Contents lists available at ScienceDirect

Electrochimica Acta

journal homepage: www.journals.elsevier.com/electrochimica-acta

NanoStat: An open source, fully wireless potentiostat

Shawn Chia-Hung Lee [a], Peter J. Burke [a,b,*]

[a] Department of Biomedical Engineering, University of California, Irvine, CA 92697, USA
[b] Department of Electrical and Engineering and Computer Science, University of California, Irvine, CA 92697, USA

ARTICLE INFO

Keywords:
Potentiostat
Microcontroller
Electrochemistry

ABSTRACT

We present an open source, fully wireless potentiostat (the "NanoStat") for applications in electrochemistry, sensing, biomedical diagnostics, and nanotechnology, based on only 2 integrated circuit chips: A digital microcontroller with integrated on board WiFi and file/web server hardware/software, and an analog front end. This versatile platform is fully capable of all modern electrochemistry assays, including cyclic voltammetry, square wave voltammetry, chronoamperometry, and normal pulse voltammetry. The user interface is a web browser connected over http. All the code (firmware, HTML5, JavaScript) is hosted by the NanoStat itself without the need for any additional software. The total size is 4 × 40 × 20 mm and battery operation for 6 h is demonstrated, possible to extend to weeks or months in sleep mode. We anticipate that the applications of this could be very broad, from biomedical sensing in the clinic, to remote monitoring of unattended "motes", to even possibly sensing aerial pathogens such as COVID in large public spaces without the need for anything other than a web browser for remote monitoring from anywhere in the world. Finally, we propose to use this software suite as a basis (kernel) of a fully open source, general purpose, web based electrochemistry software suite, abstracted from the hardware, which we call "OpenEChem".

1. Introduction

Electrochemcal techniques interface the world of electricity to the world of chemistry utilizing a mature, broad, and powerful suite of tools for quantitative analysis of composition of analytes for sensing and biomedical diagnostics [1]. These include DNA, RNA, proteins, aptamers, peptides lipids, metabolites, small molecules, trace metals, and even virions, such as the COVID19 virion. The simplest of the techniques require only resistance measurement whereas the most sophisticated techniques utilize signal processing and precise, coordinated timing of voltage and current waveforms (such as steps, pulses, etc. used in cyclic voltammetry, square wave voltammetry, chronoamperometry, and normal pulse voltammetry) with ultra low noise and ultra high gain analog circuitry to extract every possible piece of information about the analyte of interest.

While typical tools are benchtop standalone instruments costings thousands to tens of thousands of dollars, recent trends in the literature have used smaller and smaller custom boards with fewer and fewer components [2]. In the extreme, all signal processing and data storage in the cloud is handled off-board, with the potentiostat only doing the most low level work, the digitized signals immediately broadcast as unprocessed digitized voltages for could based processing on custom software on wired or wireless linked PCs or smartphones. This often is sold as "wearable" electronics, although the battery is still an issue, and an external processor with signal processing capability, memory, and storage capacity is required, typically referred to ambiguously as "the cloud". The control machine is either a smartphone of PC. It is advertised as "the cloud" because the PC /smartphone is connected directly to the internet, but the electrochemical sensor is NOT. While this technology definitely has its place and advantages, this is a bit misleading, because all smartphones and PCs are internet connected these days, so to call the electrochemical sensor connected to a PC or smartphone "cloud connected" is almost a tautology.

Furthermore, this approach requires custom, often proprietary software specific to the operating system of the off-chip architecture (e.g. Windows, Linux, Android, iOS, etc.). The trend has been to use the lowest possible architecture on the sensing chip such as 8 bit Atmega or at best 32 bit ARM (STM32) microcontrollers, with minimal on board intelligence, memory for at most a few traces, no internet connectivity (only Bluetooth, which is peer to peer and requires custom software) and little to no ability to serve files or graphical representation of data.

Many or most such solutions are proprietary, expensive, and closed

* Corresponding author at: Department of Electrical and Engineering and Computer Science, University of California, Irvine, CA 92697, USA.
E-mail address: pburke@uci.edu (P.J. Burke).

https://doi.org/10.1016/j.electacta.2022.140481
Received 11 October 2021; Received in revised form 14 April 2022; Accepted 16 April 2022
Available online 10 May 2022

source. The only open source solutions (reviewed in detail later in this paper) require the user to source the parts and separately solder them manually or place them manually and reflow solder them in their own labs. Both of these are laborious and require special skills. Manual pick and place is prone to error as the size of the components gets smaller, limiting the miniaturization potential. The alternative approach presented in this paper is to employ automated pick and place machines that, unlike human hands, can handle extremely small surface mount (SMD) parts as well as employ automated reflow soldering, which is more economical and can be ordered and manufactured from vendors at low cost without the need for any manual assembly at all. The boards come shipped to the user ready to operate "out of the box". Finally, the CAD files are based on expensive, commercial, proprietary software that needs to be installed on a desktop PC for design changes.

In this work (Fig. 1), we present a fully open source hardware and software design for a potentiostat (the "NanoStat" that 1) can serve all the results via a web page hosted on board the microcontroller itself 2) can be connected to from the internet from anywhere in the world via a web browser and 3) contains only 2 key integrated circuits (ICs) other than the voltage regulator, battery management, and USB ICs and a handful of discrete components: (a) a single microcontroller with on board WiFi, memory and signal processing with a complete file system and (b) a single chip analog front end. The entire system fits on a 20 × 40 mm board, with battery operation can run 6 h in active mode and even longer in sleep mode. As such, this is the world's smallest webserver (WiFi) enabled potentiostat. The cost is ultra low, under 25 dollars in small quantities fully assembled with no soldering needed. The CAD designs are also provided on a free, web based CAD program accessible from anywhere in the world with only a web browser.

We anticipate that the application of this could be very large, from biomedical sensing in the clinic, to remote monitoring of unattended "motes", to even possibly sensing aerial pathogens such as COVID in large public spaces without the need for anything other than a web browser for remote monitoring from anywhere in the world. Furthermore, any lab with a modest budget, including garage DIY, K-12 education, and modern university labs that need sophisticated electrochemical analysis with minimal investment of time and money, can benefit from this work. Finally, we propose to use this software suite as the kernel of a future open source, general purpose electrochemistry suite. Similar to how Linux is an open source operating system for modern PCs, abstracted from the hardware, we propose an open source project for electrochemistry, also abstracted from the hardware, and based on our initial foundational web-based work presented in this paper, which we call "OpenEChem".

2. System design

In this section we describe the overall design. Both the hardware and software source code are available on as S.I. and on a github repository as https://github.com/PeterJBurke/Nanostat.

2.1. Hardware design

2.1.1. Hardware architecture

The hardware architecture (Fig. 2) is based on two core integrated circuits (ICs): The ESP32 (Espressif Systems) IC has integrated WiFi, 520 KiB SRAM, 8 bit DAC, 12 bit ADC, and dual core 32 bit processors operating at 240 MHz. We use the SOC version (ESP32-PICO-D4) which has integrated crystal oscillator and 4 MByte of integrated SPI flash memory. Note that this is the key enabling technology of this work, as it contains processing power and connectivity orders of magnitude improved over prior microcontroller based potentiostats. (See "Prior Art" discussion later in this paper).

The analog front end (AFE) is an Analog Devices LMP91000, which has an integrated analog potentiostat to enable bipolar (both sign) cell operation from a single sign power supply. A digital (I2C) bus between the LMP91000 and the ESP32 controls the internal gain settings and other configurations of the potentiostat. Additional on board circuitry includes a chip for USB interface (which is not required if fully wireless operation is used), and a chip for battery power management to charge the battery from a USB connector, similar to a cell phone charger. An on board LED provides power status and a separate LED blinks every time a data point is taken. Fig. 7 shows the circuit board rendered. The number of distinct components is 29; the total number of components is 42.

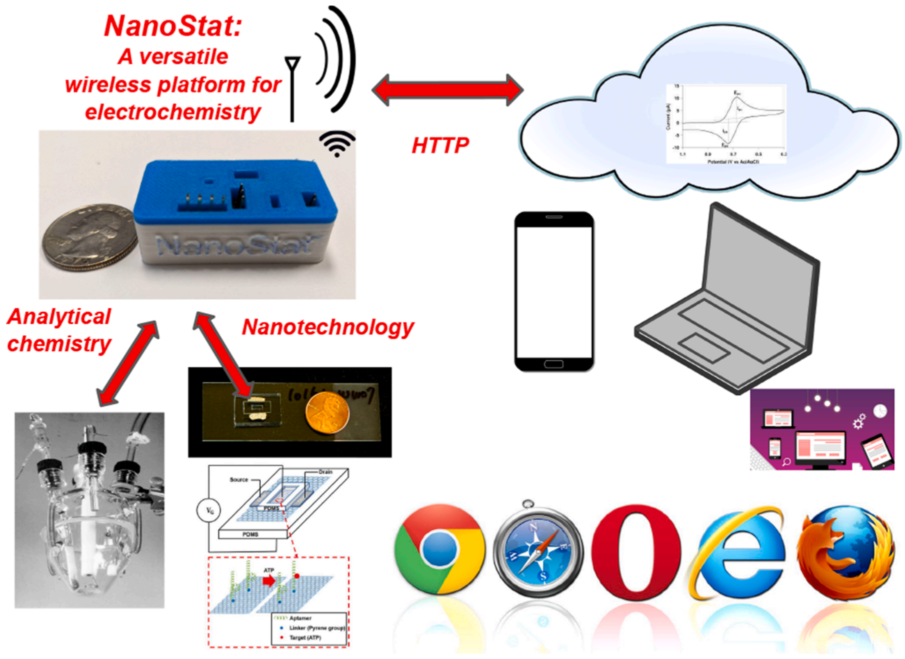


Fig. 1. NanoStat: A versatile wireless platform for electrochemistry, from traditional analytical chemistry to advanced nanotechnology based sensors (e.g. Datta et al. [3]), using only a web browser for access to a broad suite of software defined analytical tools.

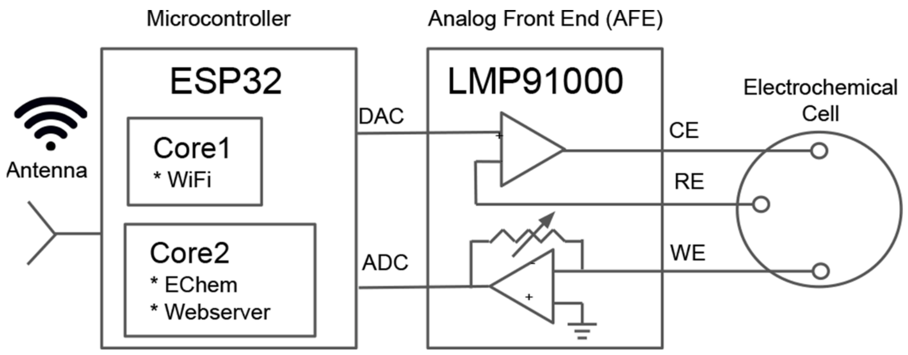


Fig. 2. 2 chip architecture schematic. One of the chips provides memory, firmware, WiFi connectivity, and hosts the website and file system for the user interface and analysis. The other chip provides the analog front end (AFE).

2.1.2. *Hardware CAD*

The CAD file is designed using a web based CAD program called EasyEDA (https://easyeda.com/editor). In addition to providing web functionality usable from anywhere in the world, it has an integrated library of parts that can be assembled by the manufacturer, with real time information about stock status, something that is increasingly important in the modern chip shortage era. The CAD files, Gerber files, bill of materials, and pick and place instructions are provided in the online repository (https://github.com/PeterJBurke/Nanostat). These are sufficient for any user to have the board manufactured from their preferred vendor. In this project, the vendor JLCPCB (https://jlcpcb.com) was chosen due to their fast turnaround time (under 2 weeks from order to delivery), integrated supply chain with real time stock information for part availability, and low cost (under 25 dollars for a fully assembled board with all parts already soldered in place, including express international shipping). However, this project is suitable for any vendor, i.e. is vendor agnostic.

PC board The PC board is FR4 with 0.5 oz. Cu traces, and is 4 layers (with ground plane and power plane the middle layers) to improve grounding, reduce V_{cc} bounce and improve shielding.

Case Although not necessary for operation, the design for a 3d printed case is provided in the supplementary information files, both for with and without battery.

Battery A lithium polymer (LiPo) battery of arbitrary size (in *x y* and *z* dimensions) can be purchased for any application. For this system we recommend matching the *x*-*y* dimensions to the board size, and *z* can be 1 to several mm. For example, model 402,040 is 4 × 40 × 20 mm, costs only a few dollars and has 280 mAh of capacity, enough for over 6 h (Fig. 3). Power through the USB is also possible using a larger battery (e.g. Voltaic Systems V25 6400 mAh USB Always On Battery), available for tens of dollars, with 6400 mAh capacity i.e. >50 h of active running (Fig. 8).

2.2. *Software design*

The software consists of the C++ code for running the microprocessor and the HTML5/JavaScript code for the web server/user interface (UI). Microsoft Visual Studio Code is used as the integrated development environment (IDE), with the PlatformIO add on, which is synced to the github repository (https://github.com/PeterJBurke/Nanostat) for easy development and web based deployment. The C++ code is about 4000 lines of code. The HTML5/JavaScript is about 1000 lines of code.

2.2.1. *Firmware design*

As usual in software development, we rely heavily on existing libraries integrated into this project. The following libraries were used:

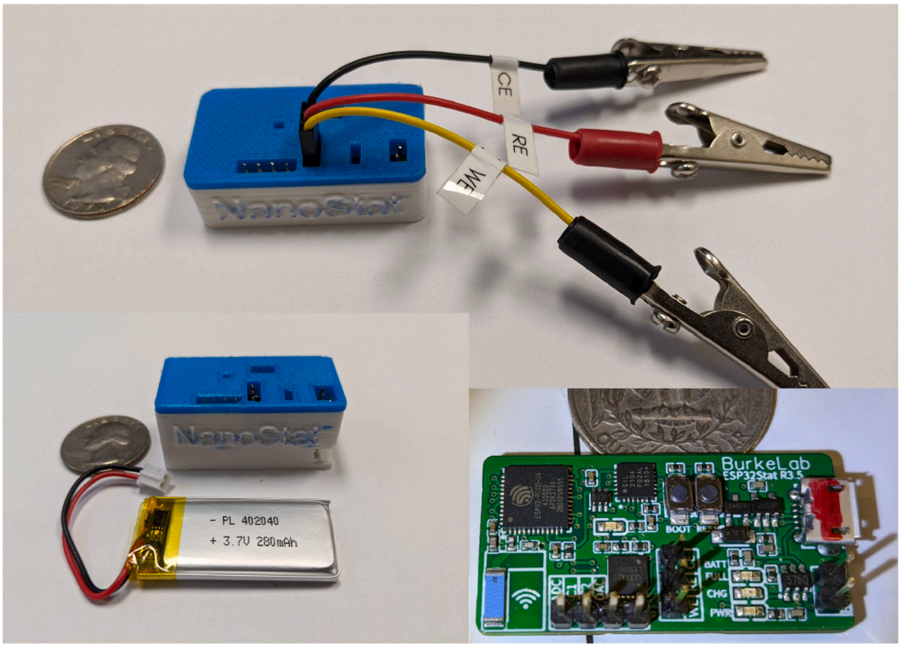


Fig. 3. Photographs of the NanoStat. Alligator clips are shown but in principle any electrode connection technique can be used. (The circuit board connector is a standard 0.1” DuPont connector.) The bottom left inset shows the 3d printed case with the optional LiPo battery taken out of the case (4 × 40 × 20 mm, 280 mAh).

linneslab/LMP91000 version 1.0.0; me-no-dev/AsyncTCP version 1.1.1; ottowinter/ESPAsyncWebServer-esphome version 1.2.7; bblanchon/ArduinoJson version 6.17.3; links2004/WebSockets version 2.3.6; bbx10/DNSServer version 1.1.0. There is a well developed software library for the LMP91000, developed initially by github users icatcu (2014) and jorgenro1 (2015), and most recently in Hoilett et al. [4]. In addition, there are several libraries for web hosting and file transfer. Because the ESP32 runs FreeRTOS operating system under the hood, and because of its dual core CPU architecture, it is possible to have simultaneous WiFi connectivity and control of the potentiostat and electrochemistry. This has not previously been the case with any other wireless connected electrochemical system (see “Prior Art” section below).

The main issue with this architecture is the limited number of bits for the resolution, and the non-linearity of the ADC and DACs. For the limited number of bits, the AFE takes the analog “drive” signal and scales it by from 2 to 24 percent, so the effective bit resolution is much smaller at the cell. For 3.3 V power supply and 8 bits the resolution is 12 mV. But when divided down to 2 percent it is effectively 0.1 mV at the cell.

The non-linearity and offset of the ADCs is far from ideal (see SI). For this purpose, a calibration routine was developed. The user connects the RE/CE, floats the WE, and runs the calibration routine once. The firmware calculates the effect slope and offset, stores these in the on board flash memory, and calls on these in each subsequent run. (See SI for description of calibration technique).

2.2.2. *Web design*

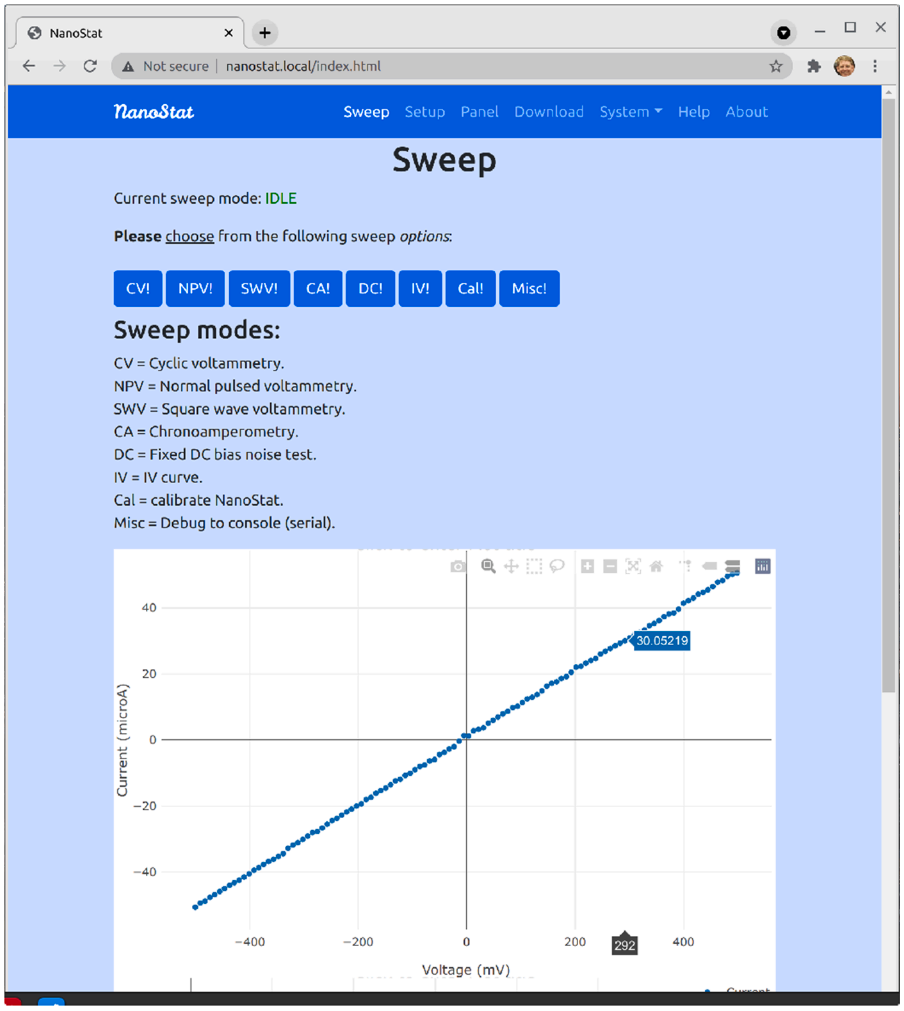
HTML5, JavaScript, and CSS (Cascading Style Sheet) are used together with the Bootstrap framework and the Plottly graphing package for an intuitive, responsive, and mobile friendly user interface on both desktop and mobile browsers. The data is transferred over a WebSocket opened between the client and the server in binary form to the browser for display.

2.2.3. *End user perspective for electrochemistry*

The following sweep types are supported in the present version: cyclic voltammetry, square wave voltammetry, chronoamperometry, and normal pulse voltammetry, current-voltage curves, “oscilloscope” mode with quantitative noise analysis (RMS, standard deviation), and a calibration mode. The user can enter various parameters for different types of sweeps, as well as simple IV curves, and noise tests (“oscilloscope mode”). The software displays “sweeping” on the browser and once the sweep is completed presents the data in a graphical format (Fig. 4), which the user can change the scale on, zoom, etc. An ASCII text data file is auto downloaded to the web browser’s disk at the end of each sweep.

2.2.4. *System administration*

If there is no WiFi connection, the device creates its own WiFi access point called “NanoStatAP”. The user can connect to that access point and enter in the credentials of up to 3 WiFi SSIDs. The user can also perform over the air “OTA” firmware upgrades by uploading the binary



Sweep

Current sweep mode: IDLE

Please <u>choose</u> from the following sweep *options*:

Sweep modes:

CV = Cyclic voltammetry.
NPV = Normal pulsed voltammetry.
SWV = Square wave voltammetry.
CA = Chronoamperometry.
DC = Fixed DC bias noise test.
IV = IV curve.
Cal = calibrate NanoStat.
Misc = Debug to console (serial).

Fig. 4. Graphical user interface (GUI) of one of the website pages. A test IV curve of a 10 *k*Ω resistor is shown immediately after measurement. The user can zoom, autoscale, read off each point, export the graph, and many other graphical options, as shown in the icons on top of the graph region.

firmware, as well as list and delete files in the on board directory structure. Fig. 10 shows the entire UI and menu tree/structure.

3. Performance

3.1. Test chemicals

In order to compare the performance of the NanoStat to industry standard, we performed electrochemistry experiments using a Ag/AgCl (3M KCl) Reference Electrode (BASi MF-2056), a platinum wire auxiliary electrode (BASi MW-1033), and a 3.0 mm diameter gold working electrode (BASi MF-2114) in a standard 100 mL Standard glass electrochemical cell (BASi MF-1051). The redox active species used was 5 mM potassium ferricyanide in 100 mM KCl at room temperature. The same cell was measured in the same session with a high end Gamry potentiostat (Reference 600) for comparison.

3.2. Results: analytical electrochemistry

Fig. 5 shows the results of the Gamry and NanoStat to measure cyclic voltammetry, square wave voltammetry, chronoamperometry, and normal pulse voltammetry. The parameters are the default parameters shown in the “Setup” page of Fig. 10. The results agree extremely well, demonstrating this work is consistent with the most high end analytical chemistry instrumentation available on the market today.

3.3. Results: noise

Given the simplicity of the system, the close proximity of digital, RF, and analog circuits on the board (and on the IC itself), and the lack of any on board analog or digital filtering, the noise performance is astounding. When operated with a battery, *the system noise is determined by the bit resolution of the ADC*. We explain.

For the trial data in Fig. 5, the Gamry is using extensive on board analog and digital filtering, whereas the simple NanoStat measures only one ADC reading per data point with no filtering. Each data measurement takes 160 μs. In the firmware, it is possible to average more readings if the rate is sufficiently slow.

In order to quantitatively test the noise, we use the “dc bias” sweep mode. In the “dc bias” sweep mode, the firmware sets the dc bias voltage to a user selectable value, then reads ADC with various number of readings per point 1, 5, 10, 50, 100, 500, 1000, at a user specified delay between points. In Fig. 6, we plot the data points with and without the battery for a 100 *k*Ω test resistor biased at 100 mV. One data point is taken every 10 ms. With the PC USB power, there is clearly 60 Hz noise (red trace). With the battery power, this 60 Hz noise is gone. For no averaging (1 ADC reading per point, beginning), the error is about 1 bit in the current. This corresponds to 2 nA, since the ADC is 12 bits (full scale voltage = 3.3 V), and the TIA resistor is set to its most sensitive value the LMP91000 offers: 350 *k*Ω. Occasional spikes in the ADC reading are seen of a few bits, a known flaw of the ESP32 with unclear origin. The later data (with 1000 ADC readings per point, corresponding

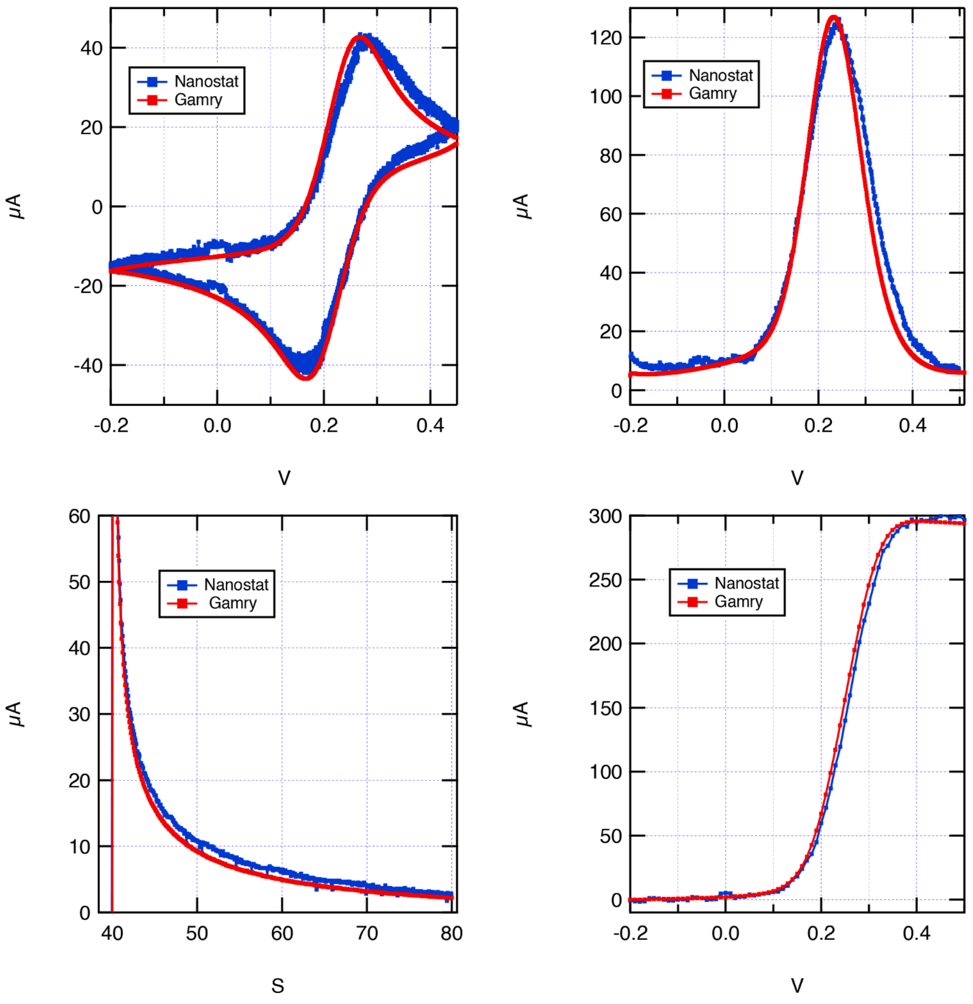


Fig. 5. Cyclic voltammetry, square wave voltammetry, chronoamperometry, and normal pulse voltammetry measured with the NanoStat (blue) and a high end Gamry commercial potentiostat (red).

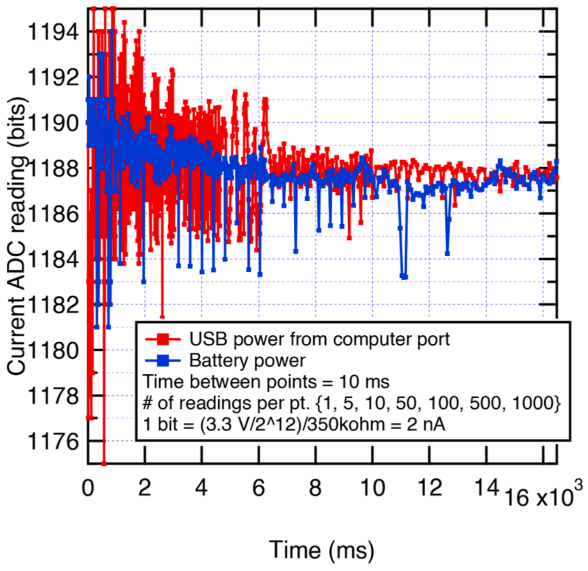


Fig. 6. Current vs. time ADC reading (output of LMP91000 TIA) with 100 $k\Omega$ test resistor biased at 100 mV. One data point is taken every 10 ms. The data point is the ADC reading with various number of readings per point 1, 5, 10, 50, 100, 500, 1000. The 60 Hz noise is clearly visible in the red trace. With battery power only, the noise is limited by the bit resolution of the ADCs.

to 160 ms per reading) shows noise much lower then the single bit level. Thus, with no signal averaging, the noise is limited by the bit resolution of the ADC, under battery use, with no 60 Hz noise. With USB power from a PC, the 60 Hz noise is much was much worse and needed to be filtered out digitally to get to the bit resolution of the ADC. This shows that a battery powered, wireless potentiostat (without any USB connections to a PC) is a wise choice for low noise operation.

Due to limits on the ADC of the ESP32, one needs to ensure $-$ 0.75 V $<$ VTIA $<$ 1 V, where VTIA $=$ RTIA * WE current. For WE currents of 10 s of microamps, we typically therefore use 35 kohm as the TIA resistance, which gives a noise of about 50 nA, as can be seen in Fig. 5. The “oscilloscope” panel allows the user to plot the readings in real time on the web browswer for a given bias voltage and number of points to read. We confirmed that 50 nA is the RMS current noise with no averaging using a 35 kohm TIA amplifier and a 10 kohm test resistor.

For applications that need more than 12 bits of resolution on the WE current, other approaches with a separate ADC IC would be needed (see below, “Prior Art” section). For applications that need better than 2 nA of current sensitivity, it is possible with the NanoStat for a user to select an external TIA resistor and connect it to the PC board labeled C1/C2, for additional current sensitivity. This option is supported in the firmware; the user just needs to select “external resistor” in the dropdown menu of TIA gain on the “setup” page (Fig. 10).

4. Discussion

4.1. Prior art

Whitesides has laid out the criteria for an analytical electrochemical system to be considered open source [5]. His criteria for open source are clear and reasonable: “Description of a truly open-source potentiostat should include all details required to replicate the system....” including circuit design, component list, board layout file, and software (microcontroller firmware and PC/tablet/smartphone control software). Many academic papers only partially meet this, meaning a lab new to the field or even an experienced lab cannot reproduce prior academic work due to lack of published details. In the old days of chemistry, if a reviewer was to accept a paper, it was required by the editor that the reviewer reproduce the results in his own lab. While this may no longer be economical, certainly publishing the details of the experimental process should be [6].

As far as we can tell, the concept of a DIY, general purpose, open source potentiostat was pioneered by Kevin Plaxco’s group at UCSB about 10 years ago [7] (the “CheapStat”). There, an open source (hardware and software) handheld potentiostat was described. The user is provided gerber files for a PC board to be manufactured, a list of components, and the user would solder the components (cost ~$80 in low volume) by hand to the PCB. The firmware and PC software for the USB control was provided. The AFE was a custom op amp and the microcontroller an Atmel XMEGA, an 8 bit microcontroller with 12 bit ADCs/DACs, 8 KB RAM and up to 128 kB flash. Voltage resolution was sub mV and current resolution sub nA.

Whitesides’ group at Harvard followed on with a cell phone connected (via audio cable) potentiostat in 2014 [5] of cost (parts only) ~$25 in large volume (>1000 pieces). There, they used an Atmega328 (Atmel) 8 bit microcontroller (popularized as an Arduino) with a wired interface to a cell phone through the audio jack, an external 16 bit ADCs/DACs, and an analog 4 pole low pass filter (LPF). Current resolution was 1 nA. The parts list was provided but the software and PC board Gerber files/CAD files were not provided, so it was not open source.

A high performance open source project with USB connection to a PC was published as “DStat” [8] in 2017. There, an ATXMEGA256A3U was used (an 8 bit microcontroller with 8 KB RAM and up to 128 kB flash), with custom AFE and 16 bit DAC IC and a 24 bit ADC IC. The AFE design was excellent: pA level currents could be measured. The user is provided the firmware, PC software, Gerber files for PCB manufacture, and a list of components to order. The user solders some components by hand and others with a flow solder technique after ordering the parts.

Dobbelaere [9] created a larger voltage version (up to 8 V) with PIC16F1459 (an 8 bit microcontroller), 22 bit IC for ADC/DAC, custom AFE, intended for battery research. All the details of the project (including the schematic, PCB design, last of components, microcontroller firmware, and host computer software) are made available to the user.

Whitesides’ group in 2018 [10] published the first wireless (Bluetooth) open source potentiostat, which simply replaced the Atmega328 with the “RFDuino”, and still used a custom AFE with separate 16 bit ADC and DAC ICs. RFDuino had an integrated BLE and Core0 Arm 32 bit processor. RFDuino, first marketed in 2013, is obsolete and almost impossible to source, and the company was acquired in 2016. A list of components to purchase as well as the PCB manufacturing files is provided (in addition to the firmware software and smartphone software), so the user can order the boards, parts, and manually solder the final system.

