1. Flipping the electronics lab: doing upper division electronics at home

Table of Contents

Introduction . . . 1
Hardware Kits . . . 3
Software & Computer requirements . . . 4
Chapter List & Concept Matrix . . . 5
Chapter Summaries . . . 6
Supplementary Materials . . . 10
References . . . 11

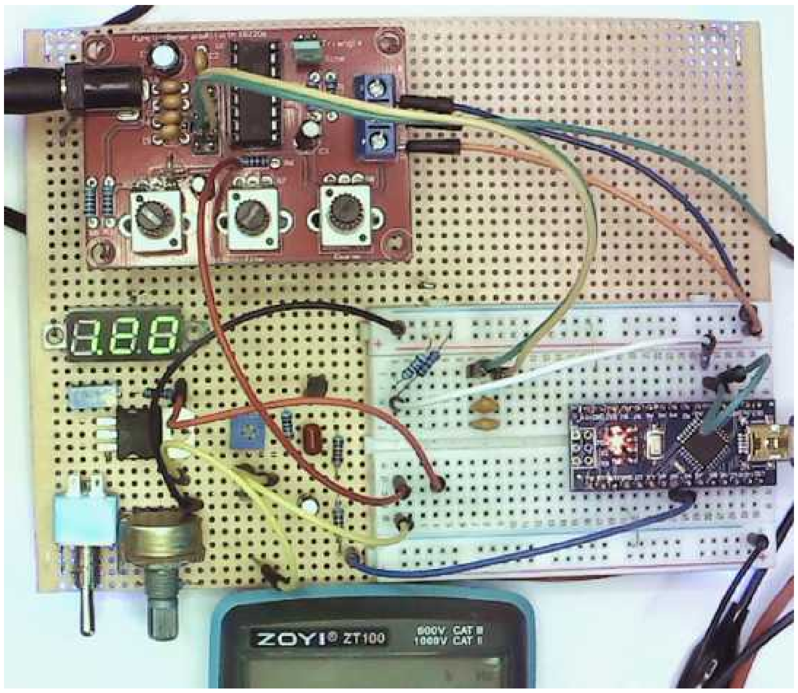


Figure 1. Final board built from our kits. This apparatus generates Bode plots using Arduino (blue, right) and MATLAB to automate sweeping a function generator (red, top) while measuring frequencies, voltages, and phase shifts.

Introduction

Learning electronics is greatly facilitated with "active learning" − building circuits, measuring electronic devices, comparing measurements and models, drawing schematics, etc. That's the philosophy driving this course. However, making lots of good measurements can be tedious, so I'm offering a solution to that problem: laboratory automation. We automate measurements, automate data reduction and visualization, and even automate (to some degree) generating rich technical reports describing the activities and their results.

This curriculum came about because of Covid 19. I'd been teaching a junior level electronics course for physics and engineering students, that had rich hands-on activities. Tired of dragging hundreds of pounds of lab power supplies and function generators out of the stockroom for each class, students constructed their own test equipment, although we still used oscilloscopes for AC phase measurements. Inexpensive Arduinos can sample AC signals up to a few kHz, and MATLAB's 5th generation programming language facilitates building a visual interface to display voltages (and/or spectra) with just a few lines of code. This would have to serve as an oscilloscope during Covid lockdown.

But why not start using the Arduino earlier, and automate measuring DC signals, like I-V curves? Instead of collecting a few data points at a manual "sample rate" ~0.05 Hz, a few lines of Arduino code can measure >8800 voltages per second, or slowed to a manageable 5-10/sec while turning a knob on the power supply. With copious amounts of data, automating analysis and visualization becomes essential, and here MATLAB excels.

But most students don't yet have experience programming. While its not practical to teach students to *write* software from scratch, they can to learn to *read*, understand, modify, and execute *simple* software sequences, especially when written in a high-level interpreted environment like MATLAB. For example, MATLAB's simple syntax, workspace environment, and powerful built-in objects and functions enable building a 2-channel oscilloscope in <20 lines of code. Here's the entire loop that updates the oscilloscope display with fresh data.

```
while runBtn.Value
    writeline(arduino,'b');
    bin = read(arduino, 802, 'uint16');
    data = reshape(bin(1:800),2,400)';
    plot(data, '.-'); grid;
    xlabel('sample #'); ylabel('voltage/ADU')
end
```

Explained in English, while the run button is checked, MATLAB tells Arduino to measure and send a block of data (by sending it a 'b' command), which is read into variable `bin`, rearranged into 2 columns, and plotted. The 12 prior lines of code create the run button and establish a serial connection to `arduino` (which we programmed to respond appropriately to the 'b' command, with a simple C program). This program's graphical interface is through an intuitive click on the graphical run button. But the powerful command line interface enables one to compute, e.g., `rms(data)`, `fft(data)`, and myriad other transformations to explore and extract meaning from the signals connected to Arduino's analog inputs. Subsequent programming exercises calibrate the time axis, add a trigger threshold, display spectra, compute amplitudes, frequency, and phase shifts, etc. Eventually, students begin to engage in interactive "dialogs" with their apparatus, data, and models.

