energia integrated development environment (IDE) were released a few days before classes started to allow students to work on the SW setup on their computers. The DSP Shield was distributed on the first day of class and with the help of this lab write-up, the students were able to:

- Verify that the DSP Shield was working properly with their computer.
- Run simple Energia examples

4.1. Lab 1: Getting to know your DSP Shield

The goal of this lab was to get students familiar with programming the DSP Shield and specifically to learn about the various input/output capabilities of the DSP Shield.
By completing this lab the students learned how to:

- Control the LEDs
- Display messages on the OLED display
- Print and scan from console
- Read the DIP switch
- Read and write audio from the microSD card
- Send and receive data from the DSP shield to MATLAB and vice versa

4.1.1. Serial (MATLAB) Interface

To facilitate co-development with host-based development environments like MATLAB, we developed a generic serial interface to send and receive data between the DSP Shield and the host. This capability is offered in the form of an Energia library and has a simple API:

- `serial_connect(baudRate)`: establishes a serial connection at the specified baud rate.

- `serial_recv_array(data, length)`: receives a data array of length elements.
- `serial_send_array(data, length)`: sends a data array of length elements.

These functions are implemented in both the host and the DSP Shield and are run in complementary pairs. The send and receive functions are overloaded for 8, 16 and 32 bit integers.
The serial interface in the host is implemented in MATLAB but it can be ported to other high level languages such as Python.
In addition, a serial command class is provided to facilitate the implementation of interfaces where the host issues commands to be executed on the DSP Shield.

4.2. Lab 2: Software Flow Control

The goal of this lab was to introduce real-time implementation issues associated with embedded processors. After completing this lab the students should understand the basics of:

- Real-time embedded flow control
- Interrupt service routines (ISR)
- Buffer management

The software flow control template uses the following functions:
- `setup()`: executes once at program entry. This function is used to initialize variables, establish serial connection and other tasks that need to execute only once.
- `loop()`: executes in an infinite loop. This function is used for background tasks.
- `ISR()`: executes once per interrupt event. A single interrupt is triggered by the direct memory access engine on completion of a block of data transfer to the DAC.

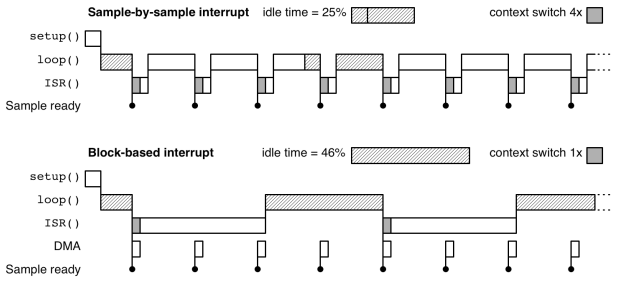

Figure 4: Flow control timing diagram.

The timing relationship between the flow control functions is illustrated in Figure 4. This figure also illustrates the advantages of block-based processing by amortizing the ISR overhead.

The students were introduced to buffer management concepts in this lab. The ping-pong scheme used throughout the labs is illustrated in Figure 5.

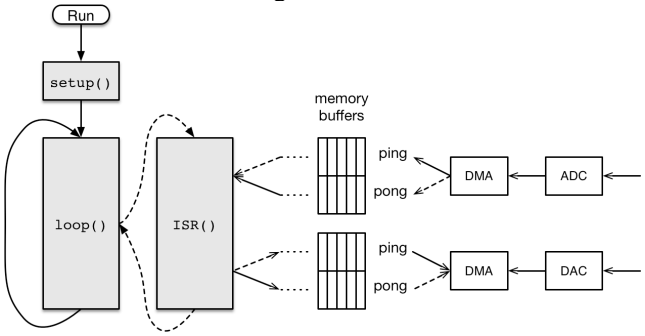

Figure 5: Buffer management.

4.3. Lab 3: Fixed-point Arithmetic

The goal of this lab was to introduce fixed-point implementation issues in embedded processors. After completing this lab the students should understand the basics of:

- Fixed-point arithmetic and modeling
- Using the C55x DSP Lib

This lab complements the class discussion on analysis of round-off quantization errors and prevention of overflow. The focus is on understanding implementation issues that arise in fixed-point implementations using two's complement arithmetic. We covered the rounding and truncation quantization and saturation and wrap overflow handling.

