A Low-Cost Platform for Teaching Real-Time Digital Signal Processing

Dr. Joseph P. Hoffbeck, University of Portland

Joseph P. Hoffbeck is a Professor of Electrical Engineering at the University of Portland in Portland, Oregon. He has a Ph.D. from Purdue University, West Lafayette, Indiana. He previously worked with digital cell phone systems at Lucent Technologies.

A Low-cost Platform for Teaching Real-time Digital Signal Processing

Abstract

The STM32F746G-DISCO Discovery kit from STMicroelectronics was chosen as the hardware platform to support a junior/senior electrical engineering real-time digital signal processing (DSP) course. The board features a STM32F746NG microcontroller with an Arm Cortex-M7 core with a floating-point unit and DSP instructions. The board includes a debugger, stereo audio line-in/line-out, two built-in digital microphones, speaker output, a built-in LCD touchscreen, a user pushbutton, and a user-controlled LED. The board can be programmed in C and assembly using several programs including STM32CubeIDE, which is a free program available for Windows, macOS, and Linux.

The course is an elective in the electrical engineering program and is taken by juniors and seniors. The prerequisites for the course are signals and systems and data structures. The course is a lecture course with no accompanying laboratory, but since the board is relatively inexpensive, a board was loaned to each student for the duration of the semester. So, students had full access to the board, and they wrote and tested DSP algorithms as homework assignments.

The author wrote starter code that demonstrates a few real-time DSP algorithms (stereo passthrough, lowpass filter, highpass filter, and reverb) and serves as a starting point for the students when they write their own programs. The starter code includes several features that make working with the board more convenient such as a menu that makes it easy to switch between different algorithms and to select the input source (either line-in or the built-in microphones). It displays an input level meter, which is helpful for setting the input gain. It has mechanisms for setting the input and output gain and measuring the gain of a filter. It also includes an indicator that alerts the user if their DSP algorithm is taking too long and is violating the real-time schedule. The starter code is available for free from the author.

The course assignments include exploring the demonstration algorithms and writing and testing several real-time DSP algorithms including mono passthrough, record and playback, generate sinusoids, finite impulse response (FIR) filters, infinite impulse response (IIR) filters, and Discrete Fourier Transform (DFT). The course also includes a project where each student selects an algorithm to implement and test on their own.

This paper describes the advantages and disadvantages of using the STM32F746G-DISCO Discovery kit to teach real-time DSP. The features of the starter code are described along with how they support the students' learning. The course learning objectives and assignments are discussed. A student survey was given to assess the effectiveness of using the board to teach real-time DSP, and the results of the survey are discussed in the paper.

Background

Traditionally many undergraduate lecture courses in electrical engineering have been paired with laboratory courses to give students hands-on experience and to supplement and reinforce the theoretical material taught in lecture courses. However, in order to overcome the cost and time constraints of laboratory courses, many faculty members have moved to a mobile studio pedagogy where the students are equipped with relatively low-cost laboratory equipment that allows them to perform experiments and measurements outside the laboratory. This approach has been implemented in a variety of individual courses, including introductory first-year courses [1], electronics [2], digital design [3], and communication systems [4], [5], [6]. Some authors have reported on the use of mobile studio pedagogy in multiple courses [7], [8], and some schools have implemented the technique throughout the electrical engineering curriculum [9], [10], [11]. A helpful discussion of the advantages and disadvantages of various logistic options is presented in [9].

Courses in real-time DSP add valuable practical experience that complements DSP theory [12]. Many different hardware options exist for teaching real-time DSP systems including development kits, single-board computers, phones/tablets, and laptops/PCs. Development kits capable of performing DSP are available from companies such as Texas Instruments, STMicroelectronics, National Instruments, AMD/Xilinx, Intel, and NXP [13]. The use of development kits to teach real-time DSP has been described by several authors [14], [15], [16]. The use of single-board computers such as Arduino, Raspberry Pi, Intel Galileo, BeagleBoard, and many others has been studied [17], [18]. It is also possible to use cellphones and tablets to teach DSP [19] and to implement real-time DSP on laptops or PCs [20].

The choice of a hardware platform for education is often sensitive to the price of the hardware and software. The cost is especially salient if the course employs a mobile studio pedagogy where each student has full access to the hardware and software to facilitate working at home or in a classroom instead of in a laboratory.

Multiple methods exist for programming these devices, including MATLAB/Simulink, Octave, Python, and C/C++. The choice of programming environments is often driven by the learning objectives of the course and the skills and knowledge of the students. High-level languages like MATLAB/Simulink and Python allow the instructor to focus on the DSP theory whereas programming in a low-level language such as C/C++ facilitates learning the concepts and skills needed to implement the low-level details required in a real-time DSP system.