Once you start automating laboratory processes, you look for throughput bottlenecks, and try to automate them. How can an Arduino *control* our power supply – essentially turn the voltage knob? Two simple transistor circuits can do this trick, while demonstrating practical utility of transistor theory, and providing more opportunities to practice soldering, programming, debugging, modeling, comparing theory and measurements, ... i.e., practice electronic engineering (and scientific) patterns and processes.

Too many upper division physics courses don't take the time for repetition, and hop from one nearly disjoint topic to the next. This curriculum covers a lot of ground, but also repeats over and over some key "design patterns". Every lab method is reduced into 3 main sections: a <u>hardware interface</u> that connects the "device under test" (DUT) in a circuit with the Arduino that does the measurements; the <u>software interface</u> that tells the Arduino what to measure and communicate; and what I call a <u>data interface</u> typically written in MATLAB that automates data reduction and visualization, and often much more. With each repetition, abstract ideas become more concrete and internalized, and the sophistication and power of the analyses rachets upward.

Most introductory college electronics texts focus on building theoretical foundations and defer integration of those ideas for later courses. Here we focus as much on integration. We need to build our own electronic test equipment to study transistors, filters, and control circuits. But we need to integrate transistors, filters, and control circuits into the test equipment that we build. So the motivation to study transistors and filters isn't (just) to build foundations, but to immediately instantiate these ideas into our apparatus for subsequent explorations. At the end of the course, instead of a few disjoint hours using complicated commerical equipment for canned experiments, students will have built, debugged, and become intimate with their own test equipment, including a programmable DC power supply, programmable function generator, oscilloscope and spectrum analyzer functionality, and AC voltmeter measuring amplitude and phase. Although limited in power, frequency, and many other features, the processes of building, calibrating, and debugging this apparatus will be a foundation and preparation to explore higher bandwidth in subsequent endeavors. Furthermore, we've emphasized and instantiated many connections between concepts (Fig. 3, Concept Matrix), e.g., *using* voltage dividers every week, *using* current mirrors to sweep frequencies, *using* complex numbers to represent AC voltages, etc.

Most students are familiar with using computers in the form of smartphones, video games, or PCs. Much of this experience centers around the machine acquiring user input – as clicks or scrolls – and responding with data displayed from the Internet or generated with programmed simulations. Some even have programming experience in this application space beyond customizing personal websites. Here we practice programming computers to measure, manipulate, and display sensor data. This is a key aspect of modern electronics – the "front end" of many signal processing tasks, and a key component of this course. Long ago computers beat us in their numerical computational abilities, and likewise their abilities to measure and interface with apparatus is far superior to ours, hence we must learn to use computers for what they excel at and combine those capabilities with our superior understanding of meaning and contexts.

Science progresses by constructing new knowledge from what we already know. We can learn from theory, e.g., if a two terminal electrical device obeys Ohm's Law, $V = iR$, and you know two of those quantities, then Ohm's Law tells you the third. We also learn from accurate and precise measurements. A big focus of this course is designing and building apparatus to make (and automate) electronic measurements, and compare the results with models in MATLAB. Although we're studying electronics, this process of leveraging theory against measurement is a common pattern applicable to many scientific fields.

Hardware Kits

This course is built on inquiry, and we provide the essential hardware to do that in a kit with components listed in Table 1, that cost ~$125. Included are: a $30 digital multimeter (DMM) capable of measuring frequency; a $21 soldering kit with a 60W temperature controlled iron; a $18 electronic project starter kit containing a solderless breadboard, variety of R's, C's, transistors, LEDs, jumper wires, etc.; a $14 function generator kit using an XR2206 integrated circuit (IC); a 12V 1A DC adapter; an LM317 adjustable voltage regulator IC; a 4"x6" perforated circuit board for soldering circuits, toolbox with electronic pliers, wire cutters and stripper, safety glasses, electret microphone, power switch, alligator clips, Arduino Nano and USB cable, dual op amp IC, and a few 0.1 and 4.7uf capacitors not in the kit. Spare parts are freely available from the Physics stockroom. If you're anticipating a long term interest in electronics, investing more in a better DMM and soldering station can make your experience more comfortable and productive.

