USING THE ARDUINO DUE FOR TEACHING DIGITAL SIGNAL PROCESSING

Joakim Jaldén, Xavier Casas Moreno*

Dept. of Information Science and Engineering,
KTH Royal Institute of Technology, Sweden.
{jalden,xaviercm}@kth.se

Isaac Skog

Dept. of Electrical Engineering,
Linköping University, Sweden.
isaac.skog@liu.se

ABSTRACT

This paper describes an Arduino Due based platform for digital signal processing (DSP) education. The platform consists of an in-house developed shield for robust interfacing with analog audio signals and user inputs, and an off-the-shelf Arduino Due that executes the students' DSP code. This combination enables direct use of the Arduino integrated development environment (IDE), with its low barrier to entry for students, its low maintenance need and cross platform interoperability, and its large user base. Relevant hardware and software features of the platform are discussed throughout, as are design choices made in relation to learning objectives, and the planned use of the platform in our own DSP course.

***Index Terms*—** Teaching, digital signal processing, Arduino

1. INTRODUCTION

Digital signal processing (DSP) is a subject that spans from the theoretical aspects of systems theory – including signals and systems concepts such as linearity, time invariance, sampling, and transform theory – to the practical details of low level programming – including timing, buffer handling, and computational limitations. Different DSP courses at different universities thus vary widely with respect to the intended learning outcomes of the course. However, even more theory-focused courses tend to have some associated lab component.

The DSP education within the electrical engineering program at the KTH Royal Institute of Technology, Sweden, is mainly theoretical but has historically included projects implemented in Matlab and a shorter lab session with Texas Instrument TMS320C7613 DSP boards. The Matlab project was designed to illustrate the usefulness of the theory for solving a concrete signal processing task, and at the same time reinforce the learning of course content. The lab session was mainly intended to inspire students by demonstrating a real-time DSP system, and it relied solely on the interaction with input and output signals of a preprogrammed DSP. The reasons for using a preprogrammed platform were: to spare the student the time consuming task of learning a complex DSP integrated development environment (IDE); to avoid the need to maintain the IDE software on the lab computers; and, above all, to avoid certain stability issues of the employed IDE. However, many students explicitly asked for DSP programming as part of the course. The absence of hands on experience with real-time DSP programming also made it challenging to successfully teach concepts related to computational complexity and hardware implementation.

To address the aforementioned drawbacks, and to enable some programming within the course, we set out to pick a new lab platform for the course. Our choice was the Arduino Due[1] board together with an in-house developed analog shield, shown in Fig. 1. The shield provides a robust and easy-to-use analog interface, while all DSP functionalities are implemented on the off-the-shelf Arduino Due board. We see a number of benefits of this approach, as outlined next.

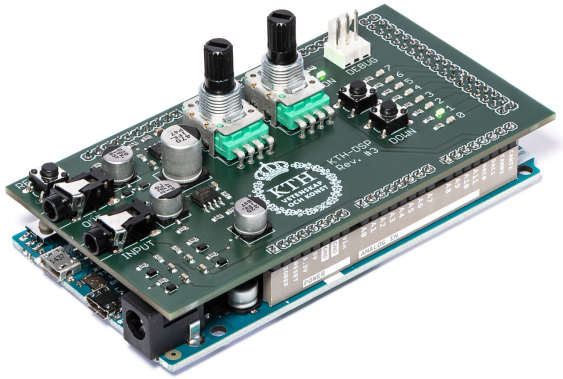

Fig. 1. An Arduino Due board with the developed shield mounted on top. The shield enables easy and safe interfacing of analog inputs and outputs, controlling of program settings via potentiometers and buttons, and measuring (via a dedicated output pin) of the number of clock cycles needed to complete a DSP task.

- The Arduino platform is a well-known platform with a massive user base. Programming is typically done in a single C or C++ file referred to as a sketch, in an open IDE, with roots in the Processing project at MIT Media Lab, and constructed with ease of use in mind. Arduinos are also frequently used in the teaching of other subjects in EE, see e.g. [1]. This means that it is likely that students are already familiar with the platform. These two reasons combined makes for a low barrier to entry.
- The large user base for the Arduino platform promotes the long term viability of the platform, and support across all major operating systems [2]. This also minimizes the maintenance costs of keeping and updating the IDE and drivers.
- The Arduino board allows for easy access to low level programming. As it does not have an operating system, it allows

*Funding for the project was provided by the School or Electrical Engineering at the KTH Royal Institute of Technology.

