IR Temperature Sensor

By Kika Arias & Elaine Ng

MIT 6.101 Final Project

Spring 2020

Introduction (Kika)

The goal of our project is to build an infrared thermometer which is a device that is used to measure the emitted energy from the surface of an object. IR thermometers are used in medical, industrial and home settings for a wide variety of applications. Many electronics retailers sell IR thermometer sensors that perform all of the processing to produce a temperature reading. We wanted to dissect this "black box" to understand how waves of energy can become a human readable temperature.

We learned that IR thermometers consist of three basic stages. A sensing stage, in which IR radiation is converted to an electrical signal, a signal conditioning stage, where filtering, amplification, and linearization are performed, and a digital output stage, where the analog signal is converted into a digital one. Our project replicates these basic stages to create a simulated IR temperature sensor.

System Overview (Kika)

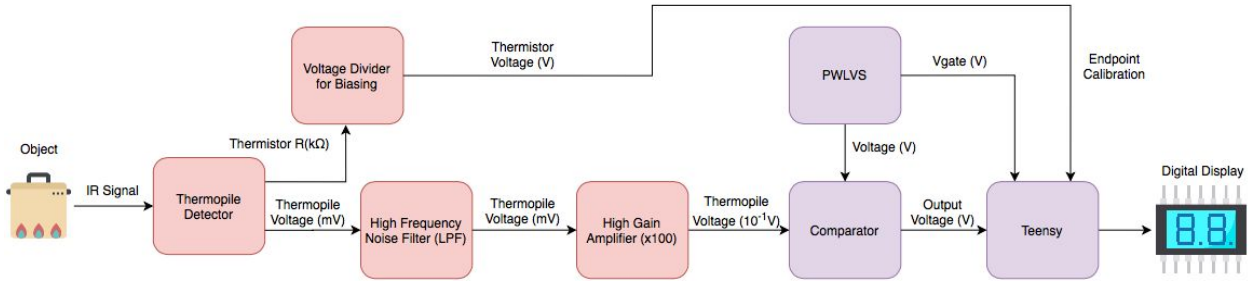


Fig 1. System Block Diagram. Thermopile signal processing in red, and temperature calculation and digital output in purple.

As mentioned in the previous section, an IR thermometer consists of three basic stages: sensing, signal conditioning, and digital output. Figure 1 shows our system block diagram with the sensing and signal conditioning stages shown in red, and the digital output and additional processing shown in purple.

The sensing stage of our project uses a thermopile sensor which has a negative temperature coefficient (NTC) thermistor for temperature compensation. The thermopile sensor absorbs infrared radiation and converts the IR to a small voltage (typically on the order of μVs). That signal has a 37 nV noise which is filtered by a high pass filter and amplified. The NTC thermistor is a temperature dependent resistor that decreases in resistivity as temperature increases. In order to turn that resistance into a processable signal, we will use a basic voltage divider.

Thermopile voltage is a nonlinear function of temperature given by:

$$V = AT^4 \qquad \textit{Equation 1}$$

Nonlinear scales are difficult to work with in displays. The process of linearizing this voltage can be quite complicated and as a result most temperature sensors do linearization entirely with digital processing. We have created a mostly analog linearization scheme that compares the thermopile voltage to this piecewise linear voltage source (PWLVS), taking advantage of the fact that the thermopile voltage is linear in limited ranges of temperature. Our PWLVS circuit design, which is the focus of many sections later, is perhaps unconventional, but is able to produce a voltage output that is piecewise linear in two regions (although the design could be easily extended to more regions).

The PWLVS will be compared to the thermopile voltage to generate a point of intersection. The resulting comparator output allows us to calculate what temperature the thermopile voltage represents since the on-time is proportional to the length of the temperature measurement cycle. This relationship will be expanded upon in later sections. The comparator signal is rectified to produce a positive signal which can be input into our microcontroller, the Teensy 3.2.

The digital output uses a Teensy 3.2 and takes the thermistor output voltage, the comparator output, and a timing signal, which denotes the length of a temperature reading cycle. The Teensy uses these signals and finds the time-stamp of any rising and falling edges to calculate the temperature reading cycle length, and the on-time of the comparator output in each reading cycle. Those values, along with the thermistor voltage, are converted to a final compensated temperature using equations which will be included in later sections. That temperature value is then output to a TFT display along with some additional graphics.

Thermopile Sensor Model (Kika)

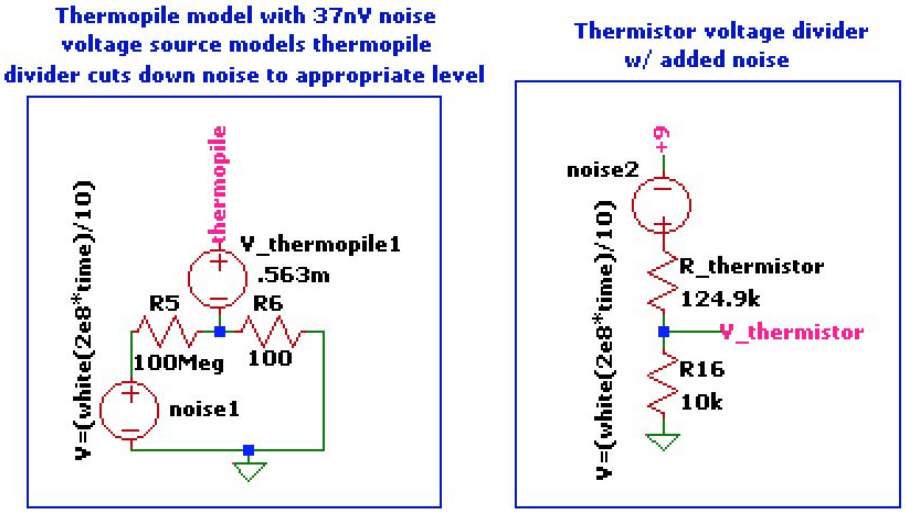


Fig 2. Thermopile and Thermistor Models. (Left) Thermopile voltage model with 37 nV noise. (Right) Thermistor model with voltage divider for signal processing.

For our project we were planning to use the ZTP-148SR Thermopile IR sensor. The sensor package contains an NTC thermistor for temperature compensation. Models for both were created based on the datasheet since Spice models were not available for either.

The thermopile sensor and NTC thermistor can be modeled as a can be modeled as a voltage source and a resistor respectively. Both change their values depending on the temperature. The thermopile voltage's relationship with temperature given by equation 1, where A is a constant of integration.

The thermistor's relationship with temperature is given by equation 2, where R_{25} is the Thermistor resistance at 25°C (given as 100 kΩ), and β is a constant given in the thermopile's datasheet (~3960 K).

$$R = R_{25} e^{\beta(\frac{1}{T}-\frac{1}{25})} \qquad \textit{Equation 2}$$

For both models, we included random noise since noise is a natural result of any input signal. This noise was generated using LTSpice's white function. In the case of the thermistor, no specific amplitude was specified for noise. For the thermopile the ZTP-148SR's datasheet lists a typical noise voltage of 37 nV RMS, so in order to achieve approximately this level of noise, a voltage divider was added to decrease the amplitude. A waveform of the thermopile signal with noise is included in Figure 4.

The thermistor model also includes a voltage divider in order to convert the resistance to a voltage that can be read by the Teensy microcontroller. In order to add noise to this model, we added the same voltage white noise used in the thermopile voltage in series with the 9 V source voltage.

Simple Signal Conditioning: Low Pass Filter and Amplifier (Kika)

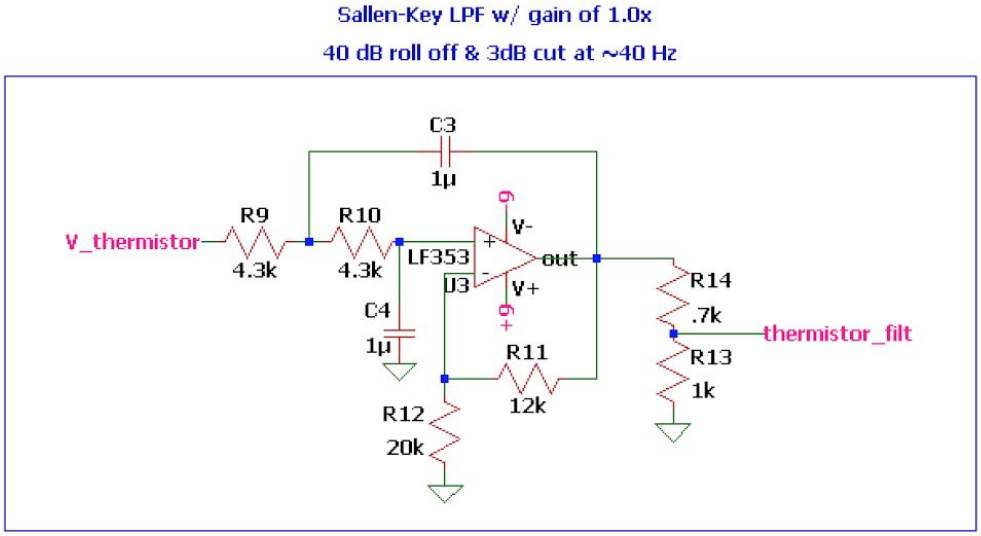


Fig 3. Sallen-Key Low Pass Filter for thermistor noise removal. Voltage divider creates a resulting 1x gain at thermistor_filt output.

