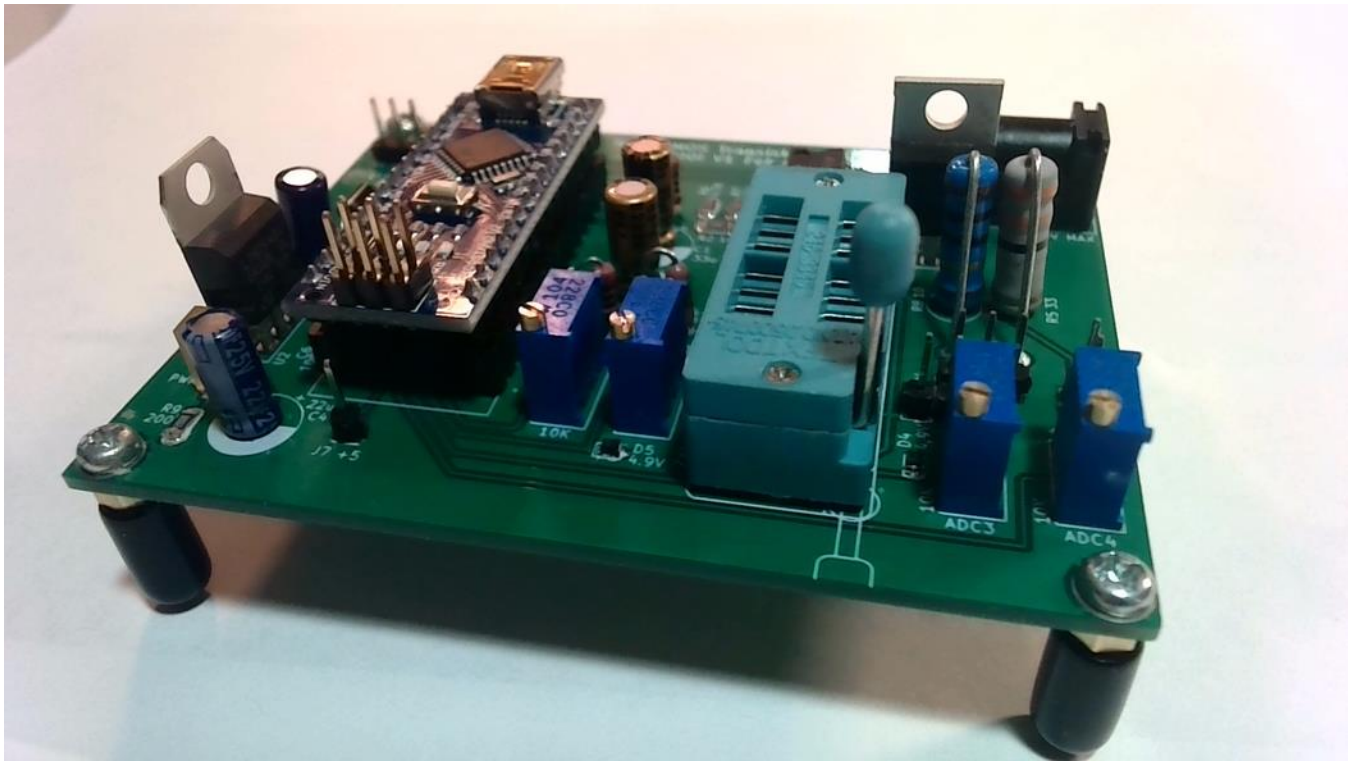
NPN-NMOS TRANSISTOR TRACER MANUAL

HARDWARE: V1 Software: V0.2b

DRAFT DOCUMENT

Dave Rajnauth, VE3OOI



Contents

Authors Notes

This device was designed only to perform analysis on NPN Bipolar Junction Transistors, N-Channel Enhancement Mode MOSFETS as well as Diodes including Zener Diodes.

Both the hardware and software are provided with NO WARRENTY or SUPPORT. You are on your own.

Various documents and files for this project are hosted on GitHub (https://github.com/drajnauth/TransistorTracer). This includes PCB boards (KiCad files and gerbers), software, and documentation (including this document).

This document includes very important information, and you need to read the contents carefully. Failure to read and follow the instructions in the document WILL cause headaches and may cause you to pitch the board against a wall.

This document uses the following syntax:

1. Anything in Red indicates something very important which could negatively impact the board. Please read carefully.
2. Anything in Deep Red indicates a particularly important note. Please pay attention.
3. Commands are highlighted in orange.
4. Changes to parameters defined by the Arduino #define directive are also highlighted in orange.

Finally, this is a powerful tool, and you are expected to understand how a transistor works and why you need the various traces detailed in this document. It has features which can make your day or frustrate the heck out of you.

Hardware Build Notes

1. Have a copy of the schematic available during the build.
2. **DO NOT INSERT THE ARDUINO AND POWER UP (i.e., Apply Vcc) BEFORE CALIBRATION IS DONE**
3. Do your best to get parts as labelled.
4. Early versions of boards have **incorrect** silkscreen values for POTS. **They MUST be 100K.**
5. The TIP120 can be replaced with a 2SC5706 power transistor.
6. The 1N5817 can be replaced with any 1 Amp Shottky Diode
7. The board can be powered from either J8 or J5. Maximum voltage is 14 V. **Exceeding this could damage your Arduino.**
8. R1, R3, R6, R7, R10, and R11 resistors cannot be substituted and need to be as close as possible to the indicated values. Substituting resistor values could dramatically alter the performance of the device.
    - ✓ R1 and R3, in combination with C1 and C2, are part of a low pass filter that converts an Arduino generated PWM signal into a DC voltage.
    - ✓ R6 and R7 set the gain for the LM386 Opamp to amply the PWM generated DC voltage to Vcc to control the TIP120/2SC5806 power transistor.
    - ✓ R10 and R11 form a voltage divider to allow the Arduino to measure the value of Vcc.
9. Capacitors C1, C2, C7, C8, C9 and C10 cannot be substituted and need to be as close as possible to the indicated values. Other capacitors can use similar values but not exact values.
    - ✓ C1 and C2 are part of a low pass filter to convert the Arduino generated PWM signal to a DC value.
    - ✓ C7, C8, C9 and C10 also form a low pass filter to smooth out the voltages read by the Arduino ADC. Values too large here will result in ADC errors.
10. The D3, D4, D5, and D6 4.9V Zener diodes **are optional** and are there to protect Arduino ADC from overvoltage. They are not necessary for the operation of the device.
11. Resistors R10 and R11 form a voltage divider to allow the Arduino to measure the value of Vcc. If you change these values, ensure the ratio will allow 14V to deliver about 4.8V to the Arduino. Measure and note the values, rounding to the nearest whole number, of the resistors you use. You will need to enter them in the Arduino code during the calibration.
12. R4, R5 and R8 set the maximum current to the base/gate or to the collector/drain of the transistor under test. You can substitute values here. Keep R4 close the 500R (i.e., within 10 or 20R). R5 and R8 **must be** 2W resistors. They dissipate heat from several hundred milliamps of current. Measure both resistors with Volt Ohm Meter and note the values, rounding to the nearest whole number, of these resistors. You will need to enter them in the Arduino code.
    - ✓ You are provided with two resistor values to limit the current delivered to the collector/drain. You can select R5 or R8 via jumper J4 (RCSEL). Whenever you change this jumper you must change the resistor value via the Arduino CLI. See the ***CLI Resistor Configuration Commands*** section.
13. Solder all SMT parts first.
14. Solder all through hole components from the underside
    - ✓ Arrange the pots such that turning clockwise increases the resistance between the wiper and the grounded leg.
15. Once you are finished **remove all** jumpers from header pins. **Do NOT insert the Arduino**. Power up and check LED operation and check for appropriate voltages at Arduino +5V pin and LM358 Vcc pin (pin 8).
16. Disconnect power and proceed to calibration.