This paper describes the goals and organization of the author's real-time DSP course, the choice of hardware and software that was selected to support the course goals, and the results of a student survey to assess the effectiveness of the choice of hardware.

Course Structure

The course is a 3-credit hour junior/senior lecture-based elective taken by electrical engineering students. The prerequisites are Signals and Systems and Data Structures (using C Language). The class met for 55 minutes three times a week for a semester (about 14 weeks). Since there was no accompanying laboratory class, the course was based on a mobile studio pedagogy where the students implemented and tested real-time DSP algorithms as homework assignments. No textbook was required for the course, but short reading assignments were assigned from Wikipedia and other sources.

The main learning objective for the course was to enable the students to be able to write C language programs to implement various DSP algorithms that run in real-time on a DSP demonstration board. The focus of the course was on floating-point algorithms, but some time was devoted to fixed-point arithmetic as well. Topics included introduction to DSP and DSP hardware, C language and compiler, signal generation, FIR filters, IIR filters, Discrete Fourier Transform (DFT) and Fast Fourier Transform (FFT), fixed-point processing, and applications of DSP.

Although the list of topics in this real-time DSP course was similar to a typical non-real-time DSP course, there are important differences. A real-time system must be designed to guarantee that the output is available within the required time frame every time or else the system malfunctions. In the case of an audio signal, if the output is not available on time, the output audio signal may have pops, clicks, or other audible distortion. In addition, since many real-time systems are designed to be low-cost or battery-powered, they often use much less powerful processors and have much less available memory compared to a laptop or computer. Being able to successfully implement real-time algorithms with these constraints requires different skills than implementing these algorithms in a high-level language like MATLAB or Python. Therefore this class, which was based on C language, included discussions of low-level details rarely found in a theoretical DSP course such as direct memory access (DMA), interrupts, double buffering techniques, circular buffers, effect of buffer size on signal delay, overflow issues, and the effect of filter order on computation time. Covering these low-level details means there is much less time for the theory of DSP, but there is a separate MATLAB-based course at the author's university that covers the theory of DSP.

In the first part of the course, there is one homework assignment due each week, and the last part of the course is dedicated to a project where the students write real-time DSP programs. Students can choose the topic of the project subject to instructor approval. Projects included a graphic equalizer, guitar tuner, waveform generator, audio player, DTMF generation/detection, midi synthesizer, and voice pitch changer. Each student was required to meet individually with the instructor to demonstrate the project and to answer questions about the project.

This course used a couple of the best practices recommended in [9]: utilization of a common platform and use of mobile platforms for homework assignments. The hardware chosen for this course was also employed in the embedded systems design course, which was convenient for those students who took both courses and reduced the overall cost to the university. Since the

students had full access to the hardware during the semester, it was used for the homework assignments in both courses. Instead of having the students purchase the hardware as was recommended in [9], the author used department funds to purchase the boards, which were loaned to the students for the duration of the semester. Unlike the approach in [9] where student support was provided by a Technical Support Center, technical support was provided by the author who swapped out damaged boards and assisted students with software issues. Fortunately, there were only a few hardware and software problems, so this arrangement was feasible.

Hardware

The choice of hardware for the course was driven by the desire to have a board that is inexpensive enough that the school could buy a board to loan to each student or else the students could purchase their own boards. An inexpensive board allows the students to have full access to their board throughout the semester and to work outside of a laboratory at convenient times. It also opens the possibility to include hands-on exercises during lectures, although the author did not take advantage of this possibility.

The author looked for a board with a stereo line-in port for input audio signals and a line-out/headphone output to monitor the output signals. Line-in input is required because it is difficult to control the input level and background noise with microphone input. It is desirable to have a microphone in addition to the line-in input. The board was required to support commercial quality audio with a sampling rate of at least 44.1 kHz (to support the standard 20 Hz – 20 kHz audio range) and at least 16-bit analog-to-digital and digital-to-analog converters. The processor on the board needed to be fast enough to implement standard DSP algorithms (signal generators, filters, FFT, etc.) running in real-time using floating-point arithmetic. A touch screen would be a bonus for implementing a convenient user interface. Table 1 contains a short summary of some of the features on some currently available boards.