In 2018 Lopin [11] made a potentiostat with a SOC. It was not wireless. It used internal integrated op amps with higher noise and offset than discreet, but was still fully functional at the microamp level. It used on chip 12 bit ADCs/DACs.

In 2019, Jenkins et al developed the open source project “ABE-Stat” [12]. This project provided both WiFi and Bluetooth wireless connectivity, as well as extended the analytical chemistry sweep methods to include electrochemical impedance spectroscopy (EIS), a valuable tool for capacitive rather than Faradaic current measurments [13]. A custom AFE was designed, and 24 bit ADC and 16 bit DAC ICs were used, as well as an IC for network analysis for EIS. Although a relatively powerful WiFi enabled microcontroller was part of the system (ESP8266), and external Bluetooth IC was used for communications with a custom Android app. As prior open source projects, a list of components to purchase as well as the PCB manufacturing files is provided (in addition to the firmware software and smartphone software), so the user can order the boards, parts, and manually solder the final system.

In 2020 Glasscott et al published the SweepStat [14]. The system used an Arduino Teensy with 16 bit integrated ADC and 12 bit integrated DAC, and a custom AFE. The interface to the PC is with USB and

Labview software environment is used for the control. As prior open source projects, a list of components to purchase as well as the PCB manufacturing files is provided (in addition to the firmware software and smartphone software), so the user can order the boards, parts, and manually solder the final system.

An AFE LMP91000 chip on an evaluation board (LMP91000EVM) and a BeagleBone microcontroller on a separate board were used to demonstrate a low-cost miniaturized potentiostat in 2014 by Bhansali [15]. Later, Hoilett [4] created a potentiostat 22 × 20 mm with an integrated AFE (LMP91000) run by an ARM based microcontroller (SAMD21, 256KB flash memory, 10 bit DAC, 12 bit ADC). It is open source, although not wireless. Their software and hardware are hosted on github and detailed assembly and operation instructions are provided. The parts and Gerber files are provided. Their recommendation is to have PCBWay manufacture and solder the components, which the user buys from DigiKey and ships to PCBWay. PCBWay allows Digikey sourced parts, they will source them for you and solder them in place. They also offer 2 side SMT, and connectors.

Mercer [16] developed a potentiostat with an ESP32 development board, used external 16 bit ADCs, and a custom op-amp AFE design. The entire system was demonstrated in a breadboard setting, not integrated onto a single PCB, and demonstrated the concept to use a microcontroller with integrated WiFi for potentiostat work. An electronically controlled (with servo) pump was also used.

An Ardruino board was used to control a custom home made AFE daughter board using discrete components and op-amps in ref. [17].

An open-source multi-channel potentiostat based on a Raspberry Pi with additional boards for each channel was demonstrated in ref. [18].

Additional related work that is not open source is given in refs. [19–25].

In addition to requiring high value skills (soldering tiny components) and being extremely inefficient, even for small numbers of boards, hand soldering has a significant limitation in the size of the board that can be made. Pick and place reflow soldering machines enable sub-mm accuracy in placement of parts whose size that far exceeds the ability of even the most trained human hand. Even compared to a technician manually soldering parts at minimum wage they also are more economical these days. And of course the reliability is much higher.

Many of the above projects require manual soldering, a custom AFE design with multiple AFE op amp chips and resistors, separate ADC and DAC chips, and typically separate wireless chips, if at all. In addition the memory and processing ability of almost all the microcontrollers used so far is not able to provide on board analysis, nor on board file systems or website hosting capabilities. Finally they are mostly manually soldered and require at least three institutions to make : 1) the parts source 2) The PCB manufacturer 3) the soldering (also many times manual, a severe disadvantage described above). We use an integrated provider that does all of 1–3 in house, as well as provides the (free) web based CAD with integrated supply chain real time stock information, with low cost and under 2 week turnaround time. Finally our work has a battery power and USB power to charge the battery, which many of the above open source projects do not have. Finally, none of the above host a website on the potentiostat system and require the user to install custom software on a pc or smartphone.

4.2. *Scalability and future directions*

4.2.1. *Security*

This version of the software uses HTTP, not HTTPs. For local area networks behind a firewall this is sufficient. However, for traffic over the public internet, it would be important to switch to enctryped traffic with the HTTPs protocol, and to enable a secure user login system.

4.2.2. *Further miniaturization*

We estimate the size of the circuit board and the number of components can be reduced by roughly a factor of two. First of all, the USB to UART interface was useful for developing the code (firmware, HTML5/JavaScript) for this project. However, now that we have demonstrated over the air (OTA) WiFi firmware update capability, the USB PC interface is superfluous and a simple UART interface (with off-chip FTDI to USB conversion) would save board space and components. Furthermore, the boot switches to put the microcontroller into firmware flash mode were found to be unnecessary with the IDE used in this work. These two changes could be easily implemented in future revisions of the board without any change in the software or functionality.

4.2.3. *Power management*

The ESP32 supports a sleep mode, and wake on LAN mode. These could be implemented to preserve power when the NanoStat was not actively taking measurements.

4.2.4. *On board analysis*

Due to the dual core 32 bit processors and generous onboard RAM, it is possible to incorporate just about any kind of signal processing and analysis a future user could design onto the microcontroller itself. In contrast to prior open source potentiostats (see below “Prior Art” section), therefore, this project is therefore very scalable to advanced signal processing and on board analysis.

4.2.5. *Device performance (resolution, shielding)*

Here we touch on how device performance (resolution, shielding) can be improved for future design. The ADC and DAC bit resolution are set by the microcontroller, so improvement here could use a separate ADC with more bits. The shielding could be improved by providing replacing the plastic 3d printed case with a metal case (Faraday cage), however the case design would need to allow WiFi antenna access to fields outside the cage.

4.3. *Towards an open source suite of electrochemistry software*

Each of the above projects listed in prior art has custom software, but it all basically performs the same core function for control/acquisition (run a sweep of voltage vs. time, measure current vs time). The UI is also similar. Therefore, many these papers are “reinventing the wheel” to a large extent. For this reason, we propose a hardware abstracted, unifying suite of software we call “OpenEChem”. With a common, unifying set of software, developers of advanced electrochemistry and sensing hardware can focus on the more challenging aspects of the field, such as (for example) electrode functionalization, miniaturization, and application specific demonstrations.

Interestingly, the use of open source is also growing in the drone community. For example, we pioneered [26] and open source web based 4G connected drone technology using a cloud server to handle routing and encryption, and that is now adopted by the French drone company Anafi in the first 4G connected commercial drone in July, 2021. The drone community has also recently started adopting the ESP32 microcontroller and the advantages of the above manufacturing capability to have an open source, long range, cheap lightweight radio control link for drones (https://github.com/openLRSng) with hundreds of miles of range for receivers costing under ten dollars and weighing less than 1 g with on board antenna.

5. Conclusion

We presented an open source, fully wireless potentiostat (the “NanoStat”) for applications in electrochemistry, sensing, biomedical diagnostics, and nanotechnology, based on only 2 integrated circuit chips. This was demonstrated to have the same functionality as modern desktop based potentiostats. While this particular project can be used by any lab or user on a very small budget, enabling numerous applications and use cases, it lays the groundwork for a more ambitious, web based, open source software suite as the kernel of a general purpose, hardware abstracted electrochemistry software suite, which we propose as “OpenEChem”.

CRediT authorship contribution statement

Shawn Chia-Hung Lee: Data curation, Formal analysis, Investigation, Methodology, Validation, Visualization, Writing – review & editing. **Peter J. Burke:** Conceptualization, Data curation, Formal analysis, Funding acquisition, Investigation, Methodology, Project administration, Resources, Software, Supervision, Validation, Visualization, Writing – original draft, Writing – review & editing.

Declaration of Competing Interest

Authors declare that they have no conflict of interest.

Acknowledgments

We acknowledge support from the Army Research Office (W911NF-18-1-0076).

Supplementary material

Supplementary material associated with this article can be found, in the online version, at doi:10.1016/j.electacta.2022.140481.

References

[1] A.J. Bard, L.R. Faulkner, Electrochemical Methods. Fundamental and Applications, second ed., John Wiley and Sons, Inc., New York, 2001 https://doi.org/10.1039/C4LC00091A. B978-0-08-098353-0.00002-6.

[2] Y. Yang, W. Gao, Wearable and flexible electronics for continuous molecular monitoring, Chem. Soc. Rev. 48 (6) (2019) 1465–1491, https://doi.org/10.1039/c7cs00730b.

[3] D. Datta, X. Meshik, S. Mukherjee, K. Sarkar, M.S. Choi, M. Mazouchi, S. Farid, Y. Y. Wang, P.J. Burke, M. Dutta, M.A. Stroscio, Submillimolar detection of adenosine monophosphate using graphene-based electrochemical aptasensor, IEEE Trans. Nanotechnol. (2017), https://doi.org/10.1109/TNANO.2016.2647715.

[4] O.S. Hoilett, J.F. Walker, B.M. Balash, N.J. Jaras, S. Boppana, J.C. Linnes, et al., Kickstat: a coin-sized potentiostat for high-resolution electrochemical analysis, Sensors 20 (8) (2020) 1–12, https://doi.org/10.3390/s20082407.

[5] A. Nemiroski, D.C. Christodouleas, J.W. Hennek, A.A. Kumar, E.J. Maxwell, M. T. Fernández-Abedul, G.M. Whitesides, et al., Universal mobile electrochemical detector designed for use in resource-limited applications, Proc. Natl. Acad. Sci. USA 111 (33) (2014) 11984–11989, https://doi.org/10.1073/pnas.1405679111.

[6] M.D.M. Dryden, R. Fobel, C. Fobel, A.R. Wheeler, et al., Upon the shoulders of giants: open-source hardware and software in analytical chemistry, Anal. Chem. 89 (8) (2017) 4330–4338, https://doi.org/10.1021/acs.analchem.7b00485.

[7] A.A. Rowe, A.J. Bonham, R.J. White, M.P. Zimmer, R.J. Yadgar, T.M. Hobza, J. W. Honea, I. Ben-Yaacov, K.W. Plaxco, et al., Cheapstat: an open-source, “do-it-yourself” potentiostat for analytical and educational applications, PLoS One 6 (9) (2011), https://doi.org/10.1371/journal.pone.0023783.

[8] M.D.M. Dryden, A.R. Wheeler, DStat: a versatile, open-source potentiostat for electroanalysis and integration, PLoS One 10 (10) (2015), https://doi.org/10.1371/journal.pone.0140349.

[9] T. Dobbelaere, P.M. Vereecken, C. Detavernier, A USB-controlled potentiostat/galvanostat for thin-film battery characterization, HardwareX 2 (2017) 34–49, https://doi.org/10.1016/j.ohx.2017.08.001.

[10] A. Ainla, M.P.S. Mousavi, M.N. Tsaloglou, J. Redston, J.G. Bell, M.T. Fernández-Abedul, G.M. Whitesides, et al., Open-source potentiostat for wireless electrochemical detection with smartphones, Anal. Chem. 90 (10) (2018) 6240–6246, https://doi.org/10.1021/acs.analchem.8b00850.

[11] P. Lopin, K.V. Lopin, PSoC-stat: a single chip open source potentiostat based on a programmable system on a chip, PLoS One 13 (7) (2018) 1–21, https://doi.org/10.1371/journal.pone.0201353.

[12] D.M. Jenkins, B.E. Lee, S. Jun, J. Reyes-De-Corcuera, E.S. McLamore, et al., ABE-Stat, a fully open-source and versatile wireless potentiostat project including electrochemical impedance spectroscopy, J. Electrochem. Soc. 166 (9) (2019) B3056–B3065, https://doi.org/10.1149/2.0061909jes.

[13] J. Li, P.J. Burke, Measurement of the combined quantum and electrochemical capacitance of a carbon nanotube, Nat. Commun. 10 (1) (2019) 3598, https://doi.org/10.1038/s41467-019-11589-9.

[14] M.W. Glasscott, M.D. Verber, J.R. Hall, A.D. Pendergast, C.J. McKinney, J.E. Dick, et al., SweepStat: a build-it-yourself, two-electrode potentiostat for macroelectrode and ultramicroelectrode studies, J. Chem. Educ. 97 (1) (2020) 265–270, https://doi.org/10.1021/acs.jchemed.9b00893.

[15] A.F.D. Cruz, N. Norena, A. Kaushik, S. Bhansali, A low-cost miniaturized potentiostat for point-of-care diagnosis, Biosens. Bioelectron. 62 (V) (2014) 249–254, https://doi.org/10.1016/j.bios.2014.06.053.

[16] C. Mercer, R. Bennett, P. Conghaile, J.F. Rusling, D. Leech, Glucose biosensor based on open-source wireless microfluidic potentiostat, Sens. Actuators, B 290 (2019) 616–624, https://doi.org/10.1016/j.snb.2019.02.031.

[17] Y.C. Li, E.L. Melenbrink, G.J. Cordonier, C. Boggs, A. Khan, M.K. Isaac, L. K. Nkhonjera, D. Bahati, S.J. Billinge, S.M. Haile, R.A. Kreuter, R.M. Crable, T. E. Mallouk, An easily fabricated low-cost potentiostat coupled with user-friendly software for introducing students to electrochemical reactions and electroanalytical techniques, J. Chem. Educ. 95 (9) (2018) 1658–1661, https://doi.org/10.1021/acs.jchemed.8b00340.

[18] P. Pansodtee, J. Selberg, M. Jia, M. Jafari, H. Dechiraju, T. Thomsen, M. Gomez, M. Rolandi, M. Teodorescu, The multi-channel potentiostat: development and evaluation of a scalable mini-potentiostat array for investigating electrochemical reaction mechanisms, PLoS One 16 (2021) 1–14, https://doi.org/10.1371/journal.pone.0257167.

[19] D.P. Rose, M.E. Ratterman, D.K. Griffin, L. Hou, N. Kelley-Loughnane, R.R. Naik, J. A. Hagen, I. Papautsky, J.C. Heikenfeld, et al., Adhesive RFID sensor patch for monitoring of sweat electrolytes, IEEE Trans. Biomed. Eng. 62 (6) (2015) 1457–1465, https://doi.org/10.1109/TBME.2014.2369991.

[20] J. Kim, S. Imani, W.R. de Araujo, J. Warchall, G. Valdés-Ramírez, T.R.L.C. Paixão, P.P. Mercier, J. Wang, Wearable salivary uric acid mouthguard biosensor with integrated wireless electronics, Biosens. Bioelectron. 74 (2015) 1061–1068, https://doi.org/10.1016/j.bios.2015.07.039.

[21] G.F. Giordano, M.B.R. Vicentini, R.C. Murer, F. Augusto, M.F. Ferrão, G.A. Helfer, A.B. da Costa, A.L. Gobbi, L.W. Hantao, R.S. Lima, et al., Point-of-use electroanalytical platform based on homemade potentiostat and smartphone for multivariate data processing, Electrochim. Acta 219 (2016) (2016) 170–177, https://doi.org/10.1016/j.electacta.2016.09.157.

[22] S. Imani, A.J. Bandodkar, A.M.V. Mohan, R. Kumar, S. Yu, J. Wang, P.P. Mercier, et al., A wearable chemical-electrophysiological hybrid biosensing system for real-time health and fitness monitoring, Nat. Commun. 7 (2016) 1–7, https://doi.org/10.1038/ncomms11650.

[23] R. Pruna, F. Palacio, A. Baraket, N. Zine, A. Streklas, J. Bausells, A. Errachid, M. López, et al., A low-cost and miniaturized potentiostat for sensing of biomolecular species such as TNF-α by electrochemical impedance spectroscopy, Biosens. Bioelectron. 100 (2018) 533–540, https://doi.org/10.1016/j.bios.2017.09.049.

[24] V. Bianchi, A. Boni, S. Fortunati, M. Giannetto, M. Careri, I. De Munari, et al., A Wi-Fi cloud-based portable potentiostat for electrochemical biosensors, IEEE Trans. Instrum. Meas. 69 (6) (2020) 3232–3240, https://doi.org/10.1109/TIM.2019.2928533.

[25] S.D. Adams, E.H. Doeven, K. Quayle, A.Z. Kouzani, MiniStat: development and evaluation of a mini-potentiostat for electrochemical measurements, IEEE Access 7 (2019) 31903–31912, https://doi.org/10.1109/ACCESS.2019.2902575.