1. electronic project starter kit – wires, breadboard, LEDs, components
2. 12v 1A power supply
3. 10k pot
4. switch
5. LM317
6. alligator clips (4x)
7. arduino nano and cable
8. 3 digit voltmeter display
9. function generator kit
10. needle nose plier
11. wire stripper

12. bare circuit board 4.5x6"
13. soldering iron, cutter, toolbox (maybe too small?)
14. toolbox 12 x7x7.5"
15. multimeter
16. 9V battery for multimeter
17. op amp
18. .1uf capacitor (4x)
19. 4.7 or 10uf capacitors (*4/student)
20. microphone
21. safety glasses
22. heat shrink kit – couple pieces of small sizes

Table 1. Electronics kit components.

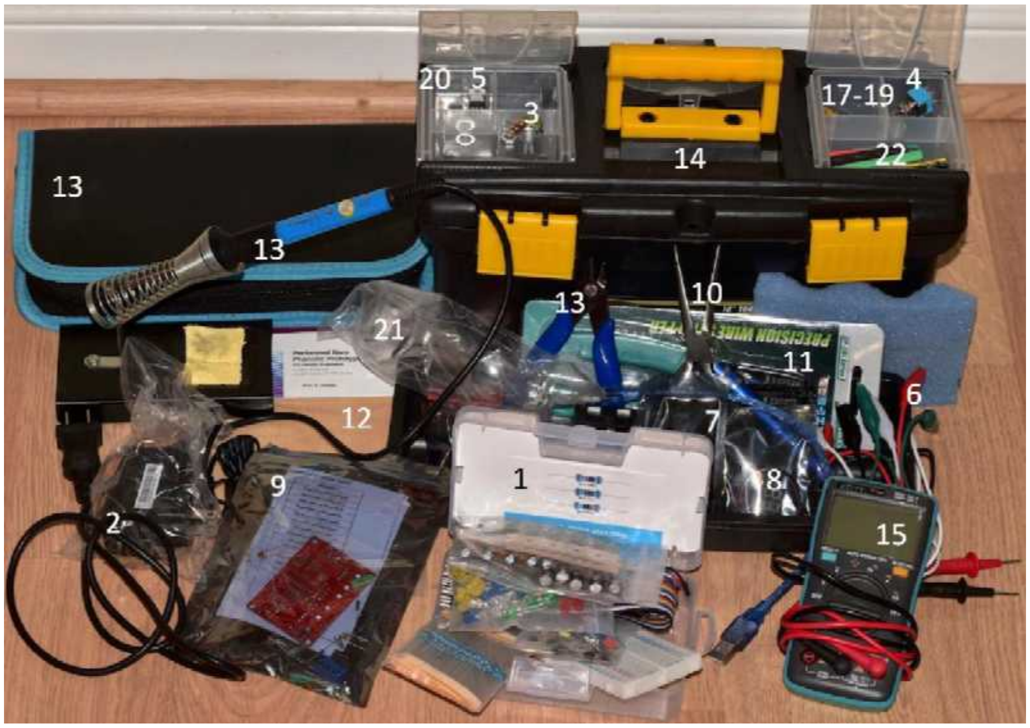


Figure 2. Toolkit components, numbered according to Table 1. All components fit inside the soldering kit (13) and toolbox (14).