Both the thermistor and thermopile voltages require filtering to remove the noise present in each model. For both, a sallen-key low pass filter is used. Both have a 1.6x gain, a 40 dB rolloff, and a -3 dB point at ~40 Hz. We chose a 40 dB rolloff to get rid of noise at frequencies as low as 60 Hz. In order to address concerns with impedance we chose the sallen-key configuration since op amps provide a high common mode rejection which helps eliminate noise. Both LPFs have a gain of 1.6x. To offset this gain the thermistor filter has a voltage divider. This step could also be eliminated with simple division in digital processing. Figure 4 shows the thermopile output voltage with noise before any signal processing. This signal is centered around 563 µV. Figure 5 then shows the output waveform of the Sallen-Key filter with a 1.6x gain applied. As shown by the graph, the resulting waveform is a signal that is a straight line centered around 903 µV (which is roughly 1.6 x 563 µV).

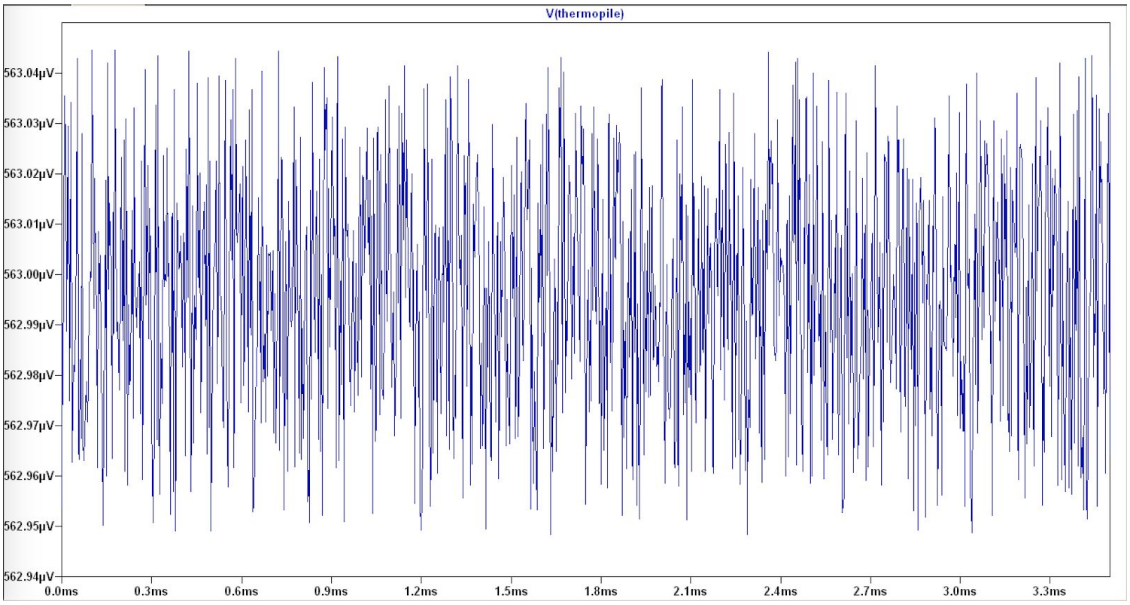


Fig 4. LTSpice plot showing the thermopile voltage model with 37 nV noise. This signal is centered around roughly 563 µV.

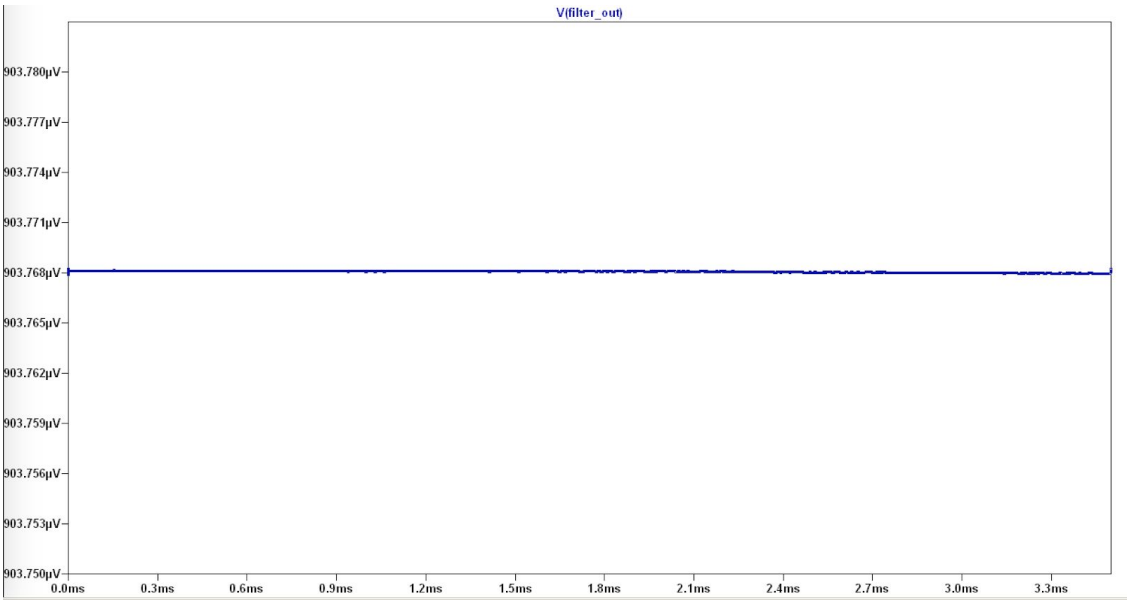


Fig 5. LTSpice plot showing the thermopile voltage after sallen-Key LPF.
This signal is centered around 903 µV due to the approx. 1.6x gain.

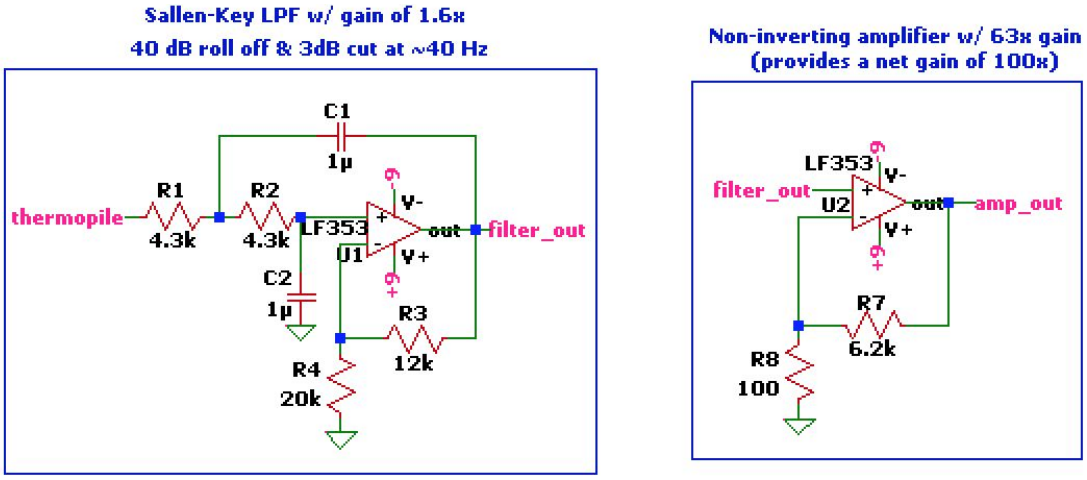


Fig 6. Sallen-Key Low Pass Filter for thermopile noise removal (left).
Non-inverting op amp circuit (right) provides a 63x gain to the output of the thermopile Sallen-Key filter.

The thermopile signal processing has an additional amplification stage with a 63x gain. The original thermopile voltage ranges from -2 mV to 10 mV. We needed a signal that was large enough to compare against other signals, but smaller than 9 V since that is the voltage that powers most of our circuit. For this reason we decided it would be best to scale the output by 100x so that it would be easy to relate the amplified values back to the original values, but small enough to use in the circuit. Since the Sallen-Key filter provides a 1.6x amplification, the non-inverting amplifier which is in series with the LPF has a 63x gain. This makes the amplified thermopile signal 100x the original raw signal output from the sensor. Figure 7 shows the output of the non-inverting amplifier circuit. The voltage is ~57 mV which is approximately 100x the original 563 µV output of the thermopile model.