[26] P.J. Burke, A safe, open source, 4G connected self-flying plane with 1h flight time and all up weight (AUW) $<$ 300 g: towards a new class of internet enabled UAVs, IEEE Access 7 (2019) 67833–67855, https://doi.org/10.1109/ACCESS.2019.2917851.

```
//https://github.com/PeterJBurke/Nanostat

bool userpause = false;             // pauses for user to press input on serial between each point in sweep
bool print_output_to_serial = true; // prints verbose output to serial

//Libraries
#include <Wire.h>
#include <WiFi.h>
#include "Arduino.h"
#include <ESPAsyncWebServer.h>
#include <ESPmDNS.h>
#include <SPIFFS.h>
#include <AsyncJson.h>
#include "wifi_credentials.h"
#include "LMP91000.h"
#include "WebSocketsServer.h"
#include "DNSServer.h"
// #include <OTA.h>
#include <Update.h>

// Microcontroller specific info (pinouts, Vcc, etc)
const uint16_t opVolt = 3300;                 //3300 mV
const uint8_t adcBits = 12;                   // ESP32 ADC is 12 bits
const uint16_t dacResolution = pow(2, 8) - 1; // ESP32 DAC is 8 bits, that is 12.9 mV for 3300 mV reference...
//LMP91000 control pins
const uint8_t dac = 25; // DAC pin for Vref to drive LMP91000. Pin 25 on BurkeLab ESP32Stat Rev 3.5
const uint8_t MENB = 5; // LMP91000 enable pin. Hard wired to ground in BurkeLab ESP32Stat Rev 3.5
//analog input pins to read voltages
const uint8_t LMP_C1 = 27; // Note C1, C2 not wired to microcontroller in BurkeLab ESP32Stat Rev 3.5
const uint8_t LMP_C2 = 39; // Note C1, C2 not wired in microcontroller in BurkeLab ESP32Stat Rev 3.5
const uint8_t LMP = 35;    // ADC pin for Vout reading. Pin 35 on BurkeLab ESP32Stat Rev 3.5
// calibration of ESP32 ADC (see calibration routine)
float a_coeff = -146.63; // hard code coeffs but they can be updated with calibration routine
float b_coeff = 7.64;    // hard code coeffs but they can be updated with calibration routine
// For BurkeLab ESP32Stat Rev 3.5, LED "blinky" pin.
int LEDPIN = 26;

// Global mode control (sweep type)
enum Sweep_Mode_Type
{
  dormant,  // not sweeping
  NPV,      // normal pulsed voltametry
  CV,       // cyclic voltametry
  SQV,      // square wave voltametry
  CA,       // chronoamperometry
  IV,       // IV curve
  CAL,      // calibrate
  DCBIAS,   // DC bias at a fixed point
  CTLPANEL, // Control panel type interface
  MISC_MODE // miscellanous
};
Sweep_Mode_Type Sweep_Mode = dormant;

// Sweep parameters (to be set by user through HTML form posts but defaults on initialization)
// (Initialized to their default values)
// General sweep parameters:
int sweep_param_lmpGain = 7;       //  (index) gain setting for LMP91000
bool sweep_param_setToZero = true; //   Boolean

// CV sweep parameters:
// runCV(sweep_param_lmpGain, sweep_param_cycles_CV, sweep_param_startV_CV,
// sweep_param_endV_CV, sweep_param_vertex1_CV, sweep_param_vertex2_CV, sweep_param_stepV_CV,
// sweep_param_rate_CV, sweep_param_setToZero);
int sweep_param_cycles_CV = 3;     //  (#)  number of times to run the scan
int sweep_param_startV_CV = 0;     //   (mV)  voltage to start the scan
int sweep_param_endV_CV = 0;       //     (mV)  voltage to stop the scan
int sweep_param_vertex1_CV = 100;  //  (mV)  edge of the scan
int sweep_param_vertex2_CV = -100; // (mV)  edge of the scan
int sweep_param_stepV_CV = 5;      // (mV)      how much to increment the voltage by
int sweep_param_rate_CV = 100;     // (mV/sec)       scanning rate

// NPV sweep parameters:
// runNPV(sweep_param_lmpGain, sweep_param_startV_NPV, sweep_param_endV_NPV,
//            sweep_param_pulseAmp_NPV, sweep_param_pulseAmp_NPV, sweep_param_period_NPV,
//            sweep_param_quietTime_NPV, uint8_t range, sweep_param_setToZero)
int sweep_param_startV_NPV = -200; //   (mV)  voltage to start the scan
int sweep_param_endV_NPV = 200;    //     (mV)  voltage to stop the scan
int sweep_param_pulseAmp_NPV = 20;
int sweep_param_width_NPV = 50;
int sweep_param_period_NPV = 200;
int sweep_param_quietTime_NPV = 1000;

// SWV sweep parameters:
// runSWV(sweep_param_lmpGain, sweep_param_startV_SWV, sweep_param_endV_SWV,
//            sweep_param_pulseAmp_SWV, sweep_param_stepV_SWV, sweep_param_freq_SWV, sweep_param_setToZero)
int sweep_param_startV_SWV = -200; //   (mV)  voltage to start the scan
int sweep_param_endV_SWV = 200;    //     (mV)  voltage to stop the scan
int sweep_param_pulseAmp_SWV = 20;
int sweep_param_stepV_SWV = 5; // (mV)      how much to increment the voltage by
int sweep_param_freq_SWV = 10;

// CA sweep parameters:
// runAmp(sweep_param_lmpGain, sweep_param_pre_stepV_CA, sweep_param_quietTime_CA,
//            sweep_param_V1_CA, sweep_param_t1_CA, sweep_param_V2_CA, sweep_param_t2_CA,
//            sweep_param_samples_CA, uint8_t range, sweep_param_setToZero)
int sweep_param_pre_stepV_CA = 50;
int sweep_param_quietTime_CA = 2000;
int sweep_param_V1_CA = 100;
int sweep_param_t1_CA = 2000;
int sweep_param_V2_CA = 50;
int sweep_param_t2_CA = 2000;
int sweep_param_samples_CA = 100;

// Noise test sweep parameters:
// testNoiseAtABiasPoint(sweep_param_biasV_noisetest, sweep_param_numPoints_noisetest,
//                           sweep_param_delayTime_ms_noisetest)
int sweep_param_biasV_noisetest = -100;
int sweep_param_numPoints_noisetest = 100;
int sweep_param_delayTime_ms_noisetest = 50;

// IV sweep parameters:
// testIV(sweep_param_startV_IV, sweep_param_endV_IV, sweep_param_numPoints_IV,
//            sweep_param_delayTime_ms_IV)
int sweep_param_startV_IV = -200; //   (mV)  voltage to start the scan
int sweep_param_endV_IV = 200;    //     (mV)  voltage to stop the scan
int sweep_param_numPoints_IV = 701;
int sweep_param_delayTime_ms_IV = 50;

// Cal sweep parameters:
// calibrateDACandADCs(sweep_param_delayTime_ms_CAL)
int sweep_param_delayTime_ms_CAL = 50;

// Arrays of IV curves etc:
const uint16_t arr_samples = 5000; //use 1000 for EIS, can use 2500 for other experiments (10k does not fit in
DRAM)
uint16_t arr_cur_index = 0;
int16_t volts[arr_samples] = {0}; // single sweep IV curve "V"
float amps[arr_samples] = {0};    // single sweep IV curve "I"
int32_t time_Voltammaogram[arr_samples] = {0};
int number_of_valid_points_in_volts_amps_array = 0; // rest of them are all zeros...

// JSON string to sent over websockets to browswer client:
// char Voltammogram_JSON_cstr[31000];
// char Voltammogram_JSON_cstr[31000];
```

```
//char Voltammogram_JSON_cstr2[31000];
int16_t volts_temp = 0;
float amps_temp = 0;
float v1_temp = 0;
float v2_temp = 0;
float analog_read_avg_bits_temp = 0;
String temp_json_string = "";

const float v_tolerance = 0.008;     //0.0075 works every other technique with 1mV step except CV which needs
minimum 2mV step
unsigned long lastTime = 0;          // global variable, last time xyz was called in  ms....
uint16_t dacVout = 1500;             // desired output of the DAC in mV
float adc_avg = 0;                   // used in noise test subroutine, better to make it local eventually...
float adc_std_dev = 0;               // used in noise test subroutine, better to make it local eventually...
int num_adc_readings_to_average = 1; // when reading ADC, how many points to average....

// LMP91000 global status settings:
// Parameters the user can set on LMP91000:
// TIA gain (keep max at 350k feedback resistor)
// Rload (keep at 10 ohms)
// Ref source int/ext (keep ext)
// Int_Z zero 50 20 67% (keep at 50%)
// Bias_Sign (plus/minus)
// Bias 1%-24%
// FET_Short (keep off)
// Mode (keep at 3-lead)
uint8_t LMPgainGLOBAL = 7; // Feedback resistor of TIA.
//void LMP91000::setGain(uint8_t gain) const
//@param          gain: the gain to be set to
//param - value - gain resistor
//0 - 000 - External resistor
//1 - 001 - 2.75 kOhm
//2 - 010 - 3.5 kOhm
//3 - 011 - 7 kOhm
//4 - 100 - 14 kOhm
//5 - 101 - 35 kOhm
//6 - 110 - 120 kOhm
//7 - 111 - 350 kOhm
uint8_t bias_setting = 0; // determines percentage of VREF applied to CE opamp, from 1% to 24%
// from LMP91000.h:
//const double TIA_BIAS[] = {0, 0.01, 0.02, 0.04, 0.06, 0.08, 0.1, 0.12, 0.14,
//    0.16, 0.18, 0.2, 0.22, 0.24};
float RFB = 1e6; // feedback resistor if using external feedback resistor

// create pStat object to control LMP91000
LMP91000 pStat = LMP91000();

// create webserver object for website:
AsyncWebServer server(80); //

// Websockets:
// Tutorial: https://www.youtube.com/watch?v=ZbX-l1Dl4N4&list=PL4sSjlE6rMIlvrllrtOVSBW8WhhMC_oI-&index=8
// Tutorial: https://www.youtube.com/watch?v=mkXsmCgvy0k
// Code tutorial: https://shawnhymel.com/1882/how-to-create-a-web-server-with-websockets-using-an-esp32-in-arduino/
// Github: https://github.com/Links2004/arduinoWebSockets
WebSocketsServer m_websocketserver = WebSocketsServer(81);
String m_websocketserver_text_to_send = "";
String m_websocketserver_text_to_send_2 = "";
int m_time_sent_websocketserver_text = millis();
int m_microsbefore_websocketsendcalled = micros();
int last_time_loop_called = millis();
int last_time_sent_websocket_server = millis();
float m_websocket_send_rate = 1.0; // Hz, how often to send a test point to websocket...
bool m_send_websocket_test_data_in_loop = false;

//WifiTool object
int WAIT_FOR_WIFI_TIME_OUT = 6000;
```

```
const char *PARAM_MESSAGE = "message"; // message server receives from client
std::unique_ptr<DNSServer> dnsServer;
std::unique_ptr<AsyncWebServer> m_wifitools_server;
const byte DNS_PORT = 53;
bool restartSystem = false;

// Control panel mode settings: (Control panel mode is browser page like a front panel of instrument)
uint8_t LMPgain_control_panel = 6; // Feedback resistor of TIA.
int num_adc_readings_to_average_control_panel = 1;
int sweep_param_delayTime_ms_control_panel = 50;
int cell_voltage_control_panel = 100;

//******************* END VARIABLE DECLARATIONS*************************8

boolean connectAttempt(String ssid, String password)
{
  boolean isWiFiConnected = false;
  // set mode
  WiFi.mode(WIFI_STA);
  // if no SSID is passed we attempt last connected wifi (from WIFI object)
  if (ssid == "")
  {
    if (WiFi.status() != WL_CONNECTED)
    {
      WiFi.begin();
    }
  }
  else
  {
    int ssidSize = ssid.length() + 1;
    int passwordSize = password.length() + 1;
    char ssidArray[ssidSize];
    char passwordArray[passwordSize];
    ssid.toCharArray(ssidArray, ssidSize);
    password.toCharArray(passwordArray, passwordSize);
    WiFi.begin(ssidArray, passwordArray);
  } //end if

  if (ssid == "")
  {
    Serial.print(F("Connecting Wifi default SSID ..."));
  }
  else if (ssid != "")
  {
    Serial.print(F("Connecting Wifi SSID = "));
    Serial.print(ssid);
    Serial.print(F(" ..."));
  }
  unsigned long now = millis();
  while (WiFi.status() != WL_CONNECTED && millis() < now + WAIT_FOR_WIFI_TIME_OUT)
  {
    Serial.print(".");
    delay(250);
  }
  // Serial.print(F("\nStatus:"));
  // Serial.println(WiFi.status());
  if (WiFi.status() == WL_CONNECTED)
  {
    Serial.println(F("\nWiFi connected"));
    Serial.println(F("IP address: "));
    Serial.println(WiFi.localIP());
    Serial.print(F("ssid: "));
    Serial.println(WiFi.SSID());
    //Serial.print(F("password: "));
    //Serial.println(WiFi.psk());
    isWiFiConnected = true;
  }
  return isWiFiConnected;
}

void sendTimeOverWebsocketJSON() // sends current time as JSON object to websocket
{
  String json = "{\"value\":";
  json += String(millis() / 1e3, 3);
  json += "}";
  m_websocketserver.broadcastTXT(json.c_str(), json.length());
}

void sendValueOverWebsocketJSON(int value_to_send_over_websocket) // sends integer as JSON object to websocket
{
  String json = "{\"value\":";
  json += String(value_to_send_over_websocket);
  json += "}";
  m_websocketserver.broadcastTXT(json.c_str(), json.length());
}

void sendStringOverWebsocket(String string_to_send_over_websocket)
{
  m_websocketserver.broadcastTXT(string_to_send_over_websocket.c_str(), string_to_send_over_websocket.length());
}

void send_is_sweeping_status_over_websocket(bool is_sweeping)
{
  if (is_sweeping)
  {
    // send is sweeping
    temp_json_string = "{\"is_sweeping\":true}";
  };
  if (!is_sweeping)
  {
    // send is not sweeping
    temp_json_string = "{\"is_sweeping\":false}";
  };
  m_websocketserver.broadcastTXT(temp_json_string.c_str(), temp_json_string.length());
}

void send_expect_binary_data_over_websocket(bool expect_binary_data)
{
  if (expect_binary_data)
  {
    // send is sweeping
    temp_json_string = "{\"expect_binary_data\":true}";
  };
  if (!expect_binary_data)
  {
    // send is not sweeping
    temp_json_string = "{\"expect_binary_data\":false}";
  };
  m_websocketserver.broadcastTXT(temp_json_string.c_str(), temp_json_string.length());
}

void sendVoltammogramWebsocketJSON()
{
  // old version used String, had memory probs for large arrays
  // psuedo code:
  // 1) Convert voltammagram to JSON...
  // Voltammagram JSON format will be like this:
  // { "Current" : [3,2,6,...], "Voltage": [8,6,7,....], "Time": [3,4,5,...]}
  // 2) Send JSON over websocket...

  Serial.println("sendVoltammogramWebsocketJSON called");
  Serial.print("number_of_valid_points_in_volts_amps_array=");
  Serial.println(number_of_valid_points_in_volts_amps_array);
  String current_array_string = "[";
  String voltage_array_string = "[";
  String time_array_string = "[";
  for (uint16_t i = 0; i < number_of_valid_points_in_volts_amps_array; i++)
  {

    current_array_string += amps[i];
    if (i != (number_of_valid_points_in_volts_amps_array - 1))
    {
      current_array_string += ",";
    }

    voltage_array_string += volts[i];
    if (i != (number_of_valid_points_in_volts_amps_array - 1))
    {
      voltage_array_string += ",";
    }

    time_array_string += (time_Voltammaogram[i] - time_Voltammaogram[0]); // normalize to start of sweep time
    if (i != (number_of_valid_points_in_volts_amps_array - 1))
    {
      time_array_string += ",";
    }
  }
  current_array_string += "]";
  voltage_array_string += "]";
  time_array_string += "]";

  String Voltammogram_JSON = "";
  Voltammogram_JSON += "{\"Current\":";
  Voltammogram_JSON += current_array_string;
  Voltammogram_JSON += ",\"Voltage\":";
  Voltammogram_JSON += voltage_array_string;
  Voltammogram_JSON += ",\"Time\":";
  Voltammogram_JSON += time_array_string;
  Voltammogram_JSON += "}";

  if (print_output_to_serial)
  {
    Serial.println("####################################");
    Serial.println("Just created Voltammogram_JSON string.");
    Serial.println("Ingredients of string:");
    Serial.println("current_array_string:");
    Serial.println(current_array_string);
    Serial.println("voltage_array_string:");
    Serial.println(voltage_array_string);
    Serial.println("time_array_string:");
    Serial.println(time_array_string);

    Serial.println("####################################");
    Serial.println("Beginning sending Voltammogram_JSON over websocket.");
    if (print_output_to_serial)
    {
      Serial.println(Voltammogram_JSON);
    }
    Serial.print("Heap free memory (in bytes)= ");
    Serial.println(ESP.getFreeHeap());
  }

  m_websocketserver.broadcastTXT(Voltammogram_JSON.c_str(), Voltammogram_JSON.length());

  if (print_output_to_serial)
  {
    Serial.println("Finished sending Voltammogram_JSON over websocket.");
    Serial.println("####################################");
  }
```

```
}

void sendVoltammogramWebsocketBIN()
{
  // send data over websocket binary
  // psuedo code:
  // 1) Convert voltammagram to JSON...
  // Voltammagram JSON format will be like this:
  // { "Current" : [3,2,6,...], "Voltage": [8,6,7,....], "Time": [3,4,5,...]}
  // 2) Send JSON over websocket...
  // Version 2 tries to use char[] instead of String for memory management (String can't hold all the data)

  // Refernce arrays we are going to use to write:
  // Arrays of IV curves etc:
  // const uint16_t arr_samples = 5000; //use 1000 for EIS, can use 2500 for other experiments (10k does not fit
in DRAM)
  // uint16_t arr_cur_index = 0;
  // int16_t volts[arr_samples] = {0}; // single sweep IV curve "V"
  // float amps[arr_samples] = {0};    // single sweep IV curve "I"
  // int32_t time_Voltammaogram[arr_samples] = {0};
  // int number_of_valid_points_in_volts_amps_array = 0; // rest of them are all zeros...

  Serial.println("sendVoltammogramWebsocketBIN called");
  Serial.print("Heap free memory (in bytes)= ");
  Serial.println(ESP.getFreeHeap());
  Serial.println("Creating binary array to send to browswer");
  Serial.print("number_of_valid_points_in_volts_amps_array=");
  Serial.println(number_of_valid_points_in_volts_amps_array);

  // reset time to zero for 1st point: (unless it's already been done!)
  if (time_Voltammaogram[0] != 0)
  {
    int32_t temp_time, temp_start_time;
    temp_start_time = time_Voltammaogram[0];
    for (uint16_t i = 0; i < number_of_valid_points_in_volts_amps_array; i++)
    {
      temp_time = time_Voltammaogram[i];
      time_Voltammaogram[i] = temp_time - temp_start_time;
    }
  }

  bool m_headerToPayload = false;

  //  *********BEGIN Websocket sample code for sending binary array of floats*****************
  // int m_num_in_test_array = 10;
  // float m_test_float_array[m_num_in_test_array];
  // for (uint8_t i = 0; i < m_num_in_test_array; i += 1)
  // {
  //   m_test_float_array[i] = 3.1415 + i;
  //   Serial.print("m_test_float_array[i]=");
  //   Serial.println(m_test_float_array[i]);
  // }
  // size_t m_test_float_array_buffer_len = m_num_in_test_array * sizeof(float); // 4 byte
  // Serial.print("m_test_float_array_buffer_len=");
  // Serial.println(m_test_float_array_buffer_len);
  // m_websocketserver.broadcastBIN((uint8_t *)m_test_float_array, m_test_float_array_buffer_len,
m_headerToPayload);
  //  *********END Websocket sample code for sending binary array of floats*****************

  //  *********BEGIN Websocket  code for sending  arrays*****************
  int m_amps_array_buffer_len = sizeof(float) * number_of_valid_points_in_volts_amps_array;
  int m_volts_array_buffer_len = sizeof(int16_t) * number_of_valid_points_in_volts_amps_array;
  int m_time_array_buffer_len = sizeof(int32_t) * number_of_valid_points_in_volts_amps_array;
  Serial.print("m_amps_array_buffer_len=");
  Serial.println(m_amps_array_buffer_len);
  Serial.print("m_volts_array_buffer_len=");
  Serial.println(m_volts_array_buffer_len);
  Serial.print("m_time_array_buffer_len=");
  Serial.println(m_time_array_buffer_len);

  // Tell browswer to get ready for 3 new binary array messages:
  send_expect_binary_data_over_websocket(true);
  // Arrays are aleady in memory, no need to use more memory, just send them directly:
  m_websocketserver.broadcastBIN((uint8_t *)amps, m_amps_array_buffer_len, m_headerToPayload);
  m_websocketserver.broadcastBIN((uint8_t *)volts, m_volts_array_buffer_len, m_headerToPayload);
  m_websocketserver.broadcastBIN((uint8_t *)time_Voltammaogram, m_time_array_buffer_len, m_headerToPayload);

  //  *********END Websocket  code for sending  arrays*****************

  if (print_output_to_serial)
  {
    Serial.println("Finished sending sendVoltammogramWebsocketBIN over websocket.");
    Serial.println("####################################");
  }
}

void blinkLED(int pin, int blinkFrequency_Hz, int duration_ms)
// blinks LED for duration ms using frequency of blinkFrequency
{
  int delayPeriod_ms = (1e3) / (2 * blinkFrequency_Hz);
  int t = 0;
  while (t < duration_ms)
  {
    digitalWrite(pin, HIGH);
    delay(delayPeriod_ms);
    digitalWrite(pin, LOW);
    delay(delayPeriod_ms);
    t += 2 * delayPeriod_ms;
  }
}

void pulseLED_on_off(int pin, int on_duration_ms)
// blinks LED on for on_duration_ms
{
  digitalWrite(pin, HIGH);
  delay(on_duration_ms);
  digitalWrite(pin, LOW);
}

inline void setLMPBias(int16_t voltage)
{
  //Sets the LMP91000's bias to positive or negative
  signed char sign = (float)voltage / abs(voltage);
  if (sign < 0)
    pStat.setNegBias();
  else if (sign > 0)
    pStat.setPosBias();
}

void setOutputsToZero()
{
  //Sets the electrochemical cell to 0V bias.
  dacWrite(dac, 0); // ESP32
  pStat.setBias(0);
}

void initLMP(uint8_t lmpGain)
{
  //Initializes the LMP91000 to the appropriate settings
  //for operating MiniStat.
  pStat.disableFET();      // Don't short WE and RE.
  pStat.setGain(lmpGain);  // Set feedback resistor of LMP gain.
  pStat.setRLoad(0);       // Low resistance (10 ohms) between WE and TIA input.
  pStat.setExtRefSource(); // Cell voltage determined by external pin.
  pStat.setIntZ(1);        // TIA uses internal resistor for gain setting.
  pStat.setThreeLead();    // 3 lead potentiostat configuration.
  pStat.setBias(0);        // Zero voltage on cell, DAC.
  pStat.setPosBias();      // Positive bias.
  setOutputsToZero();      // Zero voltage on cell, DAC.
}

inline uint16_t convertDACVoutToDACVal(uint16_t dacVout)
{
  //dacVout       voltage output to set the DAC to
  //
  //Determines the correct value to write to the digital-to-analog
  //converter (DAC) given the desired voltage, the resolution of the DAC

  // dacResolution=2^8-1 ESP32 = 255
  // opVolt =3300 mV
  // DACVal integer 0 to 255 scaled to dacVout/opVolt
  // so resolution is 13 mV!!!

  return dacVout * ((float)dacResolution / opVolt);
}

inline float analog_read_avg(int num_points, int pin_num)
{
  // Reads adc at pin_num for num_points and returns average.
  float analogRead_result = 0;

  for (int16_t j = 0; j < num_points; j += 1)
  {
    analogRead_result += analogRead(pin_num);
  }

  return analogRead_result / num_points;
}

inline void setVoltage(int16_t voltage)
{
  // Sets the DAC voltage, LMP91000 bias percentage, and LMP91000 bias sign, to get the desired cell voltage.
  // "voltage" is the desired cell voltage (RE minus WE)
  // Assumes DAC is perfect.

  // This function iterates bias_setting, dacVout until it finds a solution within v_tolerance of voltage (the
desired cell voltage)
  // End result is bias_setting (integer), dacVout (in mV) which goes to LMP91000 and ESP32
  // setV (the cell voltage the user will actually get)
  // is given by dacVout * TIA_BIAS[bias_setting]
  // TIA_BIAS[bias_setting] varies from 1% to 22% depending on setting; note initially set to 0%...

  // global variables used in this function:
  // v_tolerance: how close to desired cell voltage user can tolerate since it is digitized
  // dacVout: desired output of the DAC in mV
  // bias_setting: determines percentage of VREF applied to CE opamp, from 1% to 24%

  const uint16_t minDACVoltage = 1520; // Minimum DAC voltage that can be set;
                                       //the LMP91000 accepts a minium value of 1.5V, adding the
  //additional 20 mV for the sake of a bit of a buffer

  dacVout = minDACVoltage; // global variable, initialized to a convenient value in this method, will be changed
in iteration
  bias_setting = 0;        // global variable, initialized to a convenient value in this method, will be changed
in iteration

  //if (abs(voltage) < 15)
  //{
  // voltage = 15 * (voltage / abs(voltage));
  //voltage cannot be set to less than 15mV because the LMP91000
  //accepts a minium of 1.5V at its VREF pin and has 1% as its
  //lowest bias option 1.5V*1% = 15mV
  //}

  int16_t setV = dacVout * TIA_BIAS[bias_setting]; // "setV" is the exact cell voltage we will get, try to get it
close to "voltage"
  voltage = abs(voltage);                          // (sign handled elsewhere in the code...)

  if (abs(voltage) < 15)
  {
    // make executive decision
    if (abs(voltage) > 7.5)
    {
      //    make it 15 mV
      voltage = 15 * (voltage / abs(voltage));
    }
    else if (abs(voltage) <= 7.5)
    {
      // make it zero
      bias_setting = 0;
      setV = 0;
      dacVout = 1500;
    }
  }

  if (abs(voltage) >= 15)
  {
    // iterate
    while (setV > voltage * (1 + v_tolerance) || setV < voltage * (1 - v_tolerance)) // iterate to find
    // bias_setting (integer), dacVout (in mV) which goes to LMP91000 and ESP32
    // so that setV = dacVout * TIA_BIAS[bias_setting] is within v_tolerance of voltage (desired by user)
    {
      if (bias_setting == 0)
        bias_setting = 1;

      dacVout = voltage / TIA_BIAS[bias_setting];

      if (dacVout > opVolt)
      {
        bias_setting++;
        dacVout = 1500;

        if (bias_setting > NUM_TIA_BIAS)
          bias_setting = 0;
      }

      setV = dacVout * TIA_BIAS[bias_setting];
    }
  }
  pStat.setBias(bias_setting); // sets percentage voltage divider in LMP910000
  // dacwrite is ESP32 version of analogWrite
  dacWrite(dac, convertDACVoutToDACVal(dacVout)); // sets ESP32 DAC bits
}

inline float biasAndSample(int16_t voltage, uint32_t rate)
{
  //Sets the bias of the electrochemical cell then samples the resulting current.
  //@param        voltage: Set the bias of the electrochemical cell
  //@param        rate:    How much to time (in ms) should we wait to sample
  //                       current after biasing the cell. This parameter
  //                       sets the scan rate or the excitation frequency
  //                       based on which electrochemical technique
  //                       we're running.

  // Global variables assumed:
  // lastTime: last time in system ms this function was called (for timing reference)
  // num_adc_readings_to_average: self explanatory
  // float a = a_coeff; float b = b_coeff; // global calibration coefficients for ADC to voltage correction
  // Global variables used in this function implicit through setVoltage:
  // v_tolerance: how close to desired cell voltage user can tolerate since it is digitized
  // dacVout: desired output of the DAC in mV
  // bias_setting: determines percentage of VREF applied to CE opamp, from 1% to 24%

  setLMPBias(voltage); // Sets the LMP91000's bias to positive or negative // voltage is cell voltage
  setVoltage(voltage); // Sets the DAC voltage, LMP91000 bias percentage, and LMP91000 bias sign, to get the
desired cell "voltage".

  if (rate > 10) // we can afford the time to pulse the LED for 10 ms, just waiting anyways
  {
    pulseLED_on_off(LEDPIN, 10); // signify start of a new data point
  }
  else if (rate < 10) // we cannot afford the time to pulse the LED for 10 ms, so pulse 1 ms only
  {
    pulseLED_on_off(LEDPIN, 1); // signify start of a new data point
  }

  //delay sampling to set scan rate
  while (millis() - lastTime < rate) // wait then sample
    ;

  // Read output voltage of the transimpedance amplifier
  int adc_bits; // will be output voltage of TIA amplifier, "VOUT" on LMP91000 diagram, also C2
  // hard wired in BurkeLab ESP32Stat Rev 3.5 to LMP i.e ESP32 pin 32 (ADC1_CH7)
  adc_bits = analog_read_avg(num_adc_readings_to_average, LMP); // read a number of points and average them...

  // Now convert adc_bits to actual voltage on Vout.
  // Without calibration, we would assume that virtual ground (WE = C1) is 50% of Vref.
  // Without calibration, we would assume that C2-C1=I_WE * RTIA, and C2 is VOUT is ADC reading.
  // However, C1 is not 50% of Vref.
  // Calibration takes care of this by reading C2 as a function of ADC set voltage (assuming perfect ADC) and then
  // correcting C2 using Vout_calibrated = (adcbits/4095)*3.3V corrected by slope/offset measured during
calibration, as follows:
  // First, user opens WE so no current flows, then runs the cal function, which sets LMP cell voltage to Zero and
varies Vref i.e. Vdac.
  // That function measures Vadc as a function of Vdac and fits this formula (defining a,b):
  // adc(bits) = a + b * dac = a + b * dacvoltage * (255/3.3); here dac voltage is assumed correct
  // Correction from ADC to Vout then given by (in volts):
  // v1 = (3.3 / 255.0) * (1 / (2.0 * b)) * (float)adc_bits - (a / (2.0 * b)) * (3.3 / 255.0);

  //  Calibrate coefficients (make a local copy of the global ones):
  float a = a_coeff; // a is local copy of global a_coeff
  float b = b_coeff; // b is local copy of global a_coeff

  float v1;
  v1 = (3.3 / 255.0) * (1 / (2.0 * b)) * (float)adc_bits - (a / (2.0 * b)) * (3.3 / 255.0); // LMP is wired to
Vout of the LMP91000
  v1 = v1 * 1000;

  float v2 = dacVout * .5; //the zero of the internal transimpedance amplifier
  // V2 is not measured in  BurkeLab ESP32Stat Rev 3.5 and assumed to be half dacVout, calibration helps this see
above
  float current = 0;

  //the current is determined by the zero of the transimpedance amplifier
  //from the output of the transimpedance amplifier, then dividing
  //by the feedback resistor
  //current = (V_OUT - V_IN-) / RFB
  //v1 and v2 are in milliVolts
  if (LMPgainGLOBAL == 0)                              // using external feedback resistor, hard coded to 1e6 for
now with BurkeLab ESP32Stat Rev 3.5
    current = (((v1 - v2) / 1000) / RFB) * pow(10, 9); //scales to nA
  else
    current = (((v1 - v2) / 1000) / TIA_GAIN[LMPgainGLOBAL - 1]) * pow(10, 6); //scales to uA
  if (print_output_to_serial)
  {
    // Serial.print("******** BIAS AND SAMPLE START*******************)");
    Serial.print(int(millis())); // current time in ms
    // Serial.print(F(","));        // comma
    Serial.print(F("\t")); // tab
    Serial.print(voltage); // desired cell voltage
    // Serial.print(F(","));  // comma
    Serial.print(F("\t"));                          // tab
    Serial.print(dacVout * TIA_BIAS[bias_setting]); // SET cell voltage
    // setV = dacVout * TIA_BIAS[bias_setting]
    // Serial.print(F(",")); // comma
    Serial.print(F("\t"));                   // tab
    Serial.print(TIA_BIAS[bias_setting], 2); // scale divider
    // Serial.print(F(","));                    // comma
    Serial.print(F("\t"));      // tab
    Serial.print(dacVout, DEC); // dacVout a global variable, calculated in setVoltage(voltage);
    // Serial.print(F(","));       // comma
    Serial.print(F("\t"));     // tab
    Serial.print(adc_bits, 0); // adc_bits
    // Serial.print(F(","));      // comma
    Serial.print(F("\t")); // tab
    Serial.print(v1, 1);   // Vout
    // Serial.print(F(",")); // comma
    Serial.print(F("\t")); // tab
    Serial.print(v2, 0);   // C1, should be the zero also....
    // Serial.print(F(",")); // comma
    Serial.print(F("\t")); // tab
    Serial.print(current, 4);
    // Serial.print(F(",")); // comma
    Serial.print(F("\t")); // tab
    // Serial.print("******** BIAS AND SAMPLE END*******************)");
  }

  //update timestamp for the next measurement
  lastTime = millis();
  return current;
}

void writeVoltsCurrentArraytoFile()
{
  File file = SPIFFS.open("/data.txt", FILE_WRITE);

  if (!file)
  {
    Serial.println("There was an error opening the file for writing");
    return;
  }
  if (file.println("BurkeLab Nanostat Rev 3.5 Sweep"))
  {
    Serial.println("Header of file was written");
  }
  else
  {
    Serial.println("File write failed");
  }
  Serial.println("Writing file.");
  Serial.println("number_of_valid_points_in_volts_amps_array=");
  Serial.println(number_of_valid_points_in_volts_amps_array);

  file.print("Index");
  file.print(F("\t")); // tab
  file.print("Current_Amps");
  file.print(F("\t")); // tab
  file.print("Voltage_V");
  file.print(F("\t")); // tab
  file.println("Time_ms");
  //Wrie Arrays
  for (uint16_t i = 0; i < number_of_valid_points_in_volts_amps_array; i++)
  {
    file.print(i);
    file.print(F("\t")); // tab
    file.print(amps[i], DEC);
    file.print(F("\t")); // tab
    file.print(volts[i] / 1e3, DEC);
    file.print(F("\t"));                                     // tab
    file.println(time_Voltammaogram[i] - time_Voltammaogram[0]); // normalize to start of sweep
  }
  file.close();
}

void readFileAndPrintToSerial()
{
  File file2 = SPIFFS.open("/data.txt");

  if (!file2)
  {
    Serial.println("Failed to open file for reading");
    return;
  }

  Serial.println("File Content:");

  while (file2.available())
  {

    Serial.write(file2.read());
  }

  file2.close();
}

void writeCalFile()
{
  File m_cal_file_to_write_name = SPIFFS.open("/calibration.JSON", FILE_WRITE);

  if (!m_cal_file_to_write_name)
  {
    Serial.println("There was an error opening the file for writing");
    return;
  }

  String m_calibration_string_to_write_JSON = "";
  m_calibration_string_to_write_JSON += "{\"acoeff\":";
  m_calibration_string_to_write_JSON += a_coeff;
  m_calibration_string_to_write_JSON += ",\"bcoeff\":";
  m_calibration_string_to_write_JSON += b_coeff;
  m_calibration_string_to_write_JSON += "}";

  Serial.println("Writing this JSON string to cal file:");
  Serial.println(m_calibration_string_to_write_JSON);

  if (!m_cal_file_to_write_name.println(m_calibration_string_to_write_JSON))
  {
    Serial.println("File write failed");
  }
  m_cal_file_to_write_name.close();
}

void readCalFile()
{
  File m_cal_file_name = SPIFFS.open("/calibration.JSON");
  if (!m_cal_file_name)
  {
    Serial.println("Failed to open cal file file for reading");
    return;
  }

  if (!SPIFFS.exists("/calibration.JSON"))
  {
    Serial.println("calibration.JSON does not exist, creating one.");
    writeCalFile();
    return;
  }

  // Serial.println("Cal file opened.");
  // Serial.println("File Content:");

  String secretsJson;
  while (m_cal_file_name.available()) // read json from file
  {
    secretsJson += char(m_cal_file_name.read());
  } //end while

  // Serial.println("Parsing cal file: ");
  // Serial.println(secretsJson);
  StaticJsonDocument<1000> m_JSONdoc_from_payload;
  DeserializationError m_error = deserializeJson(m_JSONdoc_from_payload, secretsJson); // m_JSONdoc is now a json
object
  if (m_error)
  {
    Serial.println("deserializeJson() failed with code ");
    Serial.println(m_error.c_str());
  }
  //JsonObject m_JsonObject_from_payload = m_JSONdoc_from_payload.as<JsonObject>();
  a_coeff = m_JSONdoc_from_payload["acoeff"];
  b_coeff = m_JSONdoc_from_payload["bcoeff"];
  Serial.println("Cal file loaded with these parameters:");
  Serial.print("a_coeff = ");
  Serial.print(a_coeff);
  Serial.print(F("\t")); // tab
  Serial.print("b_coeff = ");
  Serial.println(b_coeff);

  m_cal_file_name.close();
}

bool readSSIDPWDfile(String m_pwd_filename_to_read)
{
  File m_pwd_file_to_read = SPIFFS.open(m_pwd_filename_to_read);

  if (!m_pwd_file_to_read)
  {
    Serial.println("Failed to open PWD file file for reading");
    return false;
  }

  if (!SPIFFS.exists(m_pwd_filename_to_read))
  {
    Serial.print(m_pwd_filename_to_read);
    Serial.println("  does not exist.");
    return false;
  }

  String m_pwd_file_string;
  while (m_pwd_file_to_read.available()) // read json from file
  {
    m_pwd_file_string += char(m_pwd_file_to_read.read());
  } //end while
  // Serial.print("m_pwd_file_string = ");
  // Serial.println(m_pwd_file_string);
  m_pwd_file_to_read.close();

  //parse
  StaticJsonDocument<1000> m_JSONdoc_from_pwd_file;
  DeserializationError m_error = deserializeJson(m_JSONdoc_from_pwd_file, m_pwd_file_string); // m_JSONdoc is now
a json object
  if (m_error)
  {
    Serial.println("deserializeJson() failed with code ");
    Serial.println(m_error.c_str());
  }
  // m_JSONdoc_from_pwd_file is the JSON object now we can use it.
  String m_SSID1_name = m_JSONdoc_from_pwd_file["SSID1"];
  String m_SSID2_name = m_JSONdoc_from_pwd_file["SSID2"];
  String m_SSID3_name = m_JSONdoc_from_pwd_file["SSID3"];
  String m_PWD1_name = m_JSONdoc_from_pwd_file["PWD1"];
  String m_PWD2_name = m_JSONdoc_from_pwd_file["PWD1"];
  String m_PWD3_name = m_JSONdoc_from_pwd_file["PWD1"];
  // Serial.print("m_SSID1_name = ");
  // Serial.print(m_SSID1_name);
  // Serial.print(F("\t")); // tab
  // Serial.print("m_PWD1_name = ");
  // Serial.print(F("\t")); // tab
  // Serial.print("m_SSID2_name = ");
  // Serial.print(m_SSID2_name);
  // Serial.print(F("\t")); // tab
  // Serial.print("m_PWD2_name = ");
  // Serial.print(m_PWD2_name);
  // Serial.print(F("\t")); // tab
  // Serial.print("m_SSID3_name = ");
  // Serial.print(m_SSID3_name);
  // Serial.print(F("\t")); // tab
  // Serial.print("m_PWD3_name = ");
  // Serial.println(m_PWD3_name);

  // Try connecting:
  //****************************8
  if (connectAttempt(m_SSID1_name, m_PWD1_name))
  {
    return true;
  }
  Serial.println("Failed to connect.");
  if (connectAttempt(m_SSID2_name, m_PWD2_name))
  {
    return true;
  }
  Serial.println("Failed to connect.");

  if (connectAttempt(m_SSID3_name, m_PWD3_name))
  {
    return true;
  }
  Serial.println("Failed to connect.");

  return false;
}

void setUpAPService()
{
  Serial.println(F("Starting Access Point server."));

  // DNSServer dnsServer;
  // dnsServer.reset(new DNSServer());
  WiFi.mode(WIFI_AP);
  // WiFi.softAPConfig(IPAddress(172, 217, 28, 1), IPAddress(172, 217, 28, 1), IPAddress(255, 255, 255, 0));
  WiFi.softAP("NanoStatAP");
  delay(500);

  /* Setup the DNS server redirecting all the domains to the apIP */
  // dnsServer->setErrorReplyCode(DNSReplyCode::NoError);
  // dnsServer->start(DNS_PORT, "*", IPAddress(172, 217, 28, 1));

  //Serial.println("dns server config done");
}

void process()
{
  ///DNS
  // dnsServer->processNextRequest();
  //yield
  yield();
  delay(10);
  // Reset flag/timer
  if (restartSystem)
  {
    if (restartSystem + 1000 < millis())
    {
      ESP.restart();
    } //end if
  }   //end if
}

void handleGetSavSecreteJson(AsyncWebServerRequest *request)
{
  String message;

  String m_SSID1_name;
  String m_SSID2_name;
  String m_SSID3_name;
  String m_PWD1_name;
  String m_PWD2_name;
  String m_PWD3_name;
  String m_temp_string;
  int params = request->params();
  for (int i = 0; i < params; i++)
  {
    AsyncWebParameter *p = request->getParam(i);
    if (p->isPost())
    {
      Serial.print(i);
      Serial.print(F("\t"));
      Serial.print(p->name().c_str());
      Serial.print(F("\t"));
      Serial.println(p->value().c_str());
      m_temp_string = p->name().c_str();
      if (m_temp_string == "ssid1")
      {
        m_SSID1_name = p->value().c_str();
      }
      else if (m_temp_string == "pass1")
      {
        m_PWD1_name = p->value().c_str();
      }
      else if (m_temp_string == "ssid2")
      {
        m_SSID2_name = p->value().c_str();
      }
      else if (m_temp_string == "pass2")
      {
        m_PWD2_name = p->value().c_str();
      }
      else if (m_temp_string == "ssid3")
      {
        m_SSID3_name = p->value().c_str();
      }
      else if (m_temp_string == "pass3")
      {
        m_PWD3_name = p->value().c_str();
      }
    }
  }
  if (request->hasParam(PARAM_MESSAGE, true))
  {
    message = request->getParam(PARAM_MESSAGE, true)->value();
    Serial.println(message);
  }
  else
  {
    message = "No message sent";
  }
  request->send(200, "text/HTML", "Credentials saved. Rebooting...");

  // {"SSID1":"myssid1xyz","PWD1":"mypwd1xyz",
  //     "SSID2":"myssid2xyz","PWD2":"mypwd2xyz",
  //     "SSID3":"myssid3xyz","PWD3":"mypwd3xyz"}

  String SSID_and_pwd_JSON = "";
  SSID_and_pwd_JSON += "{\"SSID1\":\"";
  SSID_and_pwd_JSON += m_SSID1_name;
  SSID_and_pwd_JSON += "\",\"PWD1\":\"";
  SSID_and_pwd_JSON += m_PWD1_name;

  SSID_and_pwd_JSON += "\",\"SSID2\":\"";
  SSID_and_pwd_JSON += m_SSID2_name;
  SSID_and_pwd_JSON += "\",\"PWD2\":\"";
  SSID_and_pwd_JSON += m_PWD2_name;

  SSID_and_pwd_JSON += "\",\"SSID3\":\"";
  SSID_and_pwd_JSON += m_SSID3_name;
  SSID_and_pwd_JSON += "\",\"PWD3\":\"";
  SSID_and_pwd_JSON += m_PWD3_name;

  SSID_and_pwd_JSON += "\"}";

  Serial.println("JSON string to write to file = ");
  Serial.println(SSID_and_pwd_JSON);

  Serial.print("m_SSID1_name = ");
  Serial.print(m_SSID1_name);
  Serial.print(F("\t")); // tab
  Serial.print("m_PWD1_name = ");
  Serial.print(m_PWD1_name);
  Serial.print(F("\t")); // tab
  Serial.print("m_SSID2_name = ");
  Serial.print(m_SSID2_name);
  Serial.print(F("\t")); // tab
  Serial.print("m_PWD2_name = ");
  Serial.print(m_PWD2_name);
  Serial.print(F("\t")); // tab
  Serial.print("m_SSID3_name = ");
  Serial.print(m_SSID3_name);
  Serial.print(F("\t")); // tab
  Serial.print("m_PWD3_name = ");
  Serial.println(m_PWD3_name);

  File m_ssid_pwd_file_to_write_name = SPIFFS.open("/credentials.JSON", FILE_WRITE);

  if (!m_ssid_pwd_file_to_write_name)
  {
    Serial.println("There was an error opening the pwd/ssid file for writing");
    return;
  }

  Serial.println("Writing this JSON string to pwd/ssid file:");
  Serial.println(SSID_and_pwd_JSON);

  if (!m_ssid_pwd_file_to_write_name.println(SSID_and_pwd_JSON))
  {
    Serial.println("File write failed");
  }
  m_ssid_pwd_file_to_write_name.close();

  request->send(200, "text/html", "<h1>Restarting .....</h1>");
  restartSystem = millis();
}

void handleGetSavSecreteJsonNoReboot(AsyncWebServerRequest *request)
{
  String message;

  String m_SSID1_name;
  String m_SSID2_name;
  String m_SSID3_name;
  String m_PWD1_name;
  String m_PWD2_name;
  String m_PWD3_name;
  String m_temp_string;
  int params = request->params();
  for (int i = 0; i < params; i++)
  {
    AsyncWebParameter *p = request->getParam(i);
    if (p->isPost())
    {
      Serial.print(i);
      Serial.print(F("\t"));
      Serial.print(p->name().c_str());
      Serial.print(F("\t"));
      Serial.println(p->value().c_str());
      m_temp_string = p->name().c_str();
      if (m_temp_string == "ssid1")
      {
        m_SSID1_name = p->value().c_str();
      }
      else if (m_temp_string == "pass1")
      {
        m_PWD1_name = p->value().c_str();
      }
      else if (m_temp_string == "ssid2")
      {
        m_SSID2_name = p->value().c_str();
      }
      else if (m_temp_string == "pass2")
      {
        m_PWD2_name = p->value().c_str();
      }
      else if (m_temp_string == "ssid3")
      {
        m_SSID3_name = p->value().c_str();
      }
      else if (m_temp_string == "pass3")
      {
        m_PWD3_name = p->value().c_str();
      }
    }
  }
  if (request->hasParam(PARAM_MESSAGE, true))
  {
    message = request->getParam(PARAM_MESSAGE, true)->value();
    Serial.println(message);
  }
  else
  {
    message = "No message sent";
  }
  // request->send(200, "text/HTML", "bla bla bla bla bla xyz xyz xyz xyz xyz ");

  // {"SSID1":"myssid1xyz","PWD1":"mypwd1xyz",
  //     "SSID2":"myssid2xyz","PWD2":"mypwd2xyz",
  //     "SSID3":"myssid3xyz","PWD3":"mypwd3xyz"}

  String SSID_and_pwd_JSON = "";
  SSID_and_pwd_JSON += "{\"SSID1\":\"";
  SSID_and_pwd_JSON += m_SSID1_name;
  SSID_and_pwd_JSON += "\",\"PWD1\":\"";
  SSID_and_pwd_JSON += m_PWD1_name;

  SSID_and_pwd_JSON += "\",\"SSID2\":\"";
  SSID_and_pwd_JSON += m_SSID2_name;
  SSID_and_pwd_JSON += "\",\"PWD2\":\"";
  SSID_and_pwd_JSON += m_PWD2_name;

  SSID_and_pwd_JSON += "\",\"SSID3\":\"";
  SSID_and_pwd_JSON += m_SSID3_name;
  SSID_and_pwd_JSON += "\",\"PWD3\":\"";
  SSID_and_pwd_JSON += m_PWD3_name;

  SSID_and_pwd_JSON += "\"}";

  Serial.println("JSON string to write to file = ");
  Serial.println(SSID_and_pwd_JSON);

  Serial.print("m_SSID1_name = ");
  Serial.print(m_SSID1_name);
  Serial.print(F("\t")); // tab
  Serial.print("m_PWD1_name = ");
  Serial.print(m_PWD1_name);
  Serial.print(F("\t")); // tab
  Serial.print("m_SSID2_name = ");
  Serial.print(m_SSID2_name);
  Serial.print(F("\t")); // tab
  Serial.print("m_PWD2_name = ");
  Serial.print(m_PWD2_name);
  Serial.print(F("\t")); // tab
  Serial.print("m_SSID3_name = ");
  Serial.print(m_SSID3_name);
  Serial.print(F("\t")); // tab
  Serial.print("m_PWD3_name = ");
  Serial.println(m_PWD3_name);

  File m_ssid_pwd_file_to_write_name = SPIFFS.open("/credentials.JSON", FILE_WRITE);

  if (!m_ssid_pwd_file_to_write_name)
  {
    Serial.println("There was an error opening the pwd/ssid file for writing");
    return;
  }

  Serial.println("Writing this JSON string to pwd/ssid file:");
  Serial.println(SSID_and_pwd_JSON);

  if (!m_ssid_pwd_file_to_write_name.println(SSID_and_pwd_JSON))
  {
    Serial.println("File write failed");
  }
  m_ssid_pwd_file_to_write_name.close();

  request->send(200, "text/html", "  <head> <meta http-equiv=\"refresh\" content=\"2; URL=wifi.html\" /> <meta
name=\"viewport\" content=\"width=device-width, initial-scale=1.0\"> </head> <body> <h1> Credentials stored to
flash on NanoStat. </h1>  </body>");
  restartSystem = millis();
}

void handleFileList(AsyncWebServerRequest *request)
{
  Serial.println("handle fle list");
  if (!request->hasParam("dir"))
  {
    request->send(500, "text/plain", "BAD ARGS");
    return;
  }

  AsyncWebParameter *p = request->getParam("dir");
  String path = p->value().c_str();
  Serial.println("handleFileList: " + path);
  String output = "[";

  File root = SPIFFS.open("/", "r");
  if (root.isDirectory())
  {
    Serial.println("here ??");
    File file = root.openNextFile();
    while (file)
    {
      if (output != "[")
      {
        output += ',';
      }
      output += "{\"type\":\"";
      output += (file.isDirectory()) ? "dir" : "file";
      output += "\",\"name\":\"";
      output += String(file.name()).substring(1);
      output += "\"}";
      file = root.openNextFile();
    }
  }

  path = String();
  output += "]";
  Serial.println("Sending file list to client.");
  // Serial.println(output);
  request->send(200, "application/json", output);
}

void handleUpload(AsyncWebServerRequest *request, String filename, String redirect, size_t index, uint8_t *data,
size_t len, bool final)
{
  Serial.println("handleUpload called");
  Serial.println(filename);
  Serial.println(redirect);
  File fsUploadFile;
  if (!index)
  {
    if (!filename.startsWith("/"))
      filename = "/" + filename;
    Serial.println((String) "UploadStart: " + filename);
    fsUploadFile = SPIFFS.open(filename, "w"); // Open the file for writing in SPIFFS (create if it doesn't exist)
  }
  for (size_t i = 0; i < len; i++)
  {
    fsUploadFile.write(data[i]);
    // Serial.write(data[i]);
  }
  if (final)
  {
    Serial.println((String) "UploadEnd: " + filename);
    fsUploadFile.close();

    request->send(200, "text/HTML", "  <head> <meta http-equiv=\"refresh\" content=\"2; URL=files.html\" /> <meta
name=\"viewport\" content=\"width=device-width, initial-scale=1.0\"> </head> <body> <h1> File uploaded! </h1> <p>
Returning to file page. </p> </body>");

    // request->redirect(redirect);
  }
}

// handle the upload of the firmware
void handleFirmwareUpload(AsyncWebServerRequest *request, String filename, size_t index, uint8_t *data, size_t
len, bool final)
{
  // handle upload and update
  if (!index)
  {
    Serial.printf("Update: %s\n", filename.c_str());
    if (!Update.begin(UPDATE_SIZE_UNKNOWN))
    { //start with max available size
      Update.printError(Serial);
    }
  }

  /* flashing firmware to ESP*/
  if (len)
  {
    Update.write(data, len);
  }

  if (final)
  {
    if (Update.end(true))
    { //true to set the size to the current progress
      Serial.printf("Update Success: %ub written\nRebooting...\n", index + len);
    }
    else
    {
      Update.printError(Serial);
    }
  }
  // alternative approach
  // https://github.com/me-no-dev/ESPAsyncWebServer/issues/542#issuecomment-508489206
}

// handle the upload of the firmware
void handleFilesystemUpload(AsyncWebServerRequest *request, String filename, size_t index, uint8_t *data, size_t
len, bool final)
{
  // handle upload and update
  if (!index)
  {
    Serial.printf("Update: %s\n", filename.c_str());
    // if (!Update.begin(UPDATE_SIZE_UNKNOWN))
    if (!Update.begin(SPIFFS.totalBytes(), U_SPIFFS))
    { //start with max available size
      Update.printError(Serial);
    }
  }

  /* flashing firmware to ESP*/
  if (len)
  {
    Update.write(data, len);
  }

  if (final)
  {
    if (Update.end(true))
    { //true to set the size to the current progress
      Serial.printf("Update Success: %ub written\nRebooting...\n", index + len);
    }
    else
    {
      Update.printError(Serial);
    }
  }
  // alternative approach
  // https://github.com/me-no-dev/ESPAsyncWebServer/issues/542#issuecomment-508489206
}

void getWifiScanJson(AsyncWebServerRequest *request)
{
  String json = "{\"scan_result\":[";
  int n = WiFi.scanComplete();
  if (n == -2)
  {
    WiFi.scanNetworks(true);
  }
  else if (n)
  {
    for (int i = 0; i < n; ++i)
    {
      if (i)
        json += ",";
      json += "{";
      json += "\"RSSI\":" + String(WiFi.RSSI(i));
      json += ",\"SSID\":\"" + WiFi.SSID(i) + "\"";
      json += "}";
    }
    WiFi.scanDelete();
    if (WiFi.scanComplete() == -2)
    {
      WiFi.scanNetworks(true);
    }
  }
  json += "]}";
  request->send(200, "application/json", json);
  json = String();
}

void runWifiPortal()
{

  m_wifitools_server.reset(new AsyncWebServer(80));

  IPAddress myIP;
  myIP = WiFi.softAPIP();
  // myIP = WiFi.localIP();
  Serial.print(F("AP IP address: "));
  Serial.println(myIP);

  // Need to tell server to accept packets from any source with any header via http methods GET, PUT:
  DefaultHeaders::Instance().addHeader("Access-Control-Allow-Origin", "*");
  DefaultHeaders::Instance().addHeader("Access-Control-Allow-Methods", "GET, PUT");
  DefaultHeaders::Instance().addHeader("Access-Control-Allow-Headers", "*");

  m_wifitools_server->serveStatic("/", SPIFFS, "/").setDefaultFile("wifi_index.html");
  // m_wifitools_server->on("/saveSecret/", HTTP_ANY, [&, this](AsyncWebServerRequest *request) {
  //   handleGetSavSecreteJson(request);
  // });

  m_wifitools_server->on("/saveSecret", HTTP_POST, [](AsyncWebServerRequest *request)
                         { handleGetSavSecreteJson(request); });

  m_wifitools_server->on("/list", HTTP_ANY, [](AsyncWebServerRequest *request)
                         { handleFileList(request); });

  m_wifitools_server->on("/wifiScan.json", HTTP_GET, [](AsyncWebServerRequest *request)
                         { getWifiScanJson(request); });

  //xyzxyzxyzxyzxyzxyzxyzxyzxyzxyzxyzxyzxyzxyzxyzxyzxyzxyzxyzxyzxyzxyzxyzxyzxyzxyzxyzxyzxyzxyzxyzxyzxyzxyz

  // m_wifitools_server->on("/saveSecret/", HTTP_ANY, [&, this](AsyncWebServerRequest *request) {
  //   handleGetSavSecreteJson(request);
  // });

  // m_wifitools_server->on("/list", HTTP_ANY, [&, this](AsyncWebServerRequest *request) {
  //   handleFileList(request);
  // });

  // // spiff delete
  // m_wifitools_server->on("/edit", HTTP_DELETE, [&, this](AsyncWebServerRequest *request) {
  //   handleFileDelete(request);