[1] General information on the range of Arduino platforms is available at https://www.arduino.cc, and specific information on the Arduino Due can be found at https://store.arduino.cc/arduino-due.

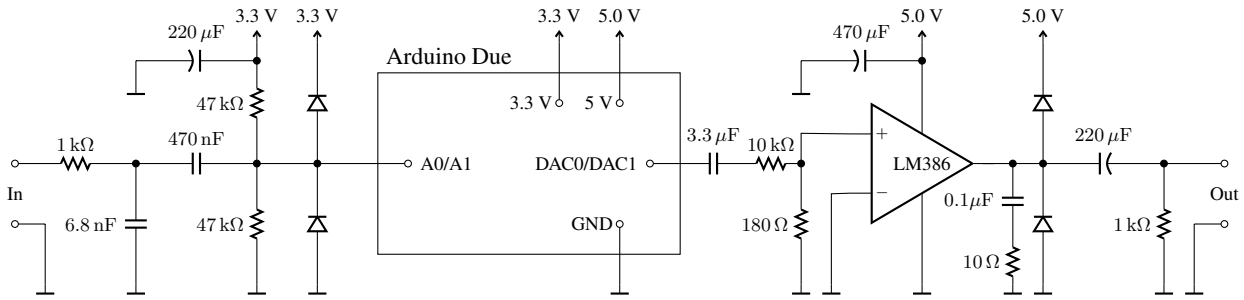

Fig. 2. Input and output circuit schematic of the developed shield. The schematic illustrate one of the two audio channels.

for direct and predicable access to for instance the ADCs and DACs, and easy configuration of timed interrupts for scheduling DSP tasks. This makes the step from theory, as presented in standard textbooks such as [3], to practice much smaller.

- The Arduino Due features a 32-bit ARM Cortex-M3 micro controller, the AT91SAM3X8E (SAM3X), clocked at 84 MHz. This enables the software implementation of a reasonable range of typical real-time signal processing tasks, in particular for audio signal processing, without at the same time being overpowered. The use of audio is particularly suitable for education, as students are intuitively familiar with this type of signals.
- The relatively low cost per unit of the platform makes it feasible to provide each student with an individual sample of the platform for the duration of the course. This enables experimentation without the need of a dedicated lab environment, e.g., to test real-time code related to specific lecture topics.

There exist a few other Arduino based approaches to DSP education. Examples include the lab-in-a-box platform [4] developed by the Kovacs lab at Stanford University, the Arduino DSP lab developed by C. Hochgraf at Norwich University [5], and the Analog Discovery board in conjuncture with the less powerful Arduino Uno or the Digilent ChipKIT UNO32 [6]. Our approach differs from [5], which use the simpler Arduino Uno 8-bit ATmega328P microcontroller board clocked at 16 MHz. While such a platform really tests the students understanding of concepts such as quantization, limit cycles, and numerical overflow [3], it is effectively impossible to use for audio signal processing. The platform in [4] also makes use of an Arduino Uno, but mainly as an interface to a Texas Instrument TMS320C5535 DSP and an IIC3204 audio codec. While this makes that platform highly suitable for audio signal processing, it required the development of a separate IDE for the DSP programming with the associated software development and maintenance costs. As noted in [7], one of the issues with that platform when used in a DSP course at Stanford in 2015 was the lack of Linux support for the IDE at that time. Such issues are largely avoided when relying on the standard off-the-shelf Arduino IDE. The setup in [6] provides capabilities most similar to the one presented herein, albeit the use of the more powerful UNO32 currently requires a third party add-on to the Arduino IDE, and still offer lower computational power than the Arduino Due.

