2021 ASEE ANNUAL CONFERENCE
Virtual Meeting | July 26–29, 2021 | Pacific Daylight Time
Paper ID #33766

Design and Outcome of a Course on Software-defined Radio Within the Computer Science Department

Dr. Marc Lichtman, University of Maryland College Park

I am an adjunct professor in the dept of Computer Science at UMD where I teach an undergrad elective that I created, introducing the CS students to digital signal processing, wireless communications, and software-defined radio. I do it in a non-traditional and hands-on manner, because the students are strong programmers but don't have the same type of signals and systems background EE students do. I have a PhD in EE from Virginia Tech where I studied wireless communications and signal processing.

Dr. Travis Fredrick Collins, Analog Devices, Inc.

Travis holds BS, MS, and PhD degrees in Electrical and Computer Engineering from WPI. He joined ADI in April 2017 in the System Development Group where he focuses on complete signal chain workflows and system architectures. Travis' expertise includes digital signal processing, communications theory, radar, and high performance compute.

Robin Getz, Analog Devices, Inc.

Robin is currently the Director of Systems Engineering at Analog Devices, and has over twenty years of diverse industry experience in engineering leadership, product marketing and sales with multi-national semiconductor firms, spending his last 15 years at Analog Devices Inc. He has a successful track record of being a highly motivated, strategic thinker, with a passion for technology, and education. Robin currently manages a multi-national, multi-disciplinary team of engineers who deliver high volume board designs, overseeing schematic capture, layouts, initial and volume manufacturing, EMI, ESD and vibration testing for regulatory compliance (CE, FCC), and production test development, and mechanical design for boxing/packaging, for both OEM customers and ADI's education outreach.

Robin obtained his Bachelor of Science in Electrical Engineering in 1994 from the University of Saskatchewan, in Saskatoon, Canada. Robin holds 4 patents in the area of acoustic / thermal control for personal computers.

Design and Outcome of a Course on Software-Defined Radio within the Computer Science Department

Marc Lichtman, Ph.D., University of Maryland
Travis Collins, Ph.D., Analog Devices, Inc.
Robin Getz, Analog Devices, Inc.

Introduction

Over the last decade there has been a surge in professional work related to Software-Defined Radio (SDR), and more broadly, Digital Signal Processing (DSP) applied to wireless communications. However, we believe there is an inefficiency when it comes to the current higher education system providing enough graduates with an appropriate background to work in these areas. It may stem from the fact that wireless communications, DSP, and SDR are all topics traditionally taught at the graduate level within Electrical and Computer Engineering (ECE). Thus, the majority of persons with the requisite knowledge and interest will be ECE MS and PhD graduates. While many ECE graduate level students are strong coders, software development skills are not the primary focus of traditional ECE programs, at least when compared to that of a typical Computer Science (CS) curriculum. This results in a small pool of candidates for positions in wireless communications and SDR, made up of MS and PhDs in ECE who happened to focus within the area of wireless communications. Only a fraction of those will have strong coding skills, and an even smaller fraction will have experience or coursework related to software development. This causes a dilemma, since the majority work performed by industry focuses on implementation of DSP in software, rather than raw mathematics or pure derivation. It is not to say that deep mathematical understanding is not required for industry, but software focused positions are more abundant. We also want to stress that it's not just a problem of background knowledge; CS students are simply unlikely to end up in these types of positions, because it's an area they have never been introduced to, unless they just happened to have taken an ECE course on the topic, or are part of a multidisciplinary graduate research group.

Within the Department of Computer Science at The University of Maryland College Park, we have created an elective at the undergraduate level, which introduces CS students to the area of wireless communications, in a hands-on manner using SDRs. Anyone familiar with the area of SDR will know that it requires a heavy background in DSP and communication system theory. Because the CS undergraduates taking this course were unlikely to have any of this background, the first half of the course acts as a DSP and wireless communications primer, essentially condensing several courses that are normally taught at the graduate level within ECE. We were able to provide CS students with the necessary background by teaching DSP theory using diagrams, animations, practical demos, and code examples rather than a mathematically rigorous theoretical approach. The remainder of the course focused on using SDRs to implement the DSP techniques they had learned (a step rarely covered in academia), as well as learning about different areas of wireless communications such as cellular, IoT, and security at the lower layers. The ultimate goal of our course, titled CMSC498X Intro to Wireless Comms and SDR, was to introduce CS students to the area of wireless communications, in a hands-on manner using

SDRs, so that they could not only build an interest in the area, but also be prepared for professional practice.