  // });

  // // spiff upload
  // m_wifitools_server->on(
  //     "/edit", HTTP_POST, [&, this](AsyncWebServerRequest *request) {},
  //     [&, this](AsyncWebServerRequest *request, const String &filename, size_t index, uint8_t *data,
  //               size_t len, bool final) {
  //       handleUpload(request, filename, "/wifi_spiffs_admin.html", index, data, len, final);
  //     });

  // m_wifitools_server->on("/wifiScan.json", HTTP_GET, [&, this](AsyncWebServerRequest *request) {
  //   getWifiScanJson(request);
  // });

  // // Simple Firmware Update Form
  // m_wifitools_server->on("/update", HTTP_GET, [&, this](AsyncWebServerRequest *request) {
  //   request->send(SPIFFS, "/wifi_upload.html");
  // });
  // m_wifitools_server->on(
  //     "/update", HTTP_POST, [&, this](AsyncWebServerRequest *request) {
  //       uint8_t isSuccess = !Update.hasError();
  //       if (isSuccess)
  //         restartSystem = millis();
  //       AsyncWebServerResponse *response = request->beginResponse(200, "text/plain", isSuccess ? "OK" : "FAIL");
  //       response->addHeader("Connection", "close");
  //       request->send(response); },
  //     [&, this](AsyncWebServerRequest *request, String filename, size_t index, uint8_t *data, size_t len, bool
final) {
  //       if (!index)
  //       {
  //         Serial.printf("Update Start: %s\n", filename.c_str());

  //         if (!Update.begin(UPDATE_SIZE_UNKNOWN))
  //         {
  //           Update.printError(Serial);
  //         }
  //       }
  //       if (!Update.hasError())
  //       {
  //         if (Update.write(data, len) != len)
  //         {
```

```
  //           Update.printError(Serial);
  //         }
  //       }
  //       if (final)
  //       {
  //         if (Update.end(true))
  //         {
  //           Serial.printf("Update Success: %uB\n", index + len);
  //         }
  //         else
  //         {
  //           Update.printError(Serial);
  //         }
  //       }
  //     });

  // m_wifitools_server->on("/restart", HTTP_GET, [&, this](AsyncWebServerRequest *request) {
  //   request->send(200, "text/html", "OK");
  //   restartSystem = millis();
  // });

  // m_wifitools_server->onNotFound([](AsyncWebServerRequest *request) {
  //   Serial.println("handle not found");
  //   request->send(404);
  // });

  // m_wifitools_server->addHandler(new CaptiveRequestHandler()).setFilter(ON_AP_FILTER); //only when requested
from AP

  //xyzxyzxyzxyzxyzxyzxyzxyzxyzxyzxyzxyzxyzxyzxyzxyzxyzxyzxyzxyzxyzxyzxyzxyzxyzxyzxyzxyzxyzxyzxyzxyzxyzxyz

  Serial.println(F("HTTP server started"));
  m_wifitools_server->begin();
  if (!MDNS.begin("nanostat")) // see https://randomnerdtutorials.com/esp32-access-point-ap-web-server/
  {
    Serial.println("Error setting up MDNS responder !");
    while (1)
      ;
    {
      delay(1000);
    }
  }
  Serial.println("MDNS started.");
  // MDNS.begin("nanostat");
  while (1) // loop until user hits restart... Once credentials saved, won't end up here again unless wifi not
connecting!
  {
    process();
  }
}

void runWifiPortal_after_connected_to_WIFI()
{
  // Don't run this after starting server or ESP32 will crash!!!
  server.on("/saveSecret/", HTTP_POST, [](AsyncWebServerRequest *request)
            { handleGetSavSecreteJsonNoReboot(request); });

  //xyzxyzxyzxyzxyzxyzxyzxyzxyzxyzxyzxyzxyzxyzxyzxyzxyzxyzxyzxyzxyzxyzxyzxyzxyzxyzxyzxyzxyzxyzxyzxyzxyzxyz

  // m_wifitools_server->on("/saveSecret/", HTTP_ANY, [&, this](AsyncWebServerRequest *request) {
  //   handleGetSavSecreteJson(request);
  // });

  // m_wifitools_server->on("/list", HTTP_ANY, [&, this](AsyncWebServerRequest *request) {
  //   handleFileList(request);
  // });
  // // spiff delete
  // m_wifitools_server->on("/edit", HTTP_DELETE, [&, this](AsyncWebServerRequest *request) {
  //   handleFileDelete(request);
  // });

  // // spiff upload
  // m_wifitools_server->on(
  //     "/edit", HTTP_POST, [&, this](AsyncWebServerRequest *request) {},
  //     [&, this](AsyncWebServerRequest *request, const String &filename, size_t index, uint8_t *data,
  //               size_t len, bool final) {
  //       handleUpload(request, filename, "/wifi_spiffs_admin.html", index, data, len, final);
  //     });

  // m_wifitools_server->on("/wifiScan.json", HTTP_GET, [&, this](AsyncWebServerRequest *request) {
  //   getWifiScanJson(request);
  // });

  // // Simple Firmware Update Form
  // m_wifitools_server->on("/update", HTTP_GET, [&, this](AsyncWebServerRequest *request) {
  //   request->send(SPIFFS, "/wifi_upload.html");
  // });
  // m_wifitools_server->on(
  //     "/update", HTTP_POST, [&, this](AsyncWebServerRequest *request) {
  //       uint8_t isSuccess = !Update.hasError();
  //       if (isSuccess)
  //         restartSystem = millis();
  //       AsyncWebServerResponse *response = request->beginResponse(200, "text/plain", isSuccess ? "OK" : "FAIL");
  //       response->addHeader("Connection", "close");
  //       request->send(response); },
  //     [&, this](AsyncWebServerRequest *request, String filename, size_t index, uint8_t *data, size_t len, bool
final) {
  //       if (!index)
  //       {
  //         Serial.printf("Update Start: %s\n", filename.c_str());

  //         if (!Update.begin(UPDATE_SIZE_UNKNOWN))
  //         {
  //           Update.printError(Serial);
  //         }
  //       }
  //       if (!Update.hasError())
  //       {
  //         if (Update.write(data, len) != len)
  //         {
  //           Update.printError(Serial);
  //         }
  //       }
  //       if (final)
  //       {
  //         if (Update.end(true))
  //         {
  //           Serial.printf("Update Success: %uB\n", index + len);
  //         }
  //         else
  //         {
  //           Update.printError(Serial);
  //         }
  //       }
  //     });

  // m_wifitools_server->on("/restart", HTTP_GET, [&, this](AsyncWebServerRequest *request) {
  //   request->send(200, "text/html", "OK");
  //   restartSystem = millis();
  // });

  // m_wifitools_server->onNotFound([](AsyncWebServerRequest *request) {
  //   Serial.println("handle not found");
  //   request->send(404);
  // });

  // m_wifitools_server->addHandler(new CaptiveRequestHandler()).setFilter(ON_AP_FILTER); //only when requested
from AP

  //xyzxyzxyzxyzxyzxyzxyzxyzxyzxyzxyzxyzxyzxyzxyzxyzxyzxyzxyzxyzxyzxyzxyzxyzxyzxyzxyzxyzxyzxyzxyzxyzxyzxyzxyz

  Serial.println(F("HTTP server started"));
  m_wifitools_server->begin();
  if (!MDNS.begin("nanostat")) // see https://randomnerdtutorials.com/esp32-access-point-ap-web-server/
  {
    Serial.println("Error setting up MDNS responder !");
    while (1)
      ;
    {
      delay(1000);
    }
  }
  Serial.println("MDNS started.");
  // MDNS.begin("nanostat");
  while (1) // loop until user hits restart... Once credentials saved, won't end up here again unless wifi not
connecting!
  {
    process();
  }
}

void listDir(const char *dirname, uint8_t levels)
{
  // from https://github.com/espressif/arduino-esp32/blob/master/libraries/SPIFFS/examples/SPIFFS_Test/-
SPIFFS_Test.ino#L9
  // see also https://techtutorialsx.com/2019/02/24/esp32-arduino-listing-files-in-a-spiffs-file-system-specific-
path/
  Serial.printf("Listing directory: %s\r\n", dirname);

  File root = SPIFFS.open(dirname);
  if (!root)
  {
    Serial.println("- failed to open directory");
    return;
  }
  if (!root.isDirectory())
  {
    Serial.println(" - not a directory");
    return;
  }

  File file = root.openNextFile();
  while (file)
  {

    Serial.print("  FILE: ");
    Serial.print(file.name());
    Serial.print("\tSIZE: ");
    Serial.println(file.size());

    file = root.openNextFile();
  }
}

void reset_Voltammogram_arrays()
{
  for (uint16_t i = 0; i < arr_samples; i++)
  {
    volts[i] = 0;
    amps[i] = 0;
    time_Voltammaogram[i] = 0;
  }
}

inline void saveVoltammogram(float voltage, float current, bool debug)
{
  //@param        voltage: voltage or time depending on type of experiment
  //                       voltage for voltammetry, time for time for
  //                       time evolution experiments like chronoamperometry
  //@param        current: current from the electrochemical cell

  //
  volts[arr_cur_index] = (int16_t)voltage;
  amps[arr_cur_index] = current;
  time_Voltammaogram[arr_cur_index] = millis();
  arr_cur_index++;
  number_of_valid_points_in_volts_amps_array++;

  if (debug)
  {
    Serial.println(" ");
    Serial.println("****************** SAVING TO VOLTAMMOGRAM******************");
    Serial.print("arr_cur_index=");
    Serial.println(arr_cur_index);
    Serial.print(voltage);
    Serial.print(F("\t"));
    Serial.print(current, 5);
    Serial.print(F("\t"));
    Serial.println(micros());
    Serial.println("****************** SAVED TO VOLTAMMOGRAM******************");
    Serial.println(" ");
  }
}
void testIV(int16_t startV, int16_t endV, int16_t numPoints,
            uint32_t delayTime_ms)
{
  //@param      startV:       voltage to start the scan
  //@param      endV:         voltage to stop the scan
  //@param      numPoints:    number of points in the scan
  //@param      delayTime_ms:  settle time after set point
  //
  // Measures IV curve.

  float i_forward = 0;
  uint32_t voltage_step;
  lastTime = millis();
  //Reset Arrays
  reset_Voltammogram_arrays(); // sets volt[i], amps[i],time_Voltammaogram[i]=0 all
  arr_cur_index = 0;
  number_of_valid_points_in_volts_amps_array = 0;

  voltage_step = (endV - startV) / numPoints;
  if (voltage_step == 0)
  {
    voltage_step = 1; // 1 mV minimum
  }

  Serial.println("*****************************************************");
  Serial.println("Starting IV sweep with these parameters:");
  Serial.println(startV);
  Serial.println(endV);
  Serial.println(numPoints);
  Serial.println(voltage_step);
  Serial.print("Heap free memory (in bytes)= ");
  Serial.println(ESP.getFreeHeap());

  // Initialize LMP91000:
```

```
  initLMP(LMPgainGLOBAL);
  setLMPBias(startV);
  setVoltage(startV);

  //xxxxxxxxxxxxxxxxxxxxxxxxxx
  if (print_output_to_serial)
  {
    Serial.println("RUNNING TEST IV:");
    Serial.print("a_coeff =");
    Serial.println(a_coeff);
    Serial.print("b_coeff =");
    Serial.println(b_coeff);
    Serial.print("startV =");
    Serial.println(startV);
    Serial.print("endV =");
    Serial.println(endV);
    Serial.print("numPoints =");
    Serial.println(numPoints);
    Serial.print("delayTime_ms =");
    Serial.println(delayTime_ms);
    Serial.print("voltage_step =");
    Serial.println(voltage_step);

    Serial.println("Column header meanings:");
    Serial.println("BIAS:");
    Serial.println("T = time in ms");
    Serial.println("Vc = desired cell voltage in mV (internal variable name = voltage)");
    Serial.println("Vset = set cell voltage in mV (internal variable name = vset = dacVout *
TIA_BIAS[bias_setting])");
    Serial.println("div = percentage scale (internal variable name = TIA_BIAS[bias_setting])");
    Serial.println("Vdac = ESP32 dac voltage in mV (internal variable name = dacVout)");
    Serial.println("adcbits = analogread(LMP)");
    Serial.println("Vout = TIA output in mV (internal variable name = v1 = pStat.getVoltage(analogRead(LMP),
opVolt, adcBits);");
    Serial.println("Vc1 = internal zero in mV also C1 (internal variable name = v2 = dacVout * .5)");
    Serial.println("i_f = forward current in uA (internal variable name = current = (((v1 - v2) / 1000) /
TIA_GAIN[LMPgain - 1]) * pow(10, 6); //scales to uA)");

    // Serial.println("xyz = xyz in xyz (internal variable name = xyz)");
    // Serial.println("xyz = xyz in xyz (internal variable name = v)");

    Serial.println("T\tVc\tVset\tdiv\tVdac\tadcbits\tVout\tVc1\ti_f");
    // Serial.println("T,Vc,Vset,div,Vdac,adcbits,Vout,Vc1,i_f");
  }

  for (int16_t this_voltage = startV; this_voltage <= endV; this_voltage += voltage_step)
  {
    i_forward = biasAndSample(this_voltage, delayTime_ms);
    saveVoltammogram(this_voltage, i_forward, false);
    if (print_output_to_serial)
    {
      Serial.println("EOL");
    }
    if (userpause) //will hold the code here until a character is sent over the Serial port
    {
      while (!Serial.available())
        ;
      Serial.read();
    }
  }
  setOutputsToZero();
  Serial.println("Finished IV sweep.");
  Serial.print("Heap free memory (in bytes)= ");
  Serial.println(ESP.getFreeHeap());
  Serial.println("*******************************************************");
}
```

```
void testNoiseAtABiasPoint(int16_t biasV, int16_t numPoints,
                           uint32_t delayTime_ms)
{
  // sets call voltage to biasV using  setVoltage(biasV) only at beginning
  // then "reads" ADC numPoints times using analog_read_avg(num_readings_to_average, LMP); where each reading is
an average
  // gives std_dev
  // internally iterates averages per reading to get idea how it effects noise...
  // dumps output to serial

  //Reset Arrays
  reset_Voltammogram_arrays(); // sets volt[i], amps[i],time_Voltammaogram[i]=0 all
  number_of_valid_points_in_volts_amps_array = 0;
  arr_cur_index = 0;

  // local variables for statistics:
  float adc_bits_array[numPoints];
  Serial.print("Heap free memory (in bytes)= ");
  Serial.println(ESP.getFreeHeap());
  float adc_bits_array_sum = 0;
  float adc_bits_array_avg = 0;
  float adc_bits_array_std_dev = 0;
  float adc_bits_minus_avg_squared_sum = 0;
  int num_readings_to_average = 1;

  // which num point to avg:
  float num_points_to_average[7] = {1, 5, 10, 50, 100, 500, 1000};

  // set LMP91000 and bias
  initLMP(LMPgainGLOBAL);
  setLMPBias(biasV);
  setVoltage(biasV);

  if (print_output_to_serial)
  {
    Serial.print("num_rdgs");
    Serial.print(F("\t"));

    Serial.print("adc_avg");
    Serial.print(F("\t"));

    Serial.println("adc_std_dev");
  };

  for (int16_t i = 0; i < 7; i += 1) // iterate over # of pts per reading
  {
    adc_bits_array_sum = 0;
    adc_bits_array_avg = 0;
    adc_bits_array_std_dev = 0;
    adc_bits_minus_avg_squared_sum = 0;
    num_readings_to_average = num_points_to_average[i];

    for (int16_t j = 0; j < numPoints; j += 1) // read the adc data
    {
      //handle pulsing and delay
      if (delayTime_ms > 10) // we can afford the time to pulse the LED for 10 ms, just waiting anyways
      {
        pulseLED_on_off(LEDPIN, 10); // signify start of a new data point
        delay(delayTime_ms - 10);
      }
      else if (delayTime_ms < 10) // we cannot afford the time to pulse the LED for 10 ms, so pulse 1 ms only
      {
        if (delayTime_ms > 2)
        {
          pulseLED_on_off(LEDPIN, 1); // signify start of a new data point
          delay(delayTime_ms - 1);
        }
        else if (delayTime_ms < 2)
        { // no LED pulse too fast....
          delay(delayTime_ms);
        }
      }
      adc_bits_array[j] = analog_read_avg(num_readings_to_average, LMP);
      //pulseLED_on_off(LEDPIN, 10);
      // Now populate Volts array etc:
      volts[arr_cur_index] = biasV;
      time_Voltammaogram[arr_cur_index] = millis();
      amps[arr_cur_index] = adc_bits_array[j];
      arr_cur_index++;
      number_of_valid_points_in_volts_amps_array++;
    }
    for (int16_t j = 0; j < numPoints; j += 1) // calculate the average
    {
      adc_bits_array_sum += adc_bits_array[j];
    }
    adc_bits_array_avg = adc_bits_array_sum / numPoints;
    for (int16_t j = 0; j < numPoints; j += 1) // calculate the std deviation
    {
      adc_bits_minus_avg_squared_sum += (adc_bits_array_avg - adc_bits_array[j]) * (adc_bits_array_avg -
adc_bits_array[j]);
    }
    adc_bits_array_std_dev = sqrt(adc_bits_minus_avg_squared_sum / numPoints);

    if (print_output_to_serial)
    {
      Serial.print(num_readings_to_average);
      Serial.print(F("\t"));

      Serial.print(adc_bits_array_avg);
      Serial.print(F("\t"));

      Serial.println(adc_bits_array_std_dev);
    }
  }
}

void testDACs(uint32_t delayTime_ms)
{

  // steps ESP32 DAC from 0 to 255 digital so you can check it on a multimeter

  //xxxxxxxxxxxxxxxxxxxxxxxxxx

  initLMP(LMPgainGLOBAL);
  setLMPBias(100);
  setVoltage(100);

  //xxxxxxxxxxxxxxxxxxxxxxxxxx
  Serial.println("RUNNING TEST DAC:");

  Serial.println("i\tdacgoal");

  float dacgoal;
  for (int16_t j = 0; j <= 255; j += 1)
  {
    dacWrite(dac, j); // sets ESP32 DAC bits
    dacgoal = 3300.0 * j / 255;
    Serial.print(j); // current time in ms
    Serial.print(F("\t"));
    Serial.println(dacgoal); // desired cell voltage
    if (userpause)           //will hold the code here until a character is sent over the Serial port
    {
      while (!Serial.available())
        ;
      Serial.read();
    }
  }
  setOutputsToZero();
}

void testDACandADCs(uint32_t delayTime_ms)
{

  // steps ESP32 DAC from 0 to 255 digital so you can check it on a multimeter
  // measures with ADC
  // prints to serial
  float adc_voltage;

  //xxxxxxxxxxxxxxxxxxxxxxxxxx

  initLMP(LMPgainGLOBAL);
  setLMPBias(100);
  setVoltage(100);

  //xxxxxxxxxxxxxxxxxxxxxxxxxx
  Serial.println("RUNNING TEST DAC:");

  Serial.println("i\tdacgoal\tadc_voltage xyz");

  float dacgoal;
  for (int16_t j = 0; j <= 255; j += 1)
  {
    pulseLED_on_off(LEDPIN, 10); // signify start of a new data point
    // dacWrite(dac, j); // sets ESP32 DAC bits
    dacWrite(dac, j); // sets ESP32 DAC bits
    // dacgoal = 3300.0 * j / 255;
    dacgoal = 3300.0 * 136 / 255;
    delay(delayTime_ms);
    // read ADC....
    adc_voltage = analogRead(LMP);
    Serial.print(j); // current time in ms
    Serial.print(F("\t"));
    Serial.print(dacgoal); // desired cell voltage
    Serial.print(F("\t"));
    Serial.println(adc_voltage); // adc voltage

    if (userpause) //will hold the code here until a character is sent over the Serial port
    {
      while (!Serial.available())
        ;
      Serial.read();
    }
  }
  setOutputsToZero();
}

void calibrateDACandADCs(uint32_t delayTime_ms)
{

  // Sets LMP91000 gain to 0%, so at Vout, voltage is always 0.5 * Vref. (Assumes WE floating so no WE current!)
  // Stores results (slope/offset calibration points) in global variables    a_coeff, b_coeff
  // Note probably need to do it for each TIA gain setting, hard coded to 7 ie. RTIA=350kohm for now.
  // Vout=DAC, Vref=ADC
  // Run DAC from 1.5 to 3 volts (in digital units with 8 bits, 116 to 255 bit)
  // Read ADC over same range. Have ADC calibratoin for DAC range.
  // Will use to correct as follows:
  // Fit ADC/2 vs DAC and return call coefficients....
  // To run: 1) Disconnect WE from RE/CE; 2) Run calibrateDACandADCs; 3) Reconnect; 4) Run testIV (biasandsample
uses coefficients)
  // Temporarily, click CV button on website for step 2 and NPV for step 4 (hardwired, later to fix UI for this...)
  float dacgoal;
  int this_voltage_adc;

  //xxxxxxxxxxxxxxxxxxxxxxxxxx
  pStat.disableFET();
  pStat.setGain(LMPgainGLOBAL); // TIA feedback resistor , global variable
  pStat.setRLoad(0);
  pStat.setExtRefSource();
  pStat.setIntZ(1);
  pStat.setThreeLead();
  pStat.setBias(0);
  pStat.setNegBias();
  pStat.setBias(0); // sets percentage voltage divider in LMP910000
                  // TIA_BIAS[bias_setting] varies from 1% to 22% depending on setting; note initially set to
0%...
                  // from LMP91000.h:
                  //const double TIA_BIAS[] = {0, 0.01, 0.02, 0.04, 0.06, 0.08, 0.1, 0.12, 0.14,
                  //    0.16, 0.18, 0.2, 0.22, 0.24};
                  //xxxxxxxxxxxxxxxxxxxxxxxxxx

  //xxxxxxxxxxxxxxxxxxxxxxxxxx
  Serial.println("RUNNING TEST ADC CAL LMP91000:");
  Serial.println("TIA_BIAS[0]=");
  Serial.println(TIA_BIAS[0]);

  // int adc_int_from_dac_int[140]; // for dac 116 to 255, what adc reads
  // assuming cell is open, no current, ADC is reading vout which is also WE voltage if no current
  // WE voltage is supposed to be 0.5* Vref, and Vref is DAC

  float sumonx = 0;
  float sumony = 0;
  float sumonxy = 0;
  float sumonxsquared = 0;
  float sumonysquared = 0;
  float n = 0;
  float a = 0;
  float b = 0;

  for (int16_t j = 116; j <= 255; j += 1) // want to go from 1.5 to 3.3 V
  // 1.5 V is (1.5/3.3)*255 = 115.909 = 116 for 8 bit dac
  {
    dacWrite(dac, j); // sets ESP32 DAC bits
    if (j == 116)
    {
      delay(250);
    }
    // read analog voltage which should be half of dac voltage...
    this_voltage_adc = analogRead(LMP);
    // adc_int_from_dac_int[j - 116] = this_voltage_adc;
    dacgoal = 3300.0 * j / 255;
    if (print_output_to_serial)
    {
      Serial.print(j);
      Serial.print(F("\t"));
      Serial.print(dacgoal);
      Serial.print(F("\t"));
      Serial.println(this_voltage_adc); // desired cell voltage
    }

    if (userpause) //will hold the code here until a character is sent over the Serial port
    {
      while (!Serial.available())
        ;
      Serial.read();
    }
    // now fill in statistics:
    sumonx += j;
    sumony += this_voltage_adc;
    sumonxy += j * this_voltage_adc;
    sumonxsquared += j * j;
    sumonysquared += this_voltage_adc * this_voltage_adc;
    n += 1;
  }

  // now I want to fit a line to adc_int_from_dac_int[i] vs i....
  // https://www.statisticshowto.com/probability-and-statistics/regression-analysis/find-a-linear-regression-
equation/
  // y=a+bx
  // y=adc
  // x=dac
  // sumonx=
  // sumony=
  // sumonxy=
  // sumonxsquared=
  // sumonysquared=
  // n=
  // a= ((sumony)*(sumonxsquared)-(sumonx)*(sumonxy))/(n*(sumonxsquared)-(sumonx)*(sumonx))
  // b= (n*(sumonxy) - (sumonx)*(sumony))/(n*(sumonxsquared)-(sumonx)*(sumonx))
  a = ((sumony) * (sumonxsquared) - (sumonx) * (sumonxy)) / (n * (sumonxsquared) - (sumonx) * (sumonx));
  b = (n * (sumonxy) - (sumonx) * (sumony)) / (n * (sumonxsquared) - (sumonx) * (sumonx));
  Serial.println("y=a + bx");
  Serial.println("y=dac");
  Serial.println("y=adc");
  Serial.print("a=");
  Serial.println(a);
  Serial.print("b=");
  Serial.println(b);
  // so adc = a + b * dac
  // dac = bit # (0-255)
  // adc = bit # (0-4095)
  // so for calculation of actual voltage from adc, we can assume the calibration (taking dac as perfect)
  // dacvoltage = (dac/255)*3.3
  // adcvoltage = 0.5*dacvoltage
  // adc = a + b * dac = a + b * dacvoltage * (255/3.3); here dac voltage is assumed correct
  // so inverting, we get dacvoltage = (adc-a)*(3.3/255)*(1/b)
  // dacvoltage = ((3.3/255)*(1/b)) * adc + ((-a/b)*(3.3/255))
  // adcvoltage = ((3.3/255)*(1/2*b)) * adc + ((-a/2*b)*(3.3/255))

  a_coeff = a;
  b_coeff = b;
}

void testLMP91000(uint32_t delayTime_ms, uint8_t bias_setting_local)
{

  // PJB 4/12/2021
  // The purpose of this subroutine is to set the LMP91000 settings over the I2C
  // so that the user can manually apply an input (control)  voltage and
  // manually read the output voltages in a test breadboad.
  // It assumes the user has access to all LMP91000 pins on a breakout board of some kind, and
  // that the ESP32 controls the LMP910000 over the I2C interface.
  // For example, the ESP32 could be an ESP32 Pico Kit board, or any ESP32 board, with I2C
  // wired manually to the LMP91000.
  // P Burke made a breakout board of LMP91000 chips.

  // Parameters the user can set on LMP91000:
  // TIA gain (keep max at 350k feedback resistor)
  // Rload (keep at 10 ohms)
  // Ref source int/ext (keep ext)
  // Int_Z zero 50 20 67% (keep at 50%)
  // Bias_Sign (plus/minus)
  // Bias 1%-24%
  // FET_Short (keep off)
  // Mode (keep at 3-lead)
  bool loop_dac = false;
  float dacgoal;
  //xxxxxxxxxxxxxxxxxxxxxxxxxxx
  pStat.disableFET();
  pStat.setGain(7); // TIA feedback resistor 350k
  pStat.setRLoad(0);
  pStat.setExtRefSource();
  pStat.setIntZ(1);
  pStat.setThreeLead();
  pStat.setBias(0);
  pStat.setNegBias();
  pStat.setBias(bias_setting_local); // sets percentage voltage divider in LMP910000
  // TIA_BIAS[bias_setting] varies from 1% to 22% depending on setting; note initially set to 0%...
  // from LMP91000.h:
  //const double TIA_BIAS[] = {0, 0.01, 0.02, 0.04, 0.06, 0.08, 0.1, 0.12, 0.14,
  //    0.16, 0.18, 0.2, 0.22, 0.24};
  //xxxxxxxxxxxxxxxxxxxxxxxxxxx

  Serial.println("RUNNING TEST LMP91000:");
  Serial.println("TIA_BIAS[bias_setting_local]=");
  Serial.println(TIA_BIAS[bias_setting_local]);

  for (int16_t i = 0; i <= 13; i += 1) //loop over percentage of Vref applied to cell
  {
    pStat.setBias(i);
    Serial.println("****************************************");
    Serial.println("TIA_BIAS[i]=");
    Serial.println(TIA_BIAS[i]);
    if (userpause) //will hold the code here until a character is sent over the Serial port
    {
      while (!Serial.available())
        ;
      Serial.read();
    }
    if (loop_dac)
    {
      for (int16_t j = 0; j <= 255; j += 1)
      {
        dacWrite(dac, j); // sets ESP32 DAC bits
        dacgoal = 3300.0 * j / 255;
        Serial.print(j); // current time in ms
        Serial.print(F("\t"));
        Serial.println(dacgoal); // desired cell voltage
        if (userpause)           //will hold the code here until a character is sent over the Serial port
        {
          while (!Serial.available())
            ;
          Serial.read();
        }
      }
    }
  }
}

void runNPVForward(int16_t startV, int16_t endV, int8_t pulseAmp,
                   uint32_t pulse_width, uint32_t off_time)
{

  //@param      startV:       voltage to start the scan
  //@param      endV:         voltage to stop the scan
  //@param      pulseAmp:     amplitude of square wave
  //@param      pulse_width:  the pulse-width of the excitation voltage
  //@param      off_time:
  //
  //Runs NPV in the forward (oxidation) direction. The bias potential
  //is swept from a more negative voltage to a more positive voltage.
  float i_forward = 0;
  float i_backward = 0;

  for (int16_t j = startV; j <= endV; j += pulseAmp)
  {
    i_forward = biasAndSample(j, pulse_width);
    if (userpause) //will hold the code here until a character is sent over the Serial port
    {
      while (!Serial.available())
        ;
      Serial.read();
    }
    i_backward = biasAndSample(startV, off_time); // xxx comment out to do only forward bias

    saveVoltammogram(j, i_forward - i_backward, false); // this will print voltage, current
    if (print_output_to_serial)
    {
      Serial.println("EOL");
    }
    // Serial.println();
    // if (userpause) //will hold the code here until a character is sent over the Serial port
    // {
    //   while (!Serial.available())
    //     ;
    //   Serial.read();
    // }
  }
}

void runNPVBackward(int16_t startV, int16_t endV, int8_t pulseAmp,
                    uint32_t pulse_width, uint32_t off_time)
{
  //@param      startV:       voltage to start the scan
  //@param      endV:         voltage to stop the scan
  //@param      pulseAmp:     amplitude of square wave
  //@param      pulse_width:  the pulse-width of the excitation voltage
  //@param      off_time:
  //
  //Runs NPV in the reverse (reduction) direction. The bias potential
  //is swept from a more positivie voltage to a more negative voltage.

  float i_forward = 0;
  float i_backward = 0;

  for (int16_t j = startV; j >= endV; j -= pulseAmp)
  {
    i_forward = biasAndSample(j, pulse_width);
    //will hold the code here until a character is sent over the Serial port
    //this ensures the experiment will only run when initiated
    while (!Serial.available())
      ;
    Serial.read();
    i_backward = biasAndSample(startV, off_time);
    saveVoltammogram(j, i_forward - i_backward, false); // this will print voltage, current

    //will hold the code here until a character is sent over the Serial port
    //this ensures the experiment will only run when initiated
    while (!Serial.available())
      ;
    Serial.read();
    saveVoltammogram(j, i_forward - i_backward, true); // this will print voltage, currnt
    // Serial.println();
  }
}

void runNPV(uint8_t lmpGain, int16_t startV, int16_t endV,
            int8_t pulseAmp, uint32_t pulse_width, uint32_t pulse_period,
            uint32_t quietTime, uint8_t range, bool setToZero)
{
  //@param      lmpGain:      gain setting for LMP91000
  //@param      startV:       voltage to start the scan
  //@param      endV:         voltage to stop the scan
  //@param      pulseAmp:     amplitude of square wave
  //@param      pulse_period: the  of the excitation voltage
  //@param      pulse_width:  the pulse-width of the excitation voltage
  //@param      quietTime:    initial wait time before the first pulse
  //@param      range:        the expected range of the measured current
  //@param      setToZero:    Boolean determining whether the bias potential of
  //                          the electrochemical cell should be set to 0 at the
  //                          end of the experiment.
  //
  //Runs the electrochemical technique, Normal Pulse Voltammetry. In this
  //technique the electrochemical cell is biased at increasing superimposed
  //voltages. The current is sampled at the end of each step potential. The
  //potential is returned to the startV at the end of each pulse period.
  //https://www.basinc.com/manuals/EC_epsilon/techniques/Pulse/pulse#normal
  //Reset Arrays
  reset_Voltammogram_arrays(); // sets volt[i], amps[i],time_Voltammaogram[i]=0 all

  arr_cur_index = 0;
  number_of_valid_points_in_volts_amps_array = 0;

  if (pulse_width > pulse_period)
  {
    uint32_t temp = pulse_width;
    pulse_width = pulse_period;
    pulse_period = temp;
  }

  if (print_output_to_serial)
  {

    //Print column headers
    //  String current = "";
    //  if(range == 12) current = "Current(pA)";
    //  else if(range == 9) current = "Current(nA)";
    //  else if(range == 6) current = "Current(uA)";
    //  else if(range == 3) current = "Current(mA)";
    //  else current = "SOME ERROR";

    // TIA_BIAS[bias_setting]
    // dacVout*TIA_BIAS[bias_setting] is the exact voltage we will get

    Serial.println("Column header meanings:");
    Serial.println("FORWARD BIAS:");
    Serial.println("T = time in ms");
    Serial.println("Vc = desired cell voltage in mV (internal variable name = voltage)");
    Serial.println("Vset = set cell voltage in mV (internal variable name = vset = dacVout *
TIA_BIAS[bias_setting])");
    Serial.println("div = percentage scale (internal variable name = TIA_BIAS[bias_setting])");
    Serial.println("Vdac = ESP32 dac voltage in mV (internal variable name = dacVout)");
    Serial.println("adc_bits = TIA output in adc_bits;");
    Serial.println("(V1) Vout = TIA output in mV (internal variable name = v1 = pStat.getVoltage(analogRead(LMP),
opVolt, adcBits);");
    Serial.println("(V2) Vc1 = internal zero in mV also C1 (internal variable name = v2 = dacVout * .5)");
    Serial.println("i_f = forward current in uA (internal variable name = current = (((v1 - v2) / 1000) /
TIA_GAIN[LMPgain - 1]) * pow(10, 6); //scales to uA)");

    Serial.println("REVERSE BIAS:");
    Serial.println("T = time in ms");
    Serial.println("Vc = desired cell voltage in mV (internal variable name = voltage)");
    Serial.println("Vset = set cell voltage in mV (internal variable name = vset = dacVout *
TIA_BIAS[bias_setting])");
```

```
    Serial.println("div = percentage scale (internal variable name = TIA_BIAS[bias_setting])");
    Serial.println("Vdac = ESP32 dac voltage in mV (internal variable name = dacVout)");
    Serial.println("Vout = TIA output in mV (internal variable name = v1 = pStat.getVoltage(analogRead(LMP),
opVolt, adcBits);");
    Serial.println("Vc1 = internal zero in mV also C1 (internal variable name = v2 = dacVout * .5)");
    Serial.println("i_R = reverse current in uA (internal variable name = current = (((v1 - v2) / 1000) /
TIA_GAIN[LMPgain - 1]) * pow(10, 6); //scales to uA)");

    Serial.println("RESPONSE:");
    Serial.println("Vc = desired cell voltage in mV (internal variable name = voltage FORWARD, should be FORWARD-
REVERSE...)");
    Serial.println("I = i_f - i_R in uA (internal variable name = i_forward - i_backward)");
    // Serial.println("xyz = xyz in xyz (internal variable name = xyz)");
    // Serial.println("xyz = xyz in xyz (internal variable name = v)");


Serial.println("T\tVc\tVset\tdiv\tVdac\tadcbits\tVout\tVc1\ti_f\tT\tVc\tVset\tdiv\tVdac\tadcbits\tVout\tVc1\ti_R\tV\tI\

    // Serial.println("T,Vc,Vset,div,Vdac,adcbits,Vout,Vc1,i_f,T,Vc,Vset,div,Vdac,adcbits,Vout,Vc1,i_R,V,I,T");

    //    Serial.println("T\tVc\tVset\tdiv\tVdac\tVout\tVc1\ti_f\tT\tVc\tVset\tdiv\tVdac\tVout\tVc1\ti_R\tVc\tI");
    //  Serial.println("T\tVc\tVdac\tVout\tVc1\ti_f\tT\tVc\tZ\tVout\ti_R\tLMP\tI");
    //  Serial.println("T\tVc\tVout\tVc1\ti_f\tT\tVc\tZ\tVout\ti_R\tLMP\tI");
    //  Serial.println("T\t\tVc\tVout\tVc1\ti_f\tT\tVc\tZ\tVout\ti_R\tLMP\tI");
    //  Serial.println("T(ms)\tVcell(mV)\tZero(mV)\tLMP\ti_f\tT(ms)\tVcell(mV)\tZero(mV)\tLMP\ti_R\tV(mV)\tI");
    //  Serial.println(F("Time(ms)  Cell set voltage (mV)   Zero(mV)    dacVout TIA_BIAS[bias_setting],LMP i.e.
Vout (mV),VC1(mV),i_forward,Time(ms),Cell set voltage (mV),Zero(mV),dacVout,TIA_BIAS[bias_setting],LMP i.e. Vout
(mV),VC1(mV),i_backward,Voltage(mV),Current"));
    //  Serial.println(F("T(ms),Vcell(mV),Zero(mV),dacVout,TIA_BIAS,Vout(mV),VC1(mV),i_f,Time(ms),Cell set voltage
(mV),Zero(mV),dacVout,TIA_BIAS[bias_setting],LMP i.e. Vout (mV),VC1(mV),i_backward,Voltage(mV),Current"));
  }

  initLMP(lmpGain);
  pulseAmp = abs(pulseAmp);
  uint32_t off_time = pulse_period - pulse_width;

  setLMPBias(startV); // -200 mV to start
  setVoltage(startV); // -200 mV to start
  // Serial.println();

  unsigned long time1 = millis();
  while (millis() - time1 < quietTime)
    ;

  if (startV < endV)
  {

    runNPVForward(startV, endV, pulseAmp, pulse_width, off_time);
  }
  else
    runNPVBackward(startV, endV, pulseAmp, pulse_width, off_time);

  arr_cur_index = 0;
  if (setToZero)
    setOutputsToZero();
}

//@param      cycles:     number of times to run the scan
//@param      startV:     voltage to start the scan
//@param      endV:       voltage to stop the scan
//@param      vertex1:    edge of the scan
//                        If vertex1 is greater than starting voltage, we start
//                        the experiment by running CV in the forward
//                        (oxidation) direction.
//@param      vertex2:    edge of the scan
//                        If vertex2 is greater than starting voltage, we start
//                        the experiment by running CV in the reverse
//                         (reduction) direction.
//@param      stepV:      how much to increment the voltage by
//@param      rate:       scanning rate
//                         in the case of CV, scanning rate is in mV/s
//Runs CV in the forward (oxidation) direction first
void runCVForward(uint8_t cycles, int16_t startV, int16_t endV,
                  int16_t vertex1, int16_t vertex2, int16_t stepV, uint16_t rate)
{
  int16_t j = startV;
  float i_cv = 0;
  number_of_valid_points_in_volts_amps_array = 0;
  arr_cur_index = 0;

  for (uint8_t i = 0; i < cycles; i++)
  {
    //j starts at startV
    //    for (j;j <= vertex1; j += stepV)
    for (; j <= vertex1; j += stepV)
    {
      i_cv = biasAndSample(j, rate);
      // Serial.print(i + 1);
      // Serial.print(F(","));
      saveVoltammogram(j, i_cv, false);
      if (print_output_to_serial)
      {
        Serial.println("EOL");
      }
      // Serial.println();
    }
    j -= 2 * stepV; //increment j twice to avoid biasing at the vertex twice

    //j starts right below the first vertex
    //    for (j; j >= vertex2; j -= stepV)
    for (; j >= vertex2; j -= stepV)
    {
      i_cv = biasAndSample(j, rate);
      // Serial.print(i + 1);
      // Serial.print(F(","));
      saveVoltammogram(j, i_cv, false);
      if (print_output_to_serial)
      {
        Serial.println("EOL");
      }
      // Serial.println();
    }
    j += 2 * stepV; //increment j twice to avoid biasing at the vertex twice

    //j starts right above the second vertex
    //for (j; j <= endV; j += stepV)
    for (; j <= endV; j += stepV)
    {
      i_cv = biasAndSample(j, rate);
      // Serial.print(i + 1);
      // Serial.print(F(","));
      saveVoltammogram(j, i_cv, false);
      if (print_output_to_serial)
      {
        Serial.println("EOL");
      }
      // Serial.println();
    }
    j -= 2 * stepV; //increment j twice to avoid biasing at the vertex twice
  }
}

//@param      cycles:     number of times to run the scan
//@param      startV:     voltage to start the scan
```

```
//@param      endV:       voltage to stop the scan
//@param      vertex1:    edge of the scan
//                        If vertex1 is greater than starting voltage, we start
//                        the experiment by running CV in the forward
//                        (oxidation) direction.
//@param      vertex2:    edge of the scan
//                        If vertex2 is greater than starting voltage, we start
//                        the experiment by running CV in the reverse
//                        (reduction) direction.
//@param      stepV:      how much to increment the voltage by
//@param      rate:       scanning rate
//                        in the case of CV, scanning rate is in mV/s
//Runs CV in the reverse (reduction) direction first
void runCVBackward(uint8_t cycles, int16_t startV, int16_t endV,
                   int16_t vertex1, int16_t vertex2, int16_t stepV, uint16_t rate)
{
  int16_t j = startV;
  float i_cv = 0;
  number_of_valid_points_in_volts_amps_array = 0;
  arr_cur_index = 0;

  for (uint8_t i = 0; i < cycles; i++)
  {
    //j starts at startV
    //    for (j; j >= vertex1; j -= stepV)
    for (; j >= vertex1; j -= stepV)
    {
      i_cv = biasAndSample(j, rate);
      // Serial.print(i + 1);
      // Serial.print(F(","));
      saveVoltammogram(j, i_cv, false);
      if (print_output_to_serial)
      {
        Serial.println("EOL");
      }
      // Serial.println();
    }
    j += 2 * stepV; //increment j twice to avoid biasing at the vertex twice

    //j starts right above vertex1
    //    for (j; j <= vertex2; j += stepV)
    for (; j <= vertex2; j += stepV)
    {
      i_cv = biasAndSample(j, rate);
      // Serial.print(i + 1);
      // Serial.print(F(","));
      saveVoltammogram(j, i_cv, false);
      if (print_output_to_serial)
      {
        Serial.println("EOL");
      }
      // Serial.println();
    }
    j -= 2 * stepV; //increment j twice to avoid biasing at the vertex twice

    //j starts right below vertex2
    //for (j; j >= endV; j -= stepV)
    for (; j >= endV; j -= stepV)
    {
      i_cv = biasAndSample(j, rate);
      // Serial.print(i + 1);
      // Serial.print(F(","));
      saveVoltammogram(j, i_cv, false);
      if (print_output_to_serial)
      {
        Serial.println("EOL");
      }
      // Serial.println();
    }
    j += 2 * stepV; //increment j twice to avoid biasing at the vertex twice
  }
}

//void runCV()
//
//@param      lmpGain:    gain setting for LMP91000
//@param      cycles:     number of times to run the scan
//@param      startV:     voltage to start the scan
//@param      endV:       voltage to stop the scan
//@param      vertex1:    edge of the scan
//                        If vertex1 is greater than starting voltage, we start
//                        the experiment by running CV in the forward
//                        (oxidation) direction.
//@param      vertex2:    edge of the scan
//                        If vertex2 is greater than starting voltage, we start
//                        the experiment by running CV in the reverse
//                        (reduction) direction.
//@param      stepV:      how much to increment the voltage by
//@param      rate:       scanning rate
//                        in the case of CV, scanning rate is in mV/s
//@param      setToZero:  Boolean determining whether the bias potential of
//                        the electrochemical cell should be set to 0 at the
//                        end of the experiment.
//
//Runs the electrochemical technique, Cyclic Voltammetry. In this
//technique the electrochemical cell is biased at a series of
//voltages and the current at each subsequent voltage is measured.
void runCV(uint8_t lmpGain, uint8_t cycles, int16_t startV,
           int16_t endV, int16_t vertex1, int16_t vertex2,
           int16_t stepV, uint16_t rate, bool setToZero)
{
  if (print_output_to_serial)
  {

    Serial.println(F("Time(ms),Zero,LMP,Current,Cycle,Voltage,Current"));
  }

  initLMP(lmpGain);
  //the method automatically handles if stepV needs to be positive or negative
  //no need for the user to specify one or the other
  //this step deals with that in case the user doesn't know
  stepV = abs(stepV);
  //figures out the delay needed to achieve a given scan rate
  //delay is dependent on desired rate and number of steps taken
  //more steps = smaller delay since we'll need to go by a bit faster
  //to sample at more steps vs. less steps, but at the same rate
  rate = (1000.0 * stepV) / rate;
  // e.g. rate = 100 mV/s step = 5 mV
  // so 100/5 steps per second or 5/100 seconds between steps=50 msec
  // so new "rate" is delay in ms between points (assuming read is instantaneous)

  //Reset Arrays
  reset_Voltammogram_arrays(); // sets volt[i], amps[i],time_Voltammaogram[i]=0 all

  lastTime = millis();
  if (vertex1 > startV)
    runCVForward(cycles, startV, endV, vertex1, vertex2, stepV, rate);
  else
    runCVBackward(cycles, startV, endV, vertex1, vertex2, stepV, rate);

  //sets the bias of the electrochemical cell to 0V
  if (setToZero)
    setOutputsToZero();
  arr_cur_index = 0;
}

//void runSWVForward
//
//@param      startV:     voltage to start the scan
//@param      endV:       voltage to stop the scan
//@param      pulseAmp:   amplitude of square wave
//@param      stepV:      how much to increment the voltage by
//@param      freq:       square wave frequency
//
//Runs SWV in the forward (oxidation) direction. The bias potential is
//swept from a more negative voltage to a more positive voltage.
void runSWVForward(int16_t startV, int16_t endV, int16_t pulseAmp,
                   int16_t stepV, double freq)
{
  float i_forward = 0;
  float i_backward = 0;

  for (int16_t j = startV; j <= endV; j += stepV)
  {
    //positive pulse
    i_forward = biasAndSample(j + pulseAmp, freq);

    //negative pulse
    i_backward = biasAndSample(j - pulseAmp, freq);
    saveVoltammogram(j, i_forward - i_backward, false);
    if (print_output_to_serial)
    {
      Serial.println("EOL");
    }
    // Serial.println();
  }
}

//void runSWVBackward
//
//@param      startV:     voltage to start the scan
//@param      endV:       voltage to stop the scan
//@param      pulseAmp:   amplitude of square wave
//@param      stepV:      how much to increment the voltage by
//@param      freq:       square wave frequency
//
//Runs SWV in the backward (reduction) direction. The bias potential
//is swept from a more positivie voltage to a more negative voltage.
void runSWVBackward(int16_t startV, int16_t endV, int16_t pulseAmp,
                    int16_t stepV, double freq)
{
  float i_forward = 0;
  float i_backward = 0;

  for (int16_t j = startV; j >= endV; j -= stepV)
  {
    //positive pulse
    i_forward = biasAndSample(j + pulseAmp, freq);

    //negative pulse
    i_backward = biasAndSample(j - pulseAmp, freq);
    number_of_valid_points_in_volts_amps_array++;
    saveVoltammogram(j, i_forward - i_backward, false);
    if (print_output_to_serial)
    {
      Serial.println("EOL");
    }
    // Serial.println();
  }
}
```

```
//void runSWV()
//
//@param      lmpGain:    gain setting for LMP91000
//@param      startV:     voltage to start the scan
//@param      endV:       voltage to stop the scan
//@param      pulseAmp:   amplitude of square wave
//@param      stepV:      how much to increment the voltage by
//@param      freq:       square wave frequency
//@param      setToZero:  Boolean determining whether the bias potential of
//                        the electrochemical cell should be set to 0 at the
//                        end of the experiment.
//
//Runs the electrochemical technique, Square Wave Voltammetry. In this
//technique the electrochemical cell is biased at a series of
//voltages with a superimposed squar wave on top of the bias voltage.
//The current is sampled on the forward and the reverse pulse.
//https://www.basinc.com/manuals/EC_epsilon/Techniques/Pulse/pulse#square
//
void runSWV(uint8_t lmpGain, int16_t startV, int16_t endV,
            int16_t pulseAmp, int16_t stepV, double freq, bool setToZero)
{
  // Serial.println(F("Time(ms),Zero,LMP,Time(ms),Zero,LMP,Voltage,Current"));

  initLMP(lmpGain);
  stepV = abs(stepV);
  pulseAmp = abs(pulseAmp);
  freq = (uint16_t)(1000.0 / (2 * freq)); //converts frequency to milliseconds

  //Reset Arrays
  reset_Voltammogram_arrays(); // sets volt[i], amps[i],time_Voltammaogram[i]=0 all

  arr_cur_index = 0;
  number_of_valid_points_in_volts_amps_array = 0;

  //testing shows that time is usually off by 1 ms or so, therefore
  //we subtract 1 ms from the calculated rate to compensate
  freq -= 1;

  if (print_output_to_serial)
  {

    //Print column headers
    //  String current = "";
    //  if(range == 12) current = "Current(pA)";
    //  else if(range == 9) current = "Current(nA)";
    //  else if(range == 6) current = "Current(uA)";
    //  else if(range == 3) current = "Current(mA)";
    //  else current = "SOME ERROR";

    // TIA_BIAS[bias_setting]
    // dacVout*TIA_BIAS[bias_setting] is the exact voltage we will get

    Serial.println("Column header meanings:");
    Serial.println("FORWARD BIAS:");
    Serial.println("T = time in ms");
    Serial.println("Vc = desired cell voltage in mV (internal variable name = voltage)");
    Serial.println("Vset = set cell voltage in mV (internal variable name = vset = dacVout *
TIA_BIAS[bias_setting])");
    Serial.println("div = percentage scale (internal variable name = TIA_BIAS[bias_setting])");
    Serial.println("Vdac = ESP32 dac voltage in mV (internal variable name = dacVout)");
    Serial.println("adc_bits = TIA output in adc_bits;");
    Serial.println("(V1) Vout = TIA output in mV (internal variable name = v1 = pStat.getVoltage(analogRead(LMP),
opVolt, adcBits);");
    Serial.println("(V2) Vc1 = internal zero in mV also C1 (internal variable name = v2 = dacVout * .5)");
    Serial.println("i_f = forward current in uA (internal variable name = current = (((v1 - v2) / 1000) /
TIA_GAIN[LMPgain - 1]) * pow(10, 6); //scales to uA)");
```

```
    Serial.println("REVERSE BIAS:");
    Serial.println("T = time in ms");
    Serial.println("Vc = desired cell voltage in mV (internal variable name = voltage)");
    Serial.println("Vset = set cell voltage in mV (internal variable name = vset = dacVout *
TIA_BIAS[bias_setting])");
    Serial.println("div = percentage scale (internal variable name = TIA_BIAS[bias_setting])");
    Serial.println("Vdac = ESP32 dac voltage in mV (internal variable name = dacVout)");
    Serial.println("Vout = TIA output in mV (internal variable name = v1 = pStat.getVoltage(analogRead(LMP),
opVolt, adcBits);");
    Serial.println("Vc1 = internal zero in mV also C1 (internal variable name = v2 = dacVout * .5)");
    Serial.println("i_R = reverse current in uA (internal variable name = current = (((v1 - v2) / 1000) /
TIA_GAIN[LMPgain - 1]) * pow(10, 6); //scales to uA)");

    Serial.println("RESPONSE:");
    Serial.println("Vc = desired cell voltage in mV (internal variable name = voltage FORWARD, should be FORWARD-
REVERSE...)");
    Serial.println("I = i_f - i_R in uA (internal variable name = i_forward - i_backward)");
    Serial.println("xyz = xyz in xyz (internal variable name = xyz)");
    Serial.println("xyz = xyz in xyz (internal variable name = v)");

    // Serial.println("T,Vc,Vset,div,Vdac,adcbits,Vout,Vc1,i_f,T,Vc,Vset,div,Vdac,adcbits,Vout,Vc1,i_R,V,I,T");

Serial.println("T\tVc\tVset\tdiv\tVdac\tadcbits\tVout\tVc1\ti_f\tT\tVc\tVset\tdiv\tVdac\tadcbits\tVout\tVc1\ti_R\tV\tI\

    //    Serial.println("T\tVc\tVset\tdiv\tVdac\tVout\tVc1\ti_f\tT\tVc\tVset\tdiv\tVdac\tVout\tVc1\ti_R\tVc\tI");
    //  Serial.println("T\tVc\tVdac\tVout\tVc1\ti_f\tT\tVc\tZ\tVout\ti_R\tLMP\tI");
    //  Serial.println("T\tVc\tVout\tVc1\ti_f\tT\tVc\tZ\tVout\ti_R\tLMP\tI");
    //  Serial.println("T\t\tVc\tVout\tVc1\ti_f\tT\tVc\tZ\tVout\ti_R\tLMP\tI");
    //  Serial.println("T(ms)\tVcell(mV)\tZero(mV)\tLMP\ti_f\tT(ms)\tVcell(mV)\tZero(mV)\tLMP\ti_R\tV(mV)\tI");
    //  Serial.println(F("Time(ms),Cell set voltage (mV),Zero(mV),dacVout,TIA_BIAS[bias_setting],LMP i.e. Vout
(mV),VC1(mV),i_forward,Time(ms),Cell set voltage (mV),Zero(mV),dacVout,TIA_BIAS[bias_setting],LMP i.e. Vout
(mV),VC1(mV),i_backward,Voltage(mV),Current"));
    //  Serial.println(F("T(ms),Vcell(mV),Zero(mV),dacVout,TIA_BIAS,Vout(mV),VC1(mV),i_f,Time(ms),Cell set voltage
(mV),Zero(mV),dacVout,TIA_BIAS[bias_setting],LMP i.e. Vout (mV),VC1(mV),i_backward,Voltage(mV),Current"));
  }

  //Reset Arrays
  reset_Voltammogram_arrays(); // sets volt[i], amps[i],time_Voltammaogram[i]=0 all

  number_of_valid_points_in_volts_amps_array = 0;
  arr_cur_index = 0;

  if (startV < endV)
    runSWVForward(startV, endV, pulseAmp, stepV, freq);
  else
    runSWVBackward(startV, endV, pulseAmp, stepV, freq);

  arr_cur_index = 0;
  if (setToZero)
    setOutputsToZero();
}

void runDPVForward(int16_t startV, int16_t endV, int8_t pulseAmp,
                   int8_t stepV, uint32_t pulse_width, uint32_t off_time)
{
  float i_forward = 0;
  float i_backward = 0;

  for (int16_t j = startV; j <= endV; j += stepV)
  {
    //baseline
    i_backward = biasAndSample(j, off_time);

    //pulse
    i_forward = biasAndSample(j + pulseAmp, pulse_width);
    number_of_valid_points_in_volts_amps_array++;
    saveVoltammogram(j, i_forward - i_backward, false);
    // Serial.println();
  }
}

void runDPVBackward(int16_t startV, int16_t endV, int8_t pulseAmp,
                    int8_t stepV, uint32_t pulse_width, uint32_t off_time)
{
  float i_forward = 0;
  float i_backward = 0;

  for (int16_t j = startV; j >= endV; j -= stepV)
  {
    //baseline
