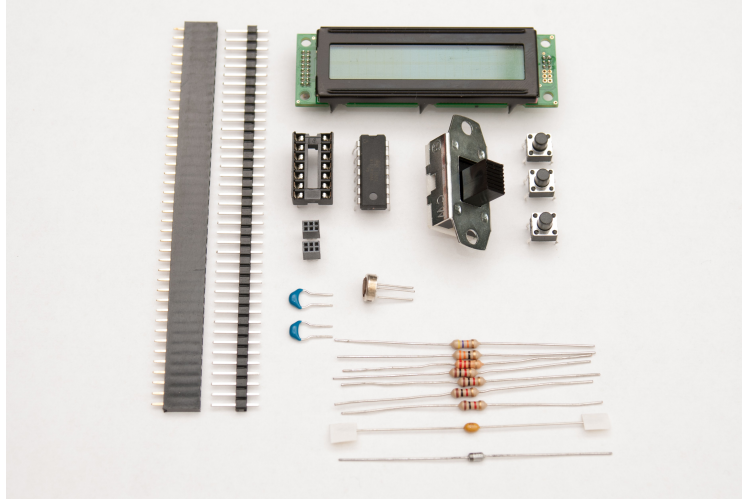


## Low Cost & Full Featured Tire Pyrometer

### Overview:

A tire pyrometer is a common tool within the racing community. The concept is simple, it's a device that measures tire temperatures thereby allowing the driver to tune for optimal traction. The goal of this project was to make a low cost alternative to some of the more costly commercial devices on the market. I knew there wasn't much to it and that I could reproduce a similar product for a fraction of the cost.

# 1: Parts & Materials



The overall part count for this project is quite low. Most of the parts are simple buttons and resistors used for interface functionality. The above photo only shows the core components to get the pyrometer up and running, it doesn't include the battery, PCB, and enclosure. Don't worry, I'm not going to leave you out in the cold though... we'll get to those later on.

Here is a basic break down of what you'll need for this project. I know it seems like a lot, but if you're even moderately into electronics, you probably already have half of these things. I've attached a more detailed spreadsheet of the full bill of materials, complete with Mouser part numbers.

## Parts:

- Thermistor
- Precision series resistor
- Character LCD
- AVR micro controller
- Female & Male headers
- Tactile pushbuttons
- Power switch
- Filter capacitors
- Contrast potentiometer
- Zener diode regulator

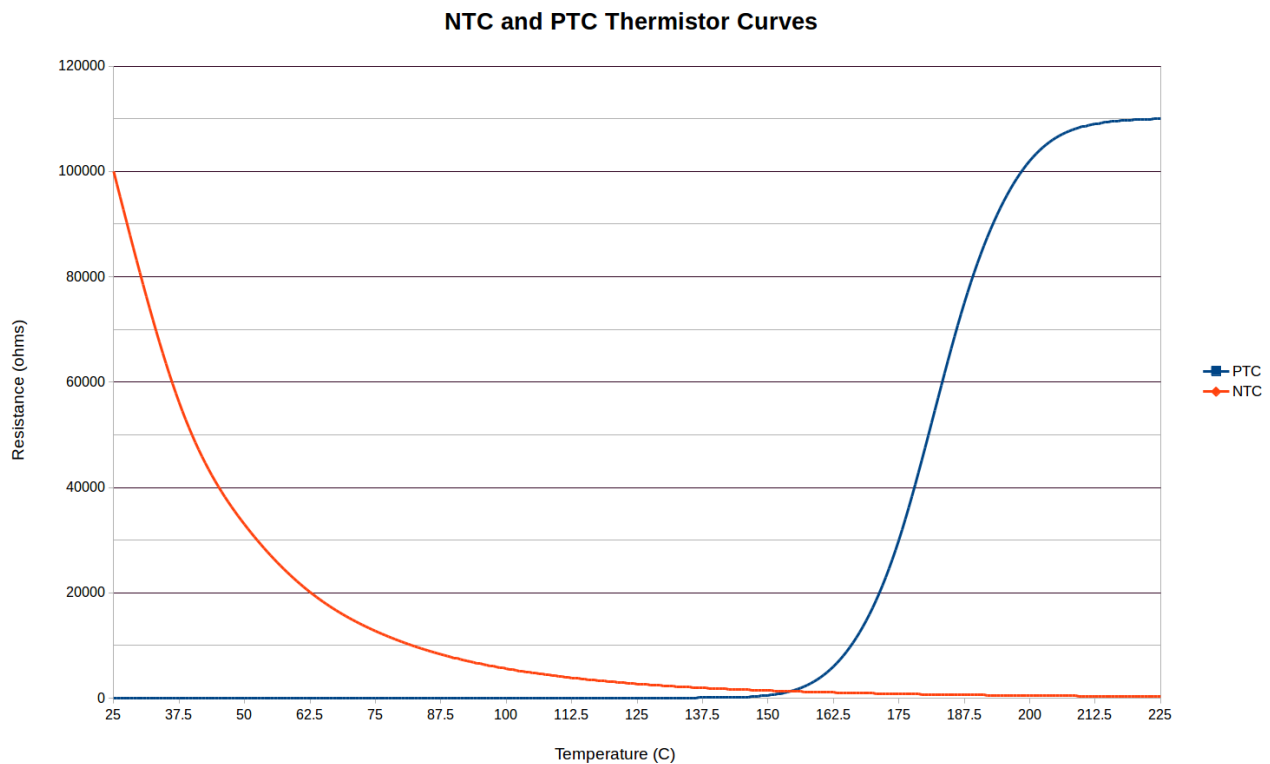
## Tools:

- Soldering iron
- Needle nose pliers
- Wire cutters
- Wire strippers
- Arduino or AVR programmer

## Materials:

- Shrink wrap tubing
- 22-24awg braided wire
- 12awg solid house wire
- Thermal compound (Arctic Silver)
- Sand paper

## 2: Thermistor Theory



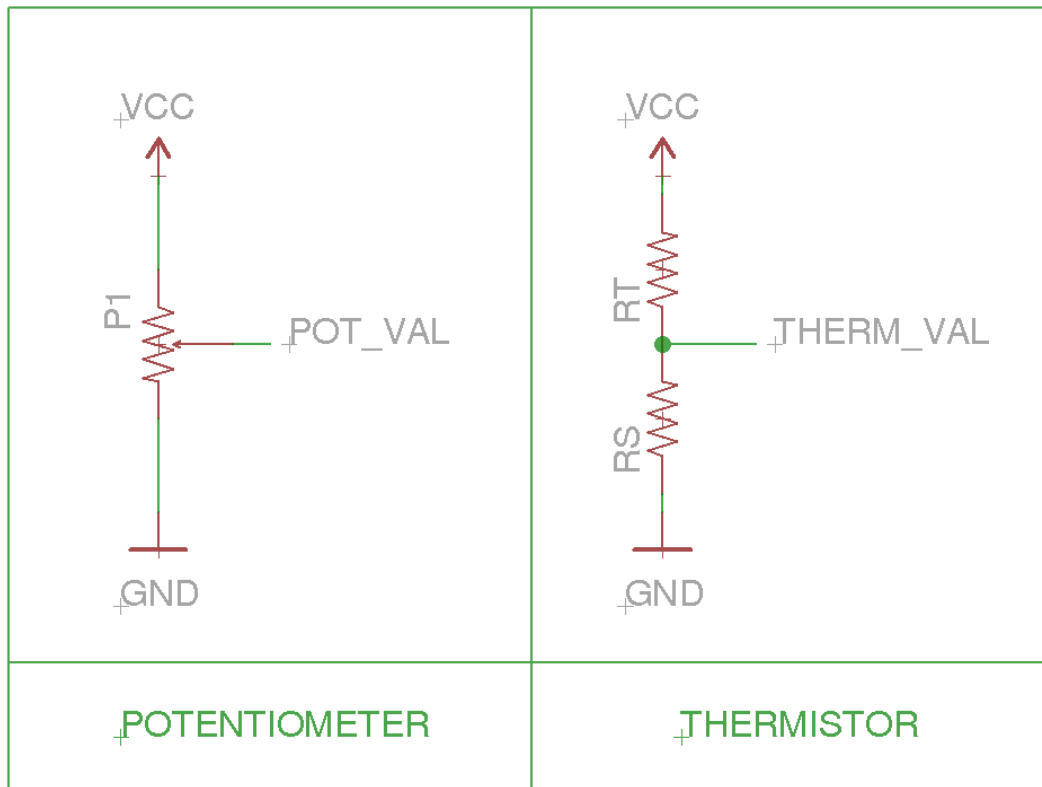
Thermistors are devices that change their electrical characteristic in response to temperature. Like the name suggests, resistance is the main characteristic that is dependent to temperature with these devices. Thermistors are very common in temperature applications because they provide high accuracy at a low cost. The above graph shows the resistance vs. temperature graph of two specific types of thermistors.