To date, we are not aware of any other SDR-based course within a CS department of a university. In fact, there are very few SDR-based courses in the US [1-5,7,8], and these are typically graduate courses taught by ECE departments with large wireless communication groups. As such, these ECE courses tend to focus more on theory, instead of being more software intensive.

Course Design and Approach

In this section we discuss how the course was designed, as well as the hardware and software we used as part of the course. From a high-level point of view, the learning objectives during the semester were organized into five sequential steps:

1. Learn basic DSP concepts
2. Get hands-on experience with SDRs
3. Learn wireless communications concepts
4. Cover system-level wireless communications analysis/design
5. Get experience creating SDR applications

What makes this course unique from other existing courses is the lack of prerequisite knowledge, and thus the number of topics that need to be concisely covered. This course includes topics that would normally be spread across several individual courses. As a result, this course only makes sense for a CS department, or ECE department without a wireless communications group, i.e., where the students have no option of taking the more comprehensive courses specific to each topic. It was for this reason that ECE students at our university were not allowed to enroll in the course, even though they did have the option of taking other CS courses as electives.

List of Topics

The following list highlights each lecture, and the topics covered. Note that some of these lectures were longer than others. Certain lectures spanned two course days, while others had to share a day with an exercise or student challenge.

1. Concept of the frequency domain, introduction to Fourier Series, Fast Fourier Transform (FFT), and power spectral density, spectral analysis of signals found in the environment around them. Students receive actual FM radio, LTE, GSM, ADS-B signals; any signals they can find using their provided SDRs and antennas.
2. Nyquist sampling, concept of "not losing information" when sampling a band-limited signal, and the concept of band-limited. They will observe the type of artifacts that occur with inadequate sampling, such as aliasing.
3. IQ sampling, and complex number review. Concept of modulating a carrier, and how to upconvert a baseband signal to real RF.

4. Digital Modulation (ASK, PSK, QAM, FSK) – Introduces the concept of modulating a carrier with bits of digital information. Students will learn how to interpret an IQ/constellation plot.
5. GNU Radio basics – Installing GNU Radio, placing and connecting blocks, running applications.
6. PlutoSDR basics – Connecting to the SDR, using it in GNU Radio and Python.
7. Digital filtering, focusing on FIR filters and how taps work, and an introduction to convolution. Students discover first-hand (i.e., "learning by doing") the purpose of a low-pass filter in the sampling process. They also learn how to filter out noise or an unwanted signal that is in an adjacent frequency band, using two different filter design methods.
8. GNU Radio advanced techniques, including creating custom blocks in Python.
9. The wireless channel and additive white Gaussian noise (AWGN) channel model, simulation of real and complex. Also introduces signal-to-noise ratio (SNR) and dB.
10. Link budget basics, which involves going into antenna types (and gain), free space path loss, atmospheric attenuation. Study of an example link budget based on ADS-B. We discuss monopole, dipole, and examples of directional antennas, without diving too deep into antenna design. Students cut their own FM radio antennas out of wire, after calculating the correct length for a ¼ wavelength whip.
11. BPSK exercise, students learn about the steps to synchronize and demodulate an actual signal, by building up a GNU Radio flowgraph using a guided class-wide exercise, where each student has their own SDR and code.
12. RF Spectrum, frequency allocation, ISM bands, cellular bands, satellite communications, military communications, garage door openers, and other common signals.
13. Channel coding (a.k.a. forward error correction) – Why we need it, where it gets placed in the transmitter and receiver, adaptive modulation and coding schemes, and an example of how Hamming codes work to detect and correct bit errors. Introduce Shannon Limit, and point out state of the art Turbo and LDPC codes.
14. RF signal file formats – How to record, play back, and store metadata of IQ signals
15. Cellular communications – Understanding LTE, as well as a brief history of 2G and 3G cellular, and what's coming in 5G
16. Multipath fading, once again using "learning by doing" by looking at actual examples of fading in real-time. This is done by observing frequency-selective fading that you would experience in a typical classroom, using a high-bandwidth signal transmitted by the professor as part of a live demo, where students can wave their hands in front of the transmitter or receiver while watching the real-time spectrogram/waterfall.
17. Additional GNU Radio tutorials and exercises, including creating out-of-tree modules (OOTs).
18. Electronic Warfare – Basics of signal detection, classification, interception, and jamming.
19. Internet of things (IoT) – IEEE 802.15.4 (ZigBee) and other wireless protocols that are enabling IoT. We dive into the PHY and MAC layer of ZigBee, and use a GNU Radio flowgraph that receives and decodes it.
20. Pulse shaping and matched filtering—why it's used in almost every wireless device

While this list of topics covered might seem immense to someone who studied within ECE, we were only able to scratch the surface of several of these areas, many of which are often their own entire course. The intent was not to make the students feel overwhelmed, but rather to give them a little exposure to the core areas of wireless communications and DSP, so that they would be more comfortable if and when they run into any of these topics in the future.

