TEACHING PRACTICAL DSP WITH OFF-THE-SHELF HARDWARE AND FREE SOFTWARE

Eric Bezzam, Adrien Hoffet, Paolo Prandoni

Audiovisual Communications Laboratory, École Polytechnique Fédérale de Lausanne, Switzerland
{firstname.lastname}@epfl.ch

ABSTRACT

In this paper we describe our approach to teaching applied digital signal processing (DSP) using freely-available software and inexpensive off-the-shelf hardware components. The pedagogical framework is built around simple real-time audio processing algorithms that provide immediate and engaging feedback to the students. At the same time, our end goal is to build a learning module that can be easily reproduced by fellow instructors and used in our Massive Open Online Course (MOOC). We use Python for the initial algorithmic prototyping and then transition to a hardware implementation using an STMicroelectronics' Nucleo-64 core with Adafruit I/O breakout boards. We employ modern documentation tools based on Git and Markdown in order to share the material and the exercises, allowing students and educators to access, reproduce, and contribute changes to the content.

***Index Terms*—** Digital signal processing, education, off-the-shelf hardware, freely-available software, audio

1. INTRODUCTION

A common problem facing most signal processing instructors today is finding a way to bridge the gap between the theory and the practice of DSP. The task is elusive because, while the theoretical fundamentals of the discipline are well-established, it is very hard to formalize a set of "best practices" that can expose the students to the fine points of practical implementations. At the same time, one would like to avoid the pitfalls of focusing too much on the specifics of a particular hardware and software platform. Ideally, one would like to borrow a page from software "hackers" and be able to use an exploratory, hands-on fashion to familiarize students with the key aspects of real-world signal processing: for instance, fixed-point arithmetic, direct memory access (DMA) transfers, and the inter-IC sound (I2S) protocol. To this end, two requisites are paramount for a practical, accessible, and reproducible DSP education module, especially in the context of remote education:

1. the tools must be industry-level;
2. only low-cost, off-the-shelf components and freely-available software should be considered.

At the Audiovisual Communications Laboratory of EPFL, we have curated a hardware and software kit that meets the above criteria. We have developed a teaching module (freely available online) that gently takes students from the familiar environment of theory and higher-level language prototyping to the realities of hardware-based real-time implementations. The main features of our approach are:

- an exposure to the key "tricks" and limitations of real-world DSP using fun applications in audio signal processing;
- the use of modern documentation tools in order to encourage collaborative efforts in designing exercises;
- the possibility to experience the main lessons in the module even without the proposed hardware, *i.e.* solely with a laptop running a simulation environment in Python.

The documentation for the module and the associated exercises are available online [1, 2]. For obvious reasons, the exercise solutions are not public but can be provided upon request.

The rest of this paper is structured as follows: in the next section we provide an overview of the decision process that led us to our final hardware and software choices, taking into account the existing efforts by fellow institutions and educators. In Sec. 3, we describe the suggested curriculum that motivates key lessons in DSP through audio applications. In Sec. 4, we introduce the documentation tools that allow others to access and modify the content to their liking. In Sec. 5, we reflect on the usage of the proposed exercises at EPFL. In Sec. 6, we provide concluding remarks.

2. HARDWARE AND SOFTWARE OVERVIEW

2.1. Previous Work

Recently, thanks to the individual efforts of motivated educators, several kits and exercises have been designed to expose

The authors would like to thank STMicroelectronics for kindly providing a supply of Nucleo boards.

university students to practical aspects of DSP [3, 4, 5]. At the same time, industrial kits are also available for education and training [6]. An extensive overview of various development kits is presented in [7]; to summarize their findings, we can identify three common hardware choices:

- FPGA development boards, *e.g.* by Altera [8] and Xilinx [9]
- inexpensive single-board computers ($25 – $150), such as Raspberry Pi [10], BeagleBone Black [11], and Wandboard [12].
- single-board microcontrollers (less than $50), such as Arduino boards [13], STMicroelectronics' (STM) Nucleo-64 boards [14], and TI's Launchpad development kits [15].