Table 1: Comparison of Some Hardware Platforms

Feature	STM32F746G-DISCO	BeagleBoard Black	Raspberry Pi 3 Model B+
Stereo Line-in/Line-Out	Yes	No	No
Headphone output	Yes	No	No
Microphones	Yes	No	No
Touch Screen	Yes	No	No
Processor Speed	200 MHz	1 GHz	1.4 GHz
Approximate Cost	$55	$63	$35

The STM32F746G-DISCO Discovery kit from STMicroelectronics was selected for use in the course (see Figure 1). It is a relatively low-cost board (about $55) that supports sampling rates up to 48 kHz with a stereo 16-bit codec. It has stereo line-in and line-out/headphone ports, two built-in digital microphones, as well as speaker outputs. It has an LCD touch screen (480 x 272 pixels). It features an STM32F746NGH6 Arm Cortex-M7 processor that runs at 200 MHz with a

hardware floating point unit and support for DSP instructions. It has 1 Mbyte of flash memory and 340 KByte RAM [21].

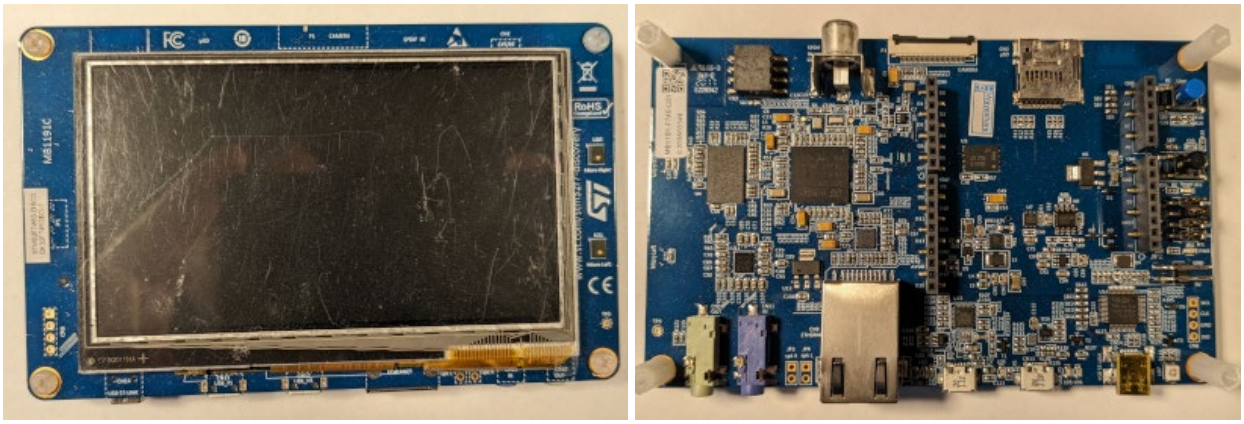

Figure 1: STM32F746G-DISCO Discovery kit (front and back)

The processor has several hardware features that support DSP algorithms such as a Harvard Architecture that allows it to transfer data between the CPU and memory in parallel with math instructions, support for two simultaneous multiply and accumulate (MAC) instructions with 8-bit or 16-bit data, zero overhead looping, and support for the CMSIS DSP library [22]. However, the processor lacks a couple of the features needed to be considered a true digital signal processor such as an accumulator with guard bits and circular and bit-reversed addressing [23]. The processor was easily capable of running the real-time DSP algorithms used in the homework assignments and in the students' projects.

The typical hardware setup used a cellphone as the audio source and earbuds or headphones to listen to the output (see Figure 2). An adaptor is required for cellphones that do not have an analog line-out/headphone port.

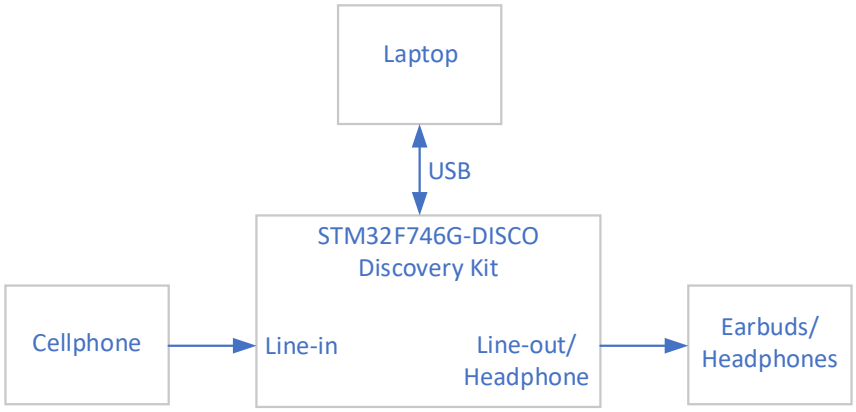

Figure 2: Typical hardware setup using a cellphone as the audio source

IDE

Several integrated development environments (IDE) can be used to program the board such as Keil MDK-ARM, IAR EWARM, GCC-based IDEs, ARM Mbed online, and STM32CubeIDE. The author chose to use STM32CubeIDE from STMicroelectronics because it is free, the code size is not limited, and it supports Windows, Mac, and Linux [24]. STM32CubeIDE (see Figure 3), is based on Eclipse and has full support for writing, debugging, and downloading programs (text editor, breakpoints, memory browser, variable viewer, register viewer, etc.).