Amount of Time Spent on Diving into Math and Theory

As discussed in the introduction, a big differentiator of this course from traditional DSP and wireless communications courses taught within university, is the way we introduced difficult DSP and communications theory to the students. Our practice is to teach concepts in a practical and intuitive manner using illustrations, figures, and analogies rather than by rigorous mathematics. A concept should be carefully introduced before covering the math behind it, as most students do not learn well through equations alone. We made heavy use of figures, animations, in-class demos, and practical coding exercises, to teach the math-heavy concepts. Below is one of the example figures used in the Nyquist sampling section, to show how if you do not sample fast enough, you can have ambiguous samples that can lead to two or more possible signals/functions.

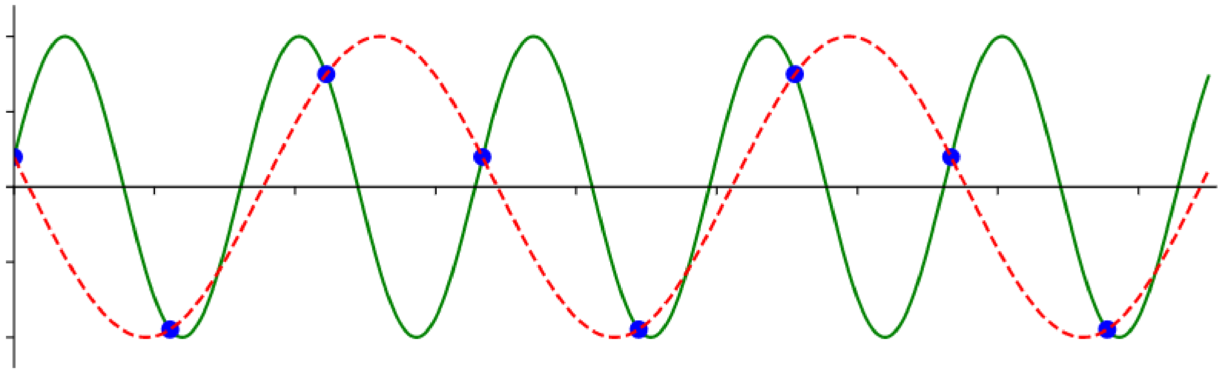


Figure 1. Sample figure used in Nyquist sampling section of the course, to demonstrate the principle of sampling fast enough to avoid ambiguity in the function recreated by the sampled points.

Hardware

The first semester CMSC498X was taught, in 2018, one RTL-SDR was provided to each student during the course of the semester. The RTL-SDR is a $20 receive-only SDR that can tune to any frequency between 25 MHz – 1700 MHz, and receive at an instantaneous bandwidth up to 2 MHz (reliably). The complex baseband samples are sent to the host computer over USB 2.0, where they can be processed using DSP, such as in GNU Radio. The RTL-SDR can be used for receiving signals like FM, ADS-B, and RDS, but it is extremely limited because it cannot go high enough to reach the 2.4 GHz ISM band, and the 2 MHz bandwidth limit means students could not see “full” signals that were wider than 2 MHz, which is a fraction of most modern signals. For example, most LTE signals have bandwidths of 10 or 20 MHz.

For the second time teaching CMSC498X, in 2019, we were able to provide each student with a PlutoSDR, a $150 SDR made by Analog Devices that can both receive and transmit. Note that this is not the first course that has made use of the PlutoSDR; see [7,8]. The PlutoSDR is able to tune between 70 MHz to 6 GHz, and have an instantaneous bandwidth up to 56 MHz, although because of its USB 2.0 bottleneck to the host, transferring samples at 56 MHz must be done using a low duty cycle (i.e., only processing a small fraction of samples, in batches), which wasn't an issue during the course. This is because the students spent a lot of time simply receiving samples and calculating/plotting the power spectral density, in order to "view" the RF spectrum, so being able to see 56 MHz at a time was extremely valuable, even if they were only processing 1 out of every 100 batches of samples, for example. The RF integrated circuit (RFIC) inside the PlutoSDR, which largely determines its capabilities and performance, is the same family of RFICs used in the Ettus Research B200 and B210, BladeRF, and many other much more expensive SDRs heavily used by industry.

