Automating Measurements In An Electronics Lab

Maximilian Seligman, Laura Medina, Edgar Hernandez, Elliot Brown, Dr. Brian Rasnow

Introduction

PHYS 310 is a hands-on electronics class. We build our own power supply, then build a function generator which enable us to "flip the lab" and conduct various experiments outside of the classroom. Although the primary focus is analog, we end by exploring Arduino microcontrollers and the digital regime of electronics. Here we demonstrate a novel way to integrate an Arduino with the "analog side" of our studies. We designed and implemented hardware and software interfaces enabling an Arduino (Nano or Uno) to control our function generator, and concurrently measure the responses of simple circuits connected to the function generator. These measurements, which historically are tediously done manually with oscilloscopes, were simply pasted from the Arduino serial monitor into Matlab, where the analysis was completed.

Methods

1. Automated frequency control: The function generator we used is based on the Exar XR-2206 function generator chip (Figure 1). It's frequency is controlled by sinking current from its pin 7 according to f(Hz) = 320 I_T(mA)/C(µf)[1]. Automated control began by using a pulse-width-modulated (PWM) output on the Arduino, which varies a duty cycle of a 980Hz digital output [2]. This output was filtered in order to maintain a steady function generator frequency using an RC low-pass filter, however much better stability was achieved by adding a signal diode in series with the R. High pulses charged the capacitor rapidly, but the diode blocked rapid discharge during low phases. (Figure 1). We used a current mirror to convert this filtered analog voltage to a current sink on pin 7 (Figure 2).

2. Automated Frequency Measurement: The function generator's TTL output was connected to an Arduino digital input pin (and a digital frequency counter for calibration). Arduino's pulseIn() function measures the half-period with a 1 microsecond clock, and the following code converted this to frequency (Hz):
```
  freq = 500000.0 /  pulseIn(TTLPIN, HIGH);
```

3. Automated waveform measurement: The function generator produces symmetric positive and negative going waveforms, but the Arduino's analog-to-digital converter (ADC) requires input between 0-5V, thus bias networks are required (Fig. 2).
4. Completed Circuit: (Figure 3).

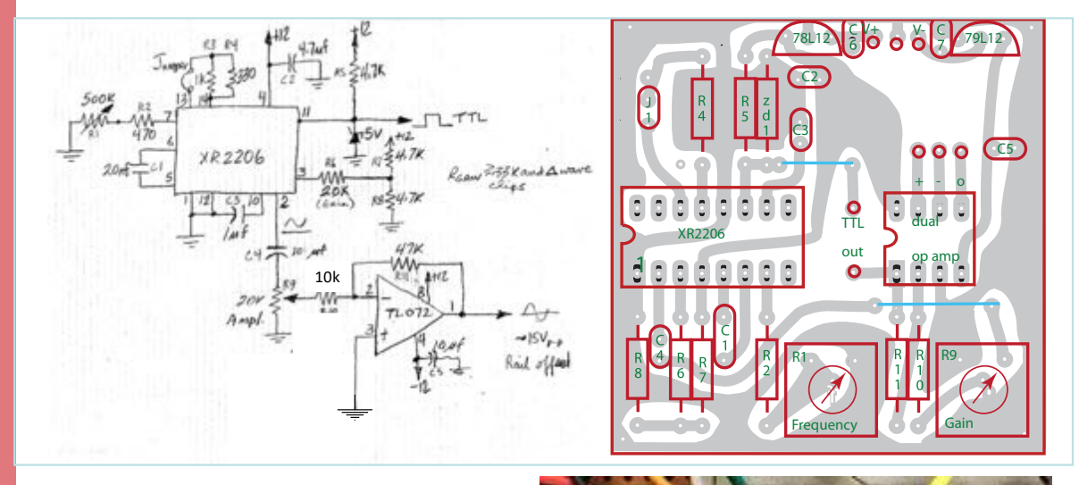


Figure 1. Function generator designed and built in Phys 310. This yellow wire enabled remote control with an Arduino.