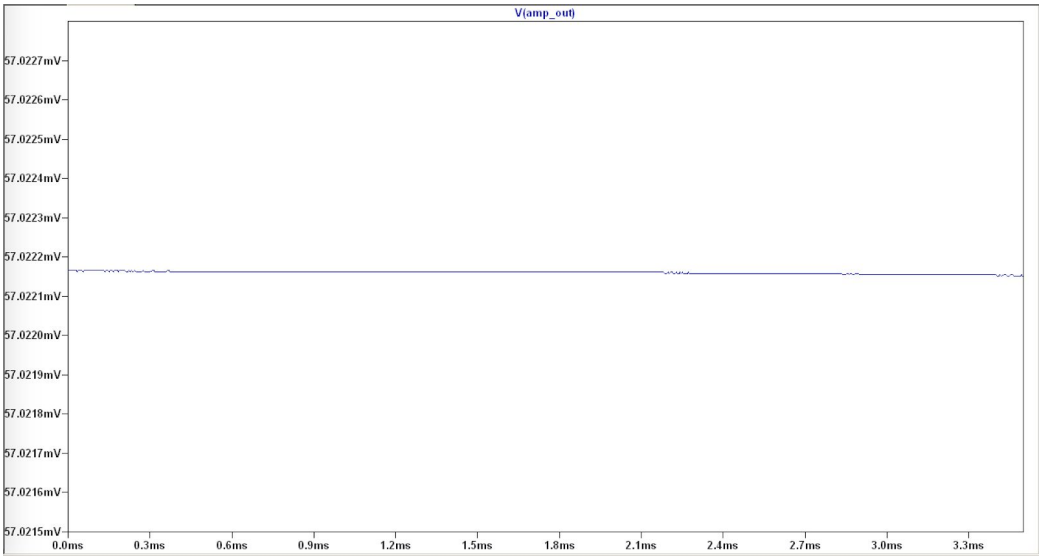


Fig 7. LTSpice plot showing the thermopile voltage after 63x gain non-inverting op amp. After this stage the resulting signal is approx. 100x the thermopile voltage output shown in Figure 4.

Piecewise Linear Voltage Source (Elaine)

General Overview

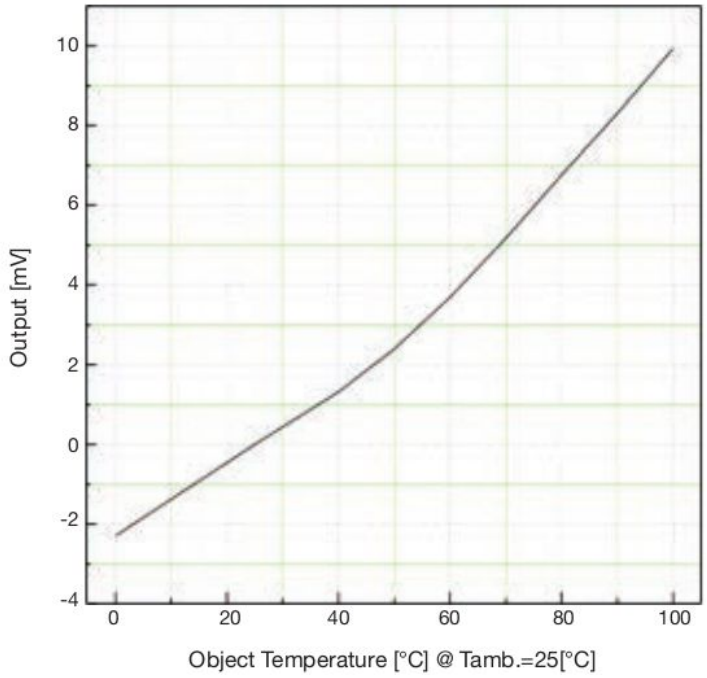


Fig 8. Thermopile voltage vs. temperature graph from the ZTP-148SR datasheet

The Piecewise Linear Voltage Source (PWLVS) circuit recreates this voltage-temperature graph provided in the thermopile sensor's datasheet. Dividing this graph in half at 50℃, it is

approximately linear in each region. The PWLVS circuit creates a piecewise linear voltage curve (in two regions) by creating two sawtooth waveforms, one stacked above the other and switching between them. This is made more clear in the following diagram.

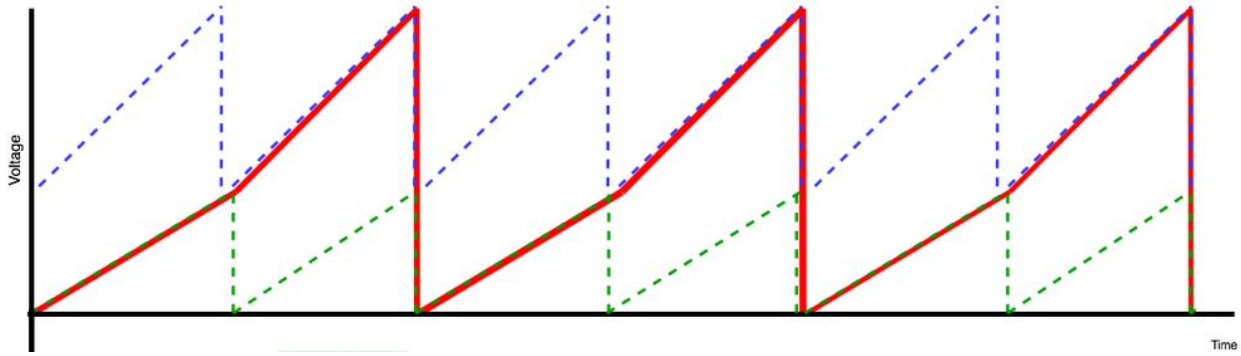


Fig 9. Diagram showing the intended piecewise linear output (in red), a low range sawtooth waveform (in green) and a high range sawtooth waveform (in blue)

Figure 9 shows that the desired piecewise linear voltage can be constructed from switching between the two sawtooths every cycle of either sawtooth.

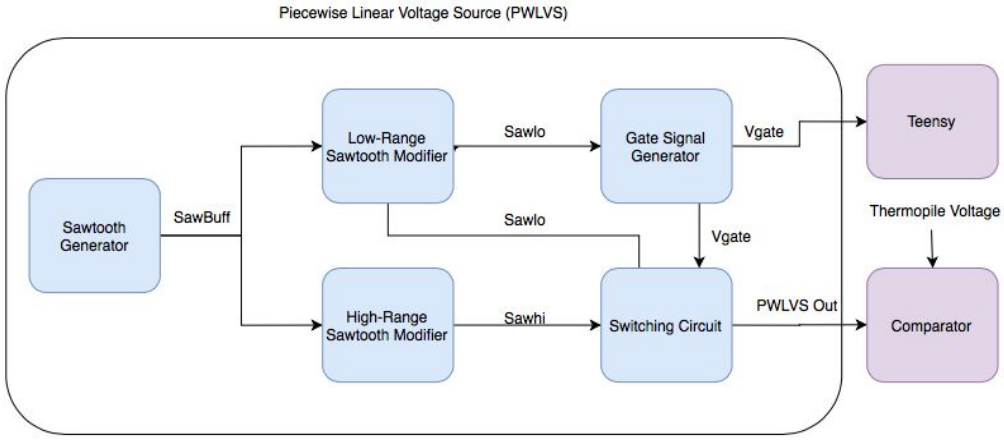


Fig 10. Block diagram of PWLVS circuit

This block diagram shows the subcircuits that make up the PWLVS circuit. First we have a sawtooth generator that generates a base sawtooth waveform with no DC. The two sawtooth modifier circuits scale (and offset for the high-range) this base sawtooth to create the two linear regions. Then, we have a circuit that switches between the low range-sawtooth (sawlo) and high-range sawtooth (sawhi). The switch timing is controlled by a gate voltage generated by the gate signal generator. Finally, the output of the switching circuit is sent to the comparator and the gate signal is sent to the Teensy for further processing.

Sawtooth Generator

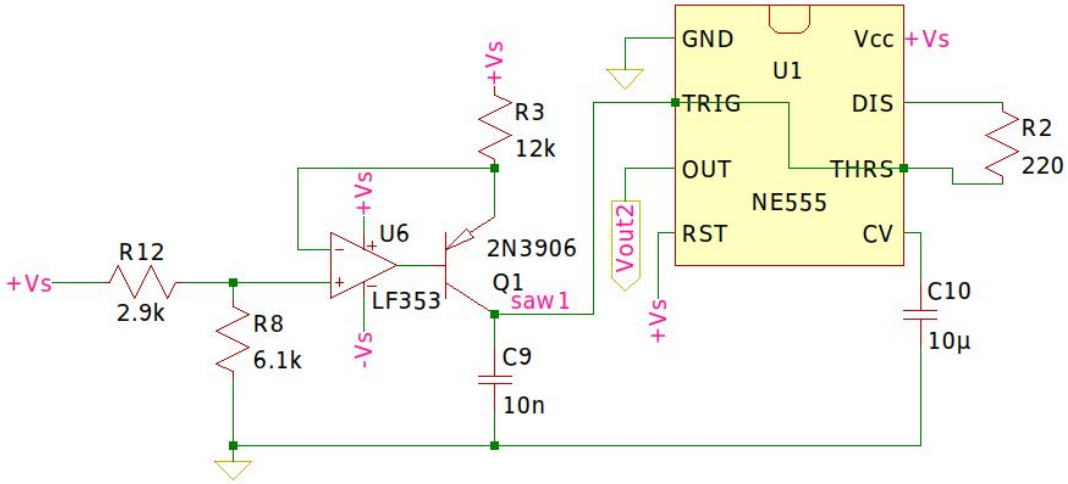


Fig 11. Schematic of sawtooth circuit based on BJT current source and 555 timer

We used the sawtooth circuit in Lab 5/6 but modified some of the values to meet our specs. Since we wanted to only use 9 V batteries for supply, this sawtooth circuit ramps from 3 V to 6 V ($\frac{1}{3}V_s$ and $\frac{2}{3}V_s$ respectively). The values of R12, R8 and C9 were used to set the frequency of the sawtooth. Since the voltage at the non-inverting input of the op amp sets the collector voltage of the 2N3906, the input voltage to the op amp must be greater than the voltage the capacitor is charging up to (6 V). As long as this condition is true, the collector current is charging the capacitor. The frequency of the sawtooth is also controlled by the input voltage to the op amp, which is why we have a voltage divider to scale down the 9 V. For this oscillator circuit, the frequency increases as the input voltage to the op amp decreases. Because we are putting the sawtooth through a high pass filter later, we have the voltage divider set to give an input voltage of 6.1 V (the smallest voltage we could reasonably make while meeting the condition) to get the highest frequency of 7.156 kHz.