Most thermistors follow an exponential relationship between resistance and temperature. The graph above shows the difference between a negative and positive coefficient thermistor; often abbreviated NTC and PTC. The main difference between the two being that NTC shows an inverse R-T relationship and the PTC shows a direct. For this project I chose an NTC thermistor for a wider temperature range and also because they tend to be a bit cheaper.

$$R_T = R_{25} \cdot \exp^{B \cdot \left( \frac{1}{T} - \frac{1}{298.15} \right)}$$

The equation above will produce the exact R-T curve for a Vishay NTC thermistor. Note the constants B and  $R_{25}$  within the equation; these values are specific for each thermistor. The  $R_{25}$  is the resistance of the thermistor at 25°C. B has units of Kelvin (temperature), and relates to how quickly the graph drops off. Both of these values will be given to you in the component's datasheet. Keep in mind T is in units of Kelvin which is simple to convert from Celsius since  $\text{Kelvin} = \text{Celsius} + 274.15$ . We will use these values and the  $R_T$  equation in the next step to make some plots and tables to help us read the temperature with the micro controller.

### 3: Thermistor Reading Technique



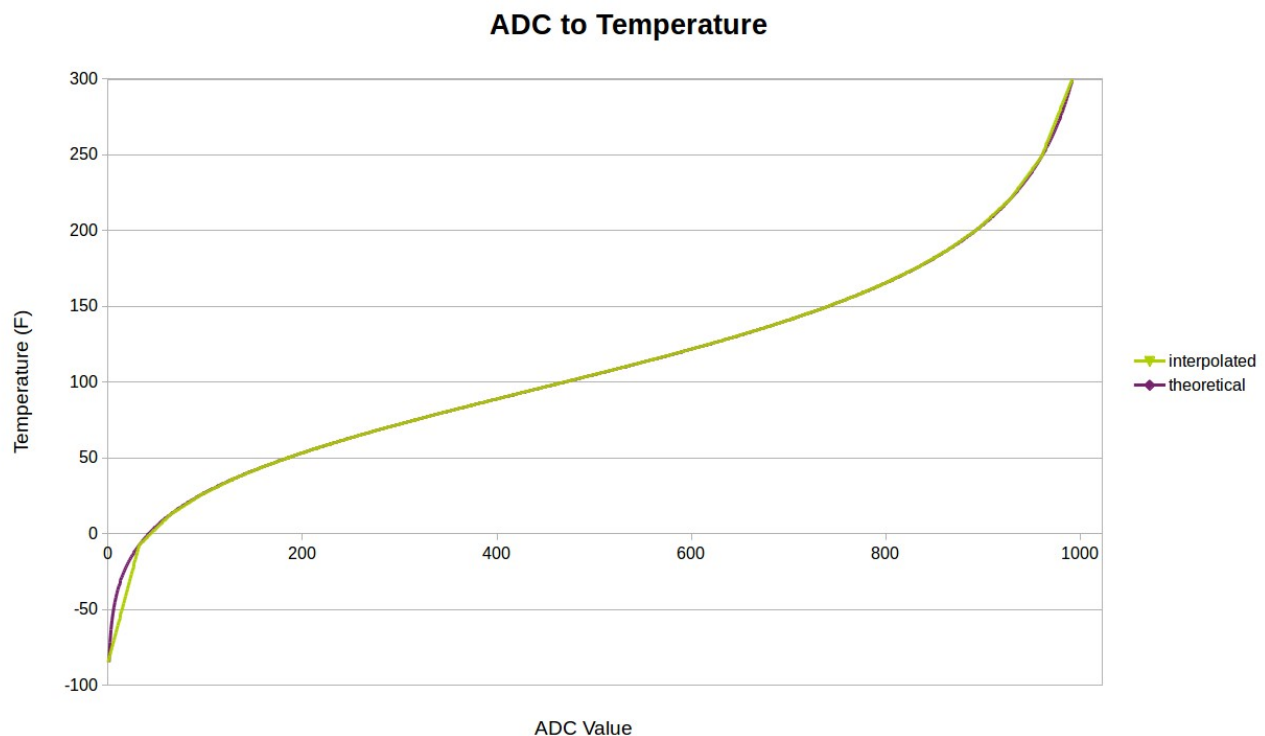
So we have this component that reacts to temperature, but how do we actually read it using a micro controller? The answer to that question is actually quite simple, in fact you've probably used a similar technique without even knowing it! If you have ever used a potentiometer with an Arduino, then you're almost an expert. The basic concept is called a voltage divider, and it utilizes the idea of series resistors having the same current.