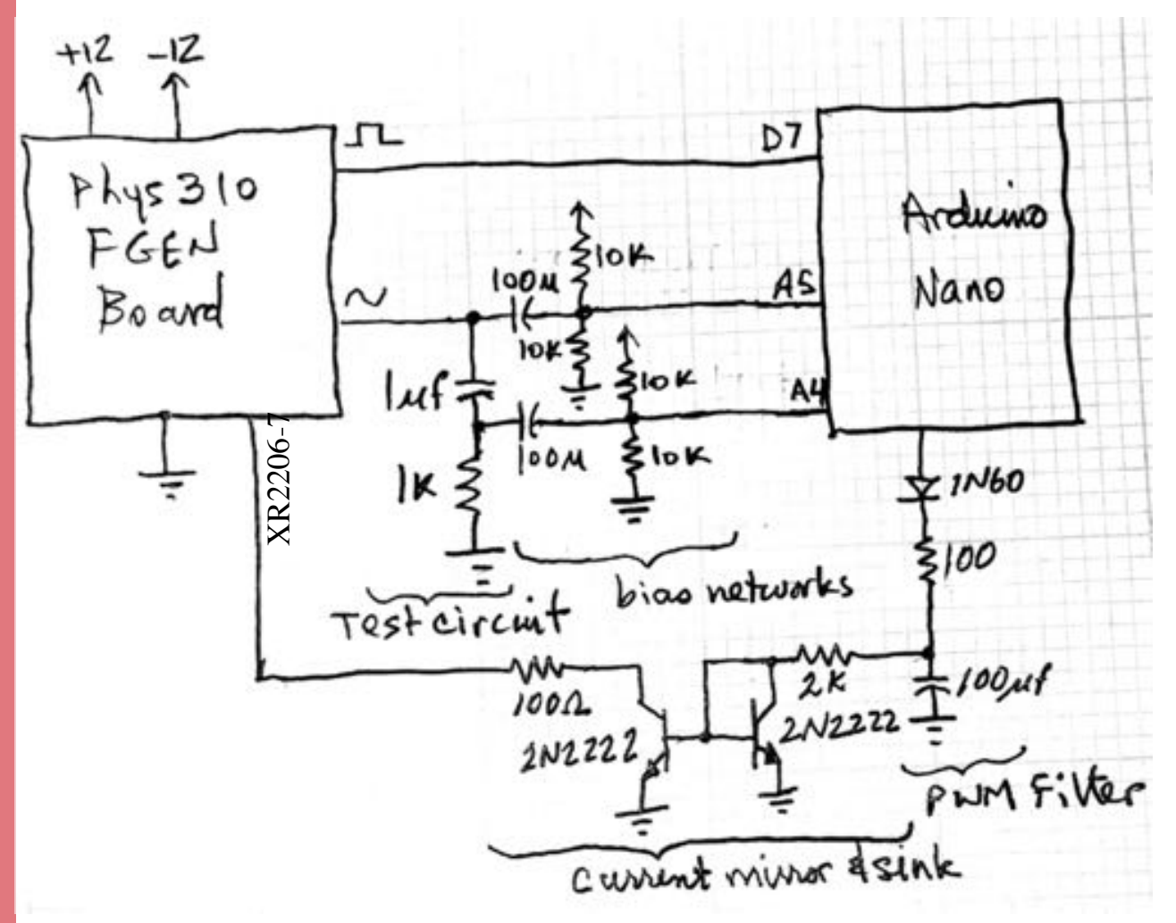


Figure 2. Schematic of Arduino-function generator hardware interface.