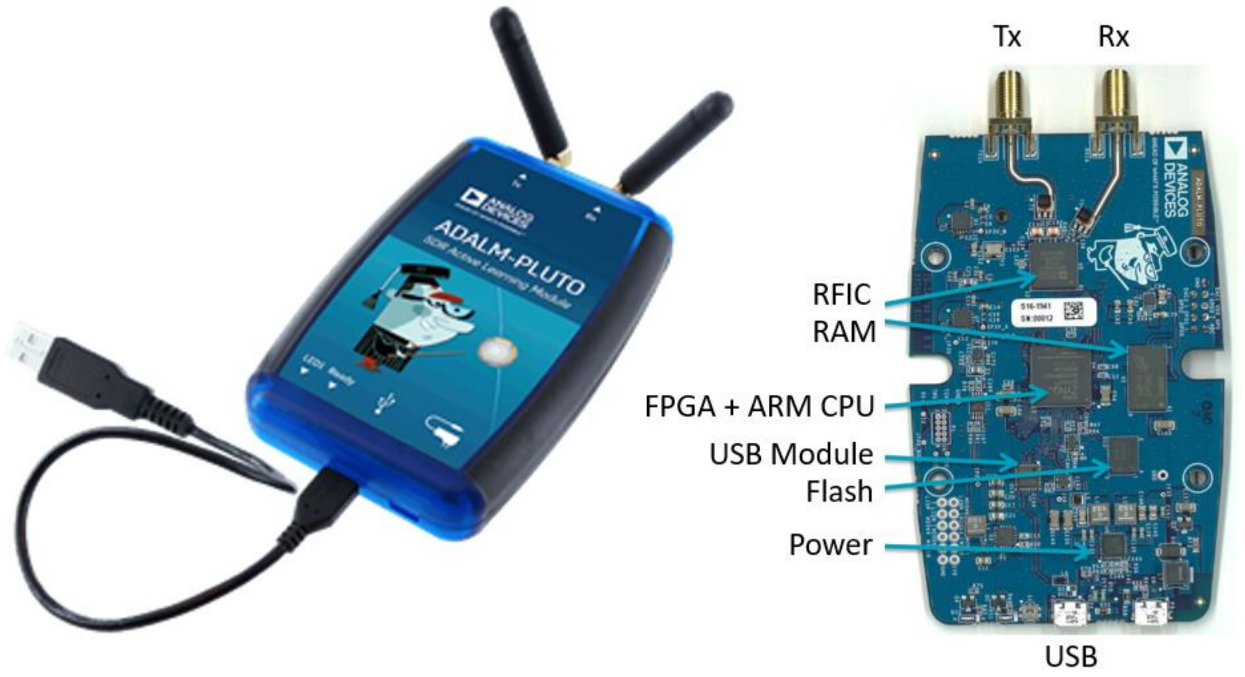
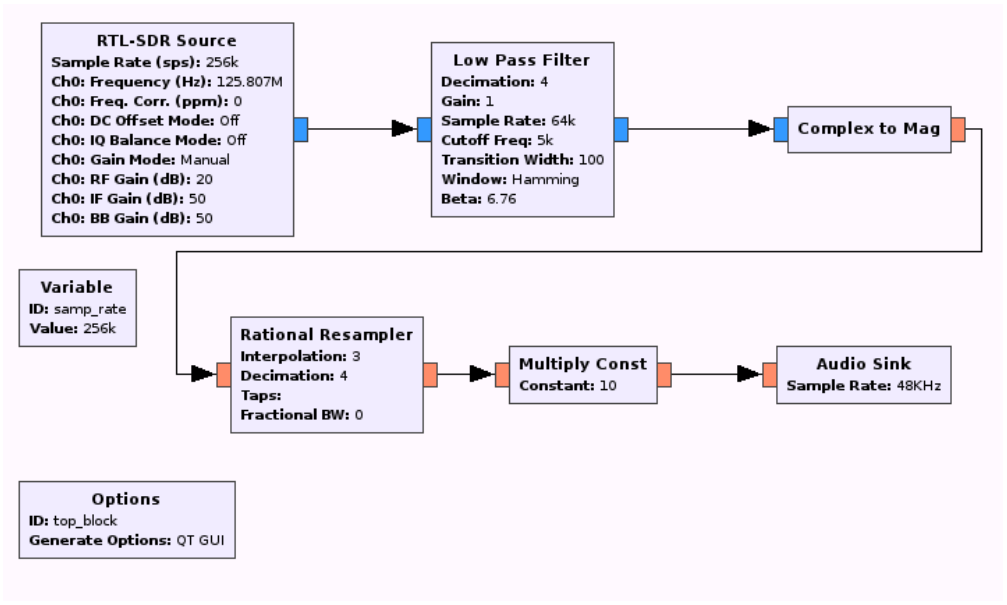


Figure 2. PlutoSDR, both external (left) and labeled PCB (right).