$$V_{RS} = \frac{VCC}{RS + RT}$$

The voltage divider is a very common circuit, found in almost all electronic designs. Any time two resistors are in series with one another, the voltage over one will always have a ratio of the total voltage over the pair. In our case we are using the fact that the thermistor resistance  $R_T$  will be varying; therefore, the voltage drop over  $R_S$  will change accordingly. This is similar to a potentiometer except a potentiometer allows full range to either VCC or GND because one of the two resistors can drop to nearly  $0\Omega$ . In our case, we can't "sweep" all the way to GND because of the constant  $R_S$  value. The equation above shows how we can determine the voltage over  $R_S$  as a function of  $R_T$ .

The nice thing about this equation is that we can physically measure  $V_{RS}$  using the ATtiny84's built-in analog-to-digital converters (ADC). Once we have that, it's simple enough to solve for  $R_T$  and use the equation from step 2 to find our temperature. The only problem is that a micro controller isn't good at doing complex math, so we have to use an alternative method of calculating the result.

## 4: Function Interpolation (approximation)



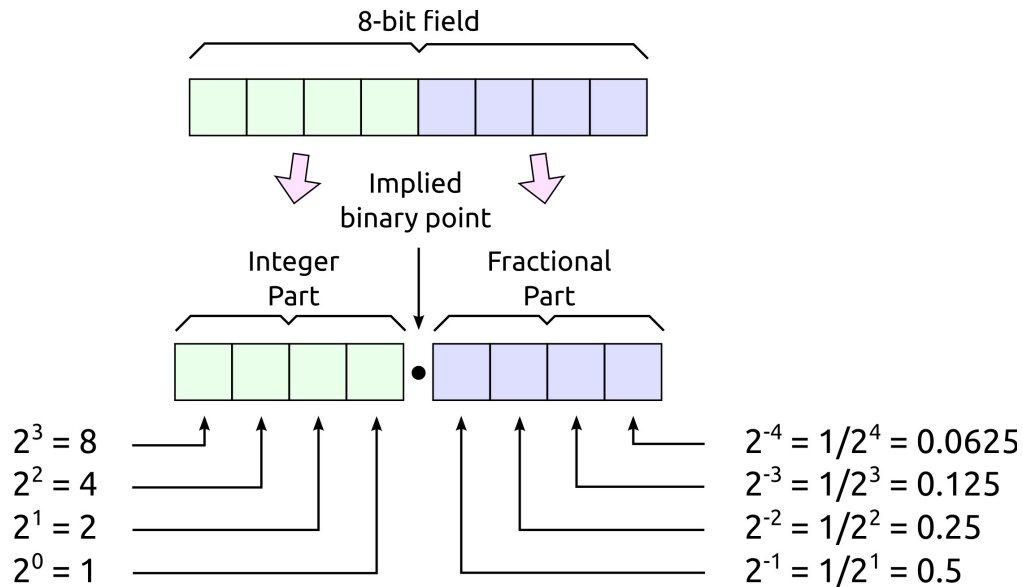
The graph above is the resulting temperature vs. ADC curve when I combine the equations from steps 2 and 3. In order to calculate all values on the graph, the ATtiny84 would have to compute a natural log as well as floating point division. Since neither of those tasks are strong points for the 8bit micro controller, I looked to something that it is good at; memory look up.

Reading from memory is fast, but in the case of the ATtiny84 it can still be resource expensive. The ADC on the chip returns a 10bit value which can range from 0V to VCC. If I were to store the entire graph in a lookup table it would take 2048 bytes, which is 25% of the ATtiny's flash memory. Keep in mind that I am going to need some of that memory later on for character strings to generate the menus. In order to make this task easier, engineers often use a technique called function interpolation.

The most common way of interpolating a function is to use line segment approximations of the original curve. In the image above, the purple line is the theoretical temperature curve, and the green line shows the line segment approximation. I chose to quantize the graph by dividing the independent variable into 32 segments. Each segment can be represented as a straight line with a given slope and y-intercept. This helps combat the memory issue because now I just have to store two arrays with 32 elements each. The final implementation of the look up table (LUT) will use 16bit integers. This means that I will only use 128 bytes of flash in comparison to 2048 if I were to store every value of the function.