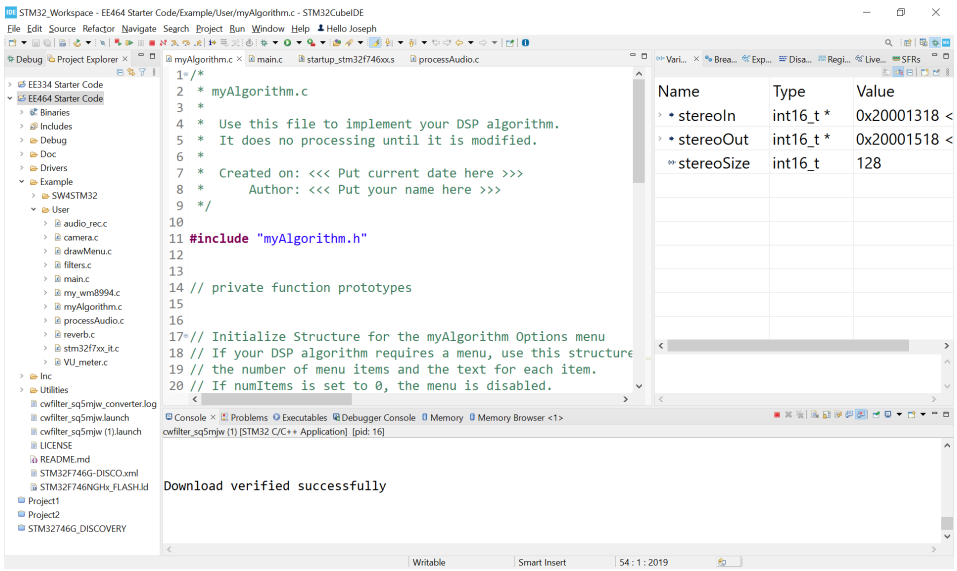

Figure 3: STM32CubeIDE

Starter Code

The students were provided with starter code that demonstrated a few DSP algorithms (see Figure 4). The starter also code implemented the low-level details needed for the passthrough algorithm, including initializing the processor, codec, DMA, and interrupts, as well as detecting when the input buffer is full and copying the input buffer to the output buffer.

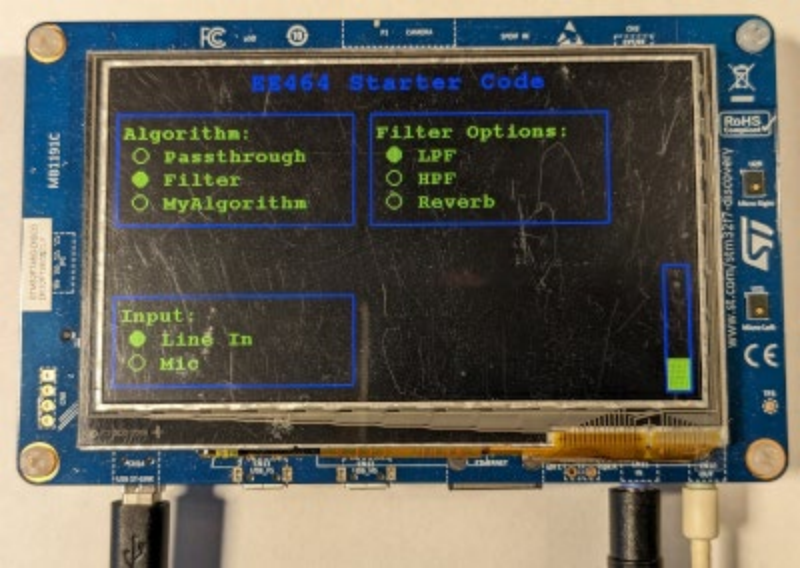

Figure 4: Starter code menu system and input level meter (in lower-right hand corner)

In addition to passthrough, the starter code also implemented a few example DSP algorithms (lowpass filter, highpass filter, and reverb) as well as a menu system to allow students to easily switch between these algorithms and passthrough. Having a quick and easy way to switch between passthrough and an algorithm was especially helpful in order to hear the possibly subtle difference that a DSP algorithm makes. The touch screen and menu system made switching back and forth very convenient.