    i_backward = biasAndSample(j, off_time);

    //pulse
    i_forward = biasAndSample(j + pulseAmp, pulse_width);
    number_of_valid_points_in_volts_amps_array++;

    saveVoltammogram(j, i_forward - i_backward, false);
    // Serial.println();
  }
}

void runDPV(uint8_t lmpGain, int16_t startV, int16_t endV,
            int8_t pulseAmp, int8_t stepV, uint32_t pulse_period,
            uint32_t pulse_width, bool setToZero)
{
  //Serial.println("Time(ms),Zero(mV),LMP,Voltage(mV)," + current);
  if (print_output_to_serial)
  {
    Serial.println(F("Time(ms),Zero(mV),LMP,Voltage(mV),Current"));
  }

  initLMP(lmpGain);
  stepV = abs(stepV);
  pulseAmp = abs(pulseAmp);
  uint32_t off_time = pulse_period - pulse_width;

  //Reset Arrays
  reset_Voltammogram_arrays(); // sets volt[i], amps[i],time_Voltammaogram[i]=0 all

  if (startV < endV)
    runDPVForward(startV, endV, pulseAmp, stepV, pulse_width, off_time);
  else
    runDPVBackward(startV, endV, pulseAmp, stepV, pulse_width, off_time);

  arr_cur_index = 0;
  if (setToZero)
    setOutputsToZero();
}

//void runAmp()
//
//@param      lmpGain:    gain setting for LMP91000
//@param      pre_stepV:  voltage to start the scan
//@param      quietTime:  initial wait time before the first pulse
//
//@param      v1:         the first step potential
//@param      t1:         how long we hold the cell at the first potential
//@param      v2:         the second step potential
//@param      t2:         how long we hold the cell at the second potential
//@param      samples:    how many times we sample the current at each potential
//@param      range:      the expected range of the measured current
//                          range = 12 is picoamperes
//                          range = 9 is nanoamperes
//                          range = 6 is microamperes
//                          range = 3 is milliamperes
//@param      setToZero:  Boolean determining whether the bias potential of
//                        the electrochemical cell should be set to 0 at the
//                        end of the experiment.
//
//Runs Amperometry technique. In Amperometry, the voltage is biased
//and held at one or maybe two potentials and the current is sampled
//while the current is held steady.
//https://www.basinc.com/manuals/EC_epsilon/Techniques/ChronoI/ca
//
void runAmp(uint8_t lmpGain, int16_t pre_stepV, uint32_t quietTime,
            int16_t v1, uint32_t t1, int16_t v2, uint32_t t2,
            uint16_t samples, uint8_t range, bool setToZero)
{
  //Print column headers
  //  String current = "";
  //  if (range == 12)
  //    current = "Current(pA)";
  //  else if (range == 9)
  //    current = "Current(nA)";
  //  else if (range == 6)
  //    current = "Current(uA)";
  //  else if (range == 3)
  //    current = "Current(mA)";
  //  else
  //    current = "SOME ERROR";
  digitalWrite(LEDPIN, HIGH); // just leave it on during sweep
  initLMP(lmpGain);

  reset_Voltammogram_arrays(); // Reset Arrays volt[i], amps[i],time_Voltammaogram[i]=0 all
  arr_cur_index = 0;
  number_of_valid_points_in_volts_amps_array = 0;

  float a = a_coeff; //    //  Calibrate coefficients (make a local copy of the global ones):
  float b = b_coeff; //    //  Calibrate coefficients (make a local copy of the global ones):

  // default samples=100
  //  if (samples > arr_samples / 3){ // max 2500/3=833; default 100
  if (samples > (arr_samples/3)) //  arr_samples is max size of amps,volts array
  { // max 2500/3=833; default 100
    // samples = (float)arr_samples / 3.0;
    samples = arr_samples/3;
    Serial.print("Samples>arr_samples, adjusting accordingly to samples = ");
    Serial.println(samples);
  }

  int16_t voltageArray[3] = {pre_stepV, v1, v2}; // default 50, 50, 50
  uint32_t timeArray[3] = {quietTime, t1, t2};   // default 2000,2000,2000

  //i = 0 is pre-step voltage
  //i = 1 is first step potential
  //i = 2 is second step potential
  for (uint8_t i = 0; i < 3; i++)
  {
    //the time between samples is determined by the
    //number of samples inputted by the user
    uint32_t fs = (double)timeArray[i] / samples; // default 2000/100=20
    unsigned long startTime = millis();

    Serial.print("samples=");
    Serial.println(samples);
    Serial.print("fs=");
    Serial.println(fs);

    //Print column headers
    if (print_output_to_serial)
    {
      Serial.println("Voltage(mV),Time(ms), Current(microA)");
    }

    //set bias potential
    setLMPBias(voltageArray[i]);
    setVoltage(voltageArray[i]);

    while (millis() - startTime < timeArray[i]) // default 2000, so for 2 seconds
    {
      // Serial.print("arr_cur_index = ");
      // Serial.print(arr_cur_index);
      // Serial.print(F("\t")); // tab
      // Serial.print("millis = ");
      // Serial.print(millis());
      // Serial.print(F("\t")); // tab
      // Serial.print("startTime = ");
      // Serial.print(startTime);
      // Serial.print(F("\t")); // tab
      // Serial.print("timeArray[i] = ");
      // Serial.print(timeArray[i]);
      // Serial.println(F("\t")); // tab

      // Read output voltage of the transimpedance amplifier
      int adc_bits;                                                 // will be output voltage of TIA amplifier,
"VOUT" on LMP91000 diagram, also C2
      adc_bits = analog_read_avg(num_adc_readings_to_average, LMP); //  hard wired in BurkeLab ESP32Stat Rev 3.5
to LMP i.e ESP32 pin 32 (ADC1_CH7)
      float v1;
      v1 = (3.3 / 255.0) * (1 / (2.0 * b)) * (float)adc_bits - (a / (2.0 * b)) * (3.3 / 255.0); // LMP is wired to
Vout of the LMP91000
      v1 = v1 * 1000.0;
      float v2 = dacVout * .5; //the zero of the internal transimpedance amplifier
      // V2 is not measured in  BurkeLab ESP32Stat Rev 3.5 and assumed to be half dacVout, calibration helps this
see above
      float current = 0;
      if (lmpGain == 0)                                   // using external feedback resistor, not (yet) suppored
with BurkeLab ESP32Stat Rev 3.5
        current = (((v1 - v2) / 1000) / RFB) * pow(10, 9); //scales to nA
      else
        current = (((v1 - v2) / 1000) / TIA_GAIN[lmpGain - 1]) * pow(10, 6); //scales to uA
                                                                              //Sample and save data
      volts[arr_cur_index] = voltageArray[i];
      time_Voltammaogram[arr_cur_index] = millis();
      amps[arr_cur_index] = current;
      number_of_valid_points_in_volts_amps_array++;
      if (print_output_to_serial)
      // if (false)
      {
        Serial.println("************BEGIN CA POINT:*******************");
        // Serial.print("adc_bits= ");
        // Serial.print(adc_bits);
        // Serial.print(F("\t")); // tab
        // Serial.print("v1= ");
        // Serial.print(v1);
        // Serial.print(F("\t")); // tab
        // Serial.print("dacVout= ");
        // Serial.print(dacVout);
        // Serial.print(F("\t")); // tab
        // Serial.print("v2= ");
        // Serial.print(v2);
        // Serial.print(F("\t")); // tab
        // Serial.print("current= ");
        // Serial.print(current);
        // Serial.print(F("\t")); // tab
        // Serial.print("lmpGain= ");
        // Serial.print(lmpGain);
        // Serial.print(F("\t")); // tab
        Serial.print("arr_cur_index = ");
        Serial.print(arr_cur_index);
        Serial.print(F("\t")); // tab
        Serial.print("Volts = ");
        Serial.print(volts[arr_cur_index]);
        Serial.print(F("\t")); // tab
        Serial.print("Amps = ");
        Serial.print(amps[arr_cur_index]);
        Serial.print(F("\t")); // tab
        Serial.print("Time = ");
        Serial.print(time_Voltammaogram[arr_cur_index]);
        // Serial.print(F("\t")); // tab
        // Serial.print("TIA_BIAS[bias_setting] = ");
        // Serial.print(TIA_BIAS[bias_setting]);
        // Serial.print(F("\t")); // tab
        // Serial.println();
        // Serial.print("TIA_GAIN[lmpGain - 1] = ");
        // Serial.print(TIA_GAIN[lmpGain - 1]);
        // Serial.print(F("\t")); // tab
        Serial.println();

        Serial.println("**************END CA POINT*****************");
      }

      // if (print_output_to_serial)
      if (false)
      {
        Serial.print("Volts = ");
        Serial.print(volts[arr_cur_index]);
        Serial.print(F("\t")); // tab
        Serial.print("Amps = ");
        Serial.print(amps[arr_cur_index]);
        Serial.print(F("\t")); // tab
        Serial.print("Time = ");
        Serial.print(time_Voltammaogram[arr_cur_index]);
        Serial.print(F("\t")); // tab
        Serial.println();
      }

      arr_cur_index++;
      delay(fs);
    }
    digitalWrite(LEDPIN, HIGH); // off at end of sweep
  }

  arr_cur_index = 0;
  if (setToZero)
    setOutputsToZero();
}

void runNPVandPrintToSerial()
// runNPVandPrintToSerial
{

  //##############################NORMAL PULSE VOLTAMMETRY##############################
  LMPgainGLOBAL = 7;
  runNPV(LMPgainGLOBAL, -200, 500, 10, 50, 200, 500, 6, true);
  // Run Normal Pulse Voltammetry with gain 350000, -200 to + 500 mV, 10 mV step, 50 microsecond width,
  //  200 microsecond period, 500 microsecond quiet time, range micro amps) // micro or milli???

  Serial.println(F("Voltage,Current")); // the final array
  for (uint16_t i = 0; i < arr_samples; i++)
  {
    Serial.print(volts[i]);
    Serial.print(F("\t"));
    Serial.println(amps[i]);
  }
}

void runCVandPrintToSerial()
// runCVandPrintToSerial
{
  //##############################CYCLIC VOLTAMMETRY##############################

  //  //void runCV(uint8_t lmpGain, uint8_t cycles, int16_t startV,
  //  //           int16_t endV, int16_t vertex1, int16_t vertex2,
  //  //           int16_t stepV, uint16_t rate, bool setToZero)
  //
  // runCV(LMPgain, 1, -100, 100, -100, 100, 1, 50, true);

  Serial.println(F("Voltage,Current")); // the final array
  for (uint16_t i = 0; i < arr_samples; i++)
  {
    Serial.print(volts[i]);
    Serial.print(F("\t"));
    Serial.println(amps[i]);
  }
}

void runSWVForwardandPrintToSerial()
// runSWVandPrintToSerial (forward)
{
  //  ##############################SQUARE WAVE VOLTAMMETRY (Forward -- Oxidation)##############################

  //  runSWV(LMPgain, -400, 500, 50, 1, 31.25, true);

  Serial.println(F("Voltage,Current"));
  for (uint16_t i = 0; i < arr_samples; i++)
  {
    Serial.print(volts[i]);
    Serial.print(F(","));
    Serial.println(amps[i]);
  }
}

void runSWVReverseandPrintToSerial()
// runCAandPrintToSerial (reverse)
{

  //  ##############################SQUARE WAVE VOLTAMMETRY (Reverse -- Reduction)##############################

  // runSWV(LMPgain, -30, -500, 50, 66, 62.5, true);
  Serial.println(F("Voltage,Current"));
  for (uint16_t i = 0; i < arr_samples; i++)
  {
    Serial.print(volts[i]);
    Serial.print(F(","));
    Serial.println(amps[i]);
  }
}
void runCAandPrintToSerial()
// LEGACY FROM TESTING DO NOT USE
// runCAandPrintToSerial
{

  //  ##############################CHRONOAMPEROMETRY##############################

  // runAmp(LMPgain, 174, 40000, -200, 40000, 200, 40000, 400, 6, true);
  for (uint16_t i = 0; i < arr_samples; i++)
  {
    Serial.print(volts[i]);
    Serial.print(F(","));
    Serial.println(amps[i]);
  }
  Serial.println("Quiet time till next Amp run");
}

void set_sweep_parameters_from_form_input(String form_id, String form_value)
{

  // else if(form_id=="xyz"){
  //   xyz=form_value.toInt();
  // }

  if (form_id == "sweep_param_lmpGain")
  {
    sweep_param_lmpGain = form_value.toInt();
  }

  else if (form_id == "sweep_param_startV_CV")
  {
    sweep_param_startV_CV = form_value.toInt();
  }

  else if (form_id == "sweep_param_endV_CV")
  {
    sweep_param_endV_CV = form_value.toInt();
  }

  else if (form_id == "sweep_param_vertex1_CV")
  {
    sweep_param_vertex1_CV = form_value.toInt();
  }

  else if (form_id == "sweep_param_vertex2_CV")
  {
    sweep_param_vertex2_CV = form_value.toInt();
  }

  else if (form_id == "sweep_param_stepV_CV")
  {
    sweep_param_stepV_CV = form_value.toInt();
  }

  else if (form_id == "sweep_param_rate_CV")
  {
    sweep_param_rate_CV = form_value.toInt();
  }

  else if (form_id == "sweep_param_startV_NPV")
  {
    sweep_param_startV_NPV = form_value.toInt();
  }

  else if (form_id == "sweep_param_endV_NPV")
  {
    sweep_param_endV_NPV = form_value.toInt();
  }

  else if (form_id == "sweep_param_pulseAmp_NPV")
  {
    sweep_param_pulseAmp_NPV = form_value.toInt();
  }

  else if (form_id == "sweep_param_width_NPV")
  {
    sweep_param_width_NPV = form_value.toInt();
  }
  else if (form_id == "sweep_param_period_NPV")
  {
    sweep_param_period_NPV = form_value.toInt();
  }

  else if (form_id == "sweep_param_quietTime_NPV")
  {
    sweep_param_quietTime_NPV = form_value.toInt();
  }

  else if (form_id == "sweep_param_startV_SWV")
  {
    sweep_param_startV_SWV = form_value.toInt();
  }

  else if (form_id == "sweep_param_endV_SWV")
  {
    sweep_param_endV_SWV = form_value.toInt();
  }

  else if (form_id == "sweep_param_pulseAmp_SWV")
  {
    sweep_param_pulseAmp_SWV = form_value.toInt();
  }

  else if (form_id == "sweep_param_stepV_SWV")
  {
    sweep_param_stepV_SWV = form_value.toInt();
  }

  else if (form_id == "sweep_param_freq_SWV")
  {
    sweep_param_freq_SWV = form_value.toInt();
  }

  else if (form_id == "sweep_param_pre_stepV_CA")
  {
    sweep_param_pre_stepV_CA = form_value.toInt();
  }

  else if (form_id == "sweep_param_quietTime_CA")
  {
    sweep_param_quietTime_CA = form_value.toInt();
  }

  else if (form_id == "sweep_param_V1_CA")
  {
    sweep_param_V1_CA = form_value.toInt();
  }

  else if (form_id == "sweep_param_t1_CA")
  {
    sweep_param_t1_CA = form_value.toInt();
  }

  else if (form_id == "sweep_param_V2_CA")
  {
    sweep_param_V2_CA = form_value.toInt();
  }

  else if (form_id == "sweep_param_t2_CA")
  {
    sweep_param_t2_CA = form_value.toInt();
  }

  else if (form_id == "sweep_param_samples_CA")
  {
    sweep_param_samples_CA = form_value.toInt();
  }

  else if (form_id == "sweep_param_biasV_noisetest")
  {
    sweep_param_biasV_noisetest = form_value.toInt();
  }

  else if (form_id == "sweep_param_delayTime_ms_noisetest")
  {
    sweep_param_delayTime_ms_noisetest = form_value.toInt();
  }

  else if (form_id == "sweep_param_startV_IV")
  {
    sweep_param_startV_IV = form_value.toInt();
  }

  else if (form_id == "sweep_param_endV_IV")
  {
    sweep_param_endV_IV = form_value.toInt();
  }

  else if (form_id == "sweep_param_numPoints_IV")
  {
    sweep_param_numPoints_IV = form_value.toInt();
  }

  else if (form_id == "sweep_param_delayTime_ms_IV")
  {
    sweep_param_delayTime_ms_IV = form_value.toInt();
  }

  else if (form_id == "sweep_param_delayTime_ms_CAL")
  {
    sweep_param_delayTime_ms_CAL = form_value.toInt();
  }

  // else if(form_id=="xyz"){
  //   xyz=form_value.toInt();
  // }

  else if (form_id == "sweep_param_cycles_CV")
  {
    sweep_param_cycles_CV = form_value.toInt();
  }

  else if (form_id == "sweep_param_setToZero")
  {
    if (form_value == "true")
    {
      sweep_param_setToZero = true;
    }
    else if (form_value != "true")
    {
      sweep_param_setToZero = false;
    }
  }
}

void handleFileDelete(AsyncWebServerRequest *request)
{
  Serial.println("in file delete");
  if (request->params() == 0)
  {
    return request->send(500, "text/plain", "BAD ARGS");
  }
  AsyncWebParameter *p = request->getParam(0);
  String path = p->value();
  Serial.println("handleFileDelete: " + path);
  if (path == "/")
  {
    return request->send(500, "text/plain", "BAD PATH");
  }

  if (!SPIFFS.exists(path))
  {
    return request->send(404, "text/plain", "FileNotFound");
  }

  SPIFFS.remove(path);
  request->send(200, "text/plain", "");
  path = String();
}

void handle_websocket_text(uint8_t *payload)
{
  // do something...
  Serial.printf("handle_websocket_text called for: %s\n", payload);

  // test JSON parsing...
  //char m_JSONMessage[] = "{\"Key1\":123,\"Key2\",345}";
  //StaticJsonDocument<1000> m_JSONdoc;
  //deserializeJson(m_JSONdoc, m_JSONMessage); // m_JSONdoc is now a json object
  //int m_key1_value = m_JSONdoc["Key1"];
  // Serial.println(m_key1_value);

  // Parse JSON payload
  StaticJsonDocument<1000> m_JSONdoc_from_payload;
  DeserializationError m_error = deserializeJson(m_JSONdoc_from_payload, payload); // m_JSONdoc is now a json
object
  if (m_error)
  {
    Serial.println("deserializeJson() failed with code ");
    Serial.println(m_error.c_str());
  }
  // Serial.println(m_key2_value);
  // now to iterate over (unknown) keys, we have to cast the StaticJsonDocument object into a JsonObject:
  // see https://techtutorialsx.com/2019/07/09/esp32-arduinojson-printing-the-keys-of-the-jsondocument/
  JsonObject m_JsonObject_from_payload = m_JSONdoc_from_payload.as<JsonObject>();
  // Iterate and print to serial:
  //   uint8_t LMPgain_control_panel = 6; // Feedback resistor of TIA.
  // int num_adc_readings_to_average_control_panel = 1;
  // int sweep_param_delayTime_ms_control_panel = 50;
  // int cell_voltage_control_panel = 100;
  for (JsonPair keyValue : m_JsonObject_from_payload)
  {
    String m_key_string = keyValue.key().c_str();

    if (m_key_string == "change_cell_voltage_to")
    {
      Serial.println("change_cell_voltage_to called");
      int m_new_cell_voltage = m_JSONdoc_from_payload["change_cell_voltage_to"];
      Serial.println(m_new_cell_voltage);
      cell_voltage_control_panel = m_new_cell_voltage;
      if (Sweep_Mode == CTLPANEL)
      {
        // set cell voltage, send params back to browswer such as percentage and dac settings
        Serial.println("Calling setLMPBias and setVoltage to:");
        Serial.println(cell_voltage_control_panel);
        setLMPBias(cell_voltage_control_panel);
        setVoltage(cell_voltage_control_panel);
        // send percent setting, dacvout back to browswer xyzxyz
        temp_json_string = "{\"dacVout\":";
        temp_json_string += String(dacVout, DEC);
        temp_json_string += ",\"percentage\":";
        temp_json_string += String(TIA_BIAS[bias_setting]);
        temp_json_string += "}";
        m_websocketserver.broadcastTXT(temp_json_string.c_str(), temp_json_string.length());
      }
    }
    if (m_key_string == "change_num_readings_to_average_per_point_to")
    {
      Serial.println("change_num_readings_to_average_per_point_to called");
      int m_new_num_readings_to_average_per_point =
m_JSONdoc_from_payload["change_num_readings_to_average_per_point_to"];
      Serial.println(m_new_num_readings_to_average_per_point);
      num_adc_readings_to_average_control_panel = m_new_num_readings_to_average_per_point;
    }
    if (m_key_string == "change_delay_between_points_ms_to")
    {
      Serial.println("change_delay_between_points_ms_to called");
      int m_new_delay_between_points_ms = m_JSONdoc_from_payload["change_delay_between_points_ms_to"];
      Serial.println(m_new_delay_between_points_ms);
      sweep_param_delayTime_ms_control_panel = m_new_delay_between_points_ms;
    }
    if (m_key_string == "change_lmpGain_to")
    {
      Serial.println("change_lmpGain_to called");
      int m_new_lmpGain = m_JSONdoc_from_payload["change_lmpGain_to"];
      LMPgainGLOBAL = m_new_lmpGain;
      if (Sweep_Mode == CTLPANEL)
      {
        Serial.println(LMPgainGLOBAL);
        pStat.setGain(LMPgainGLOBAL);
      }
    }
    // change_control_panel_is_active_to
    if (m_key_string == "change_control_panel_is_active_to")
    {
      Serial.println("change_control_panel_is_active_to called");
      bool m_new_control_panel_active_state = m_JSONdoc_from_payload["change_control_panel_is_active_to"];
      Serial.println(m_new_control_panel_active_state);
      if (m_new_control_panel_active_state)
      {
        Sweep_Mode = CTLPANEL;
        send_is_sweeping_status_over_websocket(true);
      }
      if (!m_new_control_panel_active_state)
      {
        Sweep_Mode = dormant;
        send_is_sweeping_status_over_websocket(false);
      }
      // set control panel to active or inactive, depending on message
    }
  }

  // if (true == false) // if key = "change_cell_voltage_to"
  // {
  //   m_websocket_send_rate = (float)atof((const char *)&payload[0]); // adjust data send rate used in loop
  // }
  //deserializeJson(m_JSONdoc, payload); // m_JSONdoc is now a json object that was payload delivered by websocket
message
}

// Called when receiving any WebSocket message
void onWebSocketEvent(uint8_t num,
                      WStype_t type,
                      uint8_t *payload,
                      size_t length)
{
  // Serial.println("onWebSocketEvent called");
  // Figure out the type of WebSocket event
  switch (type)
  {

  // Client has disconnected
  case WStype_DISCONNECTED:
    Serial.printf("[%u] Disconnected!\n", num);
    break;

  // New client has connected
  case WStype_CONNECTED:
  {
    IPAddress ip = m_websocketserver.remoteIP(num);
    Serial.printf("[%u] Connection from ", num);
    Serial.println(ip.toString());
  }
  break;

  // Echo text message back to client
  case WStype_TEXT:
    // Serial.println(payload[0,length-1]); // this doesn't work....
    Serial.printf("[%u] Received text: %s\n", num, payload);
    // m_websocketserver.sendTXT(num, payload);
    // if (true == false) // later change to if message has certain format:
    // {
    //   m_websocket_send_rate = (float)atof((const char *)&payload[0]); // adjust data send rate used in loop
    // }
    handle_websocket_text(payload);

    break;

  // For everything else: do nothing
  case WStype_BIN:
  case WStype_ERROR:
  case WStype_FRAGMENT_TEXT_START:
  case WStype_FRAGMENT_BIN_START:
  case WStype_FRAGMENT:
  case WStype_FRAGMENT_FIN:
  default:
    break;
  }
}

// void sendMessageToWebsocket(int num, char *MessageToSendToWebsocket)
// {
//   //  m_websocketserver.sendTXT(num,  MessageToSendToWebsocket);
//   //m_websocketserver.sendTXT("test hello world");
//   m_websocketserver.sendTXT(num, "Connected");
//   //  m_websocketserver.sendTXT(0, String &payload);
// }

void configureserver()
// configures server
{
  // Need to tell server to accept packets from any source with any header via http methods GET, PUT:
  DefaultHeaders::Instance().addHeader("Access-Control-Allow-Origin", "*");
  DefaultHeaders::Instance().addHeader("Access-Control-Allow-Methods", "GET, PUT");
  DefaultHeaders::Instance().addHeader("Access-Control-Allow-Headers", "*");

  // // Button #xyz
  // server.addHandler(new AsyncCallbackJsonWebHandler("/buttonxyzpressed", [](AsyncWebServerRequest *requestxyz,
JsonVariant &jsonxyz) {
  //   const JsonObject &jsonObjxyz = jsonxyz.as<JsonObject>();
  //   if (jsonObjxyz["on"])
  //   {
  //     Serial.println("Button xyz pressed.");
  //     // digitalWrite(LEDPIN, HIGH);
  //     Sweep_Mode = CV;
  //   }
  //   requestxyz->send(200, "OK");
  // }));

  // Button #1
  server.addHandler(new AsyncCallbackJsonWebHandler("/button1pressed", [](AsyncWebServerRequest *request1,
JsonVariant &json1)
                                                    {
                                                      const JsonObject &jsonObj1 = json1.as<JsonObject>();
                                                      if (jsonObj1["on"])
                                                      {
                                                        Serial.println("Button 1 pressed. Running CV sweep.");
                                                        // digitalWrite(LEDPIN, HIGH);
                                                        Sweep_Mode = CV;
                                                        send_is_sweeping_status_over_websocket(true);
                                                      }
                                                      request1->send(200, "OK");
                                                    }));
  // Button #2
  server.addHandler(new AsyncCallbackJsonWebHandler("/button2pressed", [](AsyncWebServerRequest *request2,
JsonVariant &json2)
                                                    {
                                                      const JsonObject &jsonObj2 = json2.as<JsonObject>();
                                                      if (jsonObj2["on"])
                                                      {
                                                        Serial.println("Button 2 pressed. Running NPV sweep.");
                                                        // digitalWrite(LEDPIN, HIGH);
                                                        Sweep_Mode = NPV;
                                                        send_is_sweeping_status_over_websocket(true);
                                                      }
                                                      request2->send(200, "OK");
                                                    }));
  // Button #3
  server.addHandler(new AsyncCallbackJsonWebHandler("/button3pressed", [](AsyncWebServerRequest *request3,
JsonVariant &json3)
                                                    {
                                                      const JsonObject &jsonObj3 = json3.as<JsonObject>();
                                                      if (jsonObj3["on"])
                                                      {
                                                        Serial.println("Button 3 pressed. Running SQV sweep.");
                                                        // digitalWrite(LEDPIN, HIGH);
                                                        Sweep_Mode = SQV;
                                                        send_is_sweeping_status_over_websocket(true);
                                                      }
                                                      request3->send(200, "OK");
                                                    }));
  // Button #4
  server.addHandler(new AsyncCallbackJsonWebHandler("/button4pressed", [](AsyncWebServerRequest *request4,
JsonVariant &json4)
                                                    {
                                                      const JsonObject &jsonObj4 = json4.as<JsonObject>();
                                                      if (jsonObj4["on"])
                                                      {
                                                        Serial.println("Button 4 pressed. Running CA sweep.");
                                                        // digitalWrite(LEDPIN, HIGH);
                                                        Sweep_Mode = CA;
                                                        send_is_sweeping_status_over_websocket(true);
                                                      }
                                                      request4->send(200, "OK");
                                                    }));
  // Button #5
  server.addHandler(new AsyncCallbackJsonWebHandler("/button5pressed", [](AsyncWebServerRequest *request5,
JsonVariant &json5)
                                                    {
                                                      const JsonObject &jsonObj5 = json5.as<JsonObject>();
                                                      if (jsonObj5["on"])
                                                      {
                                                        Serial.println("Button 5 pressed. Running DC sweep.");
                                                        // digitalWrite(LEDPIN, HIGH);
                                                        Sweep_Mode = DCBIAS;
                                                        send_is_sweeping_status_over_websocket(true);
                                                      }
                                                      request5->send(200, "OK");
                                                    }));
  // Button #6
  server.addHandler(new AsyncCallbackJsonWebHandler("/button6pressed", [](AsyncWebServerRequest *request6,
JsonVariant &json6)
                                                    {
                                                      const JsonObject &jsonObj6 = json6.as<JsonObject>();
                                                      if (jsonObj6["on"])
                                                      {
                                                        Serial.println("Button 6 pressed. Running IV sweep.");
                                                        // digitalWrite(LEDPIN, HIGH);
                                                        Sweep_Mode = IV;
                                                        send_is_sweeping_status_over_websocket(true);
                                                      }
                                                      request6->send(200, "OK");
                                                    }));

  // Button #7
  server.addHandler(new AsyncCallbackJsonWebHandler("/button7pressed", [](AsyncWebServerRequest *request7,
JsonVariant &json7)
                                                    {
                                                      const JsonObject &jsonObj7 = json7.as<JsonObject>();
                                                      if (jsonObj7["on"])
                                                      {
                                                        Serial.println("Button 7 pressed. Running CAL sweep.");
                                                        // digitalWrite(LEDPIN, HIGH);
                                                        Sweep_Mode = CAL;
                                                        send_is_sweeping_status_over_websocket(true);
                                                      }
                                                      request7->send(200, "OK");
                                                    }));
  // Button #8
  server.addHandler(new AsyncCallbackJsonWebHandler("/button8pressed", [](AsyncWebServerRequest *request8,
JsonVariant &json8)
                                                    {
                                                      const JsonObject &jsonObj8 = json8.as<JsonObject>();
                                                      if (jsonObj8["on"])
                                                      {
                                                        Serial.println("Button 8 pressed. Running MISC_MODE
sweep.");

                                                        // digitalWrite(LEDPIN, HIGH);
                                                        Sweep_Mode = MISC_MODE;
                                                        send_is_sweeping_status_over_websocket(true);
                                                      }
                                                      request8->send(200, "OK");
                                                    }));
  // Button #9
  server.addHandler(new AsyncCallbackJsonWebHandler("/button9pressed", [](AsyncWebServerRequest *request9,
JsonVariant &json9)
                                                    {
                                                      const JsonObject &jsonObj9 = json9.as<JsonObject>();
                                                      if (jsonObj9["on"])
                                                      {
                                                        Serial.println("Button 9 pressed.");
                                                        // digitalWrite(LEDPIN, HIGH);
                                                        Sweep_Mode = dormant;
                                                        send_is_sweeping_status_over_websocket(false);
                                                      }
                                                      request9->send(200, "OK");
                                                    }));
  // Button #10
  server.addHandler(new AsyncCallbackJsonWebHandler("/button10pressed", [](AsyncWebServerRequest *request10,
JsonVariant &json10)
                                                    {
                                                      const JsonObject &jsonObj10 = json10.as<JsonObject>();
                                                      if (jsonObj10["on"])
                                                      {
                                                        Serial.println("Button 10 pressed.");
                                                        // digitalWrite(LEDPIN, HIGH);
                                                        Sweep_Mode = dormant;
                                                        send_is_sweeping_status_over_websocket(false);
                                                      }
                                                      request10->send(200, "OK");
                                                    }));

  server.serveStatic("/", SPIFFS, "/").setDefaultFile("index.html");

  server.on("/downloadfile", HTTP_GET, [](AsyncWebServerRequest *request)
            { request->send(SPIFFS, "/data.txt", "text/plain", true); });

  server.on("/rebootnanostat", HTTP_GET, [](AsyncWebServerRequest *request)
            {
              // reboot the ESP32
              request->send(200, "text/HTML", "  <head> <meta http-equiv=\"refresh\" content=\"5;
URL=index.html\" /> <meta name=\"viewport\" content=\"width=device-width, initial-scale=1.0\"> </head> <body> <h1>
Rebooting! </h1>  </body>");
              delay(500);
              ESP.restart();
            });

  server.onNotFound([](AsyncWebServerRequest *request)
                    {
                      if (request->method() == HTTP_OPTIONS)
                      {
                        request->send(200); // options request typically sent by client at beginning to make sure
server can handle request
                      }
                      else
                      {
                        Serial.println("Not found");
                        request->send(404, "Not found");
                      }
                    });

  // Send a POST request to <IP>/actionpage with a form field message set to <message>
  server.on("/actionpage.html", HTTP_POST, [](AsyncWebServerRequest *request)
            {
              String message;
              Serial.println("actionpage.html, HTTP_POST actionpage received , processing....");

              //**********************************************

              // List all parameters int params = request->params();
              int params = request->params();
              for (int i = 0; i < params; i++)
              {
                AsyncWebParameter *p = request->getParam(i);
                if (p->isPost())
                {
                  Serial.print(i);
                  Serial.print(F("\t"));
                  Serial.print(p->name().c_str());
                  Serial.print(F("\t"));
                  Serial.println(p->value().c_str());
                  //Serial.print(F("\t"))

                  //Serial.println(i,'/T',p->name().c_str(),'/T',p->value().c_str());
                  // Serial.println(i,'/T',p->name().c_str(),'/T',p->value().c_str());
                  //Serial.println(i,'/T',p->name().c_str(),'/T',p->value().c_str());
                  //Serial.printf("POST[%s]: %s\n", p->name().c_str(), p->value().c_str());
                  set_sweep_parameters_from_form_input(p->name().c_str(), p->value().c_str());
                }
              }

              //**********************************************

              if (request->hasParam(PARAM_MESSAGE, true))
              {
                message = request->getParam(PARAM_MESSAGE, true)->value();
                Serial.println(message);
              }
              else
              {
                message = "No message sent";
              }
              // request->send(200, "text/HTML", "Hello, POST: " + message);
              // request->send(200, "text/HTML", "Sweep data saved. Click <a href=\"/index.html\">here</a> to
return to main page.");
              request->send(200, "text/HTML", "  <head> <meta http-equiv=\"refresh\" content=\"2;
URL=index.html\" /> <meta name=\"viewport\" content=\"width=device-width, initial-scale=1.0\"> </head> <body> <h1>
Settings saved! </h1> <p> Returning to main page. </p> </body>");
              // request->send(200, "OK");

              //   <head>
              //   <meta http-equiv="refresh" content="5; URL=https://www.bitdegree.org/" />
              // </head>
              // <body>
              //   <p>If you are not redirected in five seconds, <a href="https://www.bitdegree.org/">click here</-
a>.</p>
              // </body>

              // request->send(200, "text/URL", "www.google.com");
              // request->send(200, "text/URL", "<meta http-equiv=\"Refresh\" content=\"0; URL=https://google.com/
\">");
              // <meta http-equiv="Refresh" content="0; URL=https://example.com/">
            });

  // Wifitools stuff:
  // Save credentials:
  server.on("/saveSecret", HTTP_POST, [](AsyncWebServerRequest *request)
            { handleGetSavSecreteJsonNoReboot(request); });

  // Wifi scan:
  server.on("/wifiScan.json", HTTP_GET, [](AsyncWebServerRequest *request)
            { getWifiScanJson(request); });

  // List directory:
  server.on("/list", HTTP_ANY, [](AsyncWebServerRequest *request)
            { handleFileList(request); });

  // Delete file
  server.on(
      "/edit", HTTP_DELETE, [](AsyncWebServerRequest *request)
      { handleFileDelete(request); });

  // Peter Burke custom code:
  server.on(
      "/m_fupload", HTTP_POST, [](AsyncWebServerRequest *request) {},
      [](AsyncWebServerRequest *request, const String &filename, size_t index, uint8_t *data,
         size_t len, bool final)
      { handleUpload(request, filename, "files.html", index, data, len, final); });

  // From https://github.com/me-no-dev/ESPAsyncWebServer/issues/542#issuecomment-573445113
  // handling uploading firmware file
  server.on(
      "/m_firmware_update", HTTP_POST, [](AsyncWebServerRequest *request)
      {
        if (!Update.hasError())
        {
          AsyncWebServerResponse *response = request->beginResponse(200, "text/plain", "OK");
          response->addHeader("Connection", "close");
          request->send(response);
          ESP.restart();
        }
        else
        {
          AsyncWebServerResponse *response = request->beginResponse(500, "text/plain", "ERROR");
          response->addHeader("Connection", "close");
          request->send(response);
        }
      },
      handleFirmwareUpload);

  // handling uploading filesystem file
  // see https://github.com/espressif/arduino-esp32/blob/371f382db7dd36c470bb2669b222adf0a497600d/libraries/-
HTTPUpdateServer/src/HTTPUpdateServer.h
  server.on(
      "/m_filesystem_update", HTTP_POST, [](AsyncWebServerRequest *request)
      {
        if (!Update.hasError())
        {
          AsyncWebServerResponse *response = request->beginResponse(200, "text/plain", "OK");
          response->addHeader("Connection", "close");
          request->send(response);
          ESP.restart();
        }
        else
        {
          AsyncWebServerResponse *response = request->beginResponse(500, "text/plain", "ERROR");
          response->addHeader("Connection", "close");
          request->send(response);
        }
      },
      handleFilesystemUpload);

  // Done with configuration, begin server:
  server.begin();
}

void setup()
{
  // blinky pin for diagnostic:
  pinMode(LEDPIN, OUTPUT);

  // Start serial interface:
  Wire.begin(); // uses default microcontroller pins
  Serial.begin(115200);
  while (!Serial)
    ;

  Serial.println("Welcome to NanoStat, Firmware Rev. 0.1.2!");

  // initialize ADC:
  analogReadResolution(12);

  //############################### ENABLE THE POTENTIOSTAT #######################################
  pStat.setMENB(MENB);
  delay(50);
  pStat.standby();
  delay(50);
  initLMP(0); // Initializes the LMP91000 to the appropriate settings
  //############################### SPIFFS STARTUP #######################################
  if (!SPIFFS.begin(true))
  {
    Serial.println("An Error has occurred while mounting SPIFFS");
    return;
  }

  //############################## STATIC WIFI ####################################

  // WiFi.mode(WIFI_STA);
  // WiFi.begin(SSID, PASSWORD);
  // while (WiFi.waitForConnectResult() != WL_CONNECTED)
  // {
  //   Serial.println("Connected Failed! Rebooting...");
  //   delay(1000);
  //   ESP.restart();
  // }
  // Serial.println("Connected!");
  // Serial.println("The local IP address is:");
  // Serial.println(WiFi.localIP()); //print the local IP address

  //##############################  WIFITOOL CUSTOMIZED #####################################
  // Used this repo as a basis for ideas. https://github.com/oferzv/wifiTool

  bool m_autoconnected_attempt_succeeded = false;
  m_autoconnected_attempt_succeeded = connectAttempt("", ""); // uses SSID/PWD stored in ESP32 secret memory.....
  // Serial.print("m_autoconnected_attempt_succeeded = ");
  // Serial.println(m_autoconnected_attempt_succeeded);
  if (!m_autoconnected_attempt_succeeded)
  {
    // try SSID/PWD from file...
    Serial.println("Failed to connect.");
    String m_filenametopass = "/credentials.JSON";
    m_autoconnected_attempt_succeeded = readSSIDPWDfile(m_filenametopass);
  }
  if (!m_autoconnected_attempt_succeeded)
  {
    // start AP server
    // Serial.println("connect failed, starting AP server");
    setUpAPService();
    runWifiPortal();

    // MDNS.begin("nanostat"); // see https://randomnerdtutorials.com/esp32-access-point-ap-web-server/
  }

  // connectAttempt(SSID,PASSWORD); // uses SSID/PWD stored in ESP32 secret memory.....

  //##############################  WIFITOOL #####################################

  // wifiTool.begin(false);
  // if (!wifiTool.wifiAutoConnect())
  // {
  //   Serial.println("fail to connect to wifi!!!!");
  //   wifiTool.runApPortal();
  // }

  // Serial.println("wifitools called ");
  // delete &wifiTool;
  // delay(2000);

  //############################## DNS ####################################
  MDNS.begin("nanostat");

  //############################## WEBSERVER & WIFI ######################################

  server.reset(); // try putting this in setup
  configureserver();
  //  runWifiPortal_after_connected_to_WIFI(); // Allows some system level tools such as saving wifi credentials
and scan, OTA firmware upgrade, file directory..
  // configureserver has the code in it little by little, can't do configuration after starting server which
happens inside configureserver() method as of now...

  //############################## WEBSOCKET ######################################

  m_websocketserver.begin();
  m_websocketserver.onEvent(onWebSocketEvent); // Start WebSocket server and assign callback

  //############################## READ CALIBRATION FILE IF THERE IS ONE #####################################

  // SPIFFS.remove("/calibration.JSON"); // manual delete to test code.
  readCalFile();

  //############################## BLINK LED TO SHOW SETUP COMPLETE ######################################
  Serial.print("Heap free memory (in bytes)= ");
  Serial.println(ESP.getFreeHeap());
  Serial.println(F("Setup complete."));
  blinkLED(LEDPIN, 15, 1000); // blink LED to show setup is complete, also give settle time to LMP91000
}

void loop()
{

  //will hold the code here until a character is sent over the Serial port
  //this ensures the experiment will only run when initiated
  //  Serial.println(F("Press enter to begin a sweep."));
  //  while (!Serial.available())
  //    ;
  //  Serial.read();
  // Look for and handle WebSocket data

  m_websocketserver.loop();

  if (m_send_websocket_test_data_in_loop == true) // do things here in loop at full speed
  {
    // Pseudocode: xxx_period_in_ms_xxx=period_in_s * 1e3 = (1/freqHz)*1e3
    if (millis() - last_time_sent_websocket_server > (1000 / m_websocket_send_rate)) // every half second, print
    {
      //    sendTimeOverWebsocketJSON();
      sendValueOverWebsocketJSON(100 * 0.5 * sin(millis() / 1e3)); // value is sine wave of time , frequency 0.5
Hz, amplitude 100.
      last_time_sent_websocket_server = millis();
    }
    // m_microsbefore_websocketsendcalled=micros();
    // sendTimeOverWebsocketJSON(); // takes 2.5 ms on average, when client is connected, else 45 microseconds...
    // Serial.println(micros()-m_microsbefore_websocketsendcalled);
  }

  if (Sweep_Mode == NPV)
  {
    // void runNPV(uint8_t lmpGain, int16_t startV, int16_t endV,
    //         int8_t pulseAmp, uint32_t pulse_width, uint32_t pulse_period,
    //         uint32_t quietTime, uint8_t range, bool setToZero)
    LMPgainGLOBAL = sweep_param_lmpGain;
    runNPV(sweep_param_lmpGain, sweep_param_startV_NPV, sweep_param_endV_NPV,
           sweep_param_pulseAmp_NPV, sweep_param_width_NPV, sweep_param_period_NPV,
           sweep_param_quietTime_NPV, 1, sweep_param_setToZero);
    writeVoltsCurrentArraytoFile();
    // sendVoltammogramWebsocketJSON();
    sendVoltammogramWebsocketBIN();
    Sweep_Mode = dormant;
    send_is_sweeping_status_over_websocket(false);
  }
  else if (Sweep_Mode == CV)
  {
    LMPgainGLOBAL = sweep_param_lmpGain;
    runCV(sweep_param_lmpGain, sweep_param_cycles_CV, sweep_param_startV_CV,
          sweep_param_endV_CV, sweep_param_vertex1_CV, sweep_param_vertex2_CV, sweep_param_stepV_CV,
          sweep_param_rate_CV, sweep_param_setToZero);
    writeVoltsCurrentArraytoFile();
    // sendVoltammogramWebsocketJSON();
    sendVoltammogramWebsocketBIN();
    Sweep_Mode = dormant;
    send_is_sweeping_status_over_websocket(false);
  }
  else if (Sweep_Mode == SQV)
  {
    LMPgainGLOBAL = sweep_param_lmpGain;

    runSWV(sweep_param_lmpGain, sweep_param_startV_SWV, sweep_param_endV_SWV,
           sweep_param_pulseAmp_SWV, sweep_param_stepV_SWV, sweep_param_freq_SWV, sweep_param_setToZero);
    writeVoltsCurrentArraytoFile();
    // sendVoltammogramWebsocketJSON();
    sendVoltammogramWebsocketBIN();
    Sweep_Mode = dormant;
    send_is_sweeping_status_over_websocket(false);
  }
  else if (Sweep_Mode == CA)
  {
    LMPgainGLOBAL = sweep_param_lmpGain;
    runAmp(sweep_param_lmpGain, sweep_param_pre_stepV_CA, sweep_param_quietTime_CA,
           sweep_param_V1_CA, sweep_param_t1_CA, sweep_param_V2_CA, sweep_param_t2_CA,
           sweep_param_samples_CA, 1, sweep_param_setToZero);
    writeVoltsCurrentArraytoFile();
    // sendVoltammogramWebsocketJSON();
    sendVoltammogramWebsocketBIN();
    Sweep_Mode = dormant;
    send_is_sweeping_status_over_websocket(false);
  }
  else if (Sweep_Mode == DCBIAS)
  {
    LMPgainGLOBAL = sweep_param_lmpGain;

    testNoiseAtABiasPoint(sweep_param_biasV_noisetest, sweep_param_numPoints_noisetest,
                          sweep_param_delayTime_ms_noisetest);
    writeVoltsCurrentArraytoFile();
    // sendVoltammogramWebsocketJSON();
    sendVoltammogramWebsocketBIN();
    Sweep_Mode = dormant;
    send_is_sweeping_status_over_websocket(false);
  }
  else if (Sweep_Mode == IV)
  {
    LMPgainGLOBAL = sweep_param_lmpGain;

    testIV(sweep_param_startV_IV, sweep_param_endV_IV, sweep_param_numPoints_IV,
           sweep_param_delayTime_ms_IV);
    writeVoltsCurrentArraytoFile();
    //sendVoltammogramWebsocketJSON();
    sendVoltammogramWebsocketBIN();
    Sweep_Mode = dormant;
    send_is_sweeping_status_over_websocket(false);
  }
  else if (Sweep_Mode == CAL)
  {
    LMPgainGLOBAL = sweep_param_lmpGain;

    calibrateDACandADCs(sweep_param_delayTime_ms_CAL);
    Sweep_Mode = dormant;
    send_is_sweeping_status_over_websocket(false);
    writeCalFile();
  }
  else if (Sweep_Mode == CTLPANEL)
  {

    pStat.setGain(LMPgainGLOBAL);

    delay(sweep_param_delayTime_ms_control_panel);
    // read adc and convert to current:
    analog_read_avg_bits_temp = (float)analog_read_avg(num_adc_readings_to_average_control_panel, LMP);
    v1_temp = (3.3 / 255.0) * (1 / (2.0 * b_coeff)) * analog_read_avg_bits_temp - (a_coeff / (2.0 * b_coeff)) *
(3.3 / 255.0); // LMP is wired to Vout of the LMP91000
    v1_temp = v1_temp * 1000.0;
    v2_temp = dacVout * .5;                                                            //the zero of the
internal transimpedance amplifier
    amps_temp = (((v1_temp - v2_temp) / 1000) / TIA_GAIN[LMPgainGLOBAL - 1]) * pow(10, 6); //scales to uA
    // create json string to send to broswer:
    temp_json_string = "{\"amps\":";
    temp_json_string += String(amps_temp, DEC);
    temp_json_string += ",\"volts\":";
    temp_json_string += String(cell_voltage_control_panel);
    temp_json_string += ",\"time\":";
    temp_json_string += String(millis());
    temp_json_string += ",\"analog_read_avg_bits\":";
    temp_json_string += String(analog_read_avg_bits_temp);
    temp_json_string += ",\"analog_read_avg_mV\":";
    temp_json_string += String(v1_temp, DEC);
    temp_json_string += "}";
    // send json string to browswer
    m_websocketserver.broadcastTXT(temp_json_string.c_str(), temp_json_string.length());
  }
  else if (Sweep_Mode == MISC_MODE) // list directory to serial
  {
    listDir("/", 3);
    //    delay(250);
    // sendVoltammogramWebsocketJSON();
    // readFileAndPrintToSerial();
    Sweep_Mode = dormant;
    send_is_sweeping_status_over_websocket(false);

    sendVoltammogramWebsocketBIN();
  }
  else
  {
    delay(10);
  }
  last_time_loop_called = millis();
}
```

```
; PlatformIO Project Configuration File
;
;   Build options: build flags, source filter
;   Upload options: custom upload port, speed and extra flags
;   Library options: dependencies, extra library storages
;   Advanced options: extra scripting
;
; Please visit documentation for the other options and examples
; https://docs.platformio.org/page/projectconf.html

[env:pico32]
platform = espressif32
board = pico32
framework = arduino
monitor_speed = 115200
lib_deps =
    linneslab/LMP91000 @ ^1.0.0
    me-no-dev/AsyncTCP@^1.1.1
    ottowinter/ESPAsyncWebServer-esphome@^1.2.7
    bblanchon/ArduinoJson@^6.17.3
    links2004/WebSockets@^2.3.6
    bbx10/DNSServer@^1.1.0
```