There exist other platforms that explicitly target DSP education, suitable for real-time audio signal processing at attractive price points. These include the combination of the ST STM32F4 Discovery Board with the Wolfson Audio Card and the Cypress FM4 Starter Kit, both included in the Arm University Program[2]. These cards come with educational material as part of the Arm University Program, and are based on more capable Arm Cortex-M4 microcontrollers. The user bases of these platforms are however still small compared to the Arduino community which make the long term support of the platforms and the IDE less certain. They also currently lack native support in the Arduino IDE. A comparison of various development platforms and their suitability for real-time DSP education at different levels can be found in [8], and an excellent overview of recent ICASSP papers on real-time hardware for DSP education is provided in [9].

The installation of the Arduino IDE, as well as the compilation and deployment of code over a USB interface, is extremely simple. IDE installer packages are available for Windows, Mac OS X, and Linux. The interaction of the DSP code with real world signals can be tested by connecting a common audio source such as a smartphone or laptop, and by listening to the processed output. This means that a student can experience real-time signal processing within a few minutes after being provided the platform, and without the need for access to a dedicated lab environment.

2. THE PLATFORM

This section describes the hardware and software platform, and how the design choices made relates to the learning activities. Explicit learning activities are then further discussed in Section 3.

2.1. The shield

Since we were unable to find a suitable shield for the Arduino Due which included the functionality we were looking for, we decided to develop our own in-house. The primary purpose of the hardware shield is to provide a robust and easy way of connecting analog signals to the Arduino for DSP. We chose to use standard 3.5mm stereo audio connectors for the input and the output to simplify the use of smartphones and laptops as signal sources, and to allow the use of standard headphones as an easy way to listen to the output signal. We also made design choices to minimize the risk of damaging the Arduino by e.g., accidentally connecting a high voltage signal source to the input or the output connector, or shorting either connector to ground. Fig. 1 shows the actual hardware shield that implements

[2]For information on both the Arm university program, as well as the specific platforms, see `https://developer.arm.com/academia/arm-university-program/for-educators/digital-signal-processing`.

two analog channels, and Fig. 2 shows a schematic of the input and output circuitry.

The Arduino Due board is equipped with 12 analog inputs (ADCs) and 2 analog outputs (DACs), all with a 12 bit resolution. This reduces the amount of additional analog components needed to realize the full analog-digital-analog chain. The ADCs operate in the range 0 to 3.3V, and ground referenced input signals need to have a 1.65V DC offset added to them before they are sampled. The input circuit in Fig. 2 adds this offset and implement a second order band pass filter with pass-band gain 0.96 and lower and upper 3 dB cut-off frequencies at 13.6 Hz and 24.7 kHz, respectively. This provides DC decoupling from the source, and a sufficient suppression of high frequency noise that was otherwise aliased into the audible range by the ADC. The Arduino can amplify the analog input (around the 1.65V DC offset) by a factor of 1, 2, or 4; the gain is configurable via software. With a gain of 4 a standard consumer 0.894 V peak-to-peak audio signal just about fills the entire ADC input range.

The DACs of the Arduino Due provide simple zero-order hold digital to analog conversion. The DACs are capable of delivering 15 mA and can directly drive a high impedance input, such as a line-in audio port. The only thing needed is a decoupling capacitor to remove the output DC offset. However, the DACs are not capable of directly driving low impedance loads, such as passive speakers or headphones. To make the setup more flexible for the students we therefore decided to add an on board power amplifier based on the commonly used LM386 audio amplifier circuit. We fixed the gain so that the output is loud, but not excessively so, when driving a set of standard headphones with a signal that occupy the full DAC output range. The 220 μF output capacitor shown in Fig. 2 implements a high pass filter with a cutoff frequency of 25.9 Hz when driving a 32 Ω load. A 1 kΩ resistor at the output is included to quickly remove DC charges in the output capacitor when using a high impedance load such as an oscilloscope.

When using the board it is sometimes convenient to use signal generators for inputs. The signal generators in our lab can be set to generate signals with up to 20 V peak-to-peak amplitude and could thus potentially damage the boards if configured and connected incorrectly. We therefore added diode clamps to the input and output to provide protection against over-voltage. We also found it necessary to include voltage stabilizing capacitors to the input and output to reduce ringing in the analog signals that occurred on some copies of the Ardunio boards when combined with certain power sources.