The starter code also provided a menu to switch the input source between line-in and the built-in digital microphones. An input level meter was added to display the current input signal level, which was needed to set the input gain so that it was high enough to avoid excessive quantization error yet low enough to avoid clipping. The input and output gain on the codec was set while the program is running using the Live Expressions View in STM32CubeIDE to change the global variables InputLevel and OutputLevel. The starter code was also programmed to display a red circle in the upper-right hand corner if the student's algorithm took too long and broke the real-time schedule.

Since the students programmed the board at home instead of in a laboratory, they did not have access to an oscilloscope to measure the gain of filters. However, the starter code calculated the gain of the current algorithm based on the difference in signal amplitudes between the input and output buffers. Therefore, in order to measure the gain of a filter, the students used their cellphones to generate an input sinusoid at a given frequency and examined the global variable Gain using the Live Expressions View.

The starter code was based on a demonstration program that implemented passthrough and several real-time audio filters [25]. That demonstration program is free and is released under the GNU General Public License from the Free Software Foundation. Since the demonstration program was written using System Workbench for STM32, the author imported it to STM32CubeIDE, and modified it in several ways. Support was added for the built-in digital

microphones and a menu was created to switch the input between the microphones and the line-in port. The LPF, HPF, and reverb demonstration algorithms were implemented, as well as a menu to switch between them. A menu option called MyAlgorithm was created that switched to the algorithm that the student wrote (see Figure 4 above). A header file was set up to make it easy for the students to change the audio buffer size, the sampling rate, and the initial input and output gain. The input level meter was added along with the indicator that shows if the algorithm takes too long for the real-time schedule. Also, support for printf was added so that messages could be printed to the LCD screen for debugging purposes.

The starter code is available from the author for free (send email to hoffbeck@up.edu).

Challenges

The STM32F746G-DISCO Discovery kit and STM32CubeIDE worked well for the course, but there were a few challenges that had to be overcome. Due to a hardware bug in the STM32F746 microcontroller, the single-step feature of the debugger would get stuck in one of the interrupt service routines (ISR) [26]. One workaround for this problem is to manually disable the interrupts while single-stepping and then to re-enabling the interrupts before resuming the program. A more permanent fix for this issue is to convert the debugger on the board to the SEGGER J-Link debugger, which requires downloading new firmware to the board [27].

Another issue is that the line-in signal contained a lot of noise if the STM32F746G-DISCO board's line-in port was connected to the headphone port of the laptop that powers the board. The easiest fix for this problem was to use to a separate device such as a cellphone as the audio source instead of the laptop. Another possible fix is to connect an audio transformer between the laptop headphone port and the line-in port on the STM32F746G-DISCO Discovery kit.

The last issue was that earbuds and headphones that have a microphone did not work well because their four-connector plug did not make good contact with the line-out/headphone port on the STM32F746G-DISCO board. Therefore, inexpensive earbuds with three-connector plugs were distributed to the students who needed them.

Assessment

The students were surveyed to determine if they thought that the STM32F746G-DISCO Discovery kit was an effective tool for learning real-time DSP. The survey was voluntary and anonymous, and the instructor left the classroom while the students filled it out. Twelve of the 14 students in the class filled out the survey. The results of the first survey question show that the students overwhelmingly believed that the board was an effective tool for learning real-time DSP (see Figure 5).

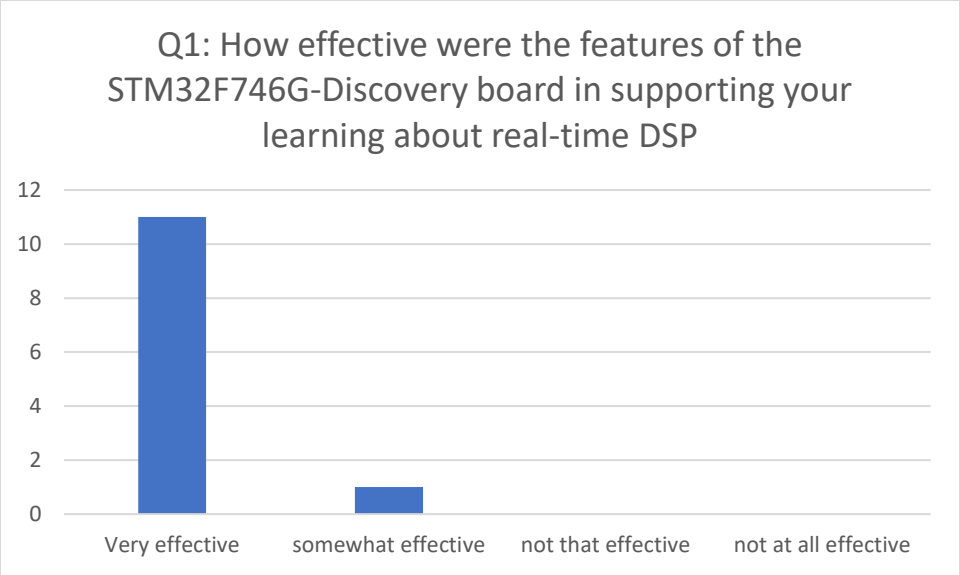

Figure 5: Survey Results for Question 1.