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>NanoStat
  </title>
  <!-- Bootstrap CSS -->
  <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.0-beta3/dist/css/bootstrap.min.css" rel="stylesheet"
    integrity="sha384-eOJMYsd53ii+scO/bJGFsiCZc+5NDVN2yr8+0RDqr0Ql0h+rP48ckxlpbzKgwra6" crossorigin="anonymous">
  <script src='https://cdn.jsdelivr.net/npm/chart.js@3.1.1/dist/chart.min.js'></script>
  <!-- <script type="text/javascript" src="websocketcode.js"></script> -->
  <!-- <script type="text/javascript" src="button_handle_JS.js"></script> -->
  <script src="https://cdn.plot.ly/plotly-latest.min.js"></script>
  <!-- <link rel="stylesheet" href="menubarstyle.css"> -->
  <!-- <link rel="stylesheet" href="CSS-Reset.css"> -->
  <!-- Google font: -->
  <link rel="preconnect" href="https://fonts.gstatic.com">
  <link href="https://fonts.googleapis.com/css2?family=Leckerli+One&display=swap" rel="stylesheet">
  <!-- MDB Bootstrap CSS -->
  <!-- Font Awesome -->
  <link href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/5.15.1/css/all.min.css" rel="stylesheet" />
  <!-- Google Fonts -->
  <link href="https://fonts.googleapis.com/css?family=Roboto:300,400,500,700&display=swap" rel="stylesheet" />
  <!-- MDB -->
  <!-- <link href="https://cdnjs.cloudflare.com/ajax/libs/mdb-ui-kit/3.3.0/mdb.min.css" rel="stylesheet" /> -->
  <!-- Custom style: -->
  <link rel="stylesheet" href="nanostatstyle.css">
</head>