Examining these options in order, working with FPGA-based boards demands a comfortable grasp of hardware description languages. These skills are beyond the scope of exercises meant to supplement a (largely theoretical) signal processing course. Similarly, single-board computers shift the focus away from DSP and put it on the underlying operating system (OS). For these reasons, we chose to go with a single-board microcontroller. Several universities have used the Arduino platform due to its ease-of-use and widespread popularity. Generally, however, in order to support analog input/output (I/O), these projects involve the design and use of custom shields [3, 4] which greatly limits the ease of reproducibility for the resulting kit. Additionally, the default Arduino IDE is a far cry from professional tools. Commercial kits such as TI's Launchpad, on the other hand, while industry-level in nature, have a much higher price point (on the order of $200 at the time of writing). This presents a significant barrier to adoption for beginners and educators. Even disregarding price, such kits often lack user-friendly software, as noted by [16], which led to the development of the *WinDSK* software. However, it is only available for Windows OS and we strive for maximum OS-independence.

2.2. Microcontroller and Breakout Boards

Our final choice rested with a single-board microcontroller from STM's Nucleo-64 family: the NUCLEO-F072RB [17]. At the time of writing, the board is listed as "Active" and can be purchased for $10.99 on Digi-Key. Our audio I/O relies on the I2S protocol [18]; we thus require two I2S buses for the microphone input and the stereo decoder output. We use Adafruit's I2S MEMS Microphone Breakout and I2S Stereo Decoder, which are both priced at $6.95 at the time of writing [19, 20]. Together with the jumper wires and power cable, priced at $3.95 [21] and $2.95 [22] respectively, our hardware kit comes to a total of $31.79 (excluding shipping costs). Our goal is to provide a low-cost "reference design" for educators and interested individuals, especially in the context of our MOOC effort [23, 24], which has a great following in countries with disadvantageous currency exchange rates. Additionally, while it is difficult to predict whether a certain piece of hardware will be discontinued,[1] STM has a ten-year commitment for its boards, which provides a clear visibility on our project's lifespan.

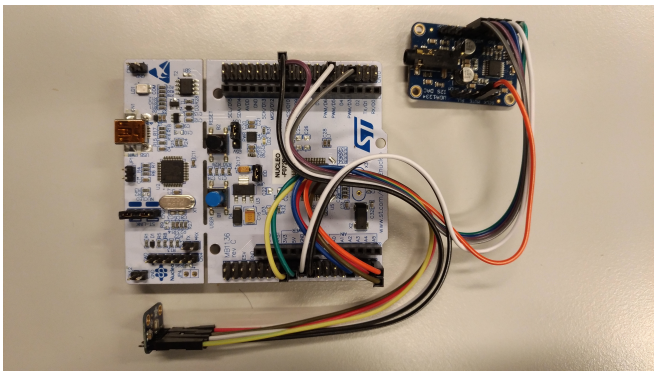

Fig. 1. NUCLEO-F072RB with Adafruit microphone and DAC breakout boards wired together.

Both STM's family of Nucleo-64 boards and TI's Launchpad development kits rely on Eclipse-based IDE's: *System Workbench for STM32* (SW4STM32) [25] and *Code Composer Studio* [26] respectively. We opted for STM's Nucleo boards as their software toolchain includes an additional program called *STM32CubeMX* [27], which significantly simplifies the process of setting up and producing the initialization code for I/O peripherals. We are therefore able to strike a balance between convenience and exposure to industry tools.

2.3. Software

Before the C implementation on the boards, we have the students understand and prototype the tasks in Python, and in a manner as close as possible as would be done with the board, *e.g. for*-loops instead of array operations and fixed-point instead of floating-point. This makes the implementation on the board primarily that of porting Python to C. Moreover, for those who are not able to obtain the hardware, the exercises can be done purely in Python, while still learning the key practical DSP lessons. A template has been prepared in Python with the *sounddevice* library [28] for users to test their implementations with their laptop's sound card.[2]

As previously mentioned, STM provides a software called *STM32CubeMX*, which greatly simplifies the process of setting up and producing the initialization code for I/O peripherals. It is a graphical tool which allows the user to enable and setup different protocols, *e.g.* I2S as is needed for our applications, and general purpose input/output (GPIO) pins of the microcontroller. Once the desired configuration is finished, the necessary C initialization code can be generated through

[1] 5 out of 28 kits / boards were discontinued from the 2016 study in [7].