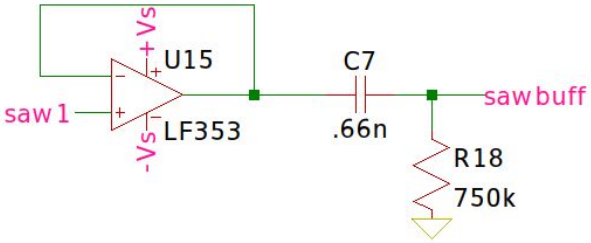


Fig 12. Schematic of high pass filter to remove DC from sawtooth

To make scaling and offsetting the sawtooth in the next stage easier, we also removed the DC value of the sawtooth by passing it through a high pass filter. The time constant formed by R18 and C7 had to be really short, so the system settled quickly. However, we could not make these values arbitrarily small or we risk having too high of a cutoff frequency. The values of the capacitor and resistor were chosen experimentally to balance between these effects.

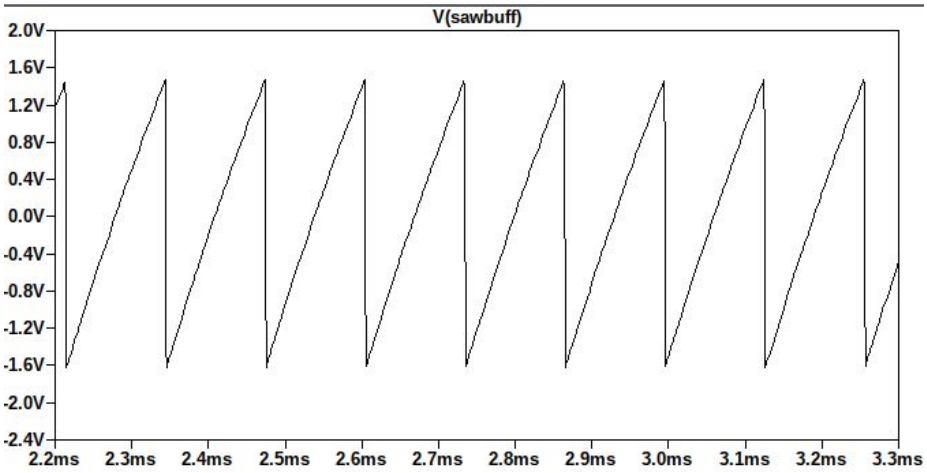


Fig 13. Plot from LTSpice simulation of the output of the high pass filter. Notice the ~3 Vpp (actually 3.1 Vpp) sawtooth with zero DC

Sawtooth Modifiers

In terms of circuit topology, these two circuits are pretty simple, just voltage dividers and the extra offset voltage source for the high range sawtooth modifier. Most of the work and design for these circuits was the process of figuring out how we were scaling the graph from the datasheet to voltage ranges that we could feasibly produce.

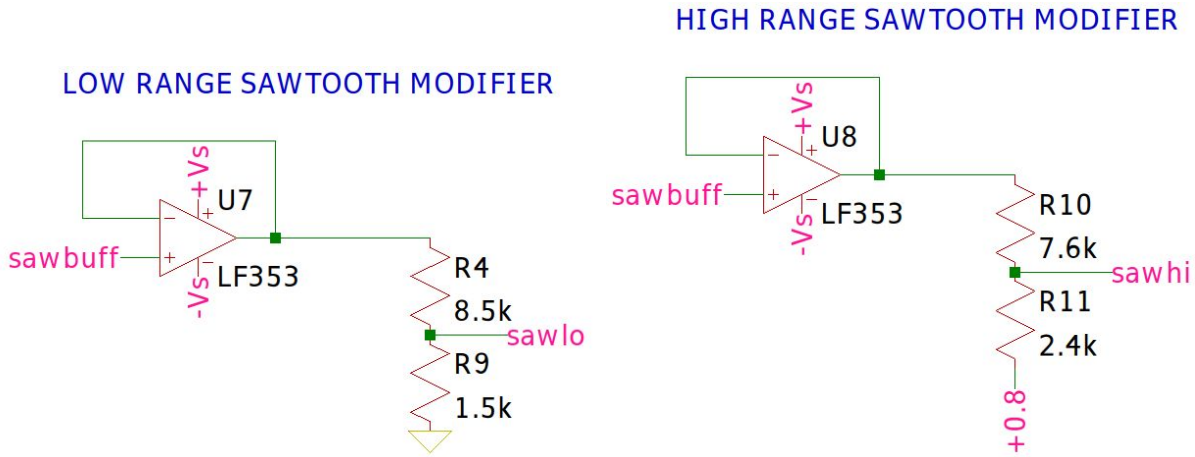


Fig 14. Schematic for low range and high range sawtooth modifiers

For the low range sawtooth, we only wanted to use a voltage divider to scale down the incoming 3.1 Vpp sawtooth. In the graph from the datasheet (Figure 8), the first linear range goes from -2.25 mV to 2.438 mV. After this ideal thermopile voltage goes through our simple signal conditioning stage, it will be amplified by 100x and so we would want the low range sawtooth to "plot" a range that spans $243.8 - (-225) = 468.8\ mV$. The voltage divider was designed to convert the original 3.1 Vpp to 468.8 mVpp i.e. it scales the incoming sawtooth voltages by $0.4688/3.1 = 0.1512$.

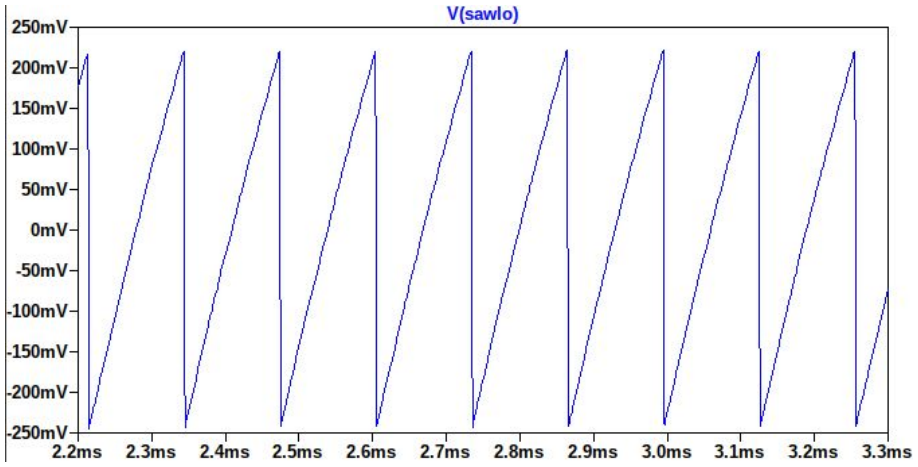


Fig 15. Plot from LTSpice simulation of the output of the low range sawtooth modifier (sawlo)

For the high range sawtooth, we used a similar strategy to determine the resistors of the voltage divider. After the 100x gain, the high range sawtooth should span $996.9 - 243.8 = 753.1\ mV$. This voltage divider was designed to scale the incoming sawtooth voltages by $0.7531/3.1 = 0.2429$. To make the high range sawtooth start at the value of the end of the low range sawtooth, we had to add an offset voltage of $0.2429(-1.6) + 0.225 = 0.614\ V$. There is an extra 0.145 V to make a total offset of 0.789 V because the high pass filter in the previous stage does not remove all of the DC. Ultimately, we decided to use an offset voltage of 0.8 V because it was a voltage supply we could make (more on that in the Powering the System section).