Calibration

Note the instructions below are for hardware Version 1 and software version 0.1b.

1. **DO NOT CONNECT ARDUINO AND POWER UP THE BOARD BEFORE CALIBRATION IS DONE.** Failure to do so may result in damage to the Arduino.
2. For all measurements, use one consistent VOM. Try not to use different meters as they may introduce additional errors.
3. Before powering up Arduino, you need to calibrate pots to not damage the Arduino's ADC pins.
4. Remove Arduino. Remove all jumpers from all trim pot jumper headers pins.
5. Connect Vcc to the board and measure the voltage at the +5V terminal pin (J7). Record the voltage.
6. Connect a jumper wire from +5V terminal to each pot's jumper "+" pin (J1, J2, J6, J3). "+" is marked on the silkscreen. The plus pin is connected to the top end of the pot. The wiper is connected to the Arduino's ADC and forms a voltage divider.
   - ✓ Adjust each pot until you get as close as possible to 1.75V output at the pot's wiper.
7. Verify the voltage at ADC 0 pin and Digital Pin 12 (or junction of R10 and R11) is below 5 Volts. If this voltage is above 5V, you need to verify the value of Vcc (less than 14V) and the values of R10 and R11. Ensure solder joins are good.
   - ✓ Note that Resistor R10 and R11 form a voltage divider to convert Vcc to a voltage below 5V. The Arduino reads this voltage to ensure a proper Vcc is applied to the board.
8. Once all pots are adjusted and the Vcc voltage divider is working, then there is little danger in damaging the Arduino's ADC. Insert Arduino and load the code. See the software loading section of this document.
9. Connect via TTY console to Arduino. Enter the C R command to display all ADC values from each pot.
10. Note that each pot has ADC1, ADC2, ADC3, ADC4 marked on the silkscreen. Each jumper associated with the pot is marked as AD1, AD2, AD3, AD4. This corresponds to the ADC number on the Arduino.
11. Connect 5V to the same jumper in step 3. Note the ADC number of the jumper and pot.
    - ✓ Adjust the pot's wiper (fine adjust needed) until the corresponding ADC displayed by the Arduino is **EXACTLY** 364. Repeat for all jumpers/pots.
    - ✓ Measure and record the voltage at the wiper of the adjusted pot. It may be slightly different from the initial 1.75V setting.
    - ✓ Enter the R command to reset the Arduino when done.
12. Complete and record the following calculations for each pot:

    Note V(wiper) is the *updated wiper voltage* you measured in the prior step and V(applied) is the voltage you measured previously at the +5V pin (J7). ADC_Value is the value of the ADC that you set the pots to (i.e., 364).

    For the example calculations below V(wiper)=1.75, V(applied)=5, ADC_Value=364. Subsititue the values you measured.

    - ✓ Calculate VOLTAGE_DIVIDER = V(wiper)/V(applied)

e.g., 1.75/5 = 0.357

Note V(wiper) is the updated voltage you measured in the prior step.

- ✓ Calculate ADC_CONVERSION = ADC_Value/V(wiper) *e.g., 364/1.75 = 208.0*
- ✓ Calculate VOLTAGE_CONVERSION = VOLTAGE_DIVIDER * ADC_CONVERSION
  *e.g., 0.357*208 = 78.0*

13. Take the VOLTAGE_CONVERSION and update the #define ADC_CONVERSION ((float)78.0) in the defines.h file. This can be done after loading the program into the Arduino IDE.

14. Using the measured resistor values for R10 and R11 calculate the voltage divider for the Vcc voltage sensing. Calculate the divider R11/(R11+R10). Next calculate the ADC conversion ratio by taking the voltage measured at the J7 pin (+5V pin) and dividing it by 1024. Update the #define VOLTAGE_SENSE_CONVERSION parameter in the defines.h file.

For example,

Suppose resistor divider measurements are R10 is 991 and R11 is 497. Divider is 497/(497+991)=0.3340

Suppose voltage at J7 pin is 4.88V so ADC Voltage conversion is 4.88V/1024 = 4.766mV. i.e., 1 ADC count is 4.766mV

With resistive divider each count is 4.766mV/0.3340 = 14.26mV = 0.01426V

#define VOLTAGE_SENSE_CONVERSION 0.01426

15. Use your measurements for the Base and Collector resistors to update the following defines in the defines.h file.

    For example, RC resistors are 32 and 10. The base resistor is 462.

#define RC_RESISTORHIGH 32

#define RC_RESISTORLOW 10

#define RB_RESISTOR 462

1. Reset the Arduino and measure the max voltage generated by the Arduino PWM circuit for the base/gate and collector/drain. Ensure no jumpers are connected. Enter the P 3 255 command and measure and record the voltage after the 200R resistor connected to D3 pin (i.e., the side of the R1 200R resistor connected to the electrolytic capacitor). Reset the Arduino and repeat for pin 10. Enter the P 10 255 and measure and record the voltage after the R3 200R resistor is connected to D10 pin. Again, measure the voltage at the side of the resistor that is connected to the electrolytic capacitor. Update the following defines in the defines.h file with the voltages you measured. In the example below, 4.6 volts was measured.

#define MAX_VC_VOLTAGE 4.6

#define MAX_VB_VOLTAGE 4.6

2. Save all modified files. Compile and load the Arduino program before continuing. The above defines **must** be in place.