In the second question, most of the students reported that seeing DSP algorithms run in real-time was effective for their learning (Figure 6).

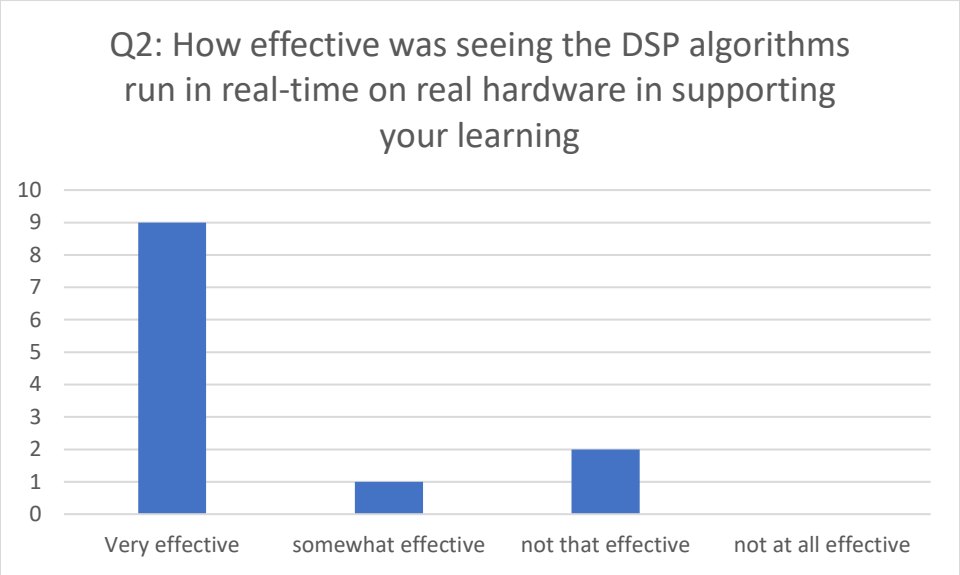

Figure 6: Survey Results for Question 2.

The students also reported that the mobile studio pedagogy was effective for learning this material (Figure 7).

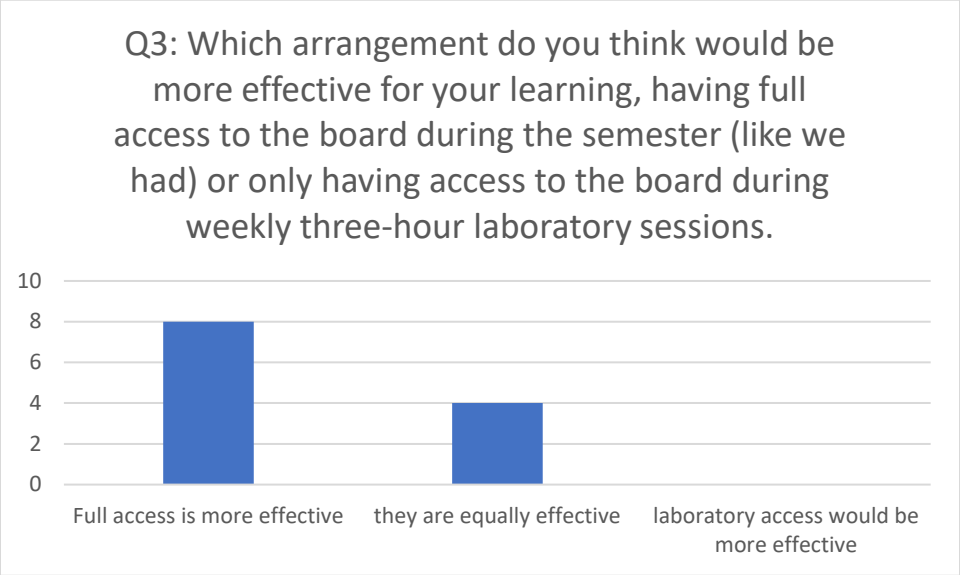

Figure 7: Survey Results for Question 3.

Most students did not think that a required textbook was important for their learning in this course (see Figure 8).