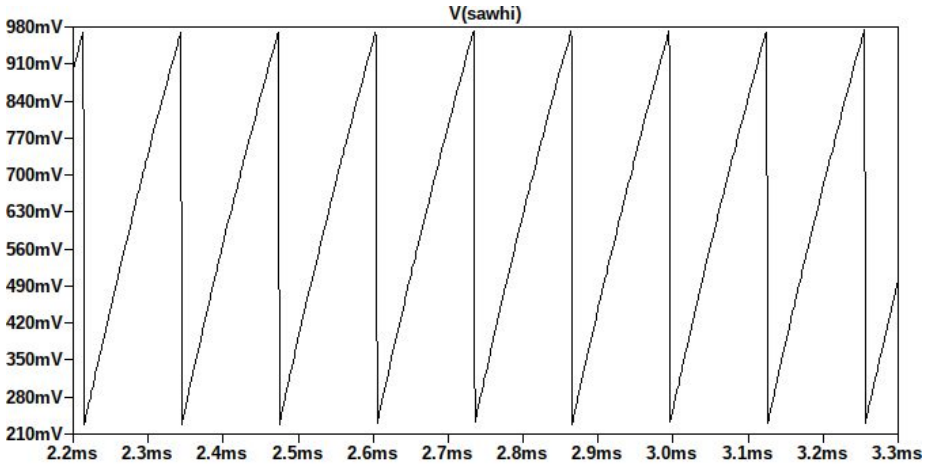


Fig 16. Plot from LTSpice simulation of the output of the high range sawtooth modifier (sawhi)

Control Signal Generator for Switching

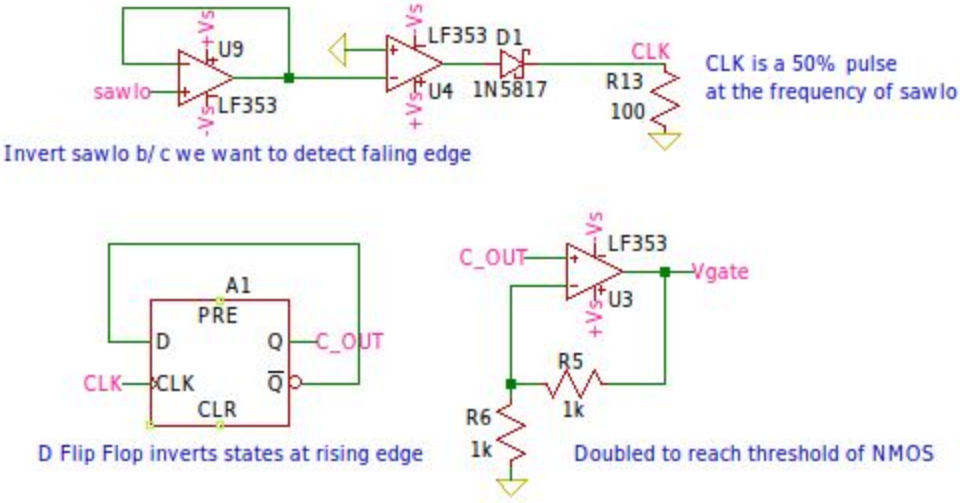


Fig 17. Schematic for the control signal generator

For the switching circuit (discussed in the next section), we needed a control signal so that the NMOS would turn sawhi on every other cycle. Thus, we designed this circuit that makes a 50% pulse with half the frequency of any of the sawtooths. First there is a comparator that compares sawlo to its DC value of zero. The reason for inverting sawlo is because we wanted the pulse to trigger at the falling edge of the sawtooth. The comparator gives a square wave that is -9 V for half of the period and 9 V for the other half. The comparator output gets rectified by the diode to create a clock signal (CLK) that is high for the half of the cycle triggered by the falling edge of the sawtooth.

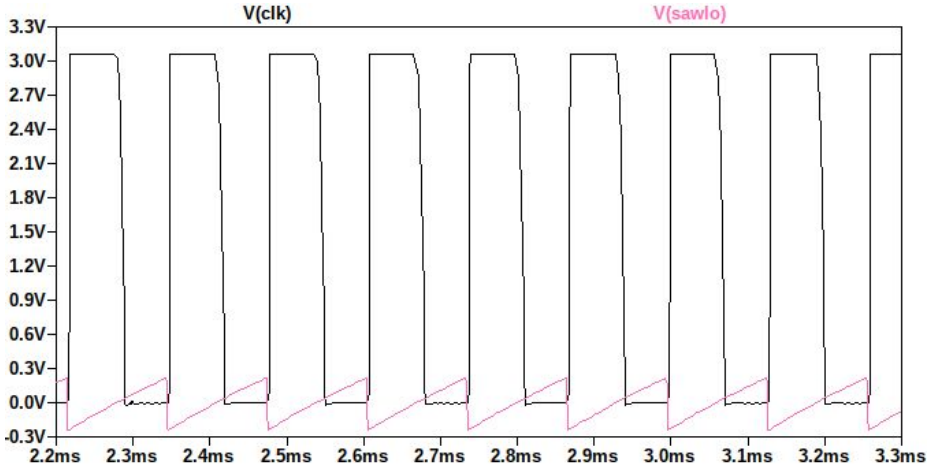


Fig 18. LTSpice plot showing the CLK signal triggered by the falling edge of sawlo with 50% duty cycle

This CLK signal goes to a D flip-flop based frequency divide-by-2 circuit. The $\overline{Q}$ and D terminals are tied together in a feedback loop that inverts the CLK signal at each rising edge. The output Q is then the input CLK signal with its frequency halved. Finally, the output of the flip-flop is passed through a non-inverting amplifier with a gain of 2 to make the voltage level of the pulse high

enough to turn on the NMOS. To be honest, the output of the flip-flop (SN74LVC2G74), which is its supply voltage of 3.3 V, is high enough to turn on the NMOS we picked. The threshold voltage of the RUC002N05 NMOS is between 0.3 V and 1 V. However, in simulation we could only get Spice's ideal flip flop to work, which has an output of 1 V, so we added the gain stage to be safe.

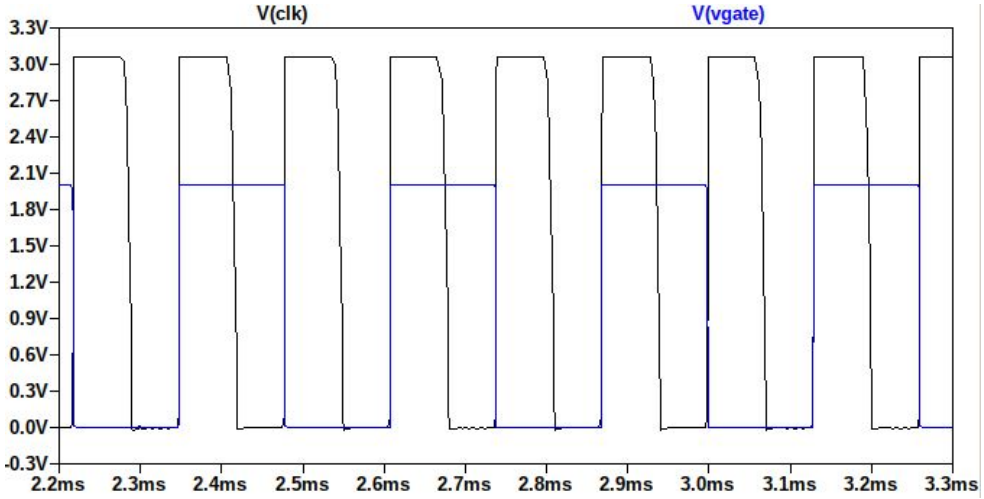


Fig 19. LTSpice plot showing the final control signal Vgate which has half the frequency of the input CLK signal and an output voltage of 2 V

Switching Circuit

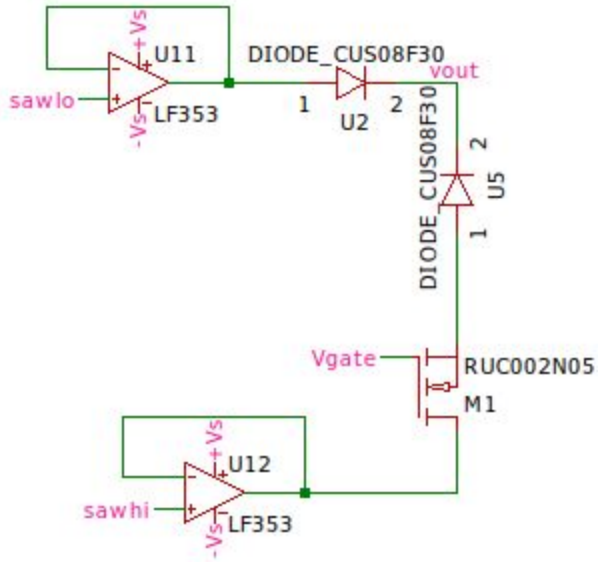


Fig 20. Schematic of the switching circuit

The switching circuit creates the piecewise linear output by switching between sawlo and sawhi. This circuit consists of sawhi and sawlo going through a diode OR gate and a NMOS that turns sawhi on and off. The gate voltage of the NMOS is the control signal from the previous stage. When the NMOS is off (and the output is sawlo), the diode on the high side (U5) is off and prevents any reverse current. The low side diode (U2) is on and carries sawlo to the output. When the NMOS is on (and the output is sawhi), U5 is forward biased and carries sawhi to the output.

Because sawhi is always greater than sawlo, U2 is off and prevents sawlo to go to the output. Thus, we have a circuit that switches between sawlo and sawhi when the NMOS turns off and on respectively.