Function interpolation drastically reduces memory resources, but as the image above shows, it does reduce accuracy towards the non-linear regions. This imprecision can be avoided as long as we select a proper  $R_s$  value that drives the linear portion of the curve within the selected temperature range (50-200°F). I've attached an excel document that contains all the calculations that I used to determine my values for the LUT. The spreadsheet allows you to use different thermistor characteristics as well as experiment with different  $R_s$  values. It also generates the plot you see above along with the percent error.

## 5: Fixed Point Arithmetic



### 8-bit binary 4.4 fixed-point representation

Representing integer numbers like 2, 5, and 17 in binary is simple. Conversely, what if I wanted to represent fractional values like 0.5 or 1.1? At first glance, this seems like a very difficult problem, but the solution actually turns out to be quite trivial.

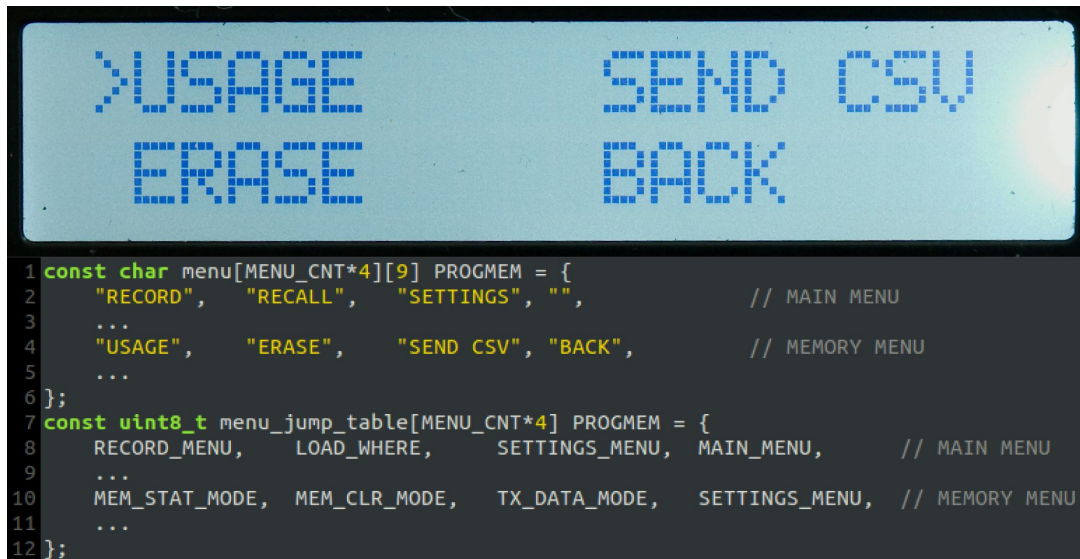
To convert a binary number to decimal we take the base value which is 2, and raise it to the current digits location from the radix point (radix being the general name for a decimal point). The same logic applies to digits to the right of the radix point, except now their exponent has to be negated. For instance, a 1 located two bits to the right of the binary point would have decimal value  $2^{-2} = 1/2^2 = 1/4 = 0.25$ . The graphic above shows this same calculation, but provides a more physical explanation than I can put into words. In the case of the pyrometer readout, we will have to make a decision as to how many bits of precision will provided sufficiently accurate data without excessive memory usage. The excel spreadsheet that I linked to in the previous step contains a self generated plot of the fixed point and interpolation error that is incurred throughout this process.

## 6: Using the Look-Up-Table

```
1 const uint16_t slope[] PROGMEM = { 0x0000,  
2                                     0x50B2,  
3                                     ...  
4                                     0xCF66,  
5                                     0x0000};  
6  
7 const int16_t y_int[] PROGMEM = { 0x0000,  
8                                   0xFE44,  
9                                   ...  
10                                  0xAE54,  
11                                  0x0000};  
12  
13 uint32_t adc2f(uint16_t adc){  
14     /* READ VALUES FROM LOOKUP TABLE */  
15     uint32_t seg_slope = pgm_read_word(&slope[adc>>5]);  
16     uint32_t seg_y_int = pgm_read_word(&y_int[adc>>5]);  
17     /* COMPUTER THE TEMPERATURE */  
18     return (((seg_slope*(uint32_t)adc)>>11)+seg_y_int);  
19 }// adc2f(uint16_t adc)
```

Now that we have the look up table for these line segments, we need to write code that will utilize them to calculate the temperature. The process starts with the analog to digital converter measuring the voltage at THERM\_VAL in step three's schematic. The ADC will return a 10bit value that we can use to index the array and find the given slope and y-intercept pair. To get this index, simply mask off the upper 5bits from the ADC value, and use the found slope, y-intercept, and full ADC value to compute the temperature. It basically boils down to a single multiply and add operation in C code. The key thing to pay attention to in the function above, is the shift right 11 operation which keeps the binary points aligned just like in the previous step.