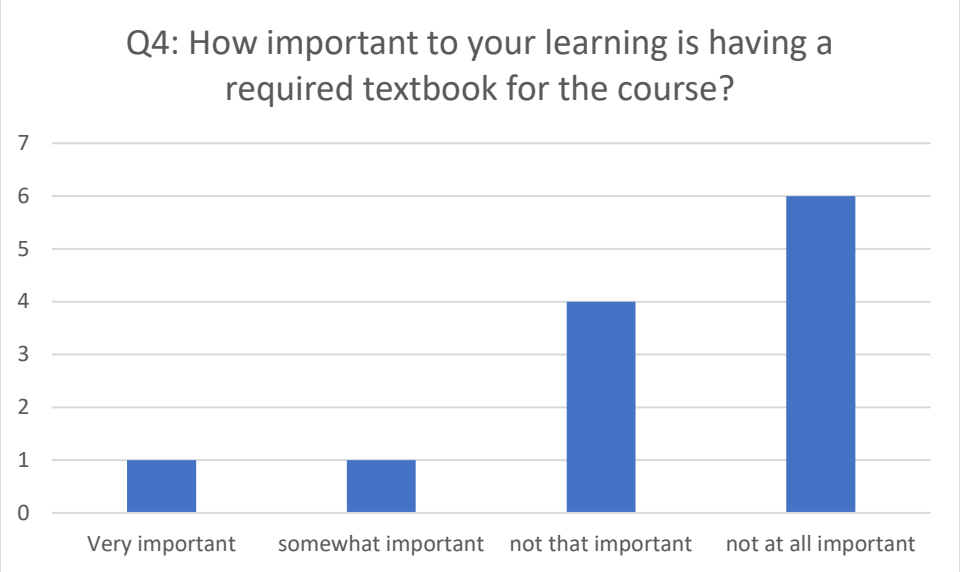

Figure 8: Survey Results for Question 4.

The last survey question asked the following: Do you have any comments or suggestions regarding the use of the STM32F746G-Discovery board in this course? (open answer). Six of the students wrote comments:

- Don't know the exact reason why the board would spontaneously act up, but possibly investing into better software and hardware would help.
- I would (have) loved to be able to see signals that are generated as run.
- I appreciate not having an expensive required textbook. Thank you!
- Maybe cover the built-in systems and functions more. That would help with understanding its operations.
- Spend a single class period building a step-by-step simple project in which the students mirror you from their board and devices.
- We needed more time for the C-code part of the project. To make our own project requires really wrestling with the board.

The first comment listed above showed that the student was frustrated by some aspect of the board, and the second one suggests that that student would appreciate access to an oscilloscope at least for the signal generation assignment. The third comment shows appreciation for not requiring a textbook. The last three comments were suggestions for adding certain material to the lectures and changing one of the project due dates.

Conclusion

This paper described the use of the STM32F746G-DISCO Discovery kit for teaching a real-time DSP course with a mobile studio pedagogy. Although there were some challenges, the board worked well for implementing standard DSP algorithms in real-time using stereo audio signals. The relatively low cost of the board enabled students to have full access to the board throughout the semester. The board can be programmed in C language or assembly using a free IDE that runs on multiple operating systems. The touch screen enabled an intuitive user interface, which made working with the board much more convenient.

References

[1] J.O. Attia, L.D. Hobson, P.H. Obiomon, and M. Tembely, “Engaging Electrical and Computer Engineering Freshman Students with an Electrical Engineering Practicum,” in *2017 ASEE Annual Conference & Exposition*, Columbus, Ohio, June 2017.

[2] K.R. Hite, L.J. Slimak, D. Korakakis, and T.C. Ahern, “An Online Approach to the Analog Electronics Laboratory,” in *2019 ASEE Annual Conference & Exposition*, Tampa, Florida. June 2019.

[3] M.E. Radu, C. Cole, M.A. Dabacan, J. Harris, and S.M. Sexton, “The Impact of Providing Unlimited Access to Programmable Boards in Digital Design Education,” *IEEE Transactions on Education*, vol. 54, pp. 174-183, 2011.

[4] C.J. Prust, “An Introductory Communication Systems Course with MATLAB/Simulink-Based Software-Defined Radio Laboratory,” in *2019 ASEE Annual Conference & Exposition*, Tampa, Florida, June 2019.

[5] K. VonEhr, W. Neuson, and B.E. Dunne, “Software Defined Radio: Choosing the Right System for Your Communications Course,” in *2016 ASEE Annual Conference & Exposition*, New Orleans, Louisiana, June 2016.

[6] R. Musselman, “SDRs Used as Motivational Tool for Communications and DSP,” in *2018 ASEE Annual Conference & Exposition*, Salt Lake City, Utah, June 2018.