3. Complete the following. DO NO RESET OR POWER OFF ARDUINO UNTIL ALL STEPS BELOW ARE COMPLETE. Failure will result in redoing the steps.

1. Set the base resistor value using the S R B nnnn command where nnnn is the resistor value that you measured and entered previously. *E.g., enter S R B 462 to set the base resistor to 462.*
2. Set the jumper J4 (RCSEL) to a resistor for the sweep. Use the S R C nnnn command to define the Collector resistor nnnn that the jumper has selected. E.g., Enter S R C 10 to set the collector resistor set by the jumper J4 to be the 10R resistor. Alternatively, enter S R C 32 if the jumper J4 has selected the 32R resistor.
3. Connect Vcc to the board. **If you change Vcc, the *following* calibration step needs to be repeated**. E.g., if you use 12V to calibrate the board and you connect 13.8V in the future, the board will need to be recalibrated for the new Vcc value.
4. The final calibration step is to get the Arduino to measure the min and max voltages available when generating a trace. Insert jumpers in J1, J2, J6, & J3. Ensure there is nothing connected to the ZIF socket. Enter the C M command. This takes a few minutes to complete. **If you miss this step, the tracer will not work properly**.

4. Update the EEPROM with the configuration by entering the E W command. **If you miss this step, the tracer will not work properly next time it's restarted.** The Arduino will now power up with the values you defined above.
5. Your transistor curve tracer (V1) is now ready.

Transistor Tracer Software

The transistor tracer needs two software components:

1. The Tracer uses an Arduino Nana with special software that controls the hardware and produces the various traces.
2. The Tracer also requires an external plotting program. A small versitile Python program was written to generate generic plots. This program is one option that can be used to generate plots on the fly. A second option is to use another program such as Microsoft Excel or Apache OpenOffice Calc to generate the charts. The Arduino software simply displays the data to be charted on the serial console which can be copied and pasted into any other charting program.

Arduino Nano Firmware

This software for the Arduino Nano was written using Visual Code with the PlatformIO plugin instead of the Arduino IDE. The transistor tracer software is FULLY compatible with the Arduino IDE. If you are familiar with PlatformIO, I suggest you create a new Visual Studio project for Platform IO and load the source files. Directions to accomplish this using Visual Studio is beyond the scope of this document. Instead, directions for loading using Arduino IDE will be given below.

It is assumed that you understand how to use the Arduino IDE to compile and load sketches on Arduino boards. If you are new to using the Arduino IDE, I suggest that you review tutorials on the internet to prepare you. However, I strongly suggest you seek assistance from someone that is familiar with the Arduino IDE to reduce your stress and increase your chances of success.

Arduino IDE Loading Instructions

1. From the https://github.com/drajnauth/TransistorTracer github repo, download the zip file for the entire repo. This will duplicate all files in the repo.
2. Create a directory on your local hard disk and provide a name that you would like to call the Arduino sketch (e.g., “TransistorTracerSketch”).
3. Copy all the files from the “Software/Nano NPN-NMOS Transistor Tracer v0.2b” directory to the above created directory.
4. Rename the **main.cpp** file so it matches the directory name you created with an .ino extension (e.g. TransistorTracerSketch.ino)
5. Double click the ino file or open the sketch in your Arduino IDE.
6. Ensure you have the Arduino Nano board selected and the appropriate com port selected. Complete a test compile to ensure it compiles.
7. Load the software on the nano.
8. Once the software is loaded, the onboard LED should blink every second indicating Vcc is not connected. If you connect via the Arduino Console, there will be a Vcc error displayed. This is normal until the Nano is inserted onto the tracer board and the board is powered up. **DO NOT INSERT THE ARDUINO AND POWER UP THE BOARD BEFORE CALIBRATION IS DONE.**

Microcontroller Console Python Program

The console program was written to plot data from any microcontroller. It WAS NOT written exclusively for the transistor tracer. The program was meant to be a quick and dirty program that can plot charts from any TTY (serial port) connected microcontroller. A display could have been incorporated onto the Arduino, however this would make this project complex, and the Nano software would be a beast to write and maintain.

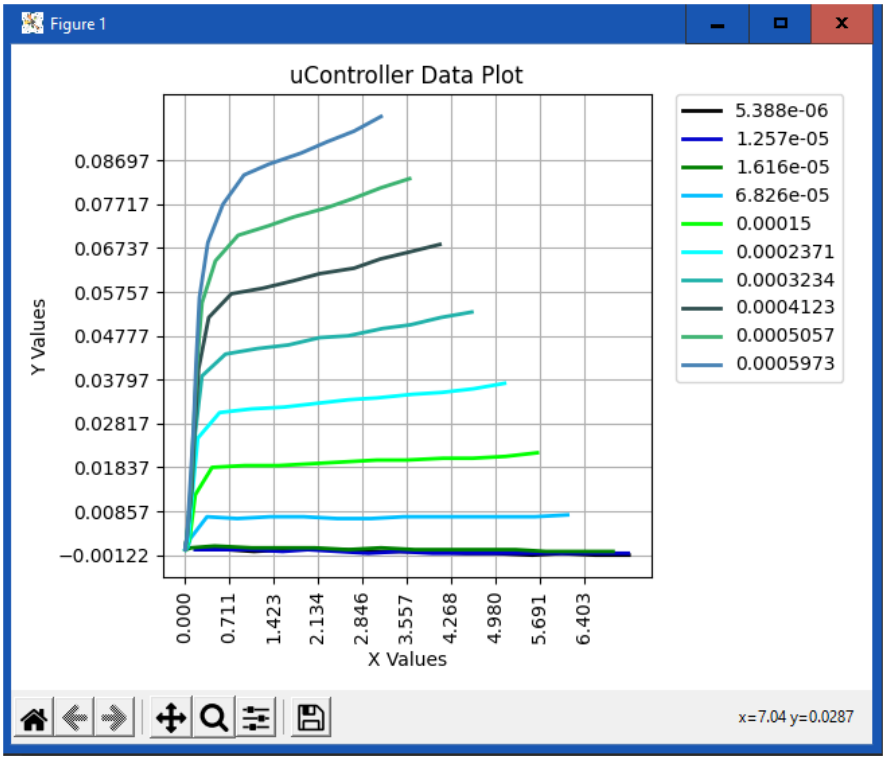


Figure 1: Sample Plot from Python Program

You can also copy and paste the generated data from the Arduino Console into Microsoft Excel or Apache OpenOffice Calc and generate the charts yourself. The Arduino simply outputs the data to the console and the python program captures it, stores it into a buffer and then plots the data for you.