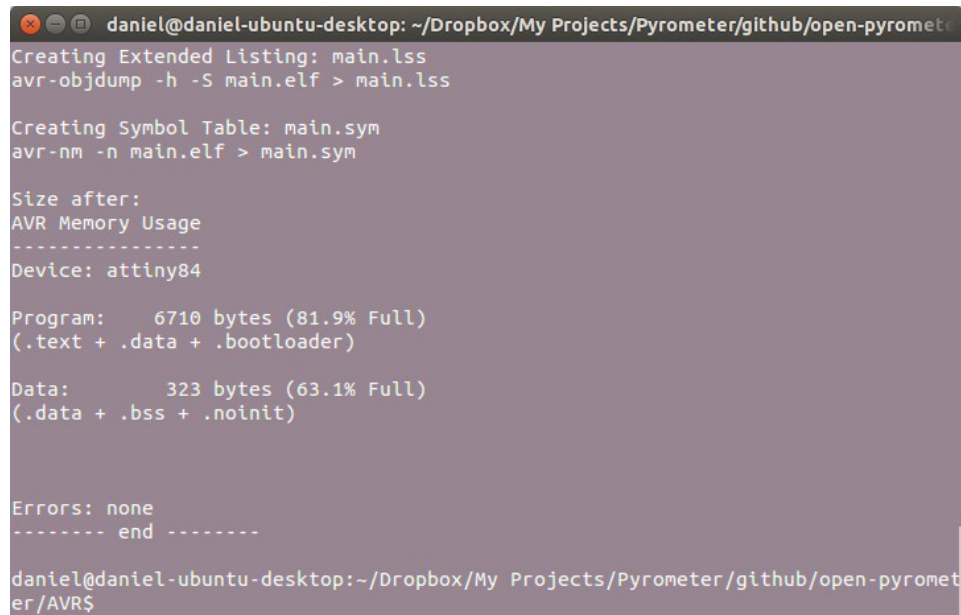
## 7: Menu Interface



In order to navigate the pyrometer's many different options, I had to develop a well structured menu system. Like most graphical user interfaces, I wanted a list of options to choose from, and also a method of “feedback” to show the user which item they have selected. With such a limited resolution display and a non object oriented language, this task proved to be a bit challenging.

The first thing I designed was the overall layout of each menu page. Due to the limited screen dimensions, I chose to divide the display into 4 quadrants that would each contain a menu option. With this in place, having a pointer character next to the user's selection restricts each option to having a length of 9 characters. As the code snippet above shows, each menu label is stored in a fixed length array with null strings placed where no option is provided. The second component to the menu system, is a jump table with #defines that tell a switch case handler where to look to display the next menu page. For every label in the menu array, there must exist an element in the jump table. With this structure, it allows me to create an easily expandable menu system with a small memory footprint.

## 8: Programming



```
daniel@daniel-ubuntu-desktop: ~/Dropbox/My Projects/Pyrometer/github/open-pyromet
Creating Extended Listing: main.lss
avr-objdump -h -S main.elf > main.lss

Creating Symbol Table: main.sym
avr-nm -n main.elf > main.sym

Size after:
AVR Memory Usage
-----
Device: attiny84

Program:   6710 bytes (81.9% Full)
(.text + .data + .bootloader)

Data:      323 bytes (63.1% Full)
(.data + .bss + .noinit)

Errors: none
----- end -----

daniel@daniel-ubuntu-desktop:~/Dropbox/My Projects/Pyrometer/github/open-pyromet
er/AVR$
```

I won't be going into too much detail in this step because there are plenty of tutorials out there that can explain the process much better than I can (links below). As I mentioned in my most recent blog (link???), I've been using an Arduino as a full blown AVR programmer for as long as I can remember (6 years). The benefit to this is that it costs absolutely nothing when using the free tools such as the AVRGCC compiler and the avrdude programmer. The code for this project can be downloaded from github. It contains all source files including the makefile needed to compile and program to the chip.