Software & Computer requirements

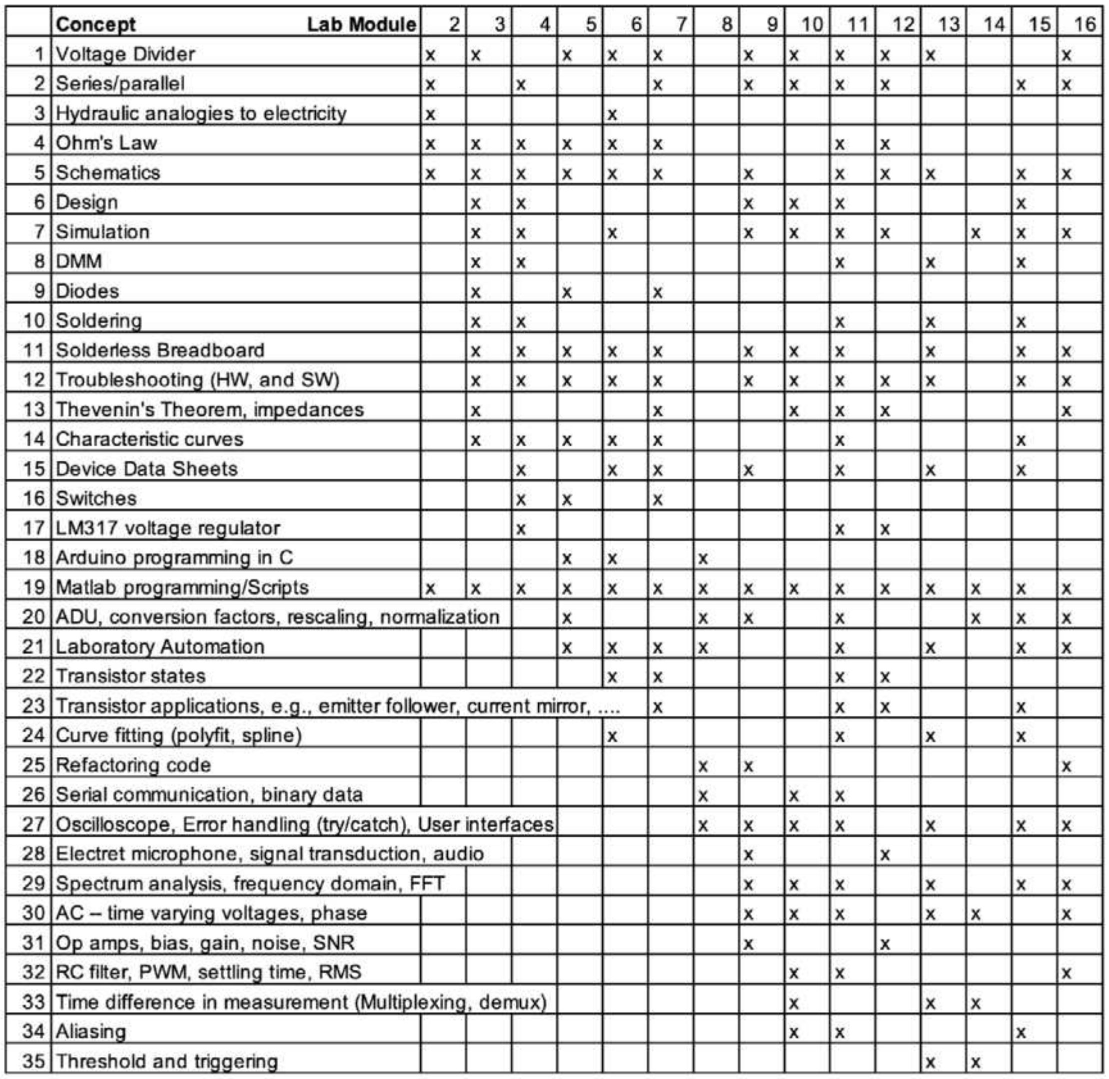
Arduino IDE is freely downloadable [1].You'll also need to install a (free) CH340 serial driver [2]. MATLAB is downloadable free through the University MATLAB Portal [3]. This is an expensive piece of software that's extraordinarily stable and well-supported. A public domain version of MATLAB exists called Octave (https://octave.org), but it requires a little more work to configure and run this in Octave. MATLAB also has a web-based version that will *not* work for this class, as it does not support interfacing with hardware like your computer's serial (USB) port – so installing the executable is necessary. If you're skilled with Python and familiar with its numerical and graphical libraries, it should be possible to translate the enclosed code, but I expect the result would be significantly more verbose. We definitely take advantage of MATLAB's expressive syntax and extensively developed built-in feature sets so the data interfaces are minimally complex and minimally obfuscate the broader goals of learning electronics.

MATLAB has significant resource requirements and is much more enjoyable running on fast computers with adequate memory. We won't be pushing MATLAB very hard for these activities. Expect that if you're not familiar with installing software, you may require assistance to get MATLAB and Arduinos working ... don't give up. Bring your laptop to the Computer Center or class for assistance.

Chapter List & Concept Matrix

1. Introduction
2. Voltage and Current – and introduction to MATLAB
3. LED nightlight – recycling a wall wart; characteristic curves
4. *LM317 DC power supply – data sheets; soldering
5. Automated I-V curves – Arduino data acquisition, voltage dividers
6. *Transistor gain – measuring transistor characteristics
7. BJT and FET switches – and photoresistor control of LED nightlight
8. MATLAB meets Arduino – serial ports and binary data
9. *Op-amps, microphone, oscilloscope, and spectra
10. PWM filters – and frequency domain analysis
11. *Arduino Controlled LM317 – applied transistors and filters
12. Modeling the Automated LM317 –explore electronic theory in MATLAB
13. Function generator – adding amplitude, frequency and phase measurements to our oscilloscope
14. Frequency domain deskewing – correcting temporal skew in the frequency domain
15. *Arduino Controlled Function generator – adding software frequency control
16. Bode plot – automated frequency domain measurements of an RC circuit