Finally, we connected: two of the analog inputs to potentiometers; two digital inputs to push buttons; and 8 digital outputs to LEDs labelled 0 to 7. These additional inputs and outputs can be read and set by software, which provides additional flexibility. Further, we added two software configurable digital general-purpose input/output (GPIO) pins which, e.g., can be used to output clock pulses and signal the start or stop of interrupt events. Specific use cases of these capabilities are discussed in Section 3. Finally, the audio jacks, the potentiometers, and the push-buttons are all chosen as through-hole components, to increase the physical robustness of the shield. The remaining components are surface mounted to simplify the production of the shields.

For use in our DSP course at KTH we have manufactured 35 shields, at a per unit cost of approximately US$55 using a printed circuit assembly (PCA) service. The Arduino Due, and USB and audio cables, add another US$40 per lab kit. Both prices can be further reduced when ordering larger quantities. Nevertheless, at this price point we forecast being able to provide each student with an individual kit for the duration of the course.

2.2. The Arduino Due

As stated in Section 1, the Arduino Due is built around a SAM3X 32-bit ARM Cortex-M3 micro controller, clocked at 84 MHz. In our current implementation, we use one of the SAM3X timers to trigger an interrupt every 1750 clock cycles for reading from ADCs and writing to the DACs, yielding a sampling rate of 48 kHz[3]. While handling input and output via an interrupt in this fashion is not ideal for high audio fidelity due to sample jitter, we perceive it to be of sufficient quality for educational purposes. In fact, the biggest disturbance does not seem to be jitter but rather a noise introduced by the DACs, even when only used to deliver a static DC signal.

The 1750 clock cycles available per sample at the 48 kHz sample rate is sufficient for a reasonable amount of real-time signal processing, as most relevant Cortex-M3 instructions of the SAM3X take either one or two clock cycles. During our tests, we have been able to realize FIR filters of more than 100 taps using convolution in the time domain, and over 200 taps when implemented in the frequency domain using overlap save with a 1024-point FFT[4].

The lack of an operating system, and single threaded execution of code on the Arduino, is helpful when learning DSP implementation. Although setting up the interrupt handling is not straightforward, it is relatively easy to visualize when a particular piece of code is executed, and thus to relate the results to signals and systems theory. Further, any of the analog or digital inputs can be read at any time in the code, making it easy for students to, e.g., use the two potentiometer values as parameters in their implementation. It is also straightforward to change the initial code to, say, sample at other rates or to run the ADCs and DACs at different rates.

The biggest deficit of the platform is in our view the lack of a floating point unit in the SAM3X microcontroller. While fixed-point implementations is an important topic taught in many DSP courses, and while 32-bit computations are not exactly limiting, it would sometimes be helpful with the abstraction of infinite precision real valued computations. The Arduino IDE also lacks a proper debugger. However, this is to some extent remedied by the built-in serial monitor that can be used for so-called `printf`-debugging for smaller projects.

3. FROM THEORY TO PRACTICE

In standard textbooks, a DSP system for processing of a continuous time signal, is often presented as in Fig. 3 [10]. A continuous-time (analog) signal is sampled, and possibly quantized, to produce a discrete-time (digital) signal, which undergoes some processing and is then converted back to a time-continuous signal via a process such as pulse amplitude modulation. Although not explicitly expressed, this view tends to be interpreted in the way that samples are processed one by one in the same order they are delivered by the sampling circuit. However, this is not the way most DSP systems are structured. Typically, the implementation of the ADC relies on direct memory access (DMA), where samples are accumulated in a buffer, followed by processing of complete signal blocks [11, pp. 223-224]. The DAC is implemented similarly. The simple model in Fig. 3, where the discrete-time samples are though of as time-synchronized

[3]For ADC, DAC and interrupt handling, we use a slightly modified version of a code made available by user MarkT in the Arduino forum, at `https://forum.arduino.cc/index.php?topic=205096.0`.