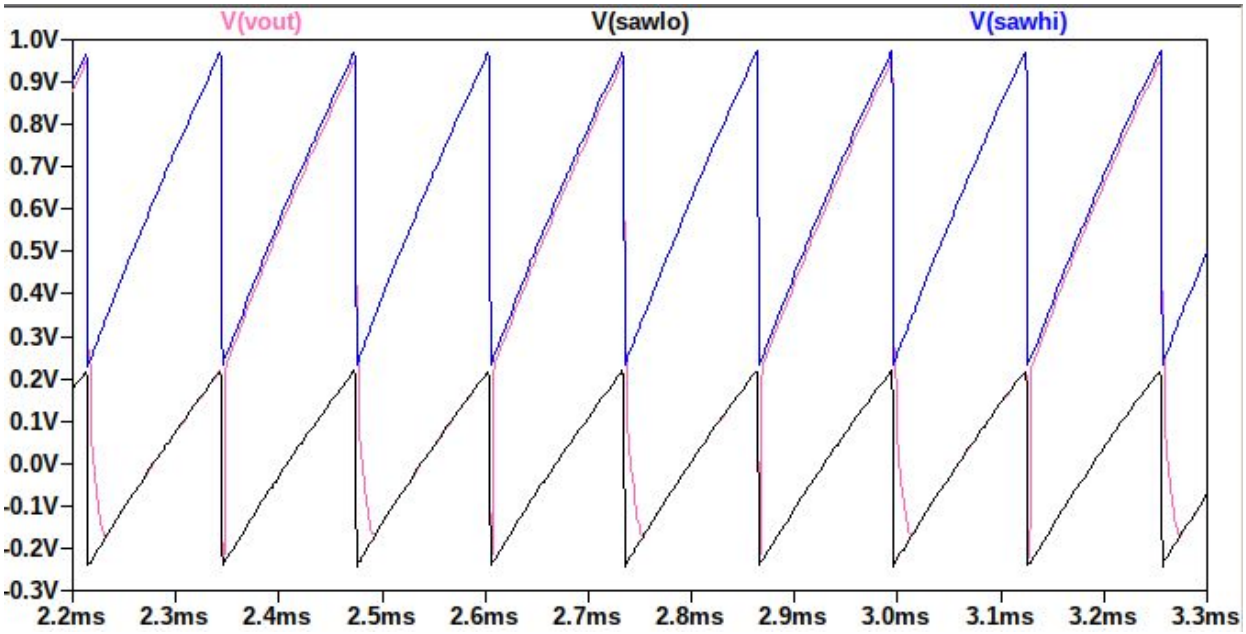


Fig 21. LTSpice plot of the output of the switching circuit (in pink), sawlo (in black) and sawhi (in blue) Note that vout follows sawlo and then sawhi in the next cycle (so the switching works!)

There are two minor issues with the output of the circuit. First, the output does not start from the lowest value of sawlo. Second, the voltage briefly spikes when the NMOS turns on. We think the spikes come from the slight delay (only ~4 µs) between the edges of sawhi and sawlo. Both of these effects are small enough to not really affect the comparator output that much and can be mitigated in post processing.

Comparator & Precision Rectifier

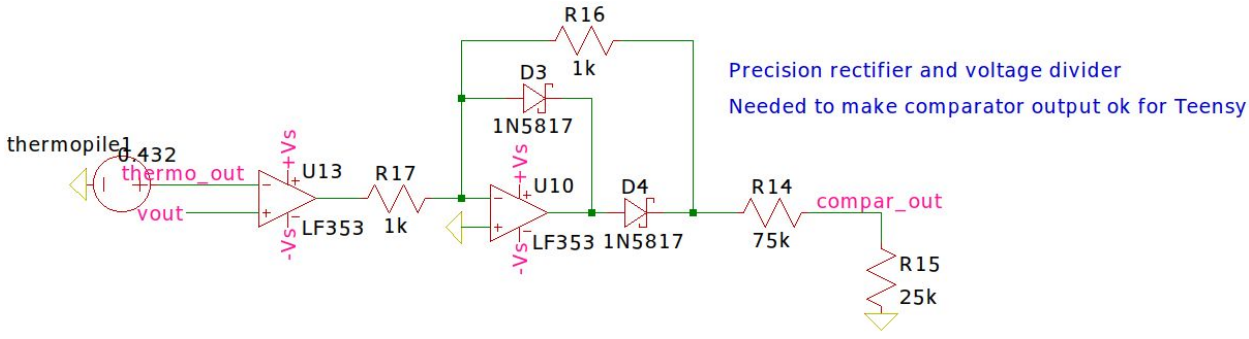


Fig 22. Schematic of the final comparator and rectifier stage

Finally, the piecewise linear output is sent to this comparator and rectifier stage. The comparator compares the piecewise linear output to the signal conditioned thermopile output. This comparator output is rectified using the improved precision rectifier from Lab 4. A normal diode

rectifier would have an extra 0.6 V drop that we did not want. The precision rectifier can rectify signals with much less error. This particular configuration prevents the op amp from saturating. The output of the rectifier is either $-V_{comparator}$ or 0 depending if $V_{comparator} < 0$. Since the analog pins of the Teensy can only take a maximum 3.3 V, we added a voltage divider to scale the 9 V rectified output to 2.25 V.

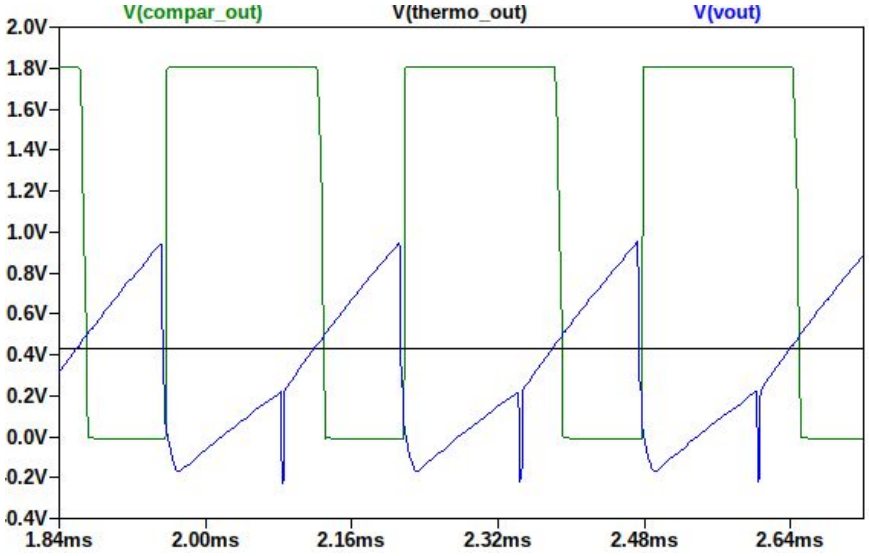


Fig 23. Plot of the final output (compar_out) in green, thermopile voltage (thermo_out) =0.432 V in black and PWLVS voltage (vout) in blue

The comparator triggers a little after the actual point of intersection of thermo_out and vout . This delay is only around 12 µs, and is compensated for in the post processing.

Teensy Processing (Kika)

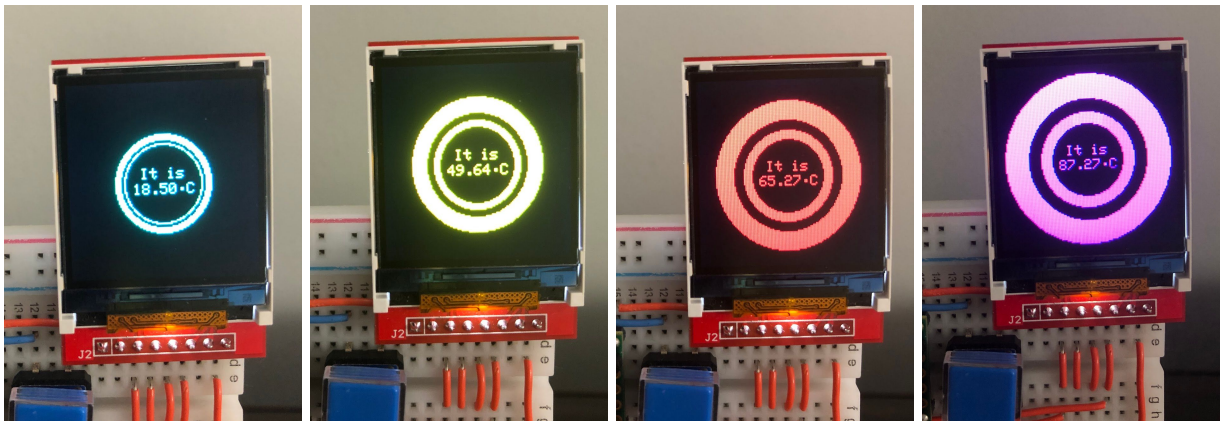


Fig 24. Examples of different temperature readings being displayed to the TFT display. Outputs are based on an LTSpice simulation output which inputs a linearly increasing voltage to the comparator.

The comparator data and the switching signal (vgate) would normally be passed to the Teensy 3.2 via digital or analog input. However, since we do not have the ability to give a live signal input,

simulation data from LTspice was used instead. LTSpice produced a time, comparator, and vgate signal which we will use in calculating the temperature. This simulation data is stored in three double arrays in a C file separately from the main IR thermometer sketch. One millisecond of simulation produces over 2000 array elements.