Even though the PlutoSDR can transmit, no part of the course required that students use the transmitting capability. However, there were in-class exercises that involved the professor transmitting a signal, and the students receiving it.

Software

While the hardware supports multiple signal processing frameworks, such as MATLAB, Simulink, or Labview; in this course, students used both GNU Radio, and Python, to control their SDRs and implement DSP techniques. GNU Radio is a free and open source SDR framework/toolkit, that comes with hundreds of premade blocks that each perform a single DSP or communications type function. Blocks are connected together in what's called a flowgraph, which represents the GNU Radio "app". Flowgraphs will often start with a block that receives samples using the SDR, and end with some sort of GUI or audio device, as shown in Figure 3. GNU Radio allows creating powerful DSP applications without writing a single line of code, as well as the ability to experiment with flowgraphs created by other people. There are existing GNU Radio flowgraphs shared online for receiving and decoding FM radio, ADS-B aircraft signals, RDS digital radio signals, WiFi, digital television, 3G and 4G cellular (the unencrypted signals), satellite signals, NOAA weather signals, and many more. Some of the DSP blocks that come with GNU Radio implement extremely sophisticated algorithms, such as synchronization techniques. Most of the time spent using GNU Radio within this course was having the students learn how to use GNU Radio, because it has a very steep learning curve, at least when you want to create your own blocks. Custom blocks can be coded in C++ or Python, although this custom code must still fit within the GNU Radio block framework, which includes a lot of GNU Radio specific boilerplate.


RTL-SDR Source
Sample Rate (sps): 256k
Ch0: Frequency (Hz): 125.807M
Ch0: Freq. Corr. (ppm): 0
Ch0: DC Offset Mode: Off
Ch0: IQ Balance Mode: Off
Ch0: Gain Mode: Manual
Ch0: RF Gain (dB): 20
Ch0: IF Gain (dB): 50
Ch0: BB Gain (dB): 50

Low Pass Filter
Decimation: 4
Gain: 1
Sample Rate: 64k
Cutoff Freq: 5k
Transition Width: 100
Window: Hamming
Beta: 6.76

Complex to Mag

Variable
ID: samp_rate
Value: 256k

Rational Resampler
Interpolation: 3
Decimation: 4
Taps:
Fractional BW: 0

Multiply Const
Constant: 10

Audio Sink
Sample Rate: 48KHz

Options
ID: top_block
Generate Options: QT GUI

Figure 3. Example GNU Radio flowgraph.

In addition to GNU Radio, the students used Python, both as a way to simulate DSP techniques, as well as a way to control their SDR and create SDR applications without needing GNU Radio. By using Python directly, without any frameworks or libraries, most of the CS students were extremely comfortable, as they had experience coding in straight Python. We made use of the numpy, scipy, and matplotlib packages, but the course itself did not provide the students with any professor-written libraries or packages. Students were also free to use any Python packages they already had experience with. As part of the course, students were taught how to use the Python API of the PlutoSDR, which is used to tune the SDR, set various parameters, and then receive samples which can then be processed in the rest of their Python code. Although GNU Radio lets the students experiment with sophisticated SDR applications (that were mostly premade for them), by using Python, they had to implement absolutely everything themselves. For example, to create a power spectral density, they had to actually perform the FFT function, take the magnitude, shift the indices, take the log, figure out the frequency domain axis labels, and then plot the values using matplotlib (or another plotting library they are familiar with). By doing it all themselves, they learned more, and gained valuable hands-on practical experience.