<!-- <body style="background-color: lightblue;"> -->

<body>

  <!-- Navigation bar -->
  <nav class="navbar navbar-expand-md navbar-dark bg-primary">
    <div class="container">
      <!-- <nav class="navbar navbar-expand-md navbar-light"> -->
      <!-- Brand icon: -->
      <a href="#" class="navbar-brand" style="font-family: 'Leckerli One', cursive;">NanoStat</a>
      <!-- Hamburger button: -->
      <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target="#toggleMobileMenu"
        aria-controls="toggleMobileMenu" aria-expanded="false" aria-table="Toggle navigation">
        <span class="navbar-toggler-icon"></span>
      </button>
      <!-- Menu items: -->
      <div class="collapse navbar-collapse" id="toggleMobileMenu">
        <ul class="navbar-nav ms-auto text-center">
          <li>
            <a class="nav-link" href="index.html">Sweep</a>
          </li>
          <li>
            <a class="nav-link" href="setup.html">Setup</a>
          </li>
          <li>
            <a class="nav-link" href="panel.html">Panel</a>
          </li>
          <li>
            <a class="nav-link" href="download.html">Download</a>
          </li>

          <li class="nav-item dropdown">
            <a href="#" class="nav-link dropdown-toggle" id="navbarDropdown" role="button" data-bs-
toggle="dropdown"
```

```
              aria-expanded="false">System</a>
            <ul class="dropdown-menu" aria-labeledby="navbarDropdown">
              <li><a href="files.html" class="dropdown-item">Files</a></li>
              <li><a href="wifi.html" class="dropdown-item">Wifi</a></li>
              <li><a href="firmware.html" class="dropdown-item">Firmware</a></li>
              <li><a href="reboot.html" class="dropdown-item">Reboot</a></li>

            </ul>

          </li>



          <li>
            <a class="nav-link" href="help.html">Help</a>
          </li>
          <li>
            <a class="nav-link active" href="about.html">About</a>
          </li>
        </ul>
      </div>
  </nav>

  <!-- #0D6EFD is color of Bootstrap menubar "primary" -->
  <!--  hsl(216, 98%, 52%) -->

  <!-- Page title -->
  <!-- <header class="container text-center mybox1"> -->
    <header class="container text-center ">
      <h1 style="font-family: 'Leckerli One', cursive;">NanoStat</h1>
      https://github.com/PeterJBurke/Nanostat
      <p>Firmware Rev. 0.1.2 </p>
      <p>     <img src="http://www.burkelab.com/wp-content/uploads/prof-peter-burke.jpg" alt="Italian Trulli">
      </p>

    </header>
  <p class="container text-center  fs-6  pt-3 fst-italic">P.J. Burke, et al, manuscript in preparation (2021).</p>

  <!-- Page footer -->
  <!-- <footer class="container text-center myfooter fs-6 fw-lighter pt-3 fst-italic"> -->
    <footer class="container text-center  fw-lighter  fst-italic">
      <p>NanoStat Rev. 3.5 @ BurkeLab 2021</p>
    </footer>

  <!-- Bootstrap JS -->
  <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.0-beta3/dist/js/bootstrap.bundle.min.js"
    integrity="sha384-JEW9xMcG8R+pH31jmWH6WWP0WintQrMb4s7ZOdauHnUtxwoG2vI5DkLtS3qm9Ekf"
    crossorigin="anonymous"></script>
  <!-- MDB Bootstrap JS -->
  <!-- MDB -->
  <!-- <script type="text/javascript" src="https://cdnjs.cloudflare.com/ajax/libs/mdb-ui-kit/3.3.0/mdb.min.js"></-
script> -->
</body>
```

```
{/* <button onclick= "window.location.href = '/wifi_manager.html'"> Wifi Setup </button> */ }

let temporary_boolean = true;
async function on_Button_1_pressed() {
  console.log(window.location.hostname)
  try {
    // const response = await fetch("http://nanostat.local/button1pressed", {
    const response = await fetch("/button1pressed", {
      method: "PUT",
      body: JSON.stringify({ on: temporary_boolean }),
      headers: {
        "Content-Type": "application/json",
      },
    });
  } catch (error) {
    alert("Request failed - check the console");
    console.error(error);
  }
}
async function on_Button_2_pressed() {
  try {
    const response = await fetch("/button2pressed", {
      method: "PUT",
      body: JSON.stringify({ on: temporary_boolean }),
      headers: {
        "Content-Type": "application/json",
      },
    });
  } catch (error) {
    alert("Request failed - check the console");
    console.error(error);
  }
}
async function on_Button_3_pressed() {
  try {
    const response = await fetch("/button3pressed", {
      method: "PUT",
      body: JSON.stringify({ on: temporary_boolean }),
      headers: {
        "Content-Type": "application/json",
      },
    });
  } catch (error) {
    alert("Request failed - check the console");
    console.error(error);
  }
}
async function on_Button_4_pressed() {
  try {
    const response = await fetch("/button4pressed", {
      method: "PUT",
      body: JSON.stringify({ on: temporary_boolean }),
      headers: {
        "Content-Type": "application/json",
      },
    });
  } catch (error) {
    alert("Request failed - check the console");
    console.error(error);
  }
}
async function on_Button_5_pressed() {
  try {
    const response = await fetch("/button5pressed", {
      method: "PUT",
      body: JSON.stringify({ on: temporary_boolean }),
      headers: {
        "Content-Type": "application/json",
      },
    });
  } catch (error) {
    alert("Request failed - check the console");
    console.error(error);
  }
}
async function on_Button_6_pressed() {
  try {
    const response = await fetch("/button6pressed", {
      method: "PUT",
      body: JSON.stringify({ on: temporary_boolean }),
      headers: {
        "Content-Type": "application/json",
      },
    });
  } catch (error) {
    alert("Request failed - check the console");
    console.error(error);
  }
}
async function on_Button_7_pressed() {
  try {
    const response = await fetch("/button7pressed", {
      method: "PUT",
      body: JSON.stringify({ on: temporary_boolean }),
      headers: {
        "Content-Type": "application/json",
      },
    });
  } catch (error) {
    alert("Request failed - check the console");
    console.error(error);
  }
}
async function on_Button_8_pressed() {
  try {
    const response = await fetch("/button8pressed", {
      method: "PUT",
      body: JSON.stringify({ on: temporary_boolean }),
      headers: {
        "Content-Type": "application/json",
      },
    });
  } catch (error) {
    alert("Request failed - check the console");
    console.error(error);
  }
}
async function on_Button_9_pressed() {
  try {
    const response = await fetch("/button9pressed", {
      method: "PUT",
      body: JSON.stringify({ on: temporary_boolean }),
      headers: {
        "Content-Type": "application/json",
      },
    });
  } catch (error) {
    alert("Request failed - check the console");
    console.error(error);
  }
}
async function on_Button_10_pressed() {
  try {
    const response = await fetch("/button10pressed", {
      method: "PUT",
      body: JSON.stringify({ on: temporary_boolean }),
      headers: {
        "Content-Type": "application/json",
      },
    });
  } catch (error) {
    alert("Request failed - check the console");
    console.error(error);
  }
}
```

```css
/* http://meyerweb.com/eric/tools/css/reset/
   v2.0 | 20110126
   License: none (public domain)
*/

html, body, div, span, applet, object, iframe,
h1, h2, h3, h4, h5, h6, p, blockquote, pre,
a, abbr, acronym, address, big, cite, code,
del, dfn, em, img, ins, kbd, q, s, samp,
small, strike, strong, sub, sup, tt, var,
b, u, i, center,
dl, dt, dd, ol, ul, li,
fieldset, form, label, legend,
table, caption, tbody, tfoot, thead, tr, th, td,
article, aside, canvas, details, embed,
figure, figcaption, footer, header, hgroup,
menu, nav, output, ruby, section, summary,
time, mark, audio, video {
    margin: 0;
    padding: 0;
    border: 0;
    font-size: 100%;
    font: inherit;
    vertical-align: baseline;
}
/* HTML5 display-role reset for older browsers */
article, aside, details, figcaption, figure,
footer, header, hgroup, menu, nav, section {
    display: block;
}
body {
    line-height: 1;
}
ol, ul {
    list-style: none;
}
blockquote, q {
    quotes: none;
}
blockquote:before, blockquote:after,
q:before, q:after {
    content: '';
    content: none;
}
table {
    border-collapse: collapse;
    border-spacing: 0;
}
```

```html
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>NanoStat
  </title>
  <!-- Bootstrap CSS -->
  <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.0-beta3/dist/css/bootstrap.min.css" rel="stylesheet"
    integrity="sha384-eOJMYsd53ii+scO/bJGFsiCZc+5NDVN2yr8+0RDqr0Ql0h+rP48ckxlpbzKgwra6" crossorigin="anonymous">
  <script src='https://cdn.jsdelivr.net/npm/chart.js@3.1.1/dist/chart.min.js'></script>
  <!-- <script type="text/javascript" src="websocketcode.js"></script> -->
  <!-- <script type="text/javascript" src="button_handle_JS.js"></script> -->
  <!-- <script src="https://cdn.plot.ly/plotly-latest.min.js"></script> -->
  <!-- <link rel="stylesheet" href="menubarstyle.css"> -->
  <!-- <link rel="stylesheet" href="CSS-Reset.css"> -->
  <!-- Google font: -->
  <link rel="preconnect" href="https://fonts.gstatic.com">
  <link href="https://fonts.googleapis.com/css2?family=Leckerli+One&display=swap" rel="stylesheet">
    <!-- Custom style: -->
    <link rel="stylesheet" href="nanostatstyle.css">
</head>


<body>

  <!-- Navigation bar -->
  <nav class="navbar navbar-expand-md navbar-dark bg-primary">
    <div class="container">
      <!-- <nav class="navbar navbar-expand-md navbar-light"> -->
      <!-- Brand icon: -->
      <a href="#" class="navbar-brand" style="font-family: 'Leckerli One', cursive;">NanoStat</a>
      <!-- Hamburger button: -->
      <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target="#toggleMobileMenu"
        aria-controls="toggleMobileMenu" aria-expanded="false" aria-table="Toggle navigation">
        <span class="navbar-toggler-icon"></span>
      </button>
      <!-- Menu items: -->
      <div class="collapse navbar-collapse" id="toggleMobileMenu">
        <ul class="navbar-nav ms-auto text-center">
          <li>
            <a class="nav-link" href="index.html">Sweep</a>
          </li>
          <li>
            <a class="nav-link" href="setup.html">Setup</a>
          </li>
          <li>
            <a class="nav-link" href="panel.html">Panel</a>
          </li>
          <li>
            <a class="nav-link active" href="download.html">Download</a>
          </li>

          <li class="nav-item dropdown">
            <a href="#" class="nav-link dropdown-toggle" id="navbarDropdown" role="button" data-bs-
toggle="dropdown"
              aria-expanded="false">System</a>
            <ul class="dropdown-menu" aria-labeledby="navbarDropdown">
              <li><a href="files.html" class="dropdown-item">Files</a></li>
              <li><a href="wifi.html" class="dropdown-item">Wifi</a></li>
              <li><a href="firmware.html" class="dropdown-item">Firmware</a></li>
              <li><a href="reboot.html" class="dropdown-item">Reboot</a></li>

            </ul>
          </li>



          <li>
            <a class="nav-link" href="help.html">Help</a>
          </li>
          <li>
            <a class="nav-link" href="about.html">About</a>
          </li>
        </ul>
      </div>
  </nav>

  <!-- <header class="container text-center mybox1"> -->
    <header class="container text-center ">
      <h1>Download</h1>
      <h2>Most recent sweep file.</h2>

    </header>



<main class="container">
  <div class="col text-center ">
    <!-- <button  class="btn btn-primary my-2">Download</button> -->
    <a class="btn btn-primary" href="downloadfile" role="button">Download</a>
  </div>
</main>

  <!-- Page footer -->
  <!-- <footer class="container text-center myfooter fs-6 fw-lighter pt-3 fst-italic"> -->
    <footer class="container text-center  fs-6 fw-lighter pt-3 fst-italic">
      <p>NanoStat Rev. 3.5 @ BurkeLab 2021</p>
    </footer>


  <!-- Bootstrap JS -->
  <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.0-beta3/dist/js/bootstrap.bundle.min.js"
    integrity="sha384-JEW9xMcG8R+pH31jmWH6WWP0WintQrMb4s7ZOdauHnUtxwoG2vI5DkLtS3qm9Ekf"
    crossorigin="anonymous"></script>
</body>
```

```html
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>NanoStat
  </title>
  <!-- Include jquery -->
  <script src="https://code.jquery.com/jquery-2.2.4.js" integrity="sha256-iT6Q9iMJYuQiMWNd9lDyBUStIq/-
8PuOW33aOqmvFpqI="
    crossorigin="anonymous"></script>
  <!-- Bootstrap CSS -->
  <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.0-beta3/dist/css/bootstrap.min.css" rel="stylesheet"
    integrity="sha384-eOJMYsd53ii+scO/bJGFsiCZc+5NDVN2yr8+0RDqr0Ql0h+rP48ckxlpbzKgwra6" crossorigin="anonymous">
  <script src='https://cdn.jsdelivr.net/npm/chart.js@3.1.1/dist/chart.min.js'></script>
  <!-- <script type="text/javascript" src="websocketcode.js"></script> -->
  <!-- <script type="text/javascript" src="button_handle_JS.js"></script> -->
  <script src="https://cdn.plot.ly/plotly-latest.min.js"></script>
  <script type="text/javascript" src="wifimanager.js"></script>
  <!-- <link rel="stylesheet" href="menubarstyle.css"> -->
  <!-- <link rel="stylesheet" href="CSS-Reset.css"> -->
  <!-- Google font: -->
  <link rel="preconnect" href="https://fonts.gstatic.com">
  <link href="https://fonts.googleapis.com/css2?family=Leckerli+One&display=swap" rel="stylesheet">
  <!-- Custom style: -->
  <link rel="stylesheet" href="nanostatstyle.css">
</head>


<body>

  <!-- Navigation bar -->
  <nav class="navbar navbar-expand-md navbar-dark bg-primary">
    <div class="container">
      <!-- <nav class="navbar navbar-expand-md navbar-light"> -->
      <!-- Brand icon: -->
      <a href="#" class="navbar-brand" style="font-family: 'Leckerli One', cursive;">NanoStat</a>
      <!-- Hamburger button: -->
      <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target="#toggleMobileMenu"
        aria-controls="toggleMobileMenu" aria-expanded="false" aria-table="Toggle navigation">
        <span class="navbar-toggler-icon"></span>
      </button>
      <!-- Menu items: -->
      <div class="collapse navbar-collapse" id="toggleMobileMenu">
        <ul class="navbar-nav ms-auto text-center">
          <li>
            <a class="nav-link" href="index.html">Sweep</a>
          </li>
          <li>
            <a class="nav-link" href="setup.html">Setup</a>
          </li>
          <li>
            <a class="nav-link" href="panel.html">Panel</a>
          </li>
          <li>
            <a class="nav-link" href="download.html">Download</a>
          </li>

          <li class="nav-item dropdown">
            <a href="#" class="nav-link dropdown-toggle active" id="navbarDropdown" role="button"
              data-bs-toggle="dropdown" aria-expanded="false">System</a>
            <ul class="dropdown-menu" aria-labeledby="navbarDropdown">
              <li><a href="files.html" class="dropdown-item active">Files</a></li>
              <li><a href="wifi.html" class="dropdown-item">Wifi</a></li>
              <li><a href="firmware.html" class="dropdown-item">Firmware</a></li>
              <li><a href="reboot.html" class="dropdown-item">Reboot</a></li>

            </ul>

          </li>



          <li>
            <a class="nav-link" href="help.html">Help</a>
          </li>
          <li>
            <a class="nav-link" href="about.html">About</a>
          </li>
        </ul>
      </div>
  </nav>

  <!-- Page title -->
  <!-- <header class="container text-center mybox1"> -->
  <header class="container text-center ">
    <h1>Files</h1>
    <h2>NanoStat directory</h2>
  </header>

  <!-- Peter Burke custom code -->
  <div class="container">
    <p>File upload</p>
    <FORM action='/m_fupload' method='post' enctype='multipart/form-data'>
      <input class='btn btn-outline-primary' type='file' name='m_fupload' id='m_fupload_id' value='m_value'>
      <button class='btn btn-primary' type='submit'>Upload File</button>
  </div>


  <main class="container ">
    <h5>Directory</h5>
    <p>Click X to delete the file.</p>

    <div id="filelist"></div>

    <script> getFileList();</script>
  </main>





  <!-- Page footer -->
  <!-- <footer class="container text-center myfooter fs-6 fw-lighter pt-3 fst-italic"> -->
  <footer class="container text-center  fs-6 fw-lighter pt-3 fst-italic">
    <p>NanoStat Rev. 3.5 @ BurkeLab 2021</p>
  </footer>


  <!-- Bootstrap JS -->
  <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.0-beta3/dist/js/bootstrap.bundle.min.js"
    integrity="sha384-JEW9xMcG8R+pH31jmWH6WWP0WintQrMb4s7ZOdauHnUtxwoG2vI5DkLtS3qm9Ekf"
    crossorigin="anonymous"></script>
</body>
```

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>NanoStat
  </title>
  <!-- Bootstrap CSS -->
  <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.0-beta3/dist/css/bootstrap.min.css" rel="stylesheet"
    integrity="sha384-eOJMYsd53ii+scO/bJGFsiCZc+5NDVN2yr8+0RDqr0Ql0h+rP48ckxlpbzKgwra6" crossorigin="anonymous">
  <script src='https://cdn.jsdelivr.net/npm/chart.js@3.1.1/dist/chart.min.js'></script>
  <!-- <script type="text/javascript" src="websocketcode.js"></script> -->
  <!-- <script type="text/javascript" src="button_handle_JS.js"></script> -->
  <script type="text/javascript" src="wifimanager.js"></script>
  <script src="https://cdn.plot.ly/plotly-latest.min.js"></script>
  <!-- <link rel="stylesheet" href="menubarstyle.css"> -->
  <!-- <link rel="stylesheet" href="CSS-Reset.css"> -->
  <!-- Google font: -->
  <link rel="preconnect" href="https://fonts.gstatic.com">
  <link href="https://fonts.googleapis.com/css2?family=Leckerli+One&display=swap" rel="stylesheet">
  <!-- Custom style: -->
  <link rel="stylesheet" href="nanostatstyle.css">
</head>


<body>

  <!-- Navigation bar -->
  <nav class="navbar navbar-expand-md navbar-dark bg-primary">
    <div class="container">
      <!-- <nav class="navbar navbar-expand-md navbar-light"> -->
      <!-- Brand icon: -->
      <a href="#" class="navbar-brand" style="font-family: 'Leckerli One', cursive;">NanoStat</a>
      <!-- Hamburger button: -->
      <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target="#toggleMobileMenu"
        aria-controls="toggleMobileMenu" aria-expanded="false" aria-table="Toggle navigation">
        <span class="navbar-toggler-icon"></span>
      </button>
      <!-- Menu items: -->
      <div class="collapse navbar-collapse" id="toggleMobileMenu">
        <ul class="navbar-nav ms-auto text-center">
          <li>
            <a class="nav-link" href="index.html">Sweep</a>
          </li>
          <li>
            <a class="nav-link" href="setup.html">Setup</a>
          </li>
          <li>
            <a class="nav-link" href="panel.html">Panel</a>
          </li>
          <li>
            <a class="nav-link" href="download.html">Download</a>
          </li>

          <li class="nav-item dropdown">
            <a href="#" class="nav-link dropdown-toggle active" id="navbarDropdown" role="button"
              data-bs-toggle="dropdown" aria-expanded="false">System</a>
            <ul class="dropdown-menu" aria-labeledby="navbarDropdown">
              <li><a href="files.html" class="dropdown-item">Files</a></li>
              <li><a href="wifi.html" class="dropdown-item">Wifi</a></li>
              <li><a href="firmware.html" class="dropdown-item active">Firmware</a></li>
              <li><a href="reboot.html" class="dropdown-item">Reboot</a></li>

            </ul>
          </li>



          <li>
            <a class="nav-link" href="help.html">Help</a>
          </li>
          <li>
            <a class="nav-link" href="about.html">About</a>
          </li>
        </ul>
      </div>
  </nav>

  <!-- Page title -->
  <!-- <header class="container text-center mybox1"> -->
  <header class="container text-center ">
    <h1>Firmware</h1>
    <h2>Upgrade the NanoStat firmware.</h2>
  </header>


  <!-- Peter Burke custom code -->
  <div class="container ">
    <h3>Firmware update</h3>
    <p>Upload firmware.bin</p>
    <FORM action='/m_firmware_update' method='post' enctype='multipart/form-data'>
      <input class='btn btn-outline-primary' type='file' name='m_fupload' id='m_fupload_id' value='m_value'>
      <button class='btn btn-primary' type='submit'>Upload firmware & update</button></FORM>
  </div>


  <div class="container my-5 ">
    <h3>Filesystem update</h3>
    <p>Upload spiffs.bin</p>
    <FORM action='/m_filesystem_update' method='post' enctype='multipart/form-data'>
      <input class='btn btn-outline-primary' type='file' name='m_filesystemupload' id='m_filesystemupload_id'
value='m_value'>
      <button class='btn btn-primary' type='submit'>Upload filesystem & update</button></FORM>
  </div>

  <!-- Page footer -->
  <!-- <footer class="container text-center myfooter fs-6 fw-lighter pt-3 fst-italic"> -->
  <footer class="container text-center  fs-6 fw-lighter pt-3 fst-italic">
    <p>NanoStat Rev. 3.5 @ BurkeLab 2021</p>
  </footer>
  <!-- Bootstrap JS -->
  <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.0-beta3/dist/js/bootstrap.bundle.min.js"
    integrity="sha384-JEW9xMcG8R+pH31jmWH6WWP0WintQrMb4s7ZOdauHnUtxwoG2vI5DkLtS3qm9Ekf"
    crossorigin="anonymous"></script>
</body>
```