* indicates graded lab reports



	Concept **Lab Module**	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	Voltage Divider	x	x		x	x	x		x	x	x	x	x			x
2	Series/parallel	x		x			x		x	x	x	x			x	x
3	Hydraulic analogies to electricity	x				x										
4	Ohm's Law	x	x	x	x	x	x				x	x				
5	Schematics	x	x	x	x	x	x		x		x	x	x		x	x
6	Design		x	x					x	x	x				x	
7	Simulation		x	x		x			x	x	x	x		x	x	x
8	DMM		x	x							x		x		x	
9	Diodes		x		x		x									
10	Soldering		x	x							x		x		x	
11	Solderless Breadboard		x	x	x	x	x		x	x	x		x		x	x
12	Troubleshooting (HW, and SW)		x	x	x	x	x		x	x	x	x	x		x	x
13	Thevenin's Theorem, impedances		x				x			x	x	x				x
14	Characteristic curves		x	x	x	x	x				x				x	
15	Device Data Sheets			x		x	x		x		x		x		x	
16	Switches			x	x		x									
17	LM317 voltage regulator			x							x	x				
18	Arduino programming in C				x	x		x								
19	Matlab programming/Scripts	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
20	ADU, conversion factors, rescaling, normalization				x			x	x		x			x	x	x
21	Laboratory Automation				x	x	x	x			x		x		x	x
22	Transistor states					x	x				x	x				
23	Transistor applications, e.g., emitter follower, current mirror, ...						x				x	x			x	
24	Curve fitting (polyfit, spline)					x					x		x		x	
25	Refactoring code							x	x							x
26	Serial communication, binary data							x		x	x					
27	Oscilloscope, Error handling (try/catch), User interfaces							x	x	x	x		x		x	x
28	Electret microphone, signal transduction, audio								x			x				
29	Spectrum analysis, frequency domain, FFT								x	x	x		x		x	x
30	AC – time varying voltages, phase								x	x	x		x	x		x
31	Op amps, bias, gain, noise, SNR								x			x				
32	RC filter, PWM, settling time, RMS									x	x					x
33	Time difference in measurement (Multiplexing, demux)									x			x	x		
34	Aliasing									x	x				x	
35	Threshold and triggering												x	x		

Figure 3. Concept Matrix shows how key concepts are repeated in many chapters (columns).

Chapter Summaries

Each chapter tends to build on the previous ones, so I highly recommend following them in order. Chapters 12 and 14 can easily be hurried or skipped. A more technical summary is in my AJP article [4].

Chapter 2 is a gentle and practical introduction to MATLAB's syntax and modeling and plotting capabilities, applied to understanding Ohm's and Kirchhoff's Laws. Even a simple circuit with two series resistors can be counter-intuitive, and we'll have to work hard to learn and unlearn concepts.

Chapter 3 presents a design challenge to make an LED nightlight from a recycled DC power supply. A goal is to show how nothing is assumed about the components. Their electrical characteristics can be measured and quantified. A design is created, prototyped, and constructed. The chapter ends by motivating construction of a regulated DC power supply.

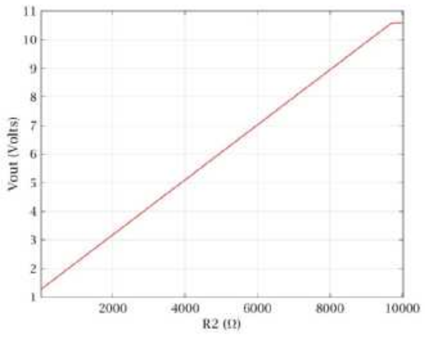
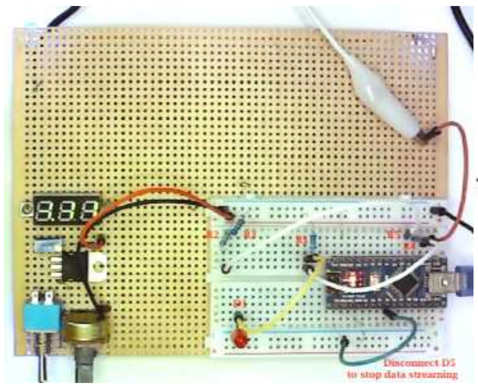


Figure 4. LED nightlight and LM317 simulation.

Chapter 4 builds our first key piece of electronic test equipment, a regulated DC power supply. Guided by the LM317 datasheet and measurements, we optimize a design with components in our kits, prototype it on a solderless breadboard, and solder the circuit on a perforated breadboard.

Chapter 5 repeats the characterization of an LED's I-V curve, but now with much better tools: a DC power supply generates a linear voltage ramp, an Arduino automates measuring all the voltages, and MATLAB converts the measurements into currents and graphs. We use automation to also consistently name variables and bind them to their data, and finally automate generation of MATLAB source code to plot the voluminous results (752 voltages measured from 9 devices shown in Fig. 5B). Repetition and comparing results help us learn.