Learning by Doing

We have already discussed how students were taught theoretical concepts using animations and coding examples, which is an example of learning by doing, but the course also involved mini Capture-the-Flag (CTF) competitions. Two to three times per semester, for about 20 minutes at the beginning of class, the students would be challenged to find/decode a signal that was being transmitted within the classroom. Using their SDRs, they would use recently learned concepts, combined with critical thinking, in order to find and/or decode an RF signal being transmitted within the classroom. These wireless-themed mini-CTFs got the students excited, especially those who were first to figure out the solution. Some of the students had a background in cyber security and had done information security type CTFs before, so they were familiar with the format. After about 20 minutes, the solution would be shared with the students, for those who were not able to figure it out. The CTFs earlier in the semester are more challenging to design because the students are only starting to learn the foundational knowledge. The first CTF involved transmitting a signal using gr-paint [6] which "paints" an image onto the spectrum, i.e. in the frequency domain, so that anyone looking at that portion of spectrum on a spectrum analyzer will see the image. An example spectrogram of gr-paint working is shown in Figure 4 below. This worked well because the students had recently learned how to create a spectrogram in pure Python, and had also been shown how to use GNU Radio as a real-time spectrum analyzer. The only real hint the students needed was the RF band the transmitter was using.

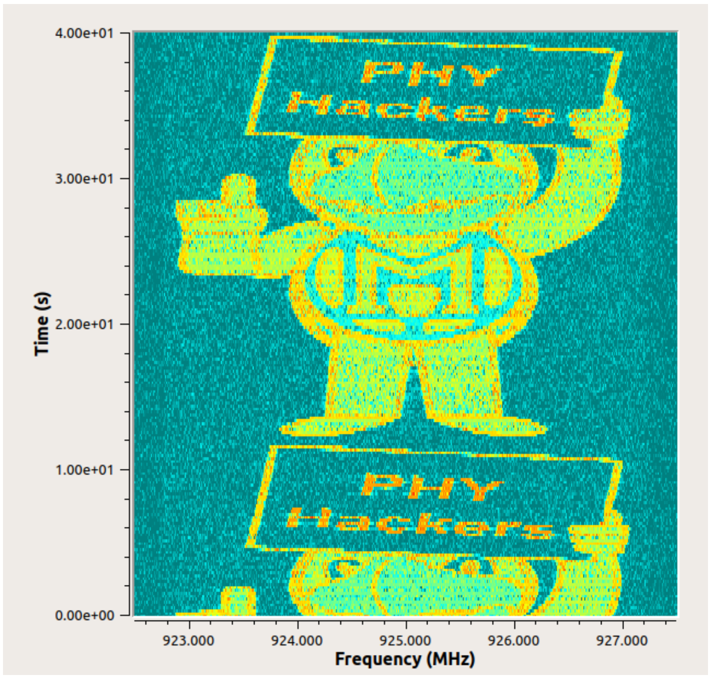


Figure 4. Screenshot of a live spectrogram, while receiving a gr-paint signal

Class Size

During the first semester this course was taught, we had 21 students, with 31 during the second semester. For our department, this class size is on the small end of the spectrum. A relatively small class size was chosen because the course involves the students learning how to use hardware; there were multiple points during the semester where the students had an in-class exercise to do and needed the professor's assistance with various errors and problems they ran into. This course, in its current form, would simply not work with a class of 200+ students, unless there were multiple knowledgeable TAs present at lectures that involved in-class exercises.

Exam Design

Both semesters involved a final exam, but no other tests or exams. The final exam was open-notes but students were not allowed any communications with others. It included a mix of multiple choice, short answer, and diagram related questions. There was simply not enough time or logistics to be able to require students to do any coding or GNU Radio problems, although many of the questions involved GNU Radio. The questions were designed so that students who did not pay attention during class and failed to fully understand concepts would have the most trouble, despite it being open-notes. One of the questions off the final exam is provided below.

1. Imagine you are a receiver, receiving the signal below (shown *before* sampling). You start sampling at time = 0 with a sample rate of 100 Hz. Draw the IQ plot of the samples you would digitize during this time interval (a blank IQ plot is provided).