The thermistor voltage is also needed in order to produce a compensated output temperature. Since the thermistor resistance is set manually in simulation, for this version of the thermometer sketch $V_{thermistor}$ is a static variable set at 0.66716 V which would be the output voltage of the thermistor voltage divider if the ambient temperature were 20 °C which is generally considered room temperature.

From this data and the equations previously described, we can calculate an accurate temperature which will be output to the TFT display. The IR thermometer sketch uses five functions to generate a temperature value. The first is *calculate_period()* which only happens once at the start of the temperature reading session and calculates the period of time it takes for one temperature reading to occur. The code detects a rising and falling edge of the vgate signal and saves the timestamp of those two events. After both are detected, the difference is multiplied by two to get the length of time of one full cycle. The *calculate_compar_ontime()* function uses the exact same mechanics to calculate the ontime of the comparator signal (however it does not multiply the resulting time by two). This function runs continuously since every comparator high is a different temperature reading.

Once the comparator on-time is calculated, the temperature can be calculated by dividing it by the temperature reading period. This relationship is justified by the fact that the PWLVS curve (which shows voltage vs time) imitates the voltage vs temperature curve on the ZTP-148SR's datasheet. The time and temperature scales in both graphs are linear, and for this reason time and temperature are proportional to each other.

The thermistor temperature must also be calculated from $V_{thermistor}$. In order to get a temperature the voltage must be converted back to a resistance, and then to a temperature. The thermistor resistance value is calculated by using a simple voltage divider equation and substituting in 9 V as the reference voltage and 10 kΩ as the other resistor value. This calculation for resistance is shown in equation 3. Equation 2 is then used to convert this resistance to a temperature.

$$R_{th} = \frac{9\ V * 10\ k\Omega}{V_{thermistor}} - 10\ k\Omega$$

Equation 3

Once both of these temperatures are calculated, a final compensated temperature can be calculated. We can derive a final temperature by combining the temperature from the thermopile and the reference sensor (the thermistor) using equation 5. Equation 5 is derived by combining equations 1 and 4 and simplifying (equation 1 is a simplified version of equation 4) .

$$V = A(T^4 - T_{ref}^4) \qquad \textit{Equation 4}$$

$$T^4 = T_{therm}^4 + T_{ref}^4 \qquad \textit{Equation 5}$$

After the temperature output is calculated, the value is displayed on the TFT display as shown in Figure 23. Although we could just display the number, we added colors and graphics in order to provide a greater level of feedback to the user. As the temperature increases, the warmth of the display color increases with dark blue representing a temperature range from 0°C to 15°C, and white representing any temperature above 90°C. The color changes every 15°C and the progression of color goes as follows: blue, cyan, green, yellow, red, magenta, and white. In addition to the color changing, two colored circles were added to give the user a sense of the magnitude of the temperature. The size of the circles are proportional to the displayed temperature. The maximum radius for the outer circle occurs at 100°C.

Powering the System (Elaine)

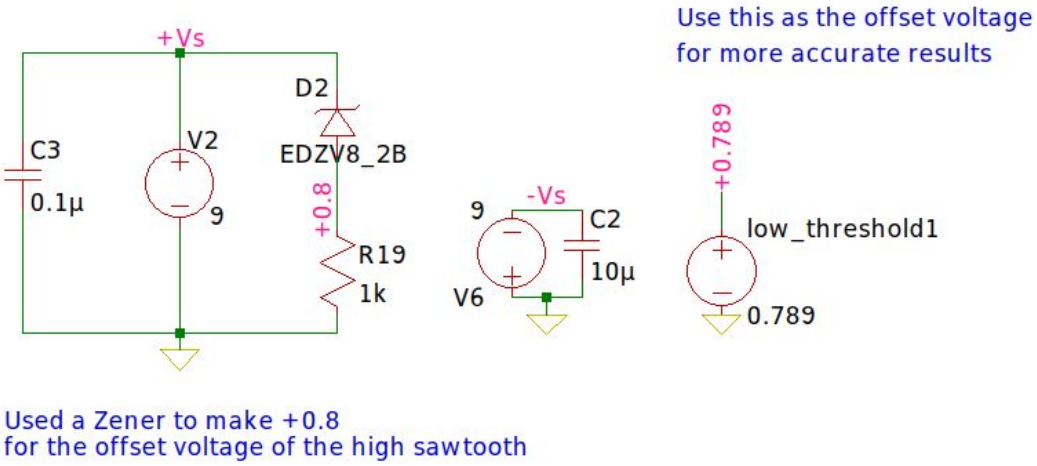


Fig 25. Schematic of the voltage supplies powering the entire system

We designed the circuit to run on 9 V batteries and the Teensy's 3.3 V analog output. The LF353 op amps we chose can be supplied by +9 V and -9 V. The D flip flop (SN74LVC2G74) can only take a maximum of supply voltage of 6.5 V, so we decided to make it run on the 3.3 V analog output from the Teensy. While most of the system can run on these more accessible sources, the offset voltage needed in the high range sawtooth modifier is 0.789 V. We were able to make a 0.8 V supply from the +9 V with a Zener diode with a Zener voltage of 8.2 V. While the Zener is regulating, the voltage across the 1 kΩ resistor is kept at 0.8 V. The 1 kΩ resistor was chosen to make sure the Zener regulates with the load to the 0.789 V voltage reference: the resistors of the high range sawtooth modifier. This meant we needed a series resistor that was $\frac{R}{R+(7.6\ k\Omega \parallel 2.4\ k\Omega)} > 0.8/9$, which is satisfied by 1 kΩ. For more accurate simulations, we have decided to include the more correct offset of 0.789 V as a separate voltage source that can be used.

Lessons Learned

Using LTSpice

Simulation software is unfortunately never perfect. Since most of our design sessions were spent with both Zoom and LTSpice running on our computers, simulations were occasionally time consuming to run. When designing the HPF in the PWLVS simulations took anywhere from 5-10 minutes. This was incredibly unfortunate since we were testing different resistance and capacitance values. In this case trial and error became an expensive process, whereas in the lab we would have received almost immediate feedback. Additionally, although many components have a spice model, a lot of frustration came from not being able to use parts that didn't have a spice model. This became especially frustrating with the flip-flop as there were limited D-type flip-flop models, only one of which could be used in LTSpice, and it didn't work.

The PWLVS

Designing the PWLVS was the most challenging technical aspect of our project. We went through many iterations continuing to build upon the original concept of stacking and switching between sawtooths. Although we received some advice on how to approach the PWLVS, we went with a design that was intuitive to us and we felt would be easiest to implement. We discovered that eliminating DC offsets is not always as easy as "just adding a cap". Since that part of the PWLVS was one of the last to be implemented, the artifacts of adding the HPF are still visible in the final PWLVS signal. The HPF eliminated our ability to have precise output in the lower range of the temperature curve. We also learned that buffers can be incredibly helpful in mitigating effects of capacitance on other areas of your circuit design, and while voltage dividers are great, they unfortunately can't regulate voltage due to their finite impedance.

PCB Layout

Laying out a board takes more time than one may think. Regrettably we may have started layout a little too late because we kept making changes to the schematic and simulations. We had to change some of the components we picked in simulation because they were too difficult to work with in layout. (For future 6.101 students, be sure to check the packaging of the parts you pick! Don't pick a huge through hole resistor if the rest of your IC's are tiny surface mounts…) Trying to lay out the power traces to all of our op amps was a very complicated game of Tetris. We tried to follow the staff's tip to keep our traces short, but that was difficult to implement with 14 dual supply op amps. If we had more time, we could probably come up with a better layout or redesign the circuit to not depend on so many dual supply op amps.

Teamwork

Although the situation surrounding the second half of the semester was not ideal, it was valuable learning to work as a team remotely. Although some parts of the circuit were made separately, the entirety of the PWLVS section was designed by both of us. We worked collaboratively to generate

ideas based on prior knowledge and research, and iteratively to eliminate problems as they arose. The PWLVS is perhaps the most noteworthy outcome of our project. It was incredibly satisfying to see that section come together piece by piece to create something that actually worked. Still getting to experience a collaborative design process, even in isolation was our favorite part of this project.