A presentation was provided giving a high-level overview of how this program works. The associated ppt has been included in the documentation folder and you should refer to this regarding how to use the python program.

If you are familiar with python and loading libraries, then load the source code and associated libraries. Python files are in the "Software/Python Microcontroller PC Console" directory.

However, if you are not familiar with python an executable file is provided in the python directory. The file's extension is. ex_. Once you have this file on your local PC, rename the file to be .exe. Some browsers will block you from downloading executable files and this is a crude attempt to circumvent such checks.

Simply double click the executable and accept the warning about the software being unverified. Do this at your own risk. The software I placed on my repo is free of malware.

Note that the program will give an error if no com ports were detected. You must plug in your Arduino to the USB port before running the console program.

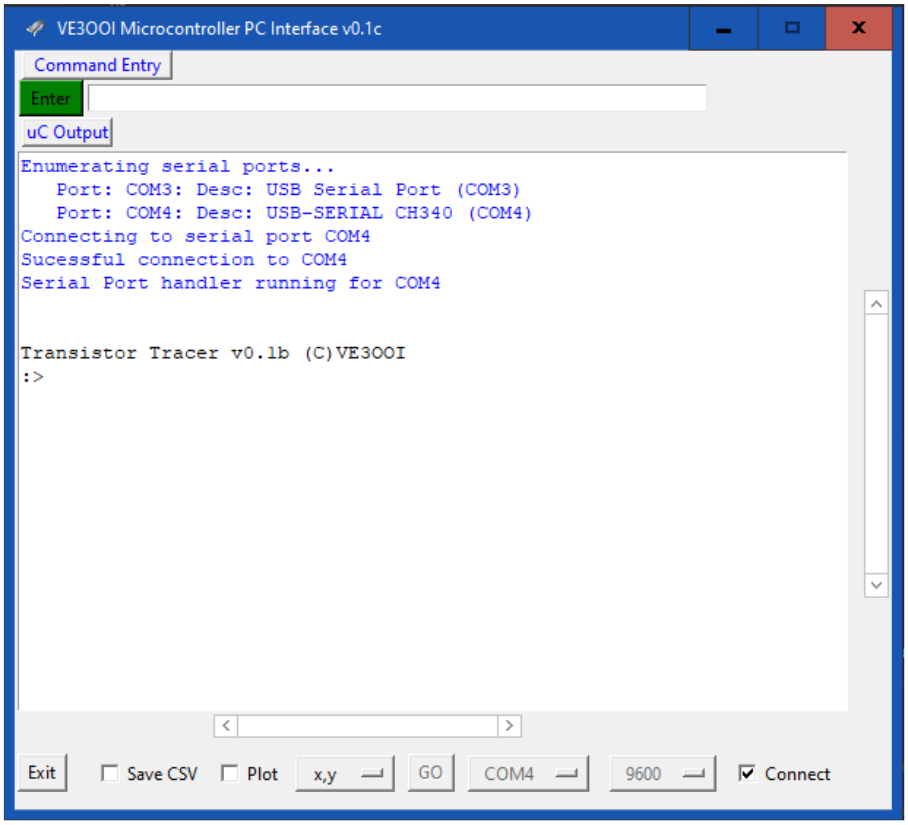


Figure 2: Python Program Interface

Command Line Interface

The Arduino has a command line interface (CLI) which is accessible via a terminal emulator connected to the Arduino USB connection. That is, a serial comport connection from your PC using either the Arduino console program or the Python console program. The CLI may be accessed via any TTY console program (e.g., putty). This is the only way to control the transistor tracer.

Performing Sweeps

1. The transistor tracer generates a trace by sweeping a set of voltages. These DC voltages are generated by the PWM circuit.
    - There is a base/gate PWM voltage (Pin 3) that is applied directly to the transistor under test.
    - There is a Vcc PWM voltage (Pin 10) that is *amplified* by the LM358 opamp and a high-power transistor (TIP120 or SC5706) in a common emitter configuration.
2. There is a "Control" voltage. There is also an "Output" voltage.
    - The "Control" voltage is generated by the Arduino PWM circuit.
    - The "Output" voltage is delivered to the device under test (DUT).
    - The conversation between the "Control" and "Output" voltages is calculated when you perform a calibration.
3. The arduino uses the "Control" voltage to sweep voltages to complete a trace. It does not use "Output" voltages. "Output" voltages are measured by the ADCs.
4. Since the base/gate "Control" voltage is fed directly to the transistor under test, the base/gate voltage is essentially the "Output" voltage to the transistor. However, since the Vcc "Control" voltage is amplified, the "Output" voltage fed to the collector/drain of the transistor under test is **larger** than the "Control".
5. ***When you set voltage ranges, you are setting the "Control" voltage that the Arduino outputs.*** However, you can set the "Output" voltage range and the Arduino will calculate the "Control" voltage for you and use that.
6. Normally, the calibration will calculate the minimum and maximum voltages available and use those values for the sweep. However, you can manually set the "Control" or "Output" sweep voltages using the S I or S O commands. Once again, if you set "Output" voltages, the Arduino converts this into "Control" voltages which are used for tracing.
7. You can also manually set the voltage delivered to the DUT using the S V command. This only uses the "Control" voltage. However, this is a one-time setting. It's not used in a sweep. **BE CAREFULL SETTING VOLTAGES MANUALLY** – you could destroy (i.e., burn out) the collector resistor or the transistor under test if you exceed its maximum ratings.
    - *When you set voltage ranges, you are restricting values that the Arduino will deliver to the DUT. This restricts values that the S V command can use.*
8. **MOST IMPORTANT:** Voltages are entered in mV (Milla Volts). i.e., to specific 1 volt you enter 1000 mV. The CLI does not support floating point or real numbers. Only integers can be entered.

The following diagram details the pin out for the Zif sockets. Ensure transistors are connected properly. Incorrect orientation would result in "wonky" charts and in extreme cases damage to the transistor or resistors on the board.

CLI Summary

Commands listed below follow a specific syntax. Anything in [ ] means that additional input is needed. Either a sub command or a number. Note that numbers MUST BE INTEGERS. Floating point or real numbers are not supported. For example, to ensure a real number multiply it by 1000 e.g., voltages are entered in millivolts.

The | character indicates an or condition. Either of the values on either side of the | can be entered buy not both. E.g., C [R|M] means that C R or C M can be entered i.e., R OR M can be entered.

The Arduino tracer store it configuration to is local EEPROM. The EEPROM is not updated when configuration is changed. You must explicitly save the running configuration to the EEPROM configuration.