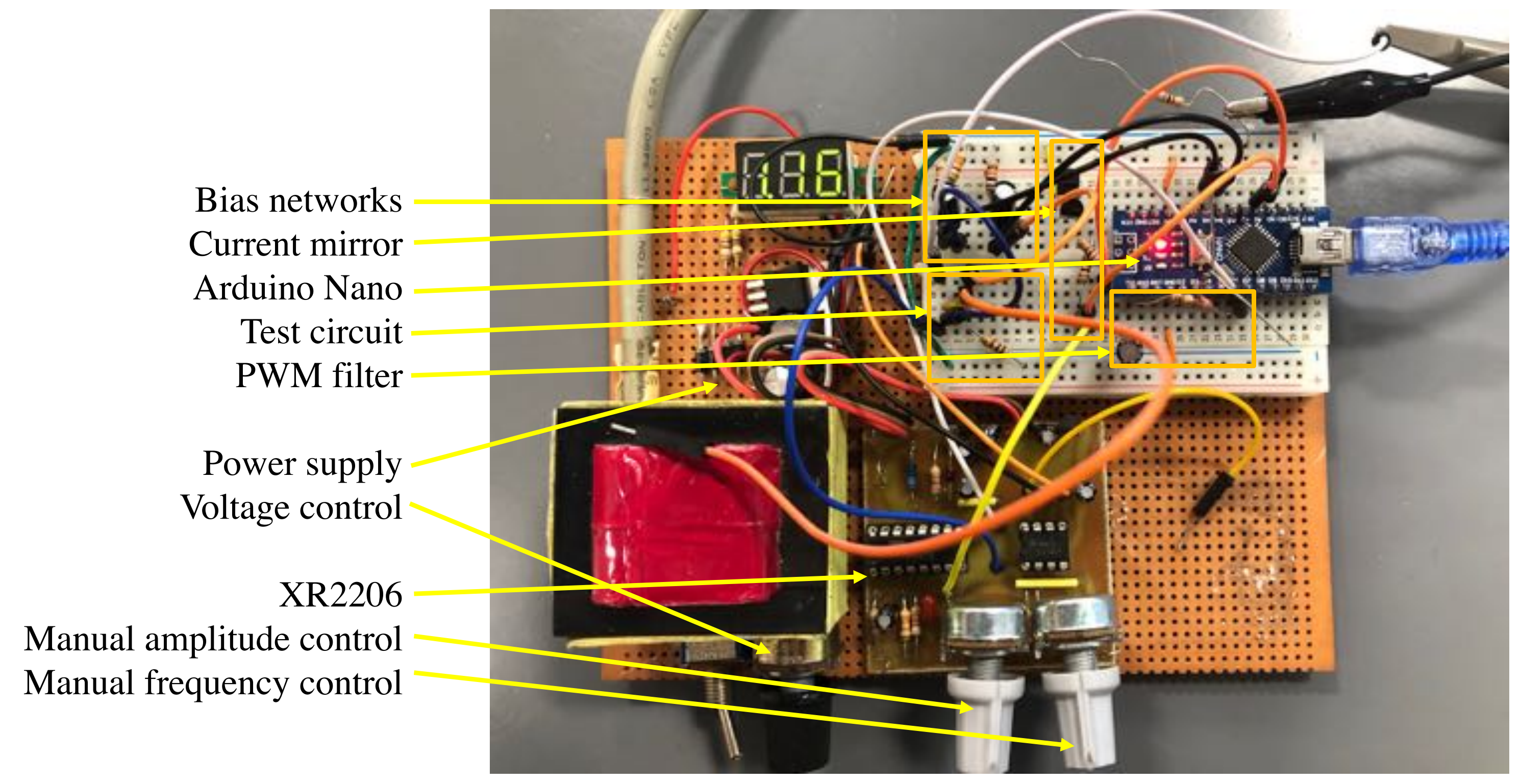
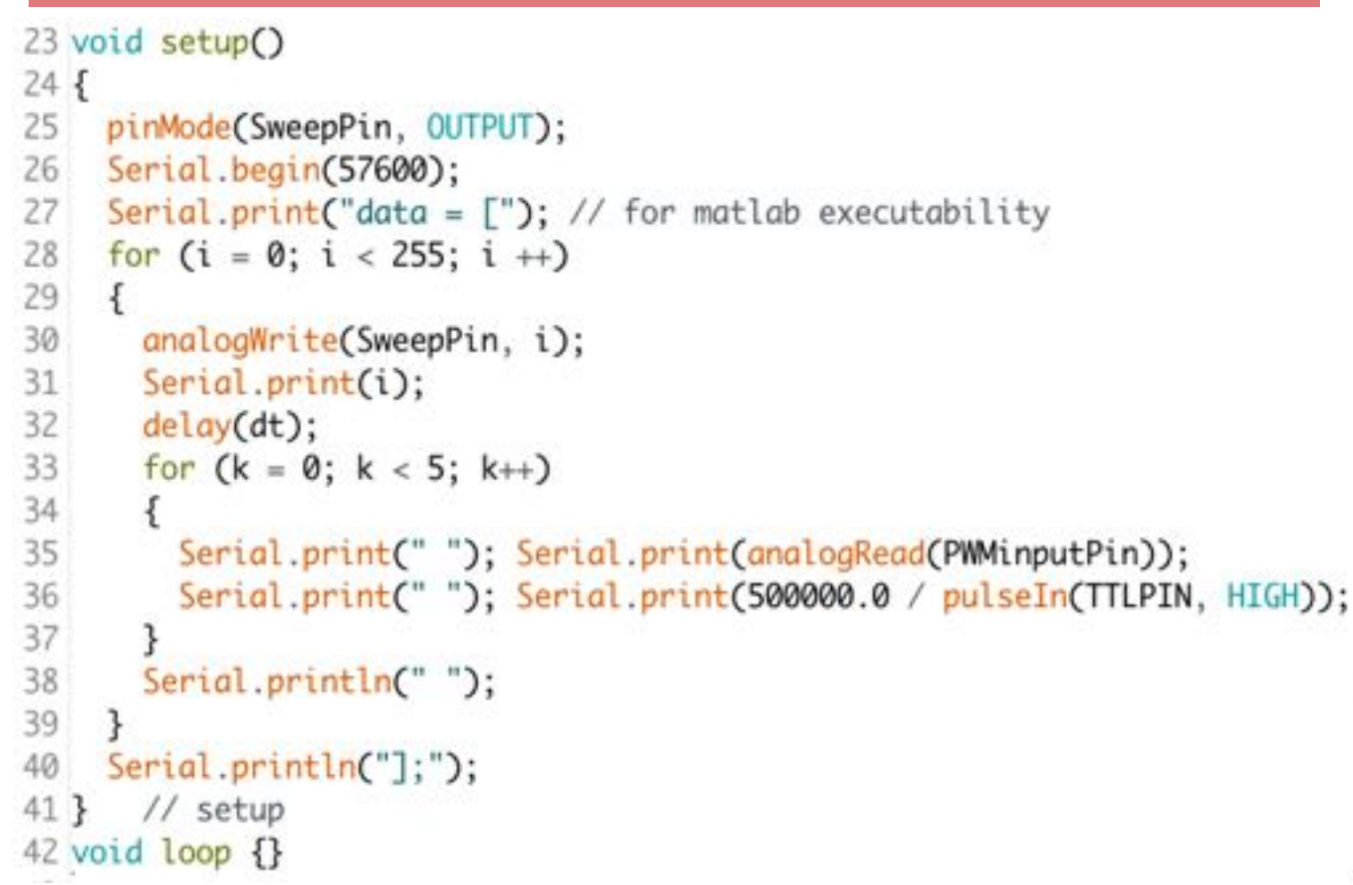