[2] In [2]: `/scripts/_templates/rt_sounddevice.py`

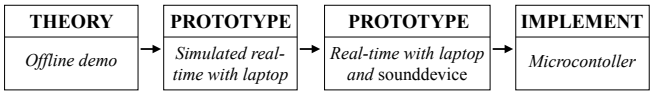

Fig. 2. Learning pipeline.

STM32CubeMX and used in *SW4STM32* with the user's application code. Although we use the NUCLEO-F072RB, other boards from STM will be compatible with the presented exercises as long as they can be used with *STM32CubeMX* and *SW4STM32* and have two I2S buses.

3. LEARNING WITH AUDIO

In order to keep up the students' motivation throughout the sometimes frustrating aspects of programming a microcontroller, we developed a series of exercises with increasing levels of difficulty centered around the idea of developing a "voice scrambler," that is, a gadget that can be used to modify a speaker's voice in real-time. There are various techniques to achieve this, from simple amplitude modulation to more complex pitch shifting based on granular synthesis or LPC coding; the theory and the algorithms are available in the learning module in the form of a Jupyter Notebook [29].

Since Python is used in the prototyping phase, the main DSP lessons are not overwhelmed by embedded systems details. As we are targeting undergraduate students in a relatively theoretical signal processing course, the emphasis of our exercises is on learning DSP skills for real-time processing. This is first done in a simulation environment with a WAV file that is read in a block-based fashion.[3] The goal of the students is to determine the initialization and processing code to apply the effect. The provided script writes the output signal to an array in a block-based fashion and subsequently to a new WAV file, which can be "debugged" by listening to it in order to determine if the processing performed the desired effect. When the output WAV file demonstrates the correctness of the processing, students can either use the *sounddevice* template to directly attempt their implementation in real-time with their laptop's sound card *or* proceed to implementing the application in C on the microcontroller. At this point, the implementation in C should largely be porting Python to C. The full learning "pipeline" is shown in Fig. 2.

As an example, let's consider the exercise of implementing the *alien voice effect*, that is, a voice scrambler based on simple amplitude modulation.

1. The students are first introduced to the theory behind the implementation, namely, multiplying the input signal with a sinusoid:

$$y[n] = x[n] \cdot \sin(2\pi(f_c/f_s)\, n), \tag{1}$$

where f_c is the modulation frequency, f_s is the sampling frequency, and n is the time index of samples that are taken at regular intervals of $1/f_s$. This operation can be done in a single line with Python, which is provided to the students [29]. This offline implementation serves as a useful reference when debugging the output of their approach.
2. Since the algorithm is extremely simple, the focus can move completely to the finer details of a real-time implementation. For instance, we introduce the concept of *lookup tables* for the efficient computation of the sinusoid's values and we introduce the notion of *state variables* for the table lookup. On the signal processing side, we remark the need to remove any *DC offset*, which would introduce a disrupting buzz in the output. With this new knowledge in mind, the students are asked to implement the *alien voice effect* in the simulated Python environment.
3. Once students have successfully implemented the effect, *i.e.* the output WAV file from the simulation environment matches that of the offline implementation, they can attempt to run their solution in real-time with their laptop's sound card and the *sounddevice* library. At this point, students may realize if their implementation can really run in real-time, or if additional optimization is required to cut down on the processing at each block.
4. With the effect running successfully in real-time on their laptop, the students can proceed to implementing it in C on the microcontroller. At this point, the students will have most of the implementation already complete and simply need to port their code to C and possibly implement any macros that might be useful.

This example demonstrates the process that we encourage when approaching a new application. In the following subsection, we give an overview of the proposed exercises and the corresponding learning objectives.

4. SHARING THE MATERIAL

Proper online dissemination of the course material is necessary for adoption by others. It also allows educators to obtain feedback on the material they have prepared. However, other than email communication, other DSP kits typically offer no other means for collaboration, modification, and feedback by others. Education in data science and machine learning has taken full advantage of modern collaboration and documentation tools[4] and we will follow these best practices as well.