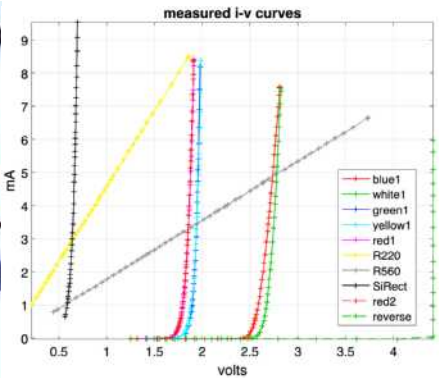


Figure 5. LM317 power supply powering I-V test circuit (left) and measured I-V curves of 9 resistors, rectifier diode, and LEDs (right).

Chapter 6 introduces NPN bipolar junction transistors (BJT) in their active region, and we program the Arduino and MATLAB to essentially become a "curve tracer", measuring the current gain, $\beta = h_{fe}$ of a transistor by sweeping across a range of base currents with various saturation currents. Repetitive measurements under varying conditions (easily done without tedium thanks to automation) demonstrate the relative constancy of β.

Chapter 7 explores switching behavior of the NPN BJT and compares it to a MOSFET. We explore two applications: switching high currents with relays and solenoids (and snubber circuits), and characterizing a CdS photoresistor from the kit to make a transistor switch that turns an LED on in the dark and off in the light.

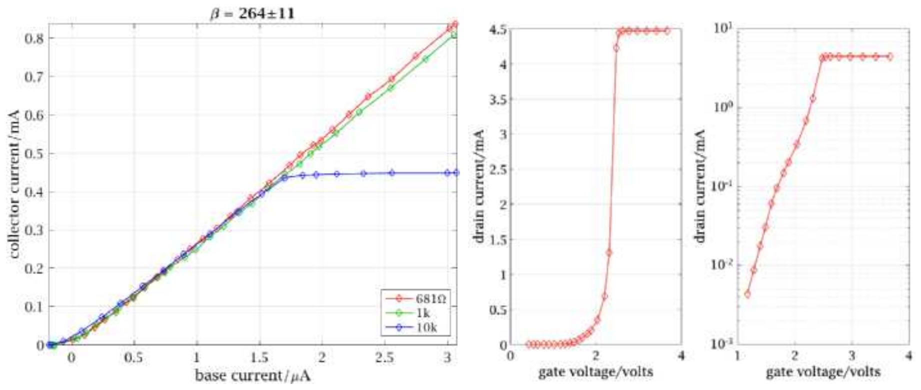


Figure 6. BJT active region (left) and FET switch currents (right).

Chapter 8 develops software so Arduino and MATLAB communicate with each other through a serial (USB) connection. Anticipating studying fast time-varying (AC) signals, we focus on maximizing the bandwidth and speed of communication and data acquisition. We program a simple command parser on the Arduino, and a simple oscilloscope display with a graphical user interface (GUI) in MATLAB.

Chapter 9 begins by measuring the output of an electret microphone using our oscilloscope display to visualize our voice. The low signal motivates design and prototyping an op-amp amplifier, exploring bias, signal-to-noise ratio (SNR), and negative gain. Again we compare measurements with theory, and conclude by tweaking our oscilloscope software to display spectra and compare signals in time and frequency domains.

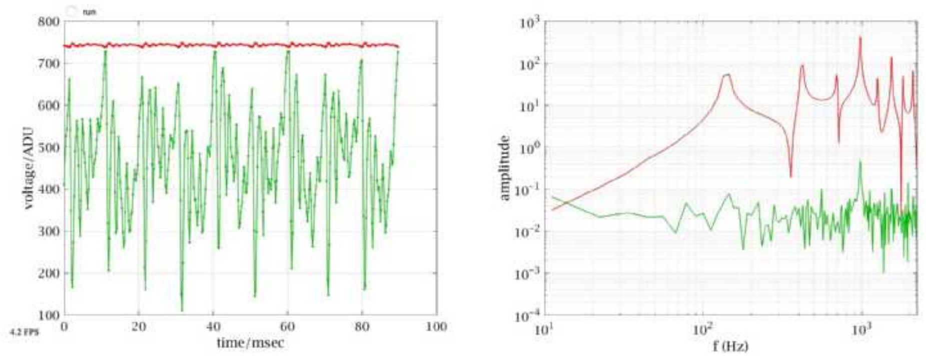


Figure 7. Raw and amplified microphone signal of voice (left) and amplitude spectra of Arduino's PWM output before and after a 2-stage low-pass filter (right).

Chapter 10 continues exploration of AC signals with our oscilloscope and spectrum analyzer to understand Arduino's pulse width modulated (PWM) `analogWrite` signals. Beginning a "bottom up" approach to controlling our LM317 with the Arduino, we explore how to convert Arduino's PWM output into clean DC with 1st and 2nd order RC filters. We use theory and measurements to explore gain and transient settling time for the filters.