4.4. Lab 4: Sampling Rate Conversion

The goal of this lab was to explore practical implementation of rate conversion, complementing the course material on interpolation, decimation and non-integer re-sampling. After completing this lab the students should understand how to implement:

- Rate conversion systems
- A basic rate converting Karaoke application

The audio codec in the DSP Shield offers very flexible sampling options and even allows independent sampling rates for the ADC and DAC. However, because of a limitation in the HW and to simplify the SW flow control, we constrain the ADC and DAC sampling rate by a rational ratio.
With the ultimate goal of developing a simple Karaoke machine, the students were guided step by step to implement the following rate conversion systems:

- Increase the sampling rate of the singing voice ADC by a factor of 3 from 16 KHz to 48 KHz.
- Increase the sampling rate of music recorded at 32 KHz to 48 KHz.
- Mix singing voice sampled at 16 KHz with the music recorded at 48 KHz and playback the result at the DAC output rate of 48 KHz.

4.5. Lab 5: Codec Hardware Accelerators

The goal of this lab was to explore the use of hardware accelerators (HWA) filters for the ADC and DAC. After completing this lab the students should understand how to:

- Develop a framework for block-based adaptation of the HWA filter coefficients from the DSP shield directly or via the host using the serial interface.

The AIC3204 audio codec offers various HWA filter configurations for the ADC and DAC [4]. Two of the configurations offered for the ADC are shown in Figures Figure 6 and Figure 7.

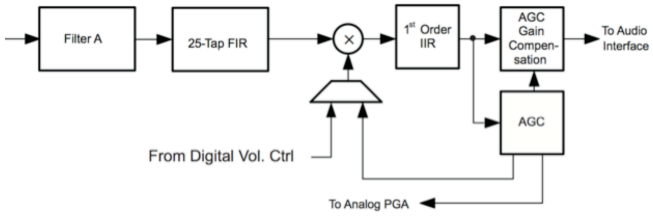

Figure 6: ADC 25-tap and 1st order IIR HWA option.

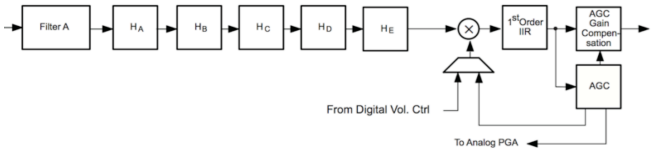

Figure 7: ADC 5 biquad and 1st order IIR HWA option.

All the HWA filters offer a 24-bit data path and double buffered filter coefficients that simplify the implementation of adaptive filtering applications.

4.6. Lab 6: DSP Lib FFT functions

The goal of this lab is to explore the use of the DSP Lib FFT functions [3]. After completing this lab, the students should understand how to:

- Use the DSP Lib FFT and inverse FFT functions
- Implement periodic convolution using the FFT and inverse FFT
- Perform basic spectral analysis
- Optionally, implement linear convolution using the FFT and inverse FFT

Although, not implemented in this version of the course, in the future we plan to incorporate a lab describing the use of the tightly coupled FFT HWA in the C5535 [5].

4.7. Project

The open-ended project allowed students to utilize concepts from the class and the lab to develop an application of their choice. The project duration was 4 weeks and the students submitted project proposals for staff approval 2 weeks before the project start date. They were given a list of project ideas that the staff considered tractable in the time allotted and by the capabilities of the DSP Shield:

- DTMF (touch tone telephone) decoder
- OFDM digital communication receiver
- Adaptive filtering demonstration
- Phase vocoder (aka, a pitch shifter)
- Heart rate detector
- Programmable audio equalizer
- Tree-structured sub-band coding of audio
- Time/frequency analyzer (spectrogram)
- Digital FM/AM communications
- Acoustic noise suppression
- Active echo cancellation

Since we did not have a dedicated lab time or facilities, we covered in lecture a typical development schedule to help the students stay on track with their projects:

- Plan / rough timeline
- Literature search
  - Key kernels / algorithms
- Reference model
  - MATLAB (or C/C++)
  - Individual module test bench
  - System test bench
- Embedded processor implementation
  - C/C++
  - Module test bench
  - System level test
- Documentation

The students were expected to have completed the reference model of their application by the end of the second week and were required to have a mid-point check with one of the staff members at this point. This would allow them to budget two weeks for the embedded implementation part of the project.