Note that Gate-Base, Collector-Drain, and Emitter-Source are the same. i.e., anything defined for base will also apply to gate.

- H — Help – Command summary
- C — Display current Calibration
- C [R|M] — Calibrate commands
  - R display raw ADC values repeatedly until the arduino is reset
  - M identifies the min and max voltage that can be provided to the transistor under test
- D — Set default values. Resets all running configuration to default.

  Reboot or reset to restore the EEPROM calibration.

  Run the C M command to reset the min and max available voltages.
- E [I|R|W] — Manage EEPROM
  - I is used to initialize the EEPROM to factory default
  - R is used to read and use the configuration in EEPROM
  - W is used to write the current running configuration to EEPROM. This will be loaded when arduino is powered on or reset.
- M — Measure and display voltages
- P [3|10] [255] — Set PWM on pin 3 or pin 10 to duty cycle from 1 to 255
- R — Reset the arduino. EEPROM configuration is loaded.
- S V [B|C] [v] — Manually set voltage v in mV
  - B is used to set the voltage for the Base/gate pin.
  - C is used to set the voltage for the collector/drain pin.
- S U [D|M|U] — Set default units for output. Decimal [D], Milla[M] or Micro [U]
- S I [B|C] [min] [max] — Set input or control min max voltage generated by Arduino
  - B sets the base/gate control voltage to sweep from min to max in mV.
  - C sets the collector/drain control voltage to sweep from min to max in mV.

- Increment auto calculated based on number of data point “P” defined below

- S O [B|C] [min] [max] Set output min max voltage delivered to transistor to sweep
  - B sets the base/gate voltage to sweep from min to max in mV.
  - C sets the collector/drain voltage to sweep from min to max in mV.
  - Increment auto calculated based on number of data point “P” defined below

- S R [B|C] [r] Set Base or Collector resistor value to r.
  - r must be configures during calibration.
  - For collector/drain, resistor is set by jumper J4 (RCSEL). Base/gate resistor is fixed and cannot be changed for this version of hardware board. Subsequent hardware versions will have different base/gate resistors.

- S P [B|C] [n] Set number of data points n to sweep between max and min
  - This is used to calculate the increment between min and max voltages defined above
  - B sets the number of data point to sweep for base/gate voltages/currents
  - C sets the number of data point to sweep for collector/drain voltages

- T N [I|IV|O|B|BC|R|G] Trace NPN Transistor
  - I - Perform an input trace. Trace Vbe vs Ic
  - IV – Perform an input trace with Vce. Trace Vbe vs Vce
  - O – Perform an output trace. Trace Vce vs Ic for various Ib values
  - B – Perform a beta trace. Trace Ib vs B
  - BC – Perform a beta trace. Trace Ic vs Bc
  - R – Perform a r_e trace. Trace Vbe vs r_e
  - G – perform a transconductance trace. Trace Vbe vs g_m

- T F [I|IV|O|K|G|R] Trace N Chan MOSFET
  - I - Perform an input trace. Trace Vgs vs Id
  - IV – Perform an input trace with Vgs. Trace Vgs vs Vds
  - O – Perform an output trace. Trace Vds vs Id for various Vgs values
  - K – Perform a FET constant K trace. Trace Id vs K
  - G – perform a transconductance trace. Trace Vgs vs g_m
  - R – Perform a R_{ds} trace. Trace Vgs vs R_{ds}

- T [I|O] Trace Diode
  - I – perform an input trace. Diode inserted between base/gate and emitter/source pins.
  - O – perform an output diode trace. A resistor must be connected in series with resistor to allow the Rc resistors not to burn up. Use resistor around 200R.

CLI Calibration Commands

This section summarizes all the commands available for calibration and configuration viewing. There are 3 commands available.

C

Display current Calibration values and the defined configuration that is currently running. This may not show the configuration that is stored in the EEPROM. E.g., you changed the configuration and did not write it to the EEPROM.

C M

Connect the Tracer to Vcc with empty socket (nothing connected) to determine the voltage sweep range for base/gate voltage and collector/drain voltage. It also calculates the conversion constants to generate output voltage as a function of control voltage.

C R

Connect the Tracer to Vcc with empty socket (nothing connected). Enter this command to display the raw ADC values repeatedly. The first column is the Vcc voltage pin (AD4), next is Vc voltage pin (AD3), Vbcontrol pin (AD1), Vb voltage pin (AD2). Press the reset button or enter the R command to reset the Arduino and stop the display.

CLI Resistor Configuration Commands

This section summarizes all the commands available for changing the resistor values.

S R C nnnn

Use the resistor value for the collector/drain (R5 or R5) specified by nnnn. Note that nnnn must be a integer number (i.e., not real or floating point number) that was set in the defines.h file during the calibration.

S R B nnnn

For this version of the transistor tracer hardware, the base/gate resistor is fixed and this command is unsupported. It will be available for a subsequent hardware revision where you can select the base/gate resistor.

S R

Display the current base/gate and collector/drain resistor that the Arduino will use for current calculation.

CLI NPN Sweep Commands

The T command is used to complete Traces. The T N command is used to complete NPN Transistor traces.

1. The following traces use the configured min and max voltages. You can adjust sweep voltages and number of data points (resolution) to be taken during a sweep. Use the S M command to adjust min & max voltages for and S P to set data points.
2. The following below outlines the various traces:

I. Input Trace: T N I

- ✓ This command sets the Vcc voltage to the maximum available voltage and then sweeps the base voltage from configured min to max at define inc (resolution). It prints out Vbe vs Ic.

II. Input Trace: T N I V

- ✓ This command is the same as above except it prints out Vbe vs Vce.

III. Output Trace: T N O

- ✓ This command sweeps the base voltage from configurated min to max with inc. For each base voltage, it then sweeps Vcc from configured min to max at defined inc. For each base voltage it prints out Vce vs Ic in the form Ib, Vce, Ic.

IV. Beta Trace: T N B

- ✓ This command sets the Vcc voltage to the maximum available voltage and then sweeps the base voltage from configured min to max at define inc (resolution). It calculates transistor beta and prints out Ib vs Beta.

V. Beta Trace: T N B C

- ✓ This command is the same as above except it prints out prints out Ic vs Beta.

VI. Beta Trace: T N R

- ✓ This command sets the Vcc voltage to the maximum available voltage and then sweeps the base voltage from configured min to max at define inc (resolution). It calculates transistor r_e (intrinsic resistance) and prints out Ic vs r_e.