Chapter 11 combines our PWM filter with a pair of transistor circuits (emitter follower and current mirror) to add software control to the LM317. The chapter concludes by writing and calibrating a simple MATLAB function to set the LM317 output voltage.

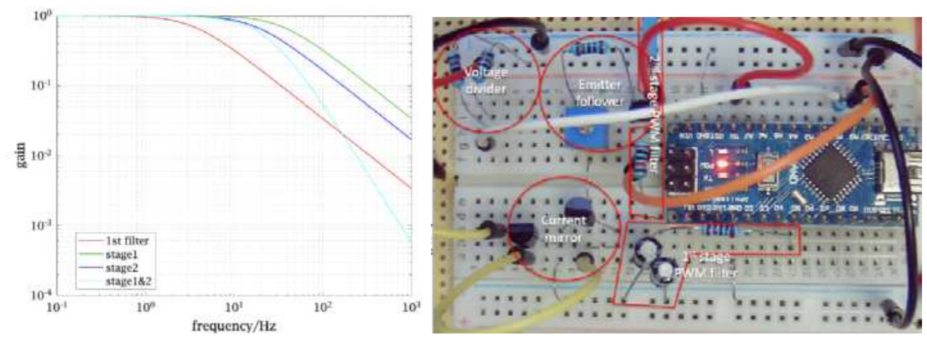


Figure 8. PWM filter models (left) and prototype 3-transistor LM317 controller (right).

Chapter 12 builds a basic MATLAB model of the LM317 controller circuit, and adds a couple of "perturbation" terms to the model to improve its fit with measurements. This chapter could be skipped if time constrained.

Chapter 13 begins by validating construction of a function generator kit using the Arduino and MATLAB oscilloscope display. The need for stable triggering immediately becomes apparent, which we implement in MATLAB, while simultaneously computing the signal's frequency and amplitude. We discover the two analog channels are interleaved, resulting in a temporal skewing or shift of one relative to the other. This interferes with phase measurements, so we fix this systematic error by adding a couple of code lines invoking a cubic spline interpolation algorithm.

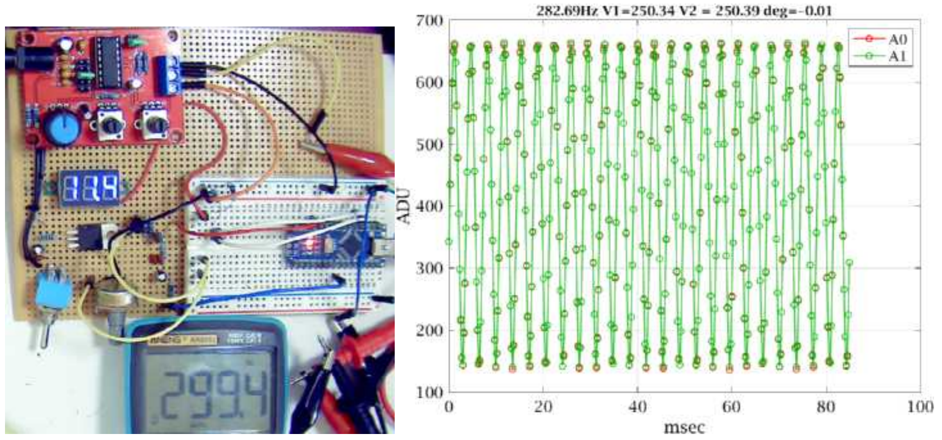
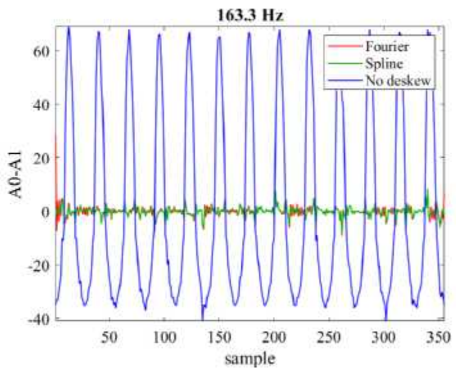


Figure 9. Function generator kit, power supply, and control circuit mounted on perforated breadboard (left), deskewed dual channel sine wave output (right), with amplitudes, frequency, and phase computed from the Arduino data and displayed above the graph. The same input on both channels results in ~0.01 degree phase shift , corrected from ~11.4 degree shift in the raw measurements due to temporal multiplexing.

Chapter 14 is a mathematical dive into correcting temporal skew between the oscilloscope channels in Fourier space. We see how radically different frequency domain algorithms are. Deskewing in the frequency domain is accomplished by phasor multiplication, and we measure the algorithm outperforms splines at higher frequencies.