5. LESSONS LEARNED

There were a total of 45 students enrolled in the class. 15 students took the 4 credit hour option of the class, which included the lab component. Two of the lab students were remote students taking the class via the Stanford Center for Professional Development. They received the lab-in-a-box by mail.
The grading policy for the lab and non-lab students is shown in Table 2.

Table 2: Grading policy.

Activity	No lab (3 credit)	Lab (4 credit)
Mid-term exam	30%	20%
Final exam	40%	25%
Homework	25%	25%
Labs 1-6		10%
Lab project		15%
Participation	5%	5%

In the following subsections we describe the key lessons learned.

5.1. Lab every where / any where

We successfully demonstrated that it is possible to incorporate a hands-on lab component involving an embedded processing board without the need for dedicated lab hours or a physical lab space. This can be attributed to the easy to use Energia IDE. The students were able to perform all the labs using their own computers and the provided hardware. Only one out of the 15 lab students required access to an on-campus computer for the first few weeks as this student is a Linux user and the Energia environment for the C55 is not yet supported on this platform. The student was able to find a compatible Mac OS or Windows computer after few weeks.

5.2. Hardware limitations

The main limitation of the hardware reported by the students is limited memory. The current version of the DSP Shield is limited by the on-chip memory of the C5535 processor of 320KB of combined program and data memory. This prevented the implementation of some of the projects that required large amount of memory. For real-time implementations involving the host, I/O throughput was also limiting.

5.3 Software limitations

The primary limitation of the Energia IDE is limited debug capabilities. There is no single step debug or ability to put breakpoints in the code to analyze processor memory and intermediate results. Instead, users must rely on commands printed to the serial console or to use the MATLAB interface to debug their code. Overall, the simplicity and ease of use of the Energia environment outweighs the disadvantages of limited debug capabilities when compared to a professional environment like Code Composer Studio (CCS)®. It would be interesting to explore the possibility of opening Energia projects in CCS to enable students ready for the next level in sophistication in DSP systems and IDE to transition seamlessly.

5.4. Lab effort

We designed each of the labs to take approximately three hours of effort per week of work for the average student. Unfortunately, many of the lab students reported that the lab portion of the class took considerable more effort than what they expected from one extra credit hour. Based on some of the questions asked during office hours, we believe the wide range of programming experience and the lack of a dedicated lab time with instructor support, contributed to students taking significant time addressing common programming pitfalls.

5.5. Project examples

The second half of the class comprised of an open-ended project. The students were given a list of project ideas to choose from or could propose one on their own, subject to instructor approval. Many of the students commented in the class survey that the open-ended aspect of the project added significant amount of time to the class.
In the rest of this section will briefly describe some of the projects that the students implemented. We gave the option to work individually or in groups of 2-3. We had a total of 13 projects, including two teams of 2 students.

5.2.1. Guitar effects
Two students implemented "guitar effects" systems, to simulate popular musical distortion systems used with electric guitars.
One student implemented a stand-alone fuzz box system, which emulates the "soft" clipping of the type seen in classic vacuum-tube amplifiers. To implement soft clipping, the student developed a nonlinear "sigmoid" signal block [7]. This nonlinear wave shaping function creates high frequency harmonics, necessitating that the student implement up- and down-sampling techniques taught in class. By using these techniques, the student ensured that the output signal did not contain these undesired harmonics.
A second student implemented a multi-effect system that included several chained effects blocks. The first of these blocks was a pair of high- and low-pass "shelving" filters, with a 2nd order Butterworth response. A "wah-wah" effect, was implemented as a state variable filter with a sinusoidal time variant cutoff frequency. The project also included a flanger effect block, implemented as a Finite Impulse Response (FIR) comb filter. Finally, the dynamic range compression block used the hardware in the audio codec. Each of these effects could be switched in real-time using a

MATLAB command line interface to instruct the DSP to enable or disable the requested functionality. The student validated his work by developing a robust MATLAB model and streaming sample data back from the DSP Shield via the serial interface.