VII. Beta Trace: T N G

- ✓ This command sets the Vcc voltage to the maximum available voltage and then sweeps the base voltage from configured min to max at define inc (resolution). It calculates transistor transconductance and prints out Ic vs g_m

CLI N-Channel MOSFET Sweep Commands

The T command is used to complete both NPN and N-Channel Enhancement Mode MOSFET Traces. The T F command is used to complete N channel MOSFET traces.

1. The following traces use the configured min and max voltages. You can adjust sweep voltages and number of data points (resolution) to be taken during a sweep. Use the S M command to adjust min & max voltages for and S P to set data points.

2. The following below outlines the various traces:

   I. Input Trace: T F I

      ✓ This command sets the Vcc voltage to the maximum available voltage and then sweeps the gate voltage from configured min to max at define inc (resolution). It prints out Vgs vs Id.

   II. Input Trace: T F I V

      ✓ This command is the same as above except it prints out Vgs vs Vds.

   III. Output Trace: T F O

      ✓ This command sweeps the base voltage from configurated min to max with inc. For each gate voltage, it then sweeps Vcc from configured min to max at defined inc. For each gate voltage it prints out Vds vs Id in the form Id, Vgs, Id.

   IV. Intrinsic Resistance Trace: T F R

      ✓ This command sets the Vcc voltage to the maximum available voltage and then sweeps the gate voltage from configured min to max at define inc (resolution). It calculates transistor re (intrinsic resistance) and prints out Id vs Rds.

   V. Transconductance Trace: T F G

      ✓ This command sets the Vcc voltage to the maximum available voltage and then sweeps the gate voltage from configured min to max at define inc (resolution). It calculates transistor transconductance and prints out Id vs g_m

   VI. Transconductance Trace: T F K

      ✓ This command is the same as above except it it calculates transistor transconductance constant K and prints out Id vs K.

CLI Diode Sweep Commands

The T command is used to complete both NPN and N-Channel Enhancement Mode MOSFET Traces. As well as diode Traces. The T D command is used to complete diode traces

The following traces use the configured min and max voltages. You can adjust sweep voltages and number of data points (resolution) to be taken during a sweep. Use the S M command to adjust min & max voltages for and S P to set data points.

1. Note that the diode **MUST** be oriented correctly in the socket.
2. The following outlines the various traces:

    i. Input Trace: T D I

    This traces the turn on voltage of the diode for it to conduct from its anode to cathode terminals. The diode must be connected between the base/gate socket and ground. The cathode is connected to emitter/source socket (ground). For a Zener diode, it's reversed, the cathode is connected to the base. The collector/drain pin of the zif socket MUST be unconnected. This command sweeps the base/gate voltage from configured min to max at define inc (resolution). It prints out Vb vs Id

    ii. Output Trace: T D O

    This trace identifes the voltage drop across the diode's anode to cathode. The diode will need a resistor connected between the cathode and the emitter/source socket (ground). The resistor is selected to NOT exceed the maximum current the diode can handle. The resistor also protects the Arduino pin used for base PWM output (pin 3). Failure to add a resistor could cause damage to the Arduino. This command sets the base/gate voltage Arduino pin to zero to sink current. It then sweeps the Vcc voltage from configurated min to max with inc. For each Vcc voltage, it measures the voltage drop across the collector/drain to the base/gate sockets along with current. It prints out Vf vs I.

Figure 3: Diode Connected to Resistor

CLI Sweep Limits Commands

The S command can be used to set various parameters that customize sweeps.

1. For base/gate voltage there is a min voltage and max voltage that can be set. The calibration command C M is used to set the default min and max voltage available.
2. As detailed previously (see ***Command Line Interface*** section) the Arduino sets a "Control" voltage which translates to an "Output" voltage that is delivered to the transistor under test. You can enter set the min and max "Control" or "Output" voltages. Recall the Control voltage is the PWM voltage output from the Arduino. The Output voltage is the voltage delivered to the Device Under Test (DUT).
3. Values MUST be entered as Millivolts (mV) e.g., 4V entered at 4000.
4. The following outlines the various configuration command options:

   I. Base/Gate Sweep: S I B xxxx yyyy

      ✓ This command sets the Control minimum voltage (xxxx) and the maximum voltage (yyyy) for the base/gate PWM pin. The voltage increment used for a trace is automatically calculated to the number of data points defined. Default is determined by C M calibration command.

   II. Base/Gate Sweep: S O B xxxx yyyy

      ✓ Same as above except you are specifying the "Output" voltages and the Arduino will calculate the "Control" voltages using the calibration that was performed.

   III. Vcc Sweep: S I C xxxx yyyy

      ✓ This command sets the Control minimum voltage (xxxx) and the maximum voltage (yyyy) for the Vcc PWM pin. The increment used for trace is automatically calculated to the number of data point defined. Default is determined by C M calibration command.

      ✓ Input sweeps use the maximum Vcc voltage. Therefore, you can manually reduce the Vcc voltage (e.g., for Vce(sat)) used for input sweeps by setting the maximum Vcc voltage.

   IV. Vcc Sweep: S O C xxxx yyyy

      ✓ Same as above except you are specifying the "Output" voltages and the Arduino will calculate the "Control" voltages using the calibration that was performed.

You need to enter the E W command to update the EEPROM configuration values or else these definitions will be lost at the next restart of the Arduino.

CLI Sweep Data Points Commands

The S command can also be used to set the number of values to sweep between the min and max you defined in the prior section. You can set the number of data points to be taken during a typical sweep.

Note that some sweeps such as the beta, g_m, K and R_{ds} and r_e sweeps **typically use 25% of the increment regardless of the defined value**. This is because it requires greater resolution to perform the calculations.

Example:

- ✓ If the min base/gate voltage is 0.01V and the max voltage is 4.6V
- ✓ If you set the number of points to 10, then the increment used during a sweep is (4.6 – 0.01)/10 = 0.45V
- ✓ Therefore, during a sweep, 10 measurements are taken with each increasing by 0.45V from min to max.