Chapter 15 explores how the function generator's frequency is controlled, and then repurposes our LM317 voltage controller circuit to automate frequency control. We modify the function generator's timing capacitors to maximize the frequency range that Arduino can set and measure, and write a simple MATLAB function to set its oscillation frequency.



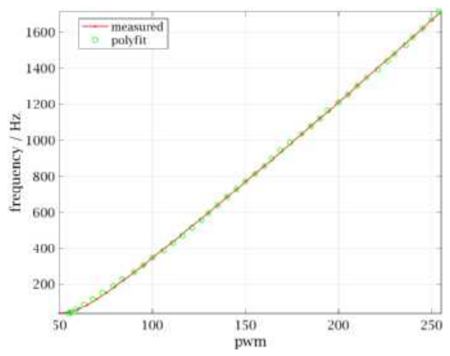
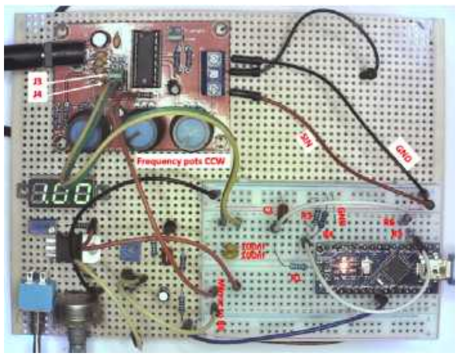


Figure 10. Systematic interchannel error is eliminated with time and frequency domain algorithms (left). Software frequency control is measured and modeled (right).

Chapter 16 begins by refactoring our oscilloscope code with data structures to clean up a complicated workspace. We then measure a Bode Plot (AC amplitudes and phases vs. frequency) in an RC series circuit using automation. Around 20 lines of MATLAB code 1) tells the Arduino to sweep 50 different frequencies and measure ~40,000 voltages in <45 seconds; 2) compute voltage amplitudes and phases across R and C at each frequency and simulate the circuit; and 3) compare results. Measurements and simulations are nearly identical, demonstrating ~1% accuracy and precision of our inexpensive apparatus.



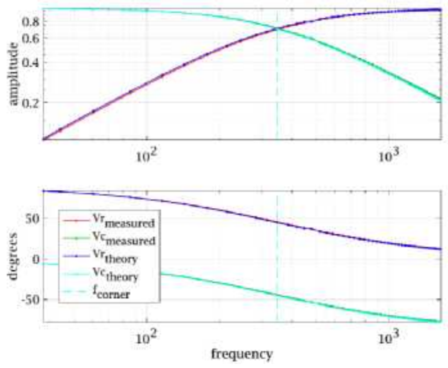


Figure 11. Automated Bode Plot of an RC series circuit was measured and simulated at 50 frequencies in <45 seconds.

Students can keep their kits and use them to continue exploring electronics. Lab reports are documentation for future reference. As is often the case, the journey is more valuable and educational than arriving at the destination. Each chapter of this journey involved constructing and debugging apparatus of increasing complexity (hardware, software in C, and MATLAB code working synergistically) to measure and enlighten understanding of electronic devices and circuits. For most students, this was also a novel way of using a computer, as a lab automation partner. Dialoging at a command line to control and debug an automated apparatus, acquire data, and perform exploratory data analyses, supercharges our intellectual capabilities and our productivity. That's why such skills are highly valued by scientists and employers.

Supplementary Materials

The chapters are relatively sparse in electronics theory, and are meant to be supplemented by other sources. Here are a few:

1. Boysen E & Kybett H (2012) "Complete Electronics Self-Teaching Guide with Projects". Offers lots of simple numerical exercises with answers.

2. Horowitz & Hill Art of Electronics (AoE, https://artofelectronics.net) is a fantastic reference book going into greater depth than most students can manage in their first upper division electronics course. This course should prepare them for AoE.
3. Learning the Art of Electronics (https://learningtheartofelectronics.com) helps with the journey through AoE.
4. Khan Academy, youtube, offer useful perspectives and explanations.
5. Analog Devices course

References

1. Download the Arduino IDE at https://www.arduino.cc/en/software.
2. Download the Arduino Serial driver at:
https://learn.sparkfun.com/tutorials/how-to-install-ch340-drivers/all
3. Download MATLAB from the University's MATLAB Portal. You only *need* basic MATLAB, any "toolboxes", Simulink, etc. are optional. MATLAB's web version does not work for this class, as it doesn't communicate with an Arduino connected to a local serial port.
4. Rasnow B (2024) Flipping the electronics lab: Learning upper division electronics at home. Amer. J. Phys. 92:809-818. https://doi.org/10.1119/5.0206534.