5.2.2. Heart Rate Estimation

Two students implemented a wavelet-based algorithm [8] to detect the R peaks of a QRS complex. Through MATLAB prototyping and DSP Shield implementation, the students reinforced decimation, filtering, and fixed point arithmetic concepts. The students then used the MATLAB interface described before to send data blocks to the DSP shield for processing. One student displayed the detected heart rate on the Shield's display screen, while the other sent back the detected R-peaks to MATLAB to quantify performance.

5.2.3. Pitch Estimation and Modification

Two students pursued pitch estimation. One student used the peaks of the cepstrum to determine pitch, requiring the use of the FFT, as well as knowledge of fixed-point arithmetic and windowing. Another student used the Fast Lifting Wavelet Transform, which used concepts similar to those used in the heart rate estimation algorithm, namely, decimation, filtering, and fixed-point arithmetic. These algorithms used audio captured by the ADC, and estimated pitch in real time.

Two students also pursued pitch modification via the Time-Domain Pitch Synchronous Overlap and Add (TD-PSOLA) method [9]. They chose this algorithm because it achieved pitch modification in a relatively simply time-domain operation, rather than through complex and time-consuming frequency-domain operations. In addition to learning about hardware constraints and efficiency, the students implementing TD-PSOLA used concepts of fixed-point arithmetic and windowing.

5.2.4. Key word recognizer pre-processor

One student implemented a filter bank pre-processor to a machine learning based keyword recognizer. The student used the DSP Shield to capture a segment of audio from the on-board converter, implement a filter bank to compute signal energy over various frequency regions and passed the heavily sub-sampled outputs of the filter bank to a keyword recognizer implemented in MATLAB.

6. CONCLUSIONS

The DSP Shield proved to be an effective means of conveying implementation concepts for DSP algorithms. Though students came from a wide range of backgrounds, the Energia interface proved to be an accessible way for students to delve into practical applications. The DSP Shield also afforded students an opportunity to learn about challenges in implementing signal processing algorithms in hardware, such as addressing memory and time constraints—many students were forced to make design choices based on the trade-off between real-time speed and algorithmic complexity.

Analysis of the students' grades in this small data set suggests that the lab option did not affect the students' ability to perform well on the theoretical part of the class.

The feedback from the students was generally positive. Most of the students agreed that the lab component enhanced their learning of the theoretical concepts. In future offerings of the class, we will consider adding dedicated lab time to offer instant feedback to students to avoid lengthy debugging on their own. We might also give 2 or 3 pre-defined project options to avoid the time consuming aspects associated with open-ended projects.

7. ACKNOWLEDGEMENTS

We would like to thank Texas Instruments and Cathy Wicks for donating the DSP Shields; the Stanford EE department for supporting the development of the lab component of the course; and the Stanford office of the Vice Provost of Teaching and Learning for providing valuable feedback from their in-class focus group and course evaluation.

8. REFERENCES

[1] W. Esposito, "The Stanford Lab in a Box", submitted to 2015 DSP/SPE Workshop, Snowbird Utah, 2015.
[2] Energia home page: www.energia.nu
[3] "TMS320C55x DSP Library Programmer's Reference," Texas Instruments, 2013. [link]
[4] "TLV320AIC3204 Application Reference Guide," Texas Instruments, 2012 [link]
[5] "TMS320C55x Data Sheet," Texas Instruments, 2014. [link]
[6] Code Composer Studio®, Texas Instruments, [link]
[7] David Te-Mao Yeh, "Digital Implementation Of Musical Distortion Circuits By Analysis And Simulation", Dissertation, Stanford University, 2009.[link]
[8] J.P. Martinez, R. Almeida, S. Olmos, A.P. Rocha, and P. Laguna, "A wavelet-based ECG delineator: evaluation on standard databases," IEEE Transactions on Biomedical Engineering, Volume 51, Issue 4, 2004.
[9] Valbret, H. ; Moulines, E. ; Tubach, J.P. , "Voice transformation using PSOLA technique," 1992 IEEE International Conference on Acoustics, Speech, and Signal Processing, 1992.