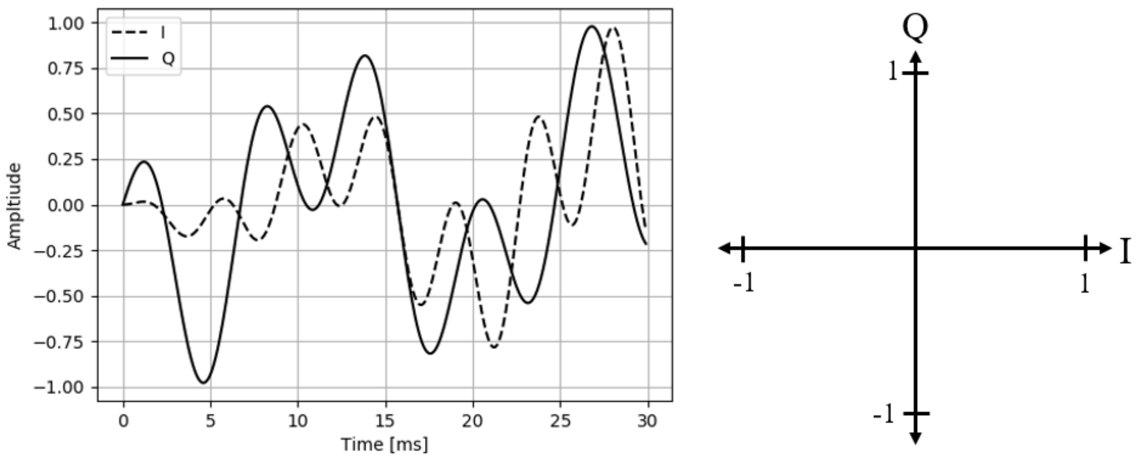


Figure 5. Example question from the final exam, asking students to show where the signal gets sampled and to translate the samples onto an IQ plot (complex plane).

There is nothing inherently difficult about this question, but a student who wasn't comfortable converting from sample rate to sample period, and/or uncomfortable with IQ signals and constellation plots, might struggle with it. Some incorrect answers involved students putting dots where the two lines intersected, which doesn't make any sense, as well as not realizing that the IQ plot on the right doesn't actually capture time, it's just a series of points (4 points in this case).

Another question off the exam was given as follows: “If I am using a sample rate of 20 MHz, and I tune to 2.44 GHz, and take FFTs using 128 samples per FFT, how many Hz does each bin of the FFT represent? I.e., if I labeled every tick of the x-axis then what would the interval between ticks be, in Hz?” The students had many homework and coding exercises that involved taking FFTs, so this was an easy one for those who had done their own work all semester. It’s also partially a trick question because it doesn’t matter what frequency the SDR tunes to, so there’s a slight critical thinking aspect to it.

Results and Discussion

Course Evaluations

Student quantitative evaluations for the two semesters this course has been taught are shown below, which include the averages at the college, dept., and course level (400, i.e., elective level, in this case). Table 1 shows Fall 2018, which had 19 evaluations submitted out of 21 students enrolled (90% response rate).

Table 1. Quantitative portion of Fall 2018 student course evaluation.

Question	Mean Numerical Value			
	This Course	College	Department	CS Elective
1. The course was intellectually challenging.	3.26	3.26	3.35	3.42
2. I learned a lot from this course.	3.53	3.08	3.22	3.21
3. The instructor treated students with respect.	3.95	3.43	3.48	3.50
4. The instructor was well-prepared for class.	3.79	3.30	3.31	3.35
5. Overall, this instructor was an effective teacher.	3.74	2.98	3.01	3.01
6. The standards the instructor set for students were...	0.95	1.15	1.17	1.16
7. Course guidelines were clearly described in the syllabus.	3.11	3.18	3.18	3.17
8. The required texts (e.g., books, course packs, online resources) helped me learn course material.	3.14	2.87	2.66	2.90
9. Based on the quality of my work in this course, the grades I earned were	1.00	0.82	0.81	0.81
10. Given the course level and number of credits, the workload was	0.95	1.14	1.18	1.12
11. How much effort did you put into the course?	1.32	1.49	1.54	1.51
12. The instructor was effective in communicating the content of the course.	3.79	3.01	3.03	3.04
13. The instructor was responsive to student concerns.	3.95	3.21	3.28	3.29
14. The instructor helped create an atmosphere that kept me engaged in course content.	3.84	2.94	2.96	2.97

These results are interesting, because the biggest challenge we had going into the initial design of the course was "how do we teach such heavy concepts to students without the typical background?" I.e., we were genuinely concerned about the course being too challenging and the students not being able to follow and learn. Because this was such a big focus, we may have gone a little too far and caused the course to be slightly too easy for the average student, at least when it comes to workload and effort required. But the extremely higher-than-average feedback for questions #2 and #9 more than make up for it. Overall, this feedback was very positive, considering it was the first time the course was taught, so the material was not yet tested, and thus rough around the edges.

Table 2 below shows the course evaluations for Fall 2019, which had 17 evaluations submitted out of 31 students enrolled (55% response rate).

Table 2. Quantitative portion of Fall 2019 student course evaluation.