[4]In our reference implementations, we apply a single real-valued FIR filter to both the left and right audio channel. All code is written in C/C++, and complied using the standard optimization setting `-Os` for the gcc/4.8.3 compiler provided with a basic installation of the Arduino 1.8.3 IDE.

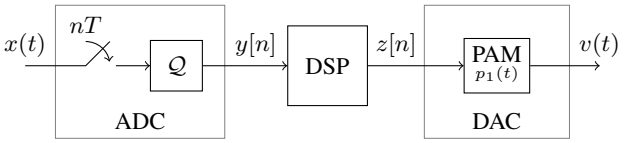

Fig. 3. A block diagram of a DSP system for processing of continuous time signals as it is commonly presented in textbooks.

with the analog signal according to $y[n] = x(nT)$, is however helpful when interpreting and understanding the mathematics. Therefore, from a learning perspective, a benefit of the proposed platform is that both the sample-by-sample and block based paradigms can easily be implemented. In what follows we illustrate how we aim to use these capabilities in teaching the DSP subject. These learning activities are similar to those described in [6], albeit using a different platform.

3.1. Sample-by-sample processing

It is often natural to start by considering sample-by-sample processing, which is conceptually closest to how the theory is usually presented. Placing a per-sample DSP code inside the interrupt routine used for reading from the ADC and writing to the DAC provides the intuitive time-synchronization between the continuous-time and the discrete-time signals. Standard IIR and FIR filters are easily implemented using a `for` loop. It is also relatively simple to illustrate and analyze the effects of limited precision (quantization), as well as topics such as overflow and limit-cycles [3].

Just as potentiometers can be used to change the characteristics of an analog filter, we can here use the them to change the coefficients of, say, a second order AR filter or a first order ARMA filter, and see the resulting changes in real-time. The digital buttons can be used to change the type of filter used, allowing instantaneous back and forth comparisons of different filter types without any intermediate platform reprogramming.

Sample-by-sample processing also provides an illustrative entry point to a discussion of computational complexity and available resources, as the processing for each sample needs to be completed before the next sample arrives. As a tool to assist this discussion the shield have one GPIO pin that can be configured to output the clock pulses of the microcontroller and a second GPIO pin that can be configured to go high (low) at the start (end) of a selected code segment. Thus, by connecting a pulse counter or an oscilloscope to these pins, the number of clock cycles needed to complete a code segment can be directly measured. Similar measurements can be obtained by reading the clock register at the start and end of the code segment, but this is not quite as illustrative in a lab setting.

3.2. Sampling and multi-rate signal processing

The sample-by-sample processing framework, in combination with the ease with which the ADCs and DAC update rates can be configured, makes it easy to explore aliasing and reconstruction, as well as to implement multi-rate signal processing applications. The analog antialias filter has, as noted, a fixed 3 dB cutoff frequency of 24.7 kHz, and only drop off at 20 dB per decade. Aliasing of signals above 24 kHz is therefore clearly visible in a spectrum measurement, although this does not seriously degrade the audio quality when listening to, e.g., music. Reducing the sampling rate below 48 kHz, however, makes the aliasing effect clearly audible as well. Finally, reading from the ADCs and writing to the DACs at different rates enables learning activities related to discrete-time rate conversion.

3.3. Block based processing

The extension to block based signal processing is straightforward, and can be done by modifying the code used for learning activities described in Section 3.1 so that the interrupt is just used for writing to and reading from input and output buffers. Code to be executed per block can be placed in the main loop of the sketch and initiated via a flag indicating when enough samples have been collected to fill a buffer. This can be used to simulate sampling via DMA.

If the samples are collected in a ring buffer it is also easy to implementing overlapping blocks of samples, as needed for example when implementing overlap save filtering using an FFT [10]. The complexity benefits of processing in the frequency domain can be illustrated using the clock counting procedure described in Section 3.1. Implementation of the FFT in fixed point arithmetics on the SAM3X is somewhat challenging, and could be a learning objective in itself, but is certainly feasible as shown by our test implementations. It is also possible to demonstrate that, e.g., longer FIR filters can be implemented using frequency domain techniques rather than straightforward convolution in the time domain.