PCB Layout

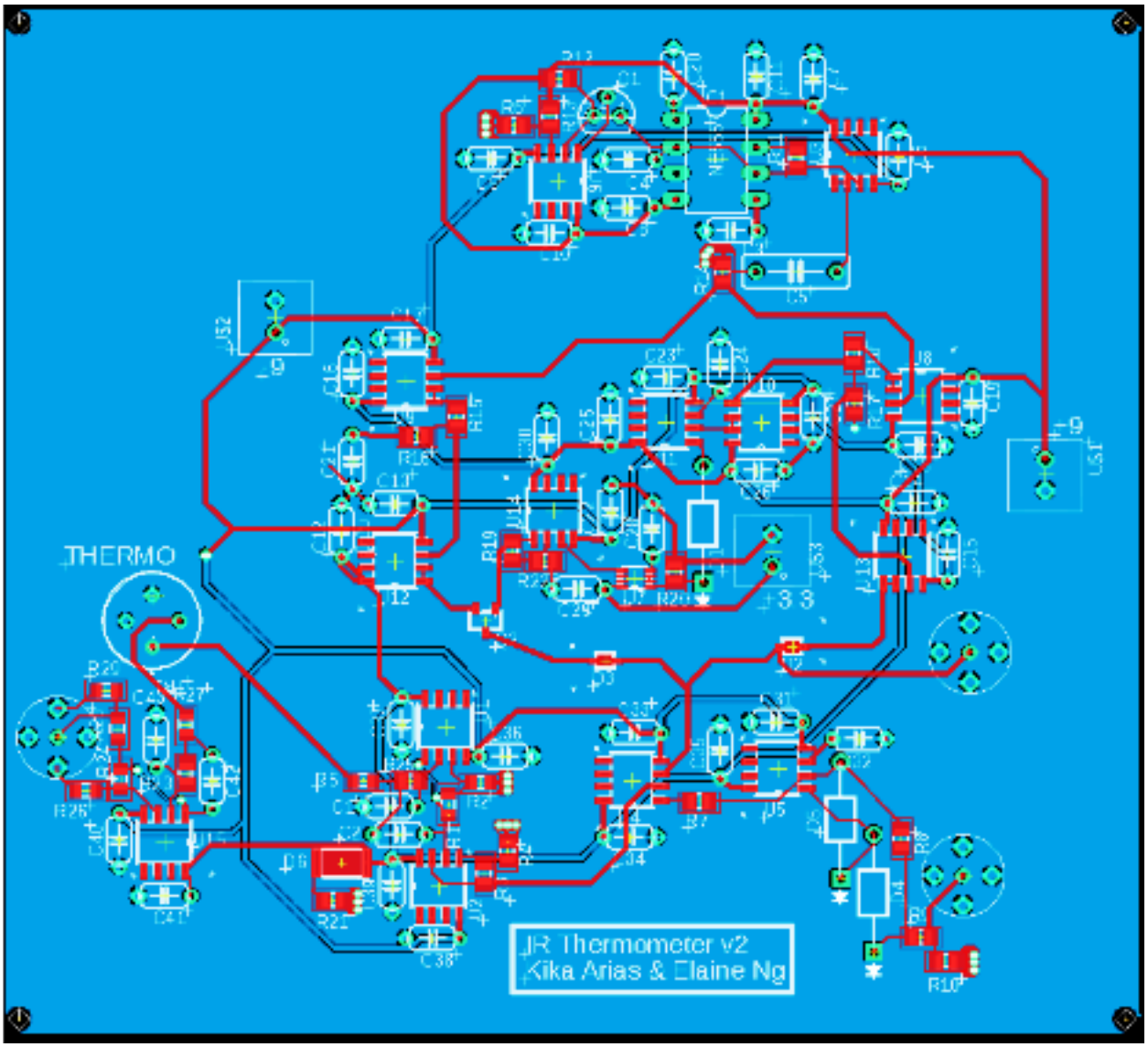


Fig 26. PCB layout of the entire system using EAGLE

Conclusion (Elaine)

We have designed and simulated in LTSpice an IR thermometer that converts the small temperature-dependent voltage reading from a thermopile sensor into a temperature value in °C. The circuit filters the noisy from thermopile voltage and applies a 100x gain. The linearization part

of the system generates a piecewise linear approximation (PWLVS) to the nonlinear voltage-temperature characteristic. Our PWLVS design is based on an unconventional approach of switching between two sawtooths that make up the lines in each region. We have also designed a comparator that can detect where the input thermopile voltage intersects the PWLVS. The Teensy is programmed to calculate the corresponding temperature through a proportion relating the comparator on time to the thermopile voltage. The Teensy is also driving a colorful display, whose colors and the size of the graphics is temperature dependent.

Although we have worked out many of the issues in this design, there is further work that can be done to make it better. For example, many of the blocks can be redesigned to not depend on multiple voltage sources. The switching circuit can be improved to reduce the voltage spikes during switching transitions. And the linearization scheme can be more accurate if we were referencing the voltage-temperature characteristics of several thermopile sensors.

Although our time in 6.101 is over, we plan on improving on these issues and even building the thermometer in the future. We like to thank the wonderful staff, Negar and Dave especially, for their guidance in this project. We've had a lot of fun struggling through this process, and we're both very proud of what we were able to share here.

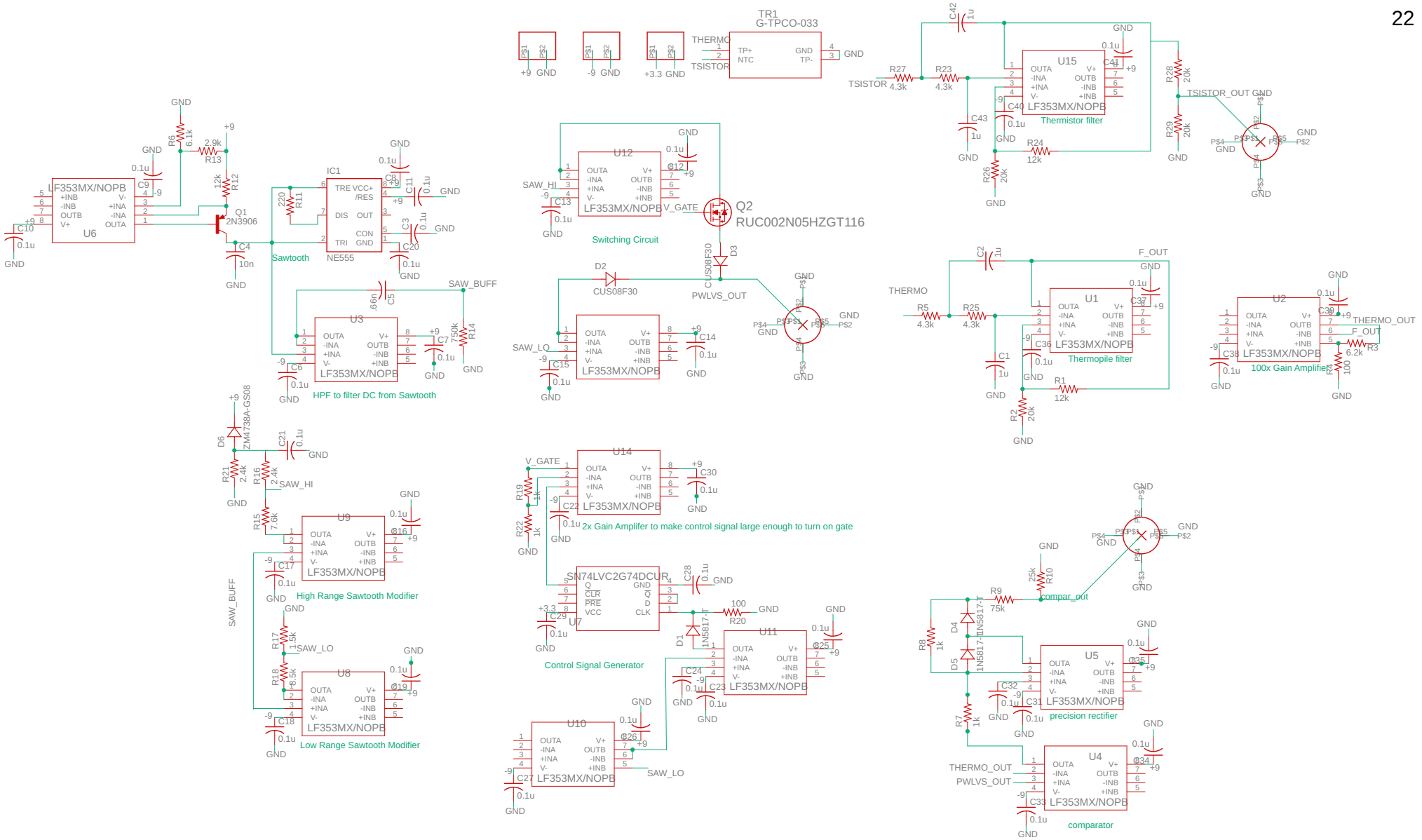
References

- GitHub repository for the project (includes Spice, Arduino, and EAGLE files)
    - https://github.mit.edu/kaarias/IR_thermometer
- Helpful application note where most of our equations came from
    - "Thermopile Sensor for Contactless Temperature"
- This article from ADI's AnalogDialogue page is about signal conditioning for thermocouples (which we found very relevant because a thermopile is just a bunch of thermocouples in series)
    - https://www.analog.com/en/analog-dialogue/articles/measuring-temp-using-thermocouples.html
- TFT display library
    - https://github.com/Bodmer/TFT_ILI9341_ESP

Parts List

- Thermopile sensor:
    - ZTP-148SR
    - TS305-11C55 (alternative)
- Zener diode

    - EDZV8.2B (in simulation)
    - ZM4738A-GS08 (another 8.2 V Zener that can work)
- NE555 timer
- 2N3906
- LF353 (14x)
- Switching diode (2x)
    - these need to have a small forward voltage and be fast:
    - CUS08F30 (high speed switching Schottky with forward drop of 0.22 V)
    - LSM115J (alternative)
- NMOS: RUC002N05
- Rectifying diode: 1N5817 (3x)
- D Flip-Flop: SN74LVC2G74
- For resistors and capacitors, refer to the schematics



Full Schematic from EAGLE