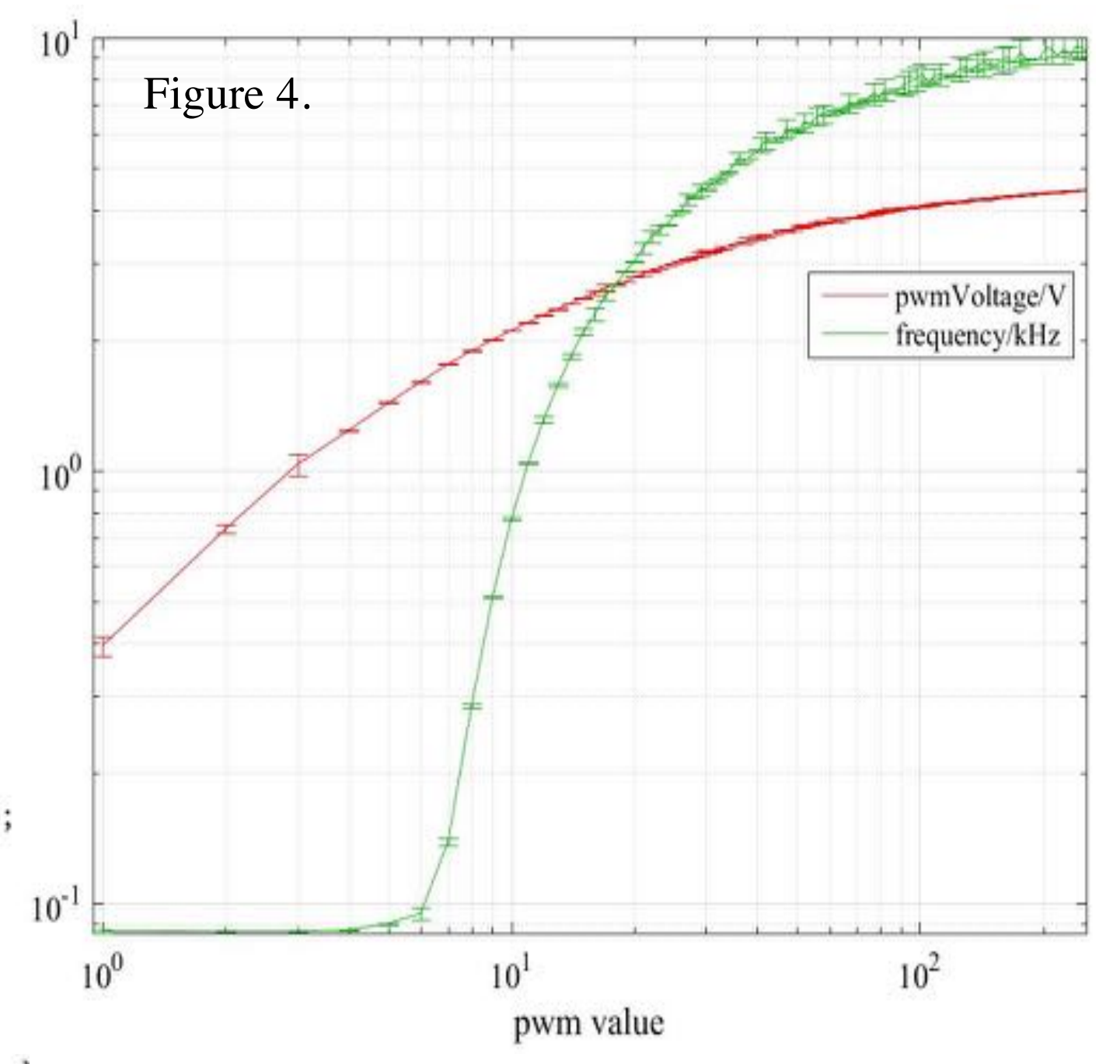
Figure 3. Photo of circuit board each Phys 310 student constructed.

Results

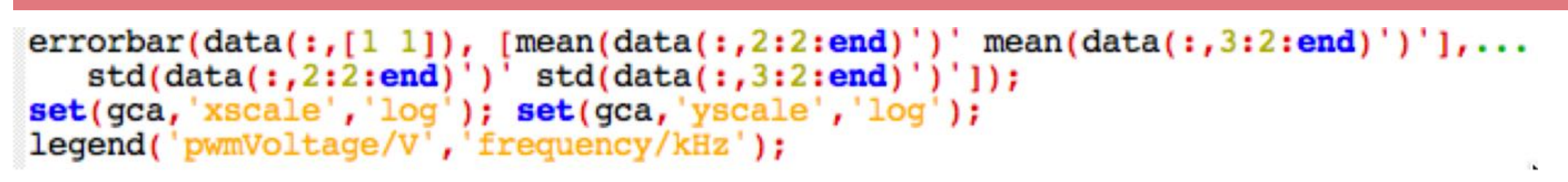
Figure 4 shows the relationship between pulse width modulation (PWM) value and the function generator frequency. Both frequencies and voltages were measured automatically by the Arduino from this code:



```
void setup()
{
  pinMode(SweepPin, OUTPUT);
  Serial.begin(57600);
  Serial.print("data = ["); // for matlab executability
  for (i = 0; i < 255; i ++)
  {
    analogWrite(SweepPin, i);
    Serial.print(i);
    delay(dt);
    for (k = 0; k < 5; k++)
    {
      Serial.print(" "); Serial.print(analogRead(PWMinputPin));
      Serial.print(" "); Serial.print(500000.0 / pulseIn(TTLPIN, HIGH));
    }
    Serial.println(" ");
  }
  Serial.println("];");
}   // setup
void loop {}
```