There are numerous ways to create and host material online. We chose Markdown for writing content due to its ease-of-use, requiring very minimal knowledge about web devel-

[3] In [2]: `scripts/_templates/rt_simulated.py`

[4] For example, this introductory course to data science at UC Berkeley [30] and the corresponding repository [31].

Exercise	Learning Objective(s)
Overview and installation	• Familiarize students with STM32's software tools (*STM32CubeMX* and *SW4STM32*) with a fully-guided Blinking LED example.
Audio passthrough	• Introduce MEMS microphones, DACs, and the I2S protocol. • How to program pins and define useful macros to access/set their values in C. • Reading datasheets and wiring components! • Terminology: direct memory access (DMA) callbacks, sample, frame, buffer.
Alien voice effect	• Problems/solutions that arise in real-time implementations: lookup tables, state variables, fixed vs. floating point, DC offset. • Setting up a timer for benchmarking.
Filter design	• Simple high-pass filter from previous exercise motivates better design. • Finite impulse response (FIR) vs. infinite impulse response (IIR), circular buffer.
Pitch shifting with granular synthesis	• Multiple lookup tables, tapering window (for smoothing discontinuities between processed blocks), state variables.

Table 1. Proposed curriculum with expected learning objectives.

opment. With all the material in Markdown files, the next choice is in deciding the platform for hosting the content. Free possibilities include GitHub Pages, GatsbyJS, Netlify, and GitBook. They all allow synchronization with a GitHub repository so changes and suggestions can be made by numerous collaborators and even students! At the moment, we use GitBook due to a few attractive features: simple layout, the ability to edit directly from the browser, and easy-to-use plug-ins, *e.g.* for embedding formulas and code.

Finally, our goal is to include the material as part of LCAV's ongoing MOOC offering in DSP [23, 24]; the class, free of charge and online since 2013, has totaled over 50,000 enrollments since its inception. We feel that a hands-on module would represent a particularly useful addition to the online curriculum especially for students who are seeking to expand their marketable skill set. The low price point of our reference hardware kit and the free availability of the associated software tools will allow students to experiment with applied DSP everywhere in the world.

5. INITIAL TEACHING EXPERIENCE

The first tryout of this applied DSP learning module took place during the Spring 2018 edition of the Bachelor course "Signal processing for communications" (COM-303) at EPFL [32]. Five practical sessions (see Table 1) were offered as optional ungraded coursework, mainly in the interest of receiving an initial feedback from participants. Each session lasted two hours, during which we would present relevant theory in the first (roughly) 30 minutes, and the students worked individually or in pairs for the remainder of the session. Out of 84 students registered for the course, 22 joined for the first exercise session; students were able to install the necessary software on Windows, MacOS, and Linux. For the second session only 14 students remained: wiring the components proved to be a true challenge for a majority of the students, mostly Computer Science and Communication Systems majors, as this type of work is not typical of their curriculum. In the third session nine students returned but only two of them were able to implement the effect on the microcontroller. This low success rate and the overall dropout rate motivated us to rethink our teaching approach.

Clearly, the difficulty in achieving a working prototype after a two-hour session was the major threat to motivation. To ameliorate things we added the Python simulation component to the implementation and learning pipeline, as discussed earlier. The additional benefit was that the simulation environment would allow students who could not complete the first module on the board to still participate in the subsequent sessions. Despite our additional effort, however, only five students attended the last class. Among possible concauses for the high dropout rate were certainly the fact that the class was optional and that the increasing workload towards the semester's end deterred students from continuing with the exercises. We are eagerly waiting to offer the material in our MOOC to receive a much more substantive amount of feedback from a diverse user base.

6. CONCLUSIONS

In this paper, we presented a pedagogical module designed to teach practical aspects of DSP, using an STM32 Nucleo board and freely-available and OS-independent software. The hardware component, in spite of its very low cost, still exposes students to industry-level tools. Moreover, the module is designed so that, even in the absence of hardware, the main lessons in real-world DSP can still be appreciated. The module has been offered experimentally at EPFL and will soon be available as part of our MOOC. The material is freely available to instructors worldwide, hoping to foster adoption and receive feedback from fellow DSP instructors.

7. REFERENCES

[1] "DSP Labs," https://lcav.gitbook.io/dsp-labs/.