[7] M.F. Chouikha, K.A. Connor, and D. Newman, “Experimental Centered Pedagogy Approach to Learning in Engineering: An HBCU's Experience,” in *2016 ASEE Annual Conference & Exposition*, New Orleans, Louisiana, June 2016.

[8] J. Attia, M. Tembely, L. Hobson, and P. Obiomon, “Hands-on Learning in Multiple Courses in Electrical and Computer Engineering,” in *2018 Gulf Southwest Section Conference*, Austin, TX, April 2018.

[9] S.S. Holland, J.L. Bonniwell, J.D. Carl, B.E. Faulkner, R.W. Kelnhofer, C.J. Prust, and L.G. Weber, “It’s All About Engagement: Infusing the Mobile Studio Approach Throughout the Electrical Engineering Curriculum,” in *2021 ASEE Virtual Annual Conference*, Virtual Conference, July 2021.

[10] S.S. Holland, C.J. Prust, R.W. Kelnhofer, and J. Wierer, “Effective Utilization of the Analog Discovery Board Across Upper-Division Electrical Engineering Courses,” in *2016 ASEE Annual Conference & Exposition*, New Orleans, Louisiana, June 2016.

[11] K.A. Connor, P.M. Schoch, K.A. Gullie, D. Newman, S.S. Armand, and J. Braunstein, “Experiment-centric Pedagogy in Circuits and Electronics Courses,” in *2018 ASEE Annual Conference & Exposition*, Salt Lake City, Utah, June 2018.

[12] C. Wicks, “Lessons Learned: Teaching Real-Time Signal Processing,” *IEEE Signal Processing Magazine*, vol. 26, pp. 181 – 185, 2009.

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

[14] E. Cooney, S. Deal, A. McNeely, and H. Chaubey, “Multidisciplinary Undergraduate Research Project to Create Musical Effect Box,” in *2019 Conference for Industry and Education Collaboration, 2019 CIEC*, New Orleans, LA, February 2019.

[15] E. Bezzam, A. Hoffet, and P. Prandoni, "Teaching Practical DSP with Off-the-shelf Hardware and Free Software, "*2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Brighton, UK, 2019, pp. 7660-7664.

[16] Y. Lin and T.D. Morton, "A Microcontroller-based DSP Laboratory Curriculum Paper," in *2017 ASEE Annual Conference & Exposition*, Columbus, Ohio, USA, June 2017.

[17] K.D. Coonley and J. Miles, "Upgrading Digital Signal Processing Development Boards in an Introductory Undergraduate Signals and Systems Course," in *2015 ASEE Annual Conference & Exposition*, Seattle, WA, June 2015.

[18] G. Pasolini, A. Bazzi, and F. Zabini, "A Raspberry Pi-Based Platform for Signal Processing Education," in *IEEE Signal Processing Magazine*, vol. 34, no. 4, pp. 151-158, July 2017.

[19] T. Moon and M.N. Do, "Implement Your DSP Algorithm on Android Tablet: Real-time DSP Laboratory Course," in *2021 ASEE Annual Conference & Exposition*, Virtual Meeting, July 2021.

[20] K. Coonley and G. Chebrolu, "Enhancing a Real-time Audio Laboratory Using the MATLAB Audio System Toolbox," in *2018 ASEE Annual Conference & Exposition*, Salt Lake City, Utah, June 2018.

[21] "32F746GDISCOVERY." st.com. https://www.st.com/en/evaluation-tools/32f746gdiscovery.html (accessed Jan. 23, 2024).

[22] "CMSIS-DSP." github.io. https://arm-software.github.io/CMSIS_5/DSP/html/index.html (accessed Jan. 23, 2024).

[23] "STM32F7 workshop MOOC." st.com. https://www.st.com/content/st_com/en/support/learning/stm32-education/stm32-moocs/stm32f7-hands-on-workshop.html (accessed Jan. 23, 2024).

[24] "STM32CubeIDE." st.com. https://www.st.com/en/development-tools/stm32cubeide.html (accessed Jan. 23, 2024).

[25] "STM32F746G-DSP-Filter-Lab." github.com. https://github.com/sq5mjw/STM32F746G-DSP-Filter-Lab/blob/master/README.md (accessed Jan. 23, 2024).

[26] "ARM: Single Stepping Cortex-M7 Enters Pending Exception Handler." developer.arm.com. https://developer.arm.com/documentation/ka002944/latest (accessed Jan. 25, 2024).

[27] "ST-LINK on-board: Converting ST-LINK On-Board Into a J-Link." https://www.segger.com/products/debug-probes/j-link/models/other-j-links/st-link-on-board/ (accessed Jan. 25, 2024).