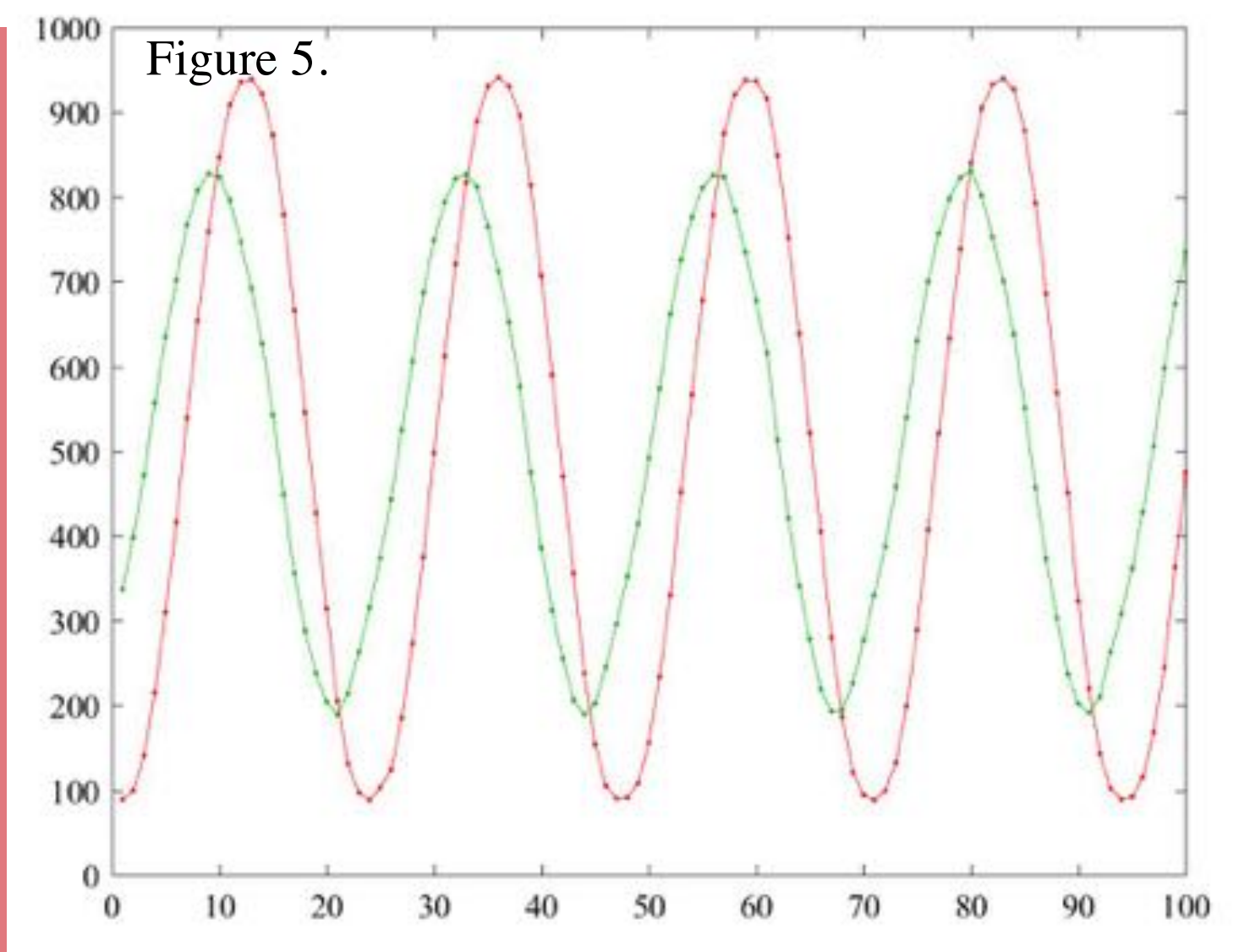


Pasting the Serial Monitor output into Matlab and running the following analysis generated Fig. 4:

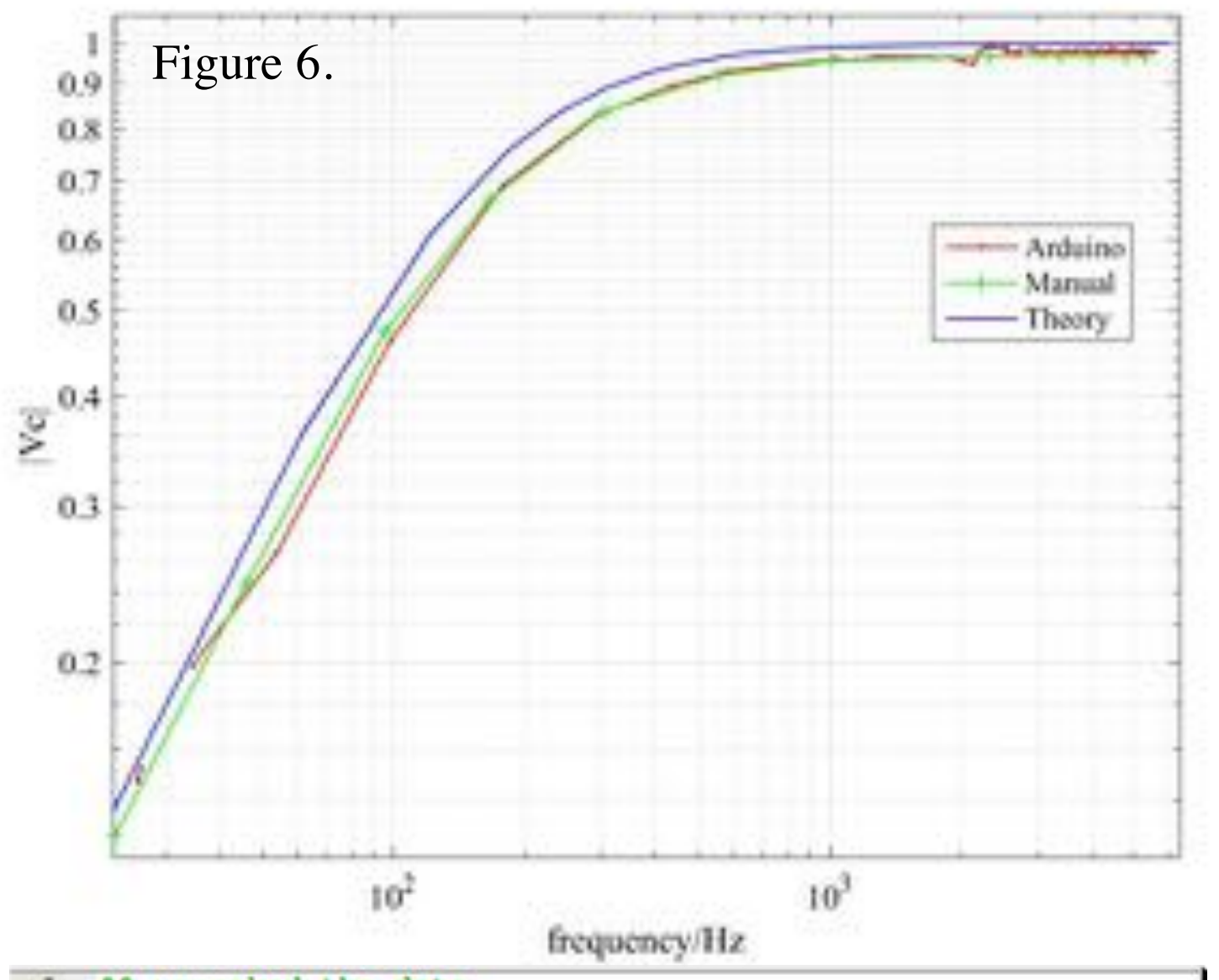


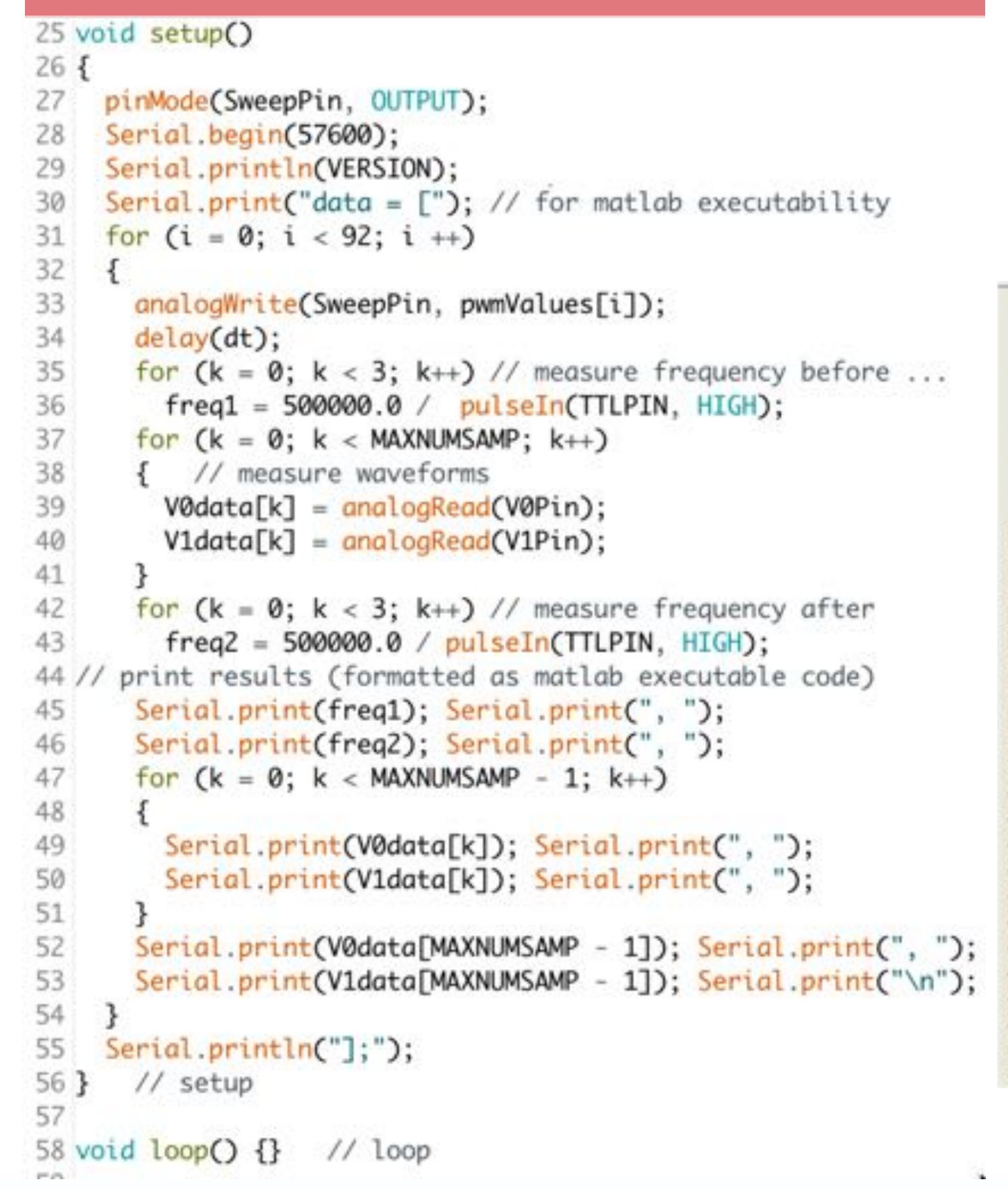
```
errorbar(data(:,[1 1]), [mean(data(:,2:2:end)')' mean(data(:,3:2:end)')'],...
   std(data(:,2:2:end)')' std(data(:,3:2:end)')']);
set(gca,'xscale','log'); set(gca,'yscale','log');
legend('pwmVoltage/V','frequency/kHz');
```

Measuring analog waveforms (Figure 5). A simple series R-C circuit was connected to the function generator. Fig. 5 shows the function generator output and the voltage across the resistor, both measured concurrently by the Arduino, through the bias network (Fig. 2). The Arduino's analog to digital converter (ADC) runs at ~9615 Hz, multiplexed between the two channels, and returns integers between 0-2^{10}=1024. The Arduino was programmed to measure 100-300 samples on each channel as fast as it could, and then print the results (interleaved) out its serial port (as ascii). The print format included characters to enable simple copy/paste from the Serial Monitor window directly into Matlab's command line or an .m file.