The following below outlines the various configuration options:

I. Base/Gate Sweep: S P B xxxx

- ✓ This command sets the number of data points (xxxx) to measure during a sweep of the base/gate voltage. The increment used for sweep is automatically calculated. This can give a smaller step size (e.g., increment of 0.1V instead of 0.4V). Default is 10 data points for base/gate.
- ✓ The range of allowed data points is defined in the defines.h file as:

```
#define MIN_NUM_VB_VOLTAGE 5        // Maximum allows points for collector/drain sweep
#define MAX_NUM_VB_VOLTAGE 20       // Maximum allows points for base/gate sweep
#define NUM_VB_VOLTAGE 10           // Default number of points
```

II. Vcc Sweep: S P C xxxx

- ✓ This command sets the number of data points (xxxx) to measure during a sweep of the collector/drain voltage. The increment used for sweep is automatically calculated. This can give a smaller step size (e.g., increment of 0.1V instead of 0.4V). Default is 10 data points for collector/drain.
- ✓ The range of allowed data points is defined in the defines.h file as:

```
#define MIN_NUM_VB_VOLTAGE 5        // Maximum allows points for collector/drain sweep
#define MAX_NUM_VB_VOLTAGE 20       // Maximum allows points for base/gate sweep
#define NUM_VC_VOLTAGE 30           // Default number of points
```

You need to enter the E W command to update the EEPROM configuration values or else this definition will be lost at the next restart of the Arduino.

CLI EEPROM Management Commands

The Arduino uses a running config that contains the various configuration parameters defined during the calibration. The running configuration is stored in RAM and is restored from a backup copy in EEPROM each time the Arduino is powered on or is reset. Each time the Arduino is reset or is restarted/powered on, it reads the configuration parameters stored in EEPROM and uses it as its running configuration. The configuration stored in EEPROM is typically not used except for the command described in this section.

If you change the running configuration, the configuration stored in the EEPOM is NOT automatically updated.

The E command is used to manage configuration values stored in the EEPROM.

The following commands outlines the various EEPROM configuration options:

I. Read EEPROM: E R

- ✓ This command reads and overwrites the current running configuration with the configuration stored in the EEPROM. Any unsaved changes made to the running configuration will be lost (i.e., changes to voltage ranges, points, resistor values, max voltages, etc.)

II. Write EEPROM: E W

- ✓ This command writes the current configuration to the EEPROM. It overwrites the EEPROM.

III. Initialize EEPROM: E I

- ✓ This command initializes the EEPROM. It first sets the default configuration values using parameters in the defines.h file, and then executes the calibration command (C M) to determine the min and max voltages. Once completed, it writes the resulting configuration to the EEPROM. Think of this as a factory reset.

CLI Miscellaneous Management Commands

There are a few other MISC commands that can be used. For example, troubleshooting.

I. Set Voltages: S V B xxxx or S V C xxxx

- ✓ This sets the PWM voltage at pin 3 (base/gate voltage) or pin 10 (collector/drain voltage). The voltage is given by xxxx and is in Millivolts (i.e., mV). If O (Oh and not zero) is added after the B or C then, the Arduino tries to set the output voltage delivered to the transistor under test based on the calibration performed. Note that this value could be off by up to 0.5 volts.
- ✓ **BE CAREFULL SETTING VOLTAGES MANUALLY** – you could destroy a resistor or the transistor under test if you exceed its maximum ratings.

II. Set Units: S U uu

- ✓ Set current units for graph for normal decimal (D), milliamps (M) or microamps (U). uu may be D, M or U. Default is Decimal
- ✓ E.g.,
  - ➢ S U D is used for normal decimal,
  - ➢ S U M is used for milli and

III. S U U is used for micro. Set PWM: P xx yyy

- ✓ This command manually sets the PWM output of pin 3 or pin 10 of the Arduino and is used for troubleshooting.
- ✓ xx is the pin number (3 or 10) and yyy is the PWM value (1-255). A higher PWM values will output a higher voltage. A value of 128 will give ½ of the maximum output voltage.

IV. Measure: M

- ✓ This reads and displays all the voltages and currents. It assumed you have manually set a voltage using the S P or S V commands.

V. Reset: R

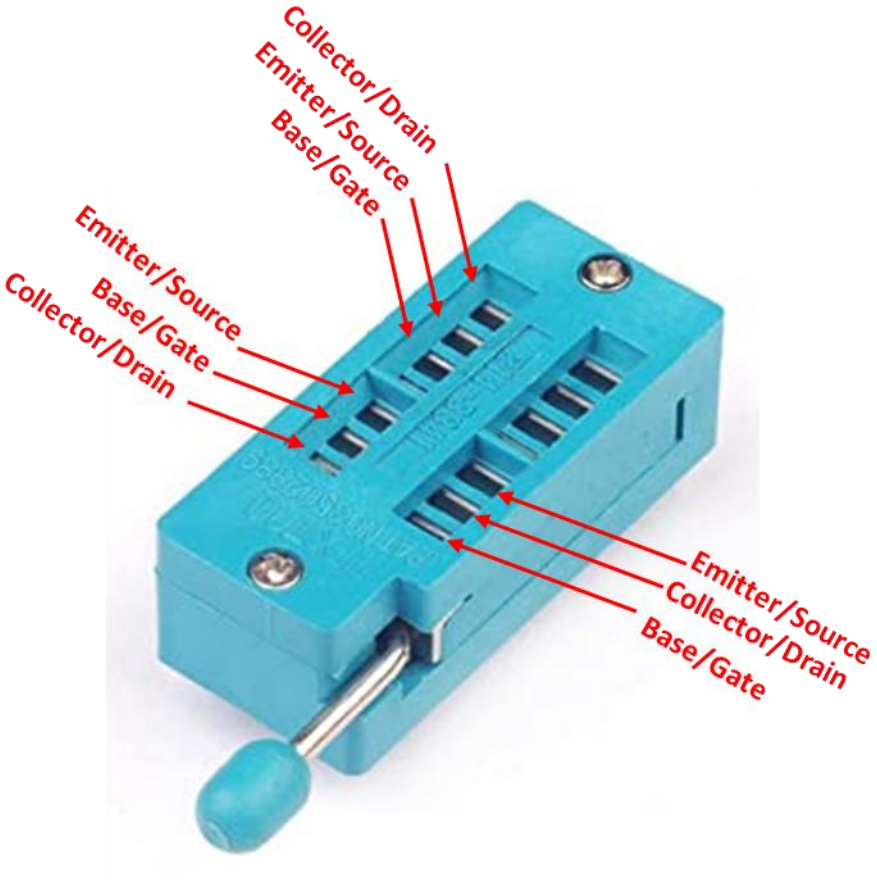
- ✓ This resets the Arduino. This is not the same as a power on. Power off/power on or pressing the Nano reset button does a “hard” reset. This command performs a “soft” reset.

VI. Set Defaults: D

- ✓ This command sets the default parameter to the running config. It does not change the EEPROM config.