Question	Mean Numerical Value			
	This Course	College	Department	CS Elective
1. The course was intellectually challenging.	3.3	3.3	3.4	3.4
2. I learned a lot from this course.	3.4	3.1	3.2	3.3
3. The instructor treated students with respect.	3.9	3.5	3.5	3.6
4. The instructor was well-prepared for class.	3.9	3.4	3.3	3.4
5. Overall, this instructor was an effective teacher.	3.8	3.1	3.0	3.1
6. The standards the instructor set for students were...	1.0	1.1	1.2	1.1
7. Course guidelines were clearly described in the syllabus.	3.2	3.2	3.1	3.2
8. The required texts (e.g., books, course packs, online resources) helped me learn course material.	3.2	2.9	2.7	3.0
9. Based on the quality of my work in this course, the grades I earned were	1.0	0.8	0.8	0.8
10. Given the course level and number of credits, the workload was	0.9	1.1	1.2	1.1
11. How much effort did you put into the course?	1.4	1.5	1.5	1.5
12. The instructor was effective in communicating the content of the course.	3.8	3.1	3.0	3.1
13. The instructor was responsive to student concerns.	3.9	3.3	3.2	3.3
14. The instructor helped create an atmosphere that kept me engaged in course content.	3.8	3.0	2.9	3.1

During this second semester of teaching the course, the effort required and "intellectually challenging" metrics were much closer to the average compared to the previous year. This year also had exceptionally high feedback for effectiveness of teaching and communicating the course content.

Challenges

The main challenge of the course has already been discussed; having to fit in so many heavy concepts into a single semester worth of time, while having time left to play around with SDRs. But there were other challenges as well. One that became a recurring theme had to do with students that were not strong programmers. Although the students were assumed to have no prerequisite knowledge in the areas of signal processing and wireless communications, the department indicated it was fair to assume all students were familiar with programming in

Python. While most students were, there were a small fraction that struggled through all of the coding exercises and homework, simply due to a lack of Python programming skills (or possibly programming skills in general). It is tough to grasp new concepts related to DSP and SDR when bogged down by problems forming correct Python syntax, logic, loops, etc. Based on the types of problems these students ran into, we believe that it was not an issue with the students being familiar enough with Python, but more of a lacking in general programming skills. Once again, it was only a very small fraction of students who struggled with the programming component.

Lastly, as with any course, there is always the issue of students working together on homework assignments, where one student doesn't fully do it themselves. This is especially common in courses that are light on exams, and/or have open-note exams. This challenge is best addressed by having more exams, but we simply did not have the time to allocate extra lectures (or half lectures) to exams, leading to a conundrum.

Conclusion

Overall, we found that we *were* able to teach all the prerequisite DSP and communications concepts, with time left to focus on SDR. This was only possible by skipping the tens-of-minutes usually taken for each mathematical derivation, and by focusing on the concepts that mattered most to the SDR component of the class. The two semesters CMSC498X was taught were considered successes, and since then, a similar course has been taught with a machine learning emphasis added on, at the master's level. In addition, an online companion resource to CMSC498X was created, and then subsequently made freely available for anyone to use, called **PySDR: A Guide to SDR and DSP using Python**. It can be accessed at https://pysdr.org/, and the underlying content is open source and hosted on GitHub [9].

References

[1] Georgia Tech Course DEF 4013P; Software-Defined Radio Development with GNU Radio: Theory and Application, https://pe.gatech.edu/courses/software-defined-radio-development-gnu-radio-theory-and-application

[2] Rutgers Course: Software-Defined Radio: A Hands-On Approach, http://www.winlab.rutgers.edu/~spasojev/courses/sdr/

[3] University of Michigan Course EECS 398: Software Defined Radio, https://ece.engin.umich.edu/academics/course-information/course-descriptions/eecs-398-software-defined-radio/

[4] Grand Valley State University Course EGR 415 Communication Systems, https://www.gvsu.edu/catalog/course/egr-415.htm

[5] VonEhr, Kurt, William Neuson, and Bruce E. Dunne. "Software defined radio: choosing the right system for your communications course." *Proceedings of the American Society for Engineering Education (ASEE)*. 2016.

[6] R. Economos, "gr-paint: An OFDM Spectrum Painter for GNU Radio", https://github.com/drmpeg/gr-paint

[7] Silage, Dennis A. "Incorporating PlutoSDR in the Communication Laboratory and Classroom: Potential or Pitfall?."

[8] Wyglinski, Alexander M., et al. "Sample-Based Understanding of Wireless Transceivers and Digital Transmis-sion Via Software-Defined Radio."

[9] Source code for PySDR: A Guide to SDR and DSP using Python (at http://pysdr.org). Available: https://github.com/777arc/textbook