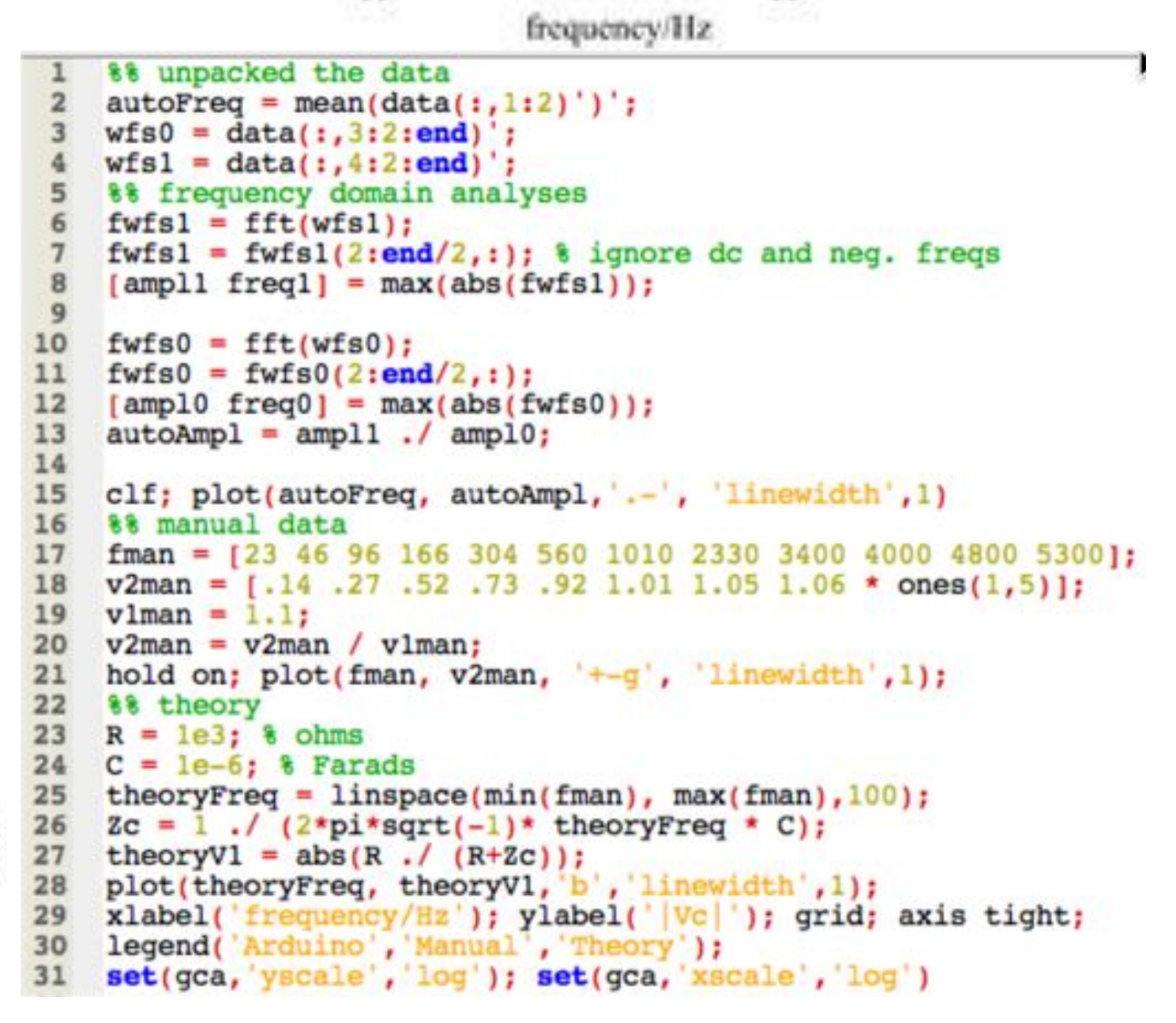
Measuring Circuit Responses in the Frequency domain (Figure 6): The primary goal of this exercise was to measure amplitude (and phase) responses of an RC circuit. The Arduino was programmed to change the frequency and measure waveforms after a 200msec delay. Arduino and Matlab source code generating Fig. 6 are below. Also shown are manually recorded data from an Owon oscilloscope, and simulations from circuit theory.





```
void setup()
{
  pinMode(SweepPin, OUTPUT);
  Serial.begin(57600);
  Serial.println(VERSION);
  Serial.print("data = ["); // for matlab executability
  for (i = 0; i < 92; i ++)
  {
    analogWrite(SweepPin, pwmValues[i]);
    delay(dt);
    for (k = 0; k < 3; k++) // measure frequency before ...
      freq1 = 500000.0 /  pulseIn(TTLPIN, HIGH);
    for (k = 0; k < MAXNUMSAMP; k++)
    {   // measure waveforms
      V0data[k] = analogRead(V0Pin);
      V1data[k] = analogRead(V1Pin);
    }
    for (k = 0; k < 3; k++) // measure frequency after
      freq2 = 500000.0 / pulseIn(TTLPIN, HIGH);
// print results (formatted as matlab executable code)
    Serial.print(freq1); Serial.print(", ");
    Serial.print(freq2); Serial.print(", ");
    for (k = 0; k < MAXNUMSAMP - 1; k++)
    {
      Serial.print(V0data[k]); Serial.print(", ");
      Serial.print(V1data[k]); Serial.print(", ");
    }
    Serial.print(V0data[MAXNUMSAMP - 1]); Serial.print(", ");
    Serial.print(V1data[MAXNUMSAMP - 1]); Serial.print("\n");
  }
  Serial.println("];");
}   // setup

void loop() {}   // loop
```



```
%% unpacked the data
autoFreq = mean(data(:,1:2)')';
wfs0 = data(:,3:2:end)';
wfs1 = data(:,4:2:end)';
%% frequency domain analyses
fwfs1 = fft(wfs1);
fwfs1 = fwfs1(2:end/2,:); % ignore dc and neg. freqs
[ampl1 freq1] = max(abs(fwfs1));

fwfs0 = fft(wfs0);
fwfs0 = fwfs0(2:end/2,:);
[ampl0 freq0] = max(abs(fwfs0));
autoAmpl = ampl1 ./ ampl0;

clf; plot(autoFreq, autoAmpl,'.-', 'linewidth',1)
%% manual data
fman = [23 46 96 166 304 560 1010 2330 3400 4000 4800 5300];
v2man = [.14 .27 .52 .73 .92 1.01 1.05 1.06 * ones(1,5)];
v1man = 1.1;
v2man = v2man / v1man;
hold on; plot(fman, v2man, '+-g', 'linewidth',1);
%% theory
R = 1e3; % ohms
C = 1e-6; % Farads
theoryFreq = linspace(min(fman), max(fman),100);
Zc = 1 ./ (2*pi*sqrt(-1)* theoryFreq * C);
theoryV1 = abs(R ./ (R+Zc));
plot(theoryFreq, theoryV1,'b','linewidth',1);
xlabel('frequency/Hz'); ylabel('|Vc|'); grid; axis tight;
legend('Arduino','Manual','Theory');
set(gca,'yscale','log'); set(gca,'xscale','log')
```