Appendix

ZIF Socket Pinout



Parts List

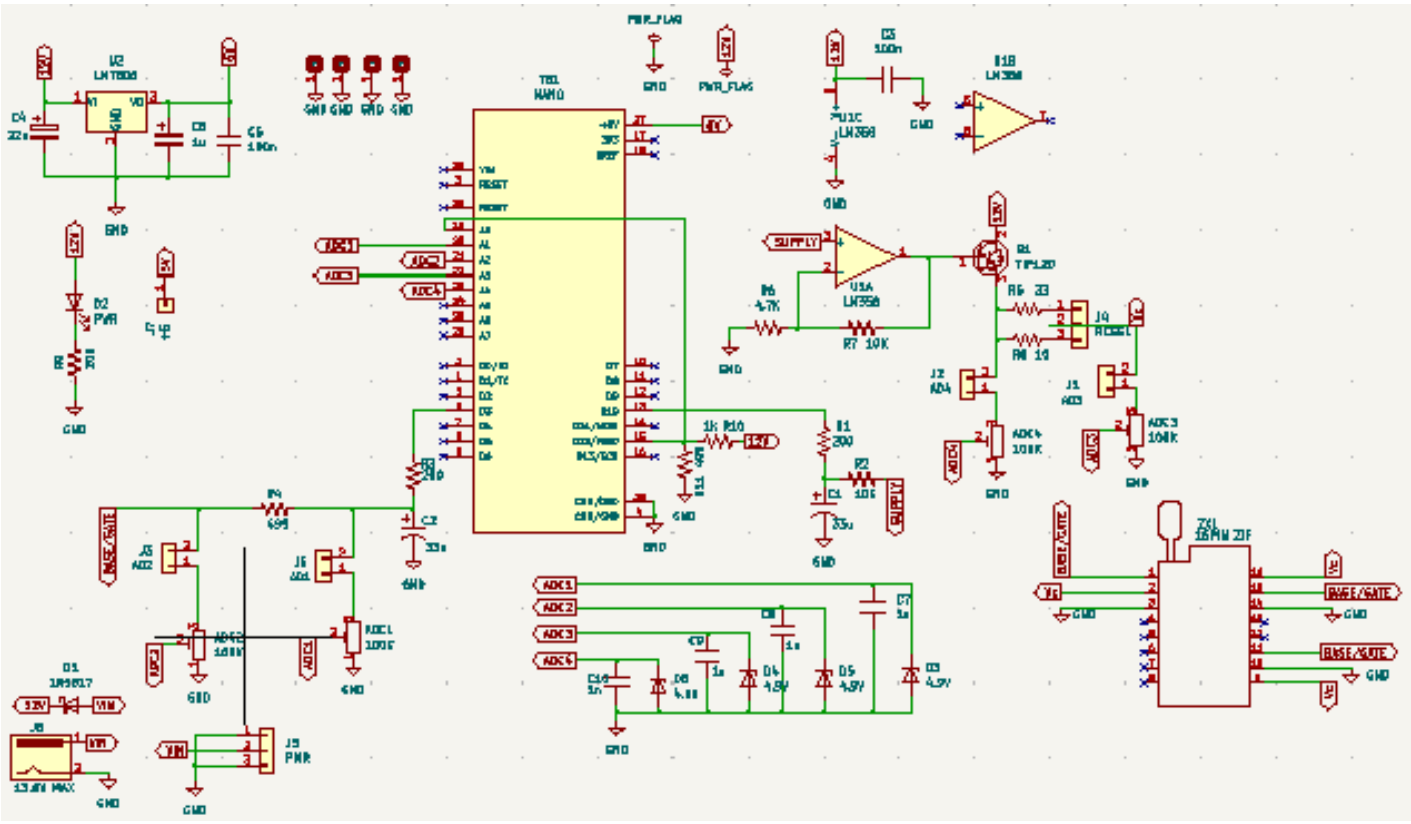
Ref	Qnty	Value	Cmp name	Notes
ADC1, ADC2, ADC3, ADC4,	4	100K	R_Potentiometer_Trim	Note that some early boards had 10K marked on silkscreen which is incorrect. Correct value is 100K
C1, C2,	2	33u	C_Polarized_US	
C3,	1	100n	C	
C4,	1	22u	C_Polarized	This is for voltage regulator. Anything around 10 or 20uF is adequate.
C4	1	1u	C_Polarized	This is for voltage regulator. Anything around 1uF is adequate.
C6,	1	100n	C	

C7, C8, C9, C10,	4	1n	C	Optional. Used to reduce noise on ADC line. If ADC values are noisy, then add capacitors around 1nf. Don't make it too large since it must charge up before ADC reads the value on the Pin
D1,	1	1N5817	1N5817	Use any Schottky Diode with low voltage drop and high current. Need about 1-2A
D2,	1	PWR	LED	Use any LED with 3mm Footprint with max current of 60mA
D3, D4, D5, D6,	4	4.9V	D_Zener	Optional. Can be omitted. Used to protect Arduino should >5V fed to ADC pins.
J1,	1	AD3	Conn_01x02	
J2,	1	AD4	Conn_01x02	
J3,	1	AD2	Conn_01x02	
J4,	1	RCSEL	Conn_01x03	
J5,	1	PWR	Conn_01x03	

J6,	1	AD1	Conn_01x02	
J7,	1	5	Conn_01x01	
J8,	1	13.8V MAX	Barrel_Jack	J5/PWR connector can be used instead. Middle pin is power.
Q1,	1	TIP120	TIP120	Can use any high current transistor. TIP122, 2SC5706 with same TO-220 footprint at TIP120
R1, R3,	2	200	R_US	
R10,	1	1K	R_US	Measure the actual value with VOM and enter it during calibration
R11,	1	499	R_US	Anything below 523R and around 500R. E.g. 475, 487,499, 511, 523. Measure the actual value with VOM and enter it during calibration
R2, R7,	2	10K	R_US	R2 can be anything around 10K. R7 MUST be 10K
R4,	1	499	R_US	Use resistor below 500R. I used 462. This limits how much current is delivered to base/gate. Measure the actual value with VOM and enter it during calibration
R5,	1	33	R_US	Must be 2 Watt resistor. Measure the actual value with VOM and enter it during calibration

R6,	1	4.7K	R_US	This resistor sets the gain of LM358. 4.7K seems to be ok. Might be a little too small (i.e., too much gain) but this value works fine. Calibration command measures the gain
R8,	1	10	R_US	Must be 2 Watt resistor. Measure the actual value with VOM and enter it during calibration
R9,	1	200	R_US	
TB1,	1	NANO	ARDUINO_NANO	
U1,	1	LM358	LM358	
U2,	1	LM7805	LM7805_TO220	
ZX1,	1	16PIN ZIF	16Pin ZIF	

Schematic Diagram



Printed Circuit Board Layout