4. SUMMARY

This paper has described an Arduino Due based platform for teaching digital signal processing, and the development of an analog shield for interfacing with audio signals. By using the Arduino Due for handling the DSP coding, it is possible to provide a student friendly DSP learning environment with a low barrier to entry. We see the use of the unmodified Arduino IDE as a key feature of the proposed platform, both when viewed from the students' perspective, as well as from the teacher's perspective in terms of platform maintainability and longevity. The platform has some drawbacks, such as low fidelity, sample jitter, lack of a floating point unit, and the absence of a debugger. Nonetheless, we believe that these deficits are overshadowed when teaching DSP by the ease of use of the platform. In particular, the Arduino Due provides an excellent platform for DSP learning activities such as: real-time filter implementation, both in the time and frequency domain; quantization and fixed point implementations; sampling, reconstruction, and aliasing; and computational complexity. Last but not least, with the low cost of the platform it is feasible to provide each student with an individual copy. This facilitates the use of hands on DSP learning activities throughout the course, in the same spirit as various other lab-in-a-box initiatives in STEM subjects. The low cost also makes the platform a viable option for use in online courses.

5. EPILOGUE

Since this paper was written, 35 copies of the platform was created and used for a lab activity in the DSP course at KTH in the fall of 2017. Overall the lab activity was very well received by the students. In particular, the use of the GPIO pins for the timing of frequency domain filtering using overlap-save was very useful in the teaching of these methods. The students' usage of the platform, and installation of the IDE, was largely without problems, although code optimization compiler directives were found to be less robust across software environments than expected. Lab instructions can be provided upon request by the first author.

6. REFERENCES

[1] J. Sarik and I. Kymissis, "Lab kits using the Arduino prototyping platform," in *Proc .of ASEE/IEEE Frontiers in Education Conference (FIE)*, Washington, DC, Oct. 2010.

[2] A. A. Galadima, "Arduino as a learning tool," in *Proc. of 11th International Conference on Electronics, Computer and Computation (ICECCO)*, Abuja, Nigeria, Sept. 2014.

[3] J. G. Proakis and D. G. Manolakis, *Digital Signal Processing – Principles, Algorithms, and Applications*, Pearson Education (US), 2006.

[4] W. J. Esposito, F. A. Mujica, D. G. Garcia, and G. T. A. Kovacs, "The lab-in-a-box project: An Arduino compatible signals and electronics teaching system," in *Proc. of IEEE Signal Processing and Signal Processing Education Workshop (SP/SPE)*, Salt Lake City, UT, Aug. 2015.

[5] C. Hochgraf, "Using Arduino to teach digital signal processing," in *Proc. of ASEE Northeast Section Conference*, Northfield, VT, Mar. 2013.

[6] M. A. Wickert, "Real-time DSP basics using Arduino and the analog shield SAR codec board," in *Proc. of IEEE Signal Processing and Signal Processing Education Workshop (SP/SPE)*, Salt Lake City, UT, Aug. 2015, pp. 59–64.

[7] F. A. Mujica, W. J. Esposito, A. Gonzalez, C. R. Qi, C. Vassos, M. Wieman, R. Wilcox, G. T. A. Kovacs, and R. W. Schafer, "Teaching digital signal processing with Stanford's lab-in-a-box," in *Proc. of IEEE Signal Processing and Signal Processing Education Workshop (SP/SPE)*, Salt Lake City, UT, Aug. 2015.

[8] D. Y. Shi and W. S. Gan, "Comparison of different development kits and its suitability in signal processing education," in *Proc. of IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Shanghai, China, Mar. 2016, pp. 6280–6284.

[9] C. H. G. Wright, T. B. Welch, and M. G Morrow, "Reinforcing signal processing theory using real-time hardware," in *Proc. of IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, New Orleans, LA, USA, Mar. 2017.

[10] S.K. Mitra, *Digital Signal Processing: A Computer-Based Approach*, Mc Graw Hill, 1998.

[11] A. Bateman and I. Paterson-Stephens, *The DSP Handbook: Algorithms, Applications and Design Techniques*, Prentice Hall, 2002.