When using an Arduino as an ISP with avrdude make sure to set the baud rate and programmer flags in the avrdude command line.

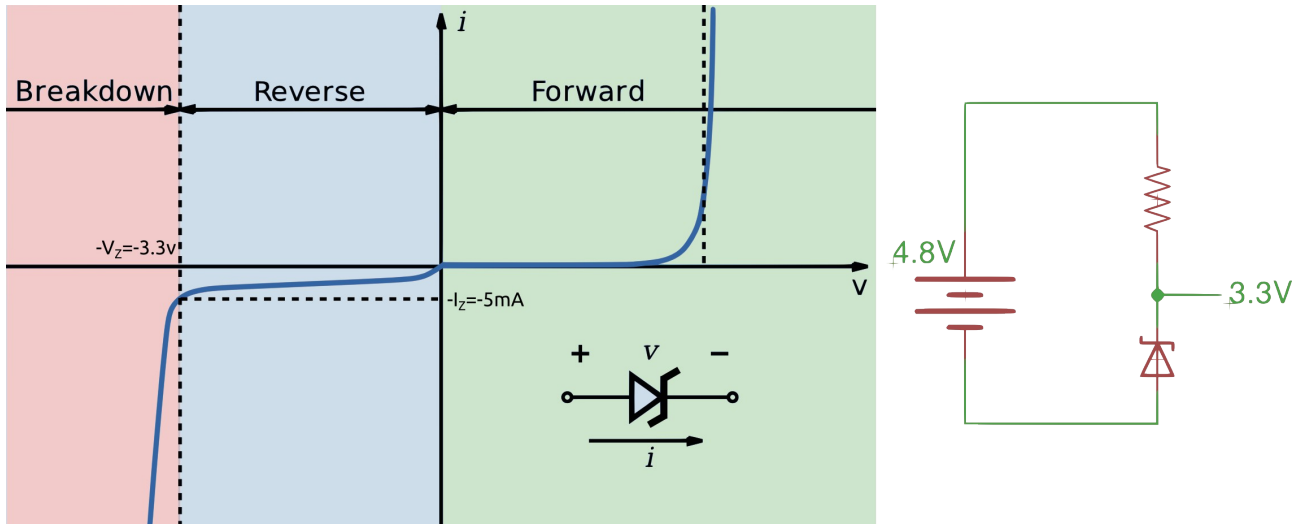
-c avrisp -b 19200

Arduino as ISP tutorial: <http://www.instructables.com/id/Turn-Your-Arduino-Into-an-ISP/>