Discussion

Figure 6 demonstrates that a $5 Arduino microcontroller with a simple analog interface (Figure 2) can accurately automate frequency domain measurements, by controlling a function generator and concurrently recording analog voltages. Within a several seconds, it recorded 27600 data points at 92 different frequencies. Automated, manual, and modeled data all agree within expected tolerances and experimental error limits. Even when sampling slightly *above* the Nyquist Frequency, fsample/2 ~ 4.8kHz, the system provided accurate results, because the frequency was measured with a 1 MHz clock via `pulseIn`, and aliasing doesn't affect the *amplitude* of the measured signal.

The advantage of automating the collection of data with the Arduino is that not only is it faster to collect the data, but it also makes it a lot faster to repeat the experiment with different values of different test circuits and take the same measurements. We can then integrate this data and create graphs with many different result curves. While coding the program to control and automate the Arduino takes some time, it can easily be used, reused, and repurposed to collect data in a variety of experiments where measuring frequency responses is needed. This reduces the time and effort needed to collect simpler forms of data to allow us to use our efforts toward trying to analyze and process the data instead of collecting it in an otherwise time consuming manner.

Compared to manually collecting data, where we would need to change variables, measure, record data, and then input this data into a program to analyze it like Matlab, we can use the Arduino to generate an output that we can copy and paste into Matlab to conduct our data analysis.

The copy-paste interface between the Arduino's serial output and Matlab takes nearly as long as does data acquisition. With relatively little effort this interface can be automated by quitting the Arduino IDE and programming Matlab to connect to the Arduino through its `serial` command.

For future experiments with this project, it would be interesting to change the characteristics of the test circuits and explore the measureable range of the function generator. Also measuring phase is important but proved more challenging. The systematic ~1msec latency between the two channels must be corrected, as well as dealing with the multivalued nature of phase (i.e., $\theta = \theta \pm n*360^\circ$).

Conclusions

- A $5 Arduino microcontroller can automate control of a function generator and concurrently measure the frequency responses of a test circuit.
- The process of bootstrapping the building of a power supply and function generator and using them to study simple circuit behavior was accomplished by most students, teaching them many practical lessons about electronics, troubleshooting, abstraction and reductionism, etc. Adding numerous components of hardware and software interfaces between them and an Arduino and Matlab was overwhelming for many students.
- Automating measurements runs a risk of depriving students of understanding the underlying processes. This exercise showed that risk was also present in the manual approach – where students could tabulate data tediously read from the oscilloscope with perhaps even less appreciation for its meanings than the richer automated data streams.
- The process of developing an action plan to develop the circuit and troubleshoot issues was also a challenge. The main challenge with this aspect of the project was trying to determine the best way to solve a problem, either in the software, or changing the components used in the circuit.
- This project showed students in a hands-on manner the power of the components that were used as well as helped them develop problem solving skills. Through this approach, students were able to become familiar enough with a deeper understanding of components in order to use them to design circuits and solve problems.

References

1. Exar XR-2206 datasheet, e.g., https://www.sparkfun.com/datasheets/Kits/XR2206_104_020808.pdf
2. https://www.arduino.cc/reference/en/language/functions/analog-io/analogwrite/
3. https://www.futurlec.com/Diodes/1N60P.shtml
4. https://en.wikipedia.org/wiki/Current_mirror
5. https://www.ebay.com/itm/0-56- DIGITAL-Red- LED-Frequency- and-Tachometer- Rotate-Speed- Meter-DC-12-24V/280903275997?epid=1131861035&amp;hash=item416723d5dd:g:n5AAAOSwgNRV8PgN
6. https://electronics.howstuffworks.com/microcontroller.htm
7. https://en.wikipedia.org/wiki/C_(programming_language)
8. https://codewords.recurse.com/issues/one/an-introduction-to-functional-programming
9. https://www.arduino.cc/en/Tutorial/HomePage