[2] "DSP Labs (repository)," https://github.com/LCAV/dsp-labs.

[3] Jaldén Joakim, Xavier Casas Moreno, and Isaac Skog, "Using the Arduino Due for teaching digital signal processing," in *Proc. IEEE Int. Conf. Acoust. Speech, Signal Process.*, 2018, pp. 6468–6472.

[4] William J. Esposito, Fernando A. Mujica, Domingo G. Garcia, and Gregory T.A. Kovacs, "The Lab-In-A-Box project: An Arduino compatible signals and electronics teaching system," in *IEEE Signal Process. Signal Process. Educ. Work.*, 2015, pp. 301–306.

[5] Clark Hochgraf, "Using Arduino to teach digital signal processing," in *Proc. ASEE Northeast Sect. Conf.*, Atlanta, GA, USA, 2013.

[6] "TMS320C6748 DSP Development Kit (LCDK)," http://www.ti.com/tool/tmdslcdk6748.

[7] DongYuan Shi and Woon-Seng Gan, "Comparison of difference development kits and its suitability in signal processing education," in *Proc. IEEE Int. Conf. Acoust. Speech, Signal Process.*, 2016, pp. 6280–6284.

[8] "All FPGA Main Boards," https://altera-kits.terasic.com/.

[9] "Boards, Kits, and Modules," https://www.xilinx.com/products/boards-and-kits.html.

[10] "Raspberry Pi," https://www.raspberrypi.org/.

[11] "BeagleBone Black," https://beagleboard.org/black.

[12] "Wandboard," https://www.wandboard.org/.

[13] "Arduino Products," https://www.arduino.cc/en/Main/Products.

[14] "STM32 MCU Nucleo," https://www.st.com/en/evaluation-tools/stm32-mcu-nucleo.html.

[15] "TI LaunchPad™ development kits," http://www.ti.com/tools-software/launchpads/overview.html.

[16] M.G. Morrow and T.B. Welch, "winDSK: a Windows-based DSP demonstration and debugging program," in *Proc. IEEE Int. Conf. Acoust. Speech, Signal Process.*, 2000, pp. 3510–3513.

[17] "NUCLEO-F072RB," https://www.st.com/en/evaluation-tools/nucleo-f072rb.html.

[18] Philips Semiconductors, "I2S bus specification," https://www.sparkfun.com/datasheets/BreakoutBoards/I2SBUS.pdf, 1996.

[19] "Adafruit I2S MEMS Microphone Breakout - SPH0645LM4H," https://www.adafruit.com/product/3421.

[20] "Adafruit I2S Stereo Decoder - UDA1334A Breakout," https://www.adafruit.com/product/3678.

[21] "Premium Female/Female Jumper Wires - 40 x 6 inch," https://www.adafruit.com/product/266.

[22] "USB cable - 6 inch A/MiniB," https://www.adafruit.com/product/899.

[23] Thomas A. Baran, Richard G Baraniuk, Alan V. Oppenheim, Paolo Prandoni, and Martin Vetterli, "MOOC adventures in signal processing," *IEEE Signal Processing*, pp. 22, 62–83, 2016.

[24] "DSP MOOC on Coursera," https://www.coursera.org/learn/dsp.

[25] "System Workbench for STM32: free IDE on Windows, Linux and OS X," https://www.st.com/en/development-tools/sw4stm32.html.

[26] "Code Composer Studio (CCS) Integrated Development Environment (IDE)," http://www.ti.com/tool/CCSTUDIO.

[27] "STM32CubeMX," https://www.st.com/en/development-tools/stm32cubemx.html.

[28] Matthias Geier, "Play and Record Sound with Python," https://python-sounddevice.readthedocs.io, 2018.

[29] "Real-Time Voice Transformers (Jupyter Notebook)," http://nbviewer.jupyter.org/github/prandoni/COM303/blob/master/voice_transformer/voicetrans.ipynb.

[30] "Principles and Techniques of Data Science," https://www.textbook.ds100.org/.

[31] "Principles and Techniques of Data Science (repository)," https://github.com/DS-100/textbook.

[32] Paolo Prandoni and Martin Vetterli, *Signal Processing for Communications*, EPFL Press, CRC, Lausanne, 2008.