Ladyada avrdude tutorial: <http://www.ladyada.net/learn/avr/avrdude.html>

github source code: <https://github.com/hankedan000/open-pyrometer>

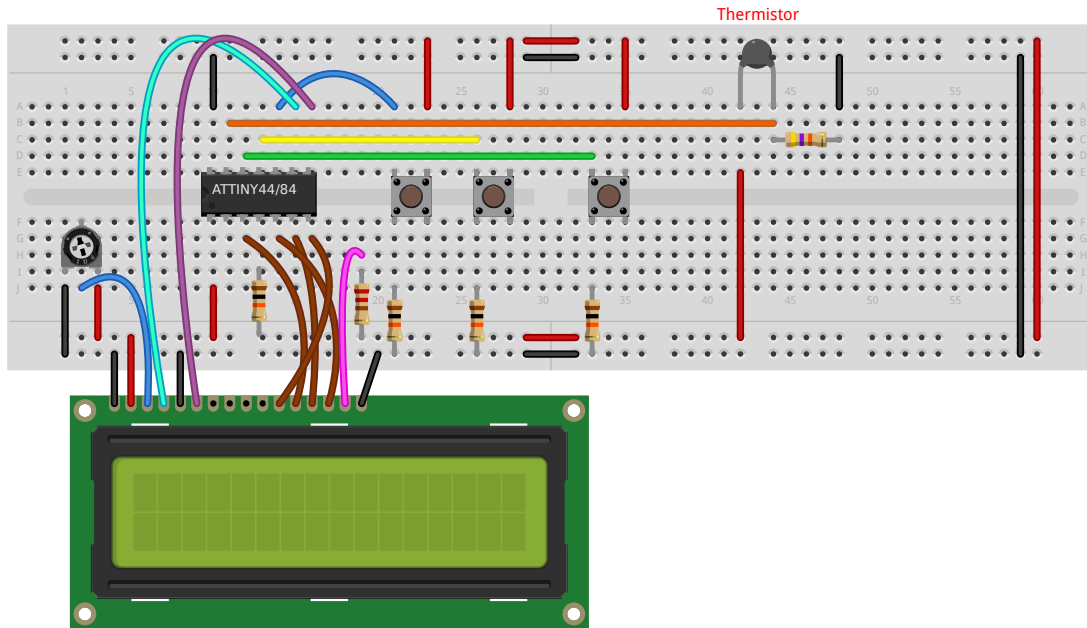
## 9: Voltage Regulation



Since this device is battery powered, I needed an efficient voltage regulator. The typical LM78xx series of linear voltage regulators simply wouldn't work since they require around 40% more voltage than their target output. I also didn't want to go with a switching power supply since that requires a high part count, and therefore higher cost. The next best regulating technique is to use a zener diode shunt regulator.

Zener diodes are device that are designed to operate quite efficiently in their break down region. A typical diode will start to build up heat once in the breakdown region, but a zener diode's characteristics allow it to hold the voltage at near constant. For this reason, a zener regulator is a perfect application for a power device. The only critical part to designing a good zener regulator, is to make sure that the diode is always operating in the breakdown region. This can be achieved by simply putting a resistor in series with the diode. The schematic shows how the ideal voltage drop across the resistor will always be 1.5V. If I want at least 5mA of current to flow through the diode, then I just need to use ohms law to solve for the resistor value. I used 10mA, just to give myself a bit of wiggle room, but ohms law says  $R = 1.5\text{V}/10\text{mA} = 150\Omega$ .

## 10: Assemble PCB



Once all the hardware and software has been finalized, it is time to set this project in stone... or solder. I chose to make a custom PCB since my university has a CNC isolation mill, but there are plenty of other options as well. Mouser sells PC prototyping boards that you can solder the component together with. The benefit to this (<http://www.mouser.com/ProductDetail/BusBoard-Prototype-Systems/SB1660/?qs=sGAEpiMZZMtgbBHFKsFQgo04jOHntgNPR1pgBh9E9%252bI%3d>) specific protoboard is that it has the exact same layout as a breadboard, so most of your circuit will be a direct crossover. I provided all the EagleCAD (MultiSIM BLUE???) project files below for you to make the exact same board that I made.

## 11: Building the Tire Probe



In order to measure the temperature of the tire properly, pyrometers usually use probes that have a sharp tips. This allows the thermistor to measure the internal temperature of the rubber, since the surface usually cools quite quickly. Considering that I can't put that much pressure onto the thermistor itself, I had to find a way to attaching some type of metal extension to it. I chose to use copper because it has a high thermal conductivity coefficient, and it is readily available.

To make the probe, I knew that I would have to make a good thermal bound between the thermistor and the probe material itself. After numerous attempts, I finally settled with the method above. It involves going into my scrap bin and pulling out a few inches of 12AWG copper house wire. I then used a hammer to flatten one end of the wire until it was nearly paper thin (surprisingly easier than it sounds). Once I had the one end flattened, I used something similar in size to the thermistor to roll the copper tab into a hollow tube. I then used some sand paper to grind off the excess copper on the ends, and then filled the hollow tube with thermal paste. Lastly I put some heat shrink tubing around the entire thing just to make sure that the thermistor and paste didn't fall out. This process resulted in a nearly continuous thermal connection from the tip of the copper to the thermistor itself.

## 12: Enclosure



The last step to this project is to put the pyrometer into some type of enclosure. When it comes to choosing an enclosure, I always focus on a few key features: power source, material, dimensions, PCB mounts, and application. In the case of this project, I wanted the pyrometer to be powered by common household batteries: AA, AAA, or 9V. I decided to use AA batteries because they cost less than 9 volts, and I already have a bunch of rechargeable AAs. Picking a size for the enclosure was difficult because I had to choose something wide enough for the LCD, but not be too thick to hold in my hand. Next up is choosing a material. ABS plastic is typically a good choice because it is soft enough to cut with a dremel tool and small files. Lastly, I knew that I wanted some way of mounting the PCB inside the case. This enclosure used standoffs to suspend the board, but some enclosures use slots or grooves to hold the PCB. Mouser provides a wide variety of enclosures to choose from, so finding one that fulfilled all these requirements was a walk in the park.