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>NanoStat
  </title>
  <!-- Bootstrap CSS -->
  <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.0-beta3/dist/css/bootstrap.min.css" rel="stylesheet"
    integrity="sha384-eOJMYsd53ii+scO/bJGFsiCZc+5NDVN2yr8+0RDqr0Ql0h+rP48ckxlpbzKgwra6" crossorigin="anonymous">
  <script src='https://cdn.jsdelivr.net/npm/chart.js@3.1.1/dist/chart.min.js'></script>
  <script src="https://cdn.plot.ly/plotly-latest.min.js"></script>
  <!-- <link rel="stylesheet" href="menubarstyle.css"> -->
  <!-- <link rel="stylesheet" href="CSS-Reset.css"> -->
  <!-- Google font: -->
  <link rel="preconnect" href="https://fonts.gstatic.com">
  <link href="https://fonts.googleapis.com/css2?family=Leckerli+One&display=swap" rel="stylesheet">
  <!-- MDB Bootstrap CSS -->
  <!-- Font Awesome -->
  <link href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/5.15.1/css/all.min.css" rel="stylesheet" />
  <!-- Google Fonts -->
  <link href="https://fonts.googleapis.com/css?family=Roboto:300,400,500,700&display=swap" rel="stylesheet" />
  <!-- MDB -->
  <!-- <link href="https://cdnjs.cloudflare.com/ajax/libs/mdb-ui-kit/3.3.0/mdb.min.css" rel="stylesheet" /> -->
  <!-- Custom style: -->
  <link rel="stylesheet" href="nanostatstyle.css">
</head>


<!-- <body style="background-color: lightblue;"> -->

<body>

  <!-- Navigation bar -->
  <nav class="navbar navbar-expand-md navbar-dark bg-primary">
    <div class="container">
      <!-- <nav class="navbar navbar-expand-md navbar-light"> -->
      <!-- Brand icon: -->
      <a href="#" class="navbar-brand" style="font-family: 'Leckerli One', cursive;">NanoStat</a>
      <!-- Hamburger button: -->
      <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target="#toggleMobileMenu"
        aria-controls="toggleMobileMenu" aria-expanded="false" aria-table="Toggle navigation">
        <span class="navbar-toggler-icon"></span>
      </button>
      <!-- Menu items: -->
      <div class="collapse navbar-collapse" id="toggleMobileMenu">
        <ul class="navbar-nav ms-auto text-center">
          <li>
            <a class="nav-link" href="index.html">Sweep</a>
          </li>
          <li>
            <a class="nav-link" href="setup.html">Setup</a>
          </li>
          <li>
            <a class="nav-link" href="panel.html">Panel</a>
          </li>
          <li>
            <a class="nav-link" href="download.html">Download</a>
          </li>

          <li class="nav-item dropdown">
            <a href="#" class="nav-link dropdown-toggle" id="navbarDropdown" role="button" data-bs-
toggle="dropdown"
              aria-expanded="false">System</a>
            <ul class="dropdown-menu" aria-labeledby="navbarDropdown">
```

```
              <li><a href="files.html" class="dropdown-item">Files</a></li>
              <li><a href="wifi.html" class="dropdown-item">Wifi</a></li>
              <li><a href="firmware.html" class="dropdown-item">Firmware</a></li>
              <li><a href="reboot.html" class="dropdown-item">Reboot</a></li>

            </ul>

          </li>



          <li>
            <a class="nav-link active" href="help.html">Help</a>
          </li>
          <li>
            <a class="nav-link" href="about.html">About</a>
          </li>
        </ul>
      </div>
  </nav>

  <!-- #0D6EFD is color of Bootstrap menubar "primary" -->
  <!--  hsl(216, 98%, 52%) -->

  <!-- Page title -->
  <!-- <header class="container text-center mybox1"> -->
  <header class="container text-center ">
    <h1>Help</h1>
  </header>

  <div class="container">

    <h3>Instructions:</h3>
    <p>On bootup, blue LED flashes until setup complete.</p>
    <p>To run a sweep, go to SWEEP page and click button for sweep type. (See below for sweep type explanations.)
During
      sweep, blue LED flashes (unless time per point less than 10 ms).</p>
    <p>To set sweep parameters, goto setup page. Click a button to run a sweep. Sweep is plotted on webpage at end
of
      sweep, and automatically downloaded.</p>
    <p>To bias at a specific point and monitor voltages, goto panel page.</p>
    <p>Zero bias: Minimum cell voltage with LMP91000 is 1% of 1.5V = 15 mV, gives slight discontinuity at origin.</-
p>
    <h3>Network:</h3>
    <p>If no WIFI connection is made with stored credentials, NanoStat goes into access point mode. Use the SSID
      "Nanostat" and enter credentials on website at 192.168.4.1.</p>
    <p>Once connected, on a laptop goto http://nanostat.local. On an Android mobile device, you need the actual IP
      address of the nanostat from your router. Contact your network administrator to find this from the router.</-
p>

    <h3>Calibration:</h3>
    <p>To calibrate, run "Cal" with WE disconnected from RE/CE. (Short RE/CE). Calibrations saved to flash, persist
      after reboot.</p>


      <h3>Choosing TIA resistance (Trans-Impedance Amplifier):</h3>
      <p>Due to limits on the ADC of the ESP32, you need to ensure -0.75 V < VTIA < 1 V, where VTIA = RTIA * WE
current.</p>


      <h3>Maximum # of points:</h3>
      <p>5000. Do not exceed. </p>

      <h3>Maximum cell voltage:</h3>
      <p>The LMP91000 can bias the cell up to 20% of the DAC output of the ESP32. The ESP32 DAC is very non-linear
above 3 V, and cannot go above 3.3 V which is the power supply voltage. Therefore the cell voltage is limited to 3
V * 20% = 600 mV, in either direction. </p>
```

```
    <h3>Sweep modes:</h3>

    <h6 class="mt-3">CV = Cyclic voltammetry.</h6>
    <a href="https://www.basinc.com/manuals/EC_epsilon/Techniques/CycVolt/cv">Bioanalytical Systems, Inc. tutorial
on
      CV</a><br>
    <img class="img-fluid" src="https://www.basinc.com/assets/img/manuals/EC_epsilon/Techniques/CycVolt/-
cv_wave_form.gif" alt="CV">


    <h6 class="mt-3">NPV = Normal pulsed voltammetry.</h6>
    <a href="https://www.basinc.com/manuals/EC_epsilon/techniques/Pulse/pulse#normal">Bioanalytical Systems, Inc.
      tutorial on NPV</a><br>
    <img class="img-fluid" src="https://www.basinc.com/assets/img/manuals/EC_epsilon/Techniques/Pulse/-
npv_wave_form.gif" alt="NPV">

    <h6 class="mt-3">SWV = Square wave voltammetry.</h6>
    <a href="https://www.basinc.com/manuals/EC_epsilon/Techniques/Pulse/pulse#square">Bioanalytical Systems, Inc.
      tutorial on SWV</a><br>
    <img class="img-fluid" src="https://www.basinc.com/assets/img/manuals/EC_epsilon/Techniques/Pulse/-
swv_wave_form.gif" alt="SWV">

    <h6 class="mt-3">CA = Chronoamperometry.</h6>
    <a href="https://www.basinc.com/manuals/EC_epsilon/Techniques/ChronoI/ca">Bioanalytical Systems, Inc. tutorial
on
      CA</a><br>
    <img class="img-fluid" src="https://www.basinc.com/assets/img/manuals/EC_epsilon/Techniques/ChronoI/-
ca_wave_form.gif" alt="CA">

    <h6 class="mt-3">DC = Fixed DC bias.</h6>
    <p>
      Sets dc bias voltage, reads ADC with various # readings per point {1, 5, 10, 50, 100, 500, 1000}, gives avg
and
      std deviation of ADC value in bits.
    </p>

    <h6 class="mt-3"6>IV = IV curve. </h6>

    <h6 class="mt-3">Cal = Calibrate NanoStat.</h6>


  </div>


  <!-- Page footer -->
  <!-- <footer class="container text-center myfooter fs-6 fw-lighter pt-3 fst-italic"> -->
  <footer class="container text-center  fs-6 fw-lighter pt-3 fst-italic">
    <p>NanoStat Rev. 3.5 @ BurkeLab 2021</p>
  </footer>

  <!-- Bootstrap JS -->
  <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.0-beta3/dist/js/bootstrap.bundle.min.js"
    integrity="sha384-JEW9xMcG8R+pH31jmWH6WWP0WintQrMb4s7ZOdauHnUtxwoG2vI5DkLtS3qm9Ekf"
    crossorigin="anonymous"></script>
  <!-- MDB Bootstrap JS -->
  <!-- MDB -->
  <!-- <script type="text/javascript" src="https://cdnjs.cloudflare.com/ajax/libs/mdb-ui-kit/3.3.0/mdb.min.js"></-
script> -->
</body>
```

```html
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>NanoStat
  </title>
  <!-- Bootstrap CSS -->
  <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.0-beta3/dist/css/bootstrap.min.css" rel="stylesheet"
    integrity="sha384-eOJMYsd53ii+scO/bJGFsiCZc+5NDVN2yr8+0RDqr0Ql0h+rP48ckxlpbzKgwra6" crossorigin="anonymous">
  <script src='https://cdn.jsdelivr.net/npm/chart.js@3.1.1/dist/chart.min.js'></script>
  <script type="text/javascript" src="websocketcode.js"></script>
  <script type="text/javascript" src="button_handle_JS.js"></script>
  <script src="https://cdn.plot.ly/plotly-latest.min.js"></script>
  <!-- <link rel="stylesheet" href="menubarstyle.css"> -->
  <!-- <link rel="stylesheet" href="CSS-Reset.css"> -->
  <!-- Google font: -->
  <link rel="preconnect" href="https://fonts.gstatic.com">
  <link href="https://fonts.googleapis.com/css2?family=Leckerli+One&display=swap" rel="stylesheet">
  <!-- Custom style: -->
  <link rel="stylesheet" href="nanostatstyle.css">
</head>


<body>

  <!-- Navigation bar -->
  <nav class="navbar navbar-expand-md navbar-dark bg-primary">
    <div class="container">
      <!-- <nav class="navbar navbar-expand-md navbar-light"> -->
      <!-- Brand icon: -->
      <a href="#" class="navbar-brand" style="font-family: 'Leckerli One', cursive;">NanoStat</a>
      <!-- Hamburger button: -->
      <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target="#toggleMobileMenu"
        aria-controls="toggleMobileMenu" aria-expanded="false" aria-table="Toggle navigation">
        <span class="navbar-toggler-icon"></span>
      </button>
      <!-- Menu items: -->
      <div class="collapse navbar-collapse" id="toggleMobileMenu">
        <ul class="navbar-nav ms-auto text-center">
          <li>
            <a class="nav-link active" href="index.html">Sweep</a>
          </li>
          <li>
            <a class="nav-link" href="setup.html">Setup</a>
          </li>
          <li>
            <a class="nav-link" href="panel.html">Panel</a>
          </li>
          <li>
            <a class="nav-link" href="download.html">Download</a>
          </li>

          <li class="nav-item dropdown">
            <a href="#" class="nav-link dropdown-toggle" id="navbarDropdown" role="button" data-bs-
toggle="dropdown"
              aria-expanded="false">System</a>
            <ul class="dropdown-menu" aria-labeledby="navbarDropdown">
              <li><a href="files.html" class="dropdown-item">Files</a></li>
              <li><a href="wifi.html" class="dropdown-item">Wifi</a></li>
              <li><a href="firmware.html" class="dropdown-item">Firmware</a></li>
              <li><a href="reboot.html" class="dropdown-item">Reboot</a></li>

            </ul>
          </li>



          <li>
            <a class="nav-link" href="help.html">Help</a>
          </li>
          <li>
            <a class="nav-link" href="about.html">About</a>
          </li>
        </ul>
      </div>
  </nav>

  <div class="container">
    <h1 class="text-center">Sweep</h1>

    <p>Current sweep mode:
    <span id="sweep_mode_id"> UNKNOWN</span>
    </p>
    <p><b>Please</b> <u>choose</u> from the following sweep <i>options</i>:</p>


    <button type="button" class="btn btn-primary my-2" onclick="on_Button_1_pressed()">CV!</button>
    <button type="button" class="btn btn-primary my-2" onclick="on_Button_2_pressed()">NPV!</button>
    <button type="button" class="btn btn-primary my-2" onclick="on_Button_3_pressed()">SWV!</button>
    <button type="button" class="btn btn-primary my-2" onclick="on_Button_4_pressed()">CA!</button>
    <button type="button" class="btn btn-primary my-2" onclick="on_Button_5_pressed()">DC!</button>
    <button type="button" class="btn btn-primary my-2" onclick="on_Button_6_pressed()">IV!</button>
    <button type="button" class="btn btn-primary my-2" onclick="on_Button_7_pressed()">Cal!</button>
    <button type="button" class="btn btn-primary my-2" onclick="on_Button_8_pressed()">Misc!</button>
    <!-- <button onclick="on_Button_9_pressed()">9!</button>
    <button onclick="on_Button_10_pressed()">10!</button> -->


    <h3>Sweep modes:</h3>
    <p>CV = Cyclic voltammetry.<br>
      NPV = Normal pulsed voltammetry.<br>
      SWV = Square wave voltammetry.<br>
      CA = Chronoamperometry.<br>
      DC = Fixed DC bias noise test.<br>
      IV = IV curve.<br>
      Cal = calibrate NanoStat.<br>
      Misc = Debug to console (serial).</p>


  </div>

  <div class="container">
    <div class="row">
      <div class="col-lg text-center">
        <!-- Current vs. voltage -->
        <div id="plotly-IV"></div>
      </div>
      <div class="col-lg text-center">
        <!-- Current vs. time -->
        <div id="plotly-IvsTime"></div>
      </div>
    </div>
  </div>


  <!-- Page footer -->
  <!-- <footer class="container text-center myfooter fs-6 fw-lighter pt-3 fst-italic"> -->
    <footer class="container text-center  fs-6 fw-lighter pt-3 fst-italic">
      <p>NanoStat Rev. 3.5 @ BurkeLab 2021</p>
    </footer>
```

```
<!-- Plotly JS -->
  <script type="text/javascr" src="plotlyplot.js"></script>



  <!-- Bootstrap JS -->
  <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.0-beta3/dist/js/bootstrap.bundle.min.js"
    integrity="sha384-JEW9xMcG8R+pH31jmWH6WWP0WintQrMb4s7ZOdauHnUtxwoG2vI5DkLtS3qm9Ekf"
    crossorigin="anonymous"></script>
</body>
```

```css
body {
    background-color: hsl(216, 99%, 90%);
}

.mybox1 {
    border: 1px white solid;
    background-color: aquamarine;
}

.mybox2 {
    border: 1px white solid;
    background-color: lightpink;
}


.mybox3 {
    border: 1px white solid;
    background-color: lightgray;
}
.mybox5 {
    border: 1px white solid;
    background-color: beige;
}
.mybox4 {
    border: 1px white solid;
    background-color: lightslategrey;
}
.mybox6 {
    border: 1px purple solid;
    background-color: beige;
    min-height: 55px;
}
.mybox7 {
    border: 1px purple solid;
    background-color: lightsalmon;
    min-height: 55px;
}


.myfooter {
    border: 1px white solid;
    background-color: pink;
}


input[type=number] {
  width: 6.375rem;
  border-radius: 0;
  /* background-color: #3CBC8D; */
}
```

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>NanoStat
  </title>
  <!-- Bootstrap CSS -->
  <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.0-beta3/dist/css/bootstrap.min.css" rel="stylesheet"
    integrity="sha384-eOJMYsd53ii+scO/bJGFsiCZc+5NDVN2yr8+0RDqr0Ql0h+rP48ckxlpbzKgwra6" crossorigin="anonymous">
  <script src='https://cdn.jsdelivr.net/npm/chart.js@3.1.1/dist/chart.min.js'></script>
  <script type="text/javascript" src="websocketcode.js"></script>
  <script type="text/javascript" src="button_handle_JS.js"></script>
  <script src="https://cdn.plot.ly/plotly-latest.min.js"></script>
  <!-- <link rel="stylesheet" href="menubarstyle.css"> -->
  <!-- <link rel="stylesheet" href="CSS-Reset.css"> -->
  <!-- Google font: -->
  <link rel="preconnect" href="https://fonts.gstatic.com">
  <link href="https://fonts.googleapis.com/css2?family=Leckerli+One&display=swap" rel="stylesheet">
  <!-- Custom style: -->
  <link rel="stylesheet" href="nanostatstyle.css">
</head>


<body>

  <!-- Navigation bar -->
  <nav class="navbar navbar-expand-md navbar-dark bg-primary">
    <div class="container">
      <!-- <nav class="navbar navbar-expand-md navbar-light"> -->
      <!-- Brand icon: -->
      <a href="#" class="navbar-brand" style="font-family: 'Leckerli One', cursive;">NanoStat</a>
      <!-- Hamburger button: -->
      <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target="#toggleMobileMenu"
        aria-controls="toggleMobileMenu" aria-expanded="false" aria-table="Toggle navigation">
        <span class="navbar-toggler-icon"></span>
      </button>
      <!-- Menu items: -->
      <div class="collapse navbar-collapse" id="toggleMobileMenu">
        <ul class="navbar-nav ms-auto text-center">
          <li>
            <a class="nav-link" href="index.html">Sweep</a>
          </li>
          <li>
            <a class="nav-link" href="setup.html">Setup</a>
          </li>
          <li>
            <a class="nav-link active" href="panel.html">Panel</a>
          </li>
          <li>
            <a class="nav-link" href="download.html">Download</a>
          </li>

          <li class="nav-item dropdown">
            <a href="#" class="nav-link dropdown-toggle" id="navbarDropdown" role="button" data-bs-
toggle="dropdown"
              aria-expanded="false">System</a>
            <ul class="dropdown-menu" aria-labeledby="navbarDropdown">
              <li><a href="files.html" class="dropdown-item">Files</a></li>
              <li><a href="wifi.html" class="dropdown-item">Wifi</a></li>
              <li><a href="firmware.html" class="dropdown-item">Firmware</a></li>
              <li><a href="reboot.html" class="dropdown-item">Reboot</a></li>

            </ul>
          </li>



          <li>
            <a class="nav-link" href="help.html">Help</a>
          </li>
          <li>
            <a class="nav-link" href="about.html">About</a>
          </li>
        </ul>
      </div>
  </nav>

  <header class="container text-center ">
    <h1>Control panel</h1>
    <h2>LMP91000</h2>
  </header>


  <div class="container">
    <div class="row mybox1">
      <!-- <div class="col-md p-0 text-center mybox1"> -->
      <div class="  col-md p-0 ">
        <h2> Controls </h2>


        <!-- Active -->
        <div class="div mybox6 mx-3">
          <div class="form-check form-check-inline">
            <input class="form-check-input p-0 border border-secondary" type="radio"
name="m_control_panel_is_active"
              id="m_control_panel_is_active_id" value="true" >
            <label class="form-check-label" for="m_control_panel_is_active_id">Active.</label>
          </div>
          <div class="form-check form-check-inline">
            <input class="form-check-input p-0 border border-secondary" type="radio"
name="m_control_panel_is_active"
              id="m_control_panel_is_inactive_id" value="false" checked>
            <label class="form-check-label" for="m_control_panel_is_inactive_id">Inactive.</label>
          </div>
        </div>

        <!--
        <input type="radio" id="m_control_panel_is_active_id" name="m_control_panel_is_active" value=true>
        <label for="m_control_panel_is_active_id">Active.</label><br>
        <input type="radio" id="m_control_panel_is_inactive_id" checked="checked" name="m_control_panel_is_active"
          value=false>
        <label for="m_control_panel_is_inactive_id">Inactive.</label><br> -->


        <!-- TIA gain -->
        <div class="div mybox6 mx-3">
          <div class="form-box ms-5 ms-md-0">
            <label class="form-label mb-0">TIA gain</label>
            <select class="form-control p-0 border border-secondary" name="sweep_param_lmpGain" id="lmpGain_id">
              <option value=7>350 kOhm</option>
              <option value=6>120 kOhm</option>
              <option value=5>35 kOhm</option>
              <option value=4>14 kOhm</option>
              <option value=3>7 kOhm</option>
              <option value=2>3.5 kOhm</option>
              <option value=1>2.75 kOhm</option>
              <option value=0>External resistor</option>
            </select>
          </div>
        </div>
        <div class="div mybox6 mx-3">
          <div class="form-box ms-5 ms-md-0">
            <label class="form-label mb-0">Cell set voltage (mV)</label>
            <input type="number" class="form-control p-0 border border-secondary" name="cell_voltage"
              id="cell_voltage_id" value=100 min=-792 max=792>
          </div>
        </div>


        <div class="div mybox6 mx-3">
          <div class="form-box ms-5 ms-md-0">
            <label class="form-label mb-0"># of readings to average per point.</label>
            <input type="number" class="form-control p-0 border border-secondary"
              name="num_readings_to_average_per_point" id="num_readings_to_average_per_point_id" value=1 min=1
              max=1000000>
          </div>
        </div>

        <div class="div mybox6 mx-3">
          <div class="form-box ms-5 ms-md-0">
            <label class="form-label mb-0">delay between points (ms)</label>
            <input type="number" class="form-control p-0 border border-secondary" name="delay_between_points_ms"
              id="delay_between_points_ms_id" value=500 min=1 max=1000>
          </div>
        </div>

        <div class="div mybox6 mx-3">
          <div class="form-box ms-5 ms-md-0">
            <label class="form-label mb-0">Max # of points to plot</label>
            <input type="number" class="form-control p-0 border border-secondary"
name="max_number_of_points_in_browser"
              id="max_number_of_points_in_browser_id" value=100 min=1 max=10000>
          </div>
        </div>


      </div>
      <div class="col-md  mybox2">
        <h2>Status</h2>

        <p>Current sweep mode:
          <span id="sweep_mode_id"> DORMANT</span>
        </p>

        <!-- determined by set voltage: -->
        <p>LMP91000 % setting =
          <span id="m_percentage_id">percent_setting</span>
        </p>
        <p>DAC voltage setting (mV) =
          <span id="m_dacVout_id">DACvoutmV</span>
        </p>
        <!-- determined at each point reading: -->
        <p> ADC average (bits) =
          <span id="analog_read_avg_bits_id">analog_read_avg_bits</span>
        </p>

        <p>ADC average (mV) =
          <span id="analog_read_avg_mV_id">analog_read_avg_mV</span>
        </p>
        <!-- for all the points on the browswer plot: -->
        <p>Average current (microA) =
          <span id="AVG_current_ID">UNKNOWN</span>
        </p>
        <p>Std. dev. current (microA) =
          <span id="STD_DEV_current_ID">UNKNOWN</span>
        </p>
      </div>

    </div>
  </div>





  </div>

  <!-- Scope -->
  <div class="container mybox3 text-center">
    <h2>Scope display</h2>
    <div id="plotly-scope-2yaxis"></div>
  </div>

    <!-- Page footer -->
  <!-- <footer class="container text-center myfooter fs-6 fw-lighter pt-3 fst-italic"> -->
    <footer class="container text-center  fs-6 fw-lighter pt-3 fst-italic">
      <p>NanoStat Rev. 3.5 @ BurkeLab 2021</p>
    </footer>

  <!-- Plotly JS -->
  <script type="text/javascr" src="plotlyscope.js"></script>
  <!-- Bootstrap JS -->
  <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.0-beta3/dist/js/bootstrap.bundle.min.js"
    integrity="sha384-JEW9xMcG8R+pH31jmWH6WWP0WintQrMb4s7ZOdauHnUtxwoG2vI5DkLtS3qm9Ekf"
    crossorigin="anonymous"></script>
  <!-- Control panel JS -->
  <script type="text/javascript" src="websckt_ctl_pan_LMP91000.js"></script>
</body>
```

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>NanoStat
  </title>
  <!-- Bootstrap CSS -->
  <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.0-beta3/dist/css/bootstrap.min.css" rel="stylesheet"
    integrity="sha384-eOJMYsd53ii+scO/bJGFsiCZc+5NDVN2yr8+0RDqr0Ql0h+rP48ckxlpbzKgwra6" crossorigin="anonymous">
  <script src='https://cdn.jsdelivr.net/npm/chart.js@3.1.1/dist/chart.min.js'></script>
  <script type="text/javascript" src="websocketcode.js"></script>
  <script type="text/javascript" src="button_handle_JS.js"></script>
  <script src="https://cdn.plot.ly/plotly-latest.min.js"></script>
  <!-- <link rel="stylesheet" href="menubarstyle.css"> -->
  <!-- <link rel="stylesheet" href="CSS-Reset.css"> -->
  <!-- Google font: -->
  <link rel="preconnect" href="https://fonts.gstatic.com">
  <link href="https://fonts.googleapis.com/css2?family=Leckerli+One&display=swap" rel="stylesheet">
    <!-- Custom style: -->
    <link rel="stylesheet" href="nanostatstyle.css">
</head>


<body>

  <!-- Navigation bar -->
  <nav class="navbar navbar-expand-md navbar-dark bg-primary">
    <div class="container">
      <!-- <nav class="navbar navbar-expand-md navbar-light"> -->
      <!-- Brand icon: -->
      <a href="#" class="navbar-brand" style="font-family: 'Leckerli One', cursive;">NanoStat</a>
      <!-- Hamburger button: -->
      <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target="#toggleMobileMenu"
        aria-controls="toggleMobileMenu" aria-expanded="false" aria-table="Toggle navigation">
        <span class="navbar-toggler-icon"></span>
      </button>
      <!-- Menu items: -->
      <div class="collapse navbar-collapse" id="toggleMobileMenu">
        <ul class="navbar-nav ms-auto text-center">
          <li>
            <a class="nav-link" href="index.html">Sweep</a>
          </li>
          <li>
            <a class="nav-link" href="setup.html">Setup</a>
          </li>
          <li>
            <a class="nav-link" href="panel.html">Panel</a>
          </li>
          <li>
            <a class="nav-link" href="download.html">Download</a>
          </li>

          <li class="nav-item dropdown">
            <a href="#" class="nav-link dropdown-toggle active" id="navbarDropdown" role="button" data-bs-
toggle="dropdown"
              aria-expanded="false">System</a>
            <ul class="dropdown-menu" aria-labeledby="navbarDropdown">
              <li><a href="files.html" class="dropdown-item">Files</a></li>
              <li><a href="wifi.html" class="dropdown-item">Wifi</a></li>
              <li><a href="firmware.html" class="dropdown-item">Firmware</a></li>
              <li><a href="reboot.html" class="dropdown-item active">Reboot</a></li>

            </ul>
          </li>



          <li>
            <a class="nav-link" href="help.html">Help</a>
          </li>
          <li>
            <a class="nav-link" href="about.html">About</a>
          </li>
        </ul>
      </div>
  </nav>


  <!-- <header class="container text-center mybox1"> -->
    <header class="container text-center ">
      <h1>Reboot</h1>
      <h2>Reboot the NanoStat.</h2>

    </header>



<main class="container">
  <div class="col text-center ">
    <!-- <button  class="btn btn-primary my-2">Download</button> -->
    <a class="btn btn-danger" href="rebootnanostat" role="button">Reboot</a>
  </div>
</main>

  <!-- Page footer -->
  <!-- <footer class="container text-center myfooter fs-6 fw-lighter pt-3 fst-italic"> -->
    <footer class="container text-center  fs-6 fw-lighter pt-3 fst-italic">
      <p>NanoStat Rev. 3.5 @ BurkeLab 2021</p>
    </footer>

  <!-- Bootstrap JS -->
  <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.0-beta3/dist/js/bootstrap.bundle.min.js"
    integrity="sha384-JEW9xMcG8R+pH31jmWH6WWP0WintQrMb4s7ZOdauHnUtxwoG2vI5DkLtS3qm9Ekf"
    crossorigin="anonymous"></script>
</body>
```

```json
{"ssid1":"",
"pass1":"",
"ssid2":"",
"pass2":"",
"ssid3":"",
"pass3":""}
```

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>NanoStat
  </title>
  <!-- Bootstrap CSS -->
  <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.0-beta3/dist/css/bootstrap.min.css" rel="stylesheet"
    integrity="sha384-eOJMYsd53ii+scO/bJGFsiCZc+5NDVN2yr8+0RDqr0Ql0h+rP48ckxlpbzKgwra6" crossorigin="anonymous">
  <script src='https://cdn.jsdelivr.net/npm/chart.js@3.1.1/dist/chart.min.js'></script>
  <!-- <script type="text/javascript" src="websocketcode.js"></script> -->
  <script type="text/javascript" src="button_handle_JS.js"></script>
  <script src="https://cdn.plot.ly/plotly-latest.min.js"></script>
  <!-- <link rel="stylesheet" href="menubarstyle.css"> -->
  <!-- <link rel="stylesheet" href="CSS-Reset.css"> -->
  <!-- Google font: -->
  <link rel="preconnect" href="https://fonts.gstatic.com">
  <link href="https://fonts.googleapis.com/css2?family=Leckerli+One&display=swap" rel="stylesheet">
  <!-- Custom style: -->
  <link rel="stylesheet" href="nanostatstyle.css">
</head>


<body>

  <!-- Navigation bar -->
  <nav class="navbar navbar-expand-md navbar-dark bg-primary">
    <div class="container">
      <!-- <nav class="navbar navbar-expand-md navbar-light"> -->
      <!-- Brand icon: -->
      <a href="#" class="navbar-brand" style="font-family: 'Leckerli One', cursive;">NanoStat</a>
      <!-- Hamburger button: -->
      <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target="#toggleMobileMenu"
        aria-controls="toggleMobileMenu" aria-expanded="false" aria-table="Toggle navigation">
        <span class="navbar-toggler-icon"></span>
      </button>
      <!-- Menu items: -->
      <div class="collapse navbar-collapse" id="toggleMobileMenu">
        <ul class="navbar-nav ms-auto text-center">
          <li>
            <a class="nav-link" href="index.html">Sweep</a>
          </li>
          <li>
            <a class="nav-link active" href="setup.html">Setup</a>
          </li>
          <li>
            <a class="nav-link" href="panel.html">Panel</a>
          </li>
          <li>
            <a class="nav-link" href="download.html">Download</a>
          </li>

          <li class="nav-item dropdown">
            <a href="#" class="nav-link dropdown-toggle" id="navbarDropdown" role="button" data-bs-
toggle="dropdown"
              aria-expanded="false">System</a>
            <ul class="dropdown-menu" aria-labeledby="navbarDropdown">
              <li><a href="files.html" class="dropdown-item">Files</a></li>
              <li><a href="wifi.html" class="dropdown-item">Wifi</a></li>
              <li><a href="firmware.html" class="dropdown-item">Firmware</a></li>
              <li><a href="reboot.html" class="dropdown-item">Reboot</a></li>

            </ul>
          </li>



          <li>
            <a class="nav-link" href="help.html">Help</a>
          </li>
          <li>
            <a class="nav-link" href="about.html">About</a>
          </li>
        </ul>
      </div>
  </nav>

  <!-- Page title -->
  <!-- <header class="container text-center mybox1"> -->
  <header class="container text-center ">
    <h1>Setup</h1>
    <h2>Sweep settings</h2>
    <p>See help for limits on # of points and max current for each gain setting.</p>
  </header>


  <!-- First buttons: -->
  <!-- <div class="container mybox2"> -->
  <div class="container">

    <form action="/actionpage.html" method="POST" enctype="application/json">

      <!-- <div class="form-box ms-5 ms-md-0-button">


        </div> -->




      <div class="container">
        <div class="row">

          <!-- <div class="col text-center border border-secondary"> -->
          <div class="col text-center ">
            <button type="reset" class="btn btn-primary my-2">Reset</button>
          </div>
          <div class="col text-center ">
            <button type="submit" class="btn btn-primary my-2">Submit</button>
          </div>

        </div>
      </div>

  </div>

  <!-- Inputs -->
  <!-- <div class="container mybox3"> -->
  <div class="container">

    <div class="row">

      <div class="col-md p-0">

        <!-- General -->
        <div class="div mybox7 ps-1 pt-1 ">
          <h6>General</h6>
        </div>
```

```
        <div class="div mybox6">
          <div class="form-box ms-5 ms-md-0">
            <label class="form-label mb-0">TIA gain</label>
            <select class="form-control p-0 border border-secondary" name="sweep_param_lmpGain" id="lmpGain_id">
              <option value=7>350 kOhm</option>
              <option value=6>120 kOhm</option>
              <option value=5>35 kOhm</option>
              <option value=4>14 kOhm</option>
              <option value=3>7 kOhm</option>
              <option value=2>3.5 kOhm</option>
              <option value=1>2.75 kOhm</option>
              <option value=0>External resistor</option>
            </select>
          </div>
        </div>


        <div class="div mybox6">
          <div class="form-check form-check-inline">
            <input class="form-check-input p-0 border border-secondary" type="radio"  name="sweep_param_setToZero"
              id="setToZero_id" value="true" checked>
            <label class="form-check-label" for="setToZero_id">Zero at end.</label>
          </div>
          <div class="form-check form-check-inline">
            <input class="form-check-input p-0 border border-secondary" type="radio"  name="sweep_param_setToZero"
              id="setToZero_id" value="false">
            <label class="form-check-label" for="setToZero_id">Not 0 at end.</label>
          </div>
        </div>


        <!-- SWV -->
        <div class="div mybox7 ps-1 pt-1 ">
          <h6>Square wave voltammetry</h6>
        </div>

        <div class="div mybox6  ">

          <div class="form-box ms-5 ms-md-0">
            <!-- <div class="form-box ms-5 ms-md-0  d-flex justify-content-center text-center"> -->
            <label class="form-label mb-0   ">Start voltage (mV)</label>
            <input type="number" class="  form-control p-0 border border-secondary" name="sweep_param_startV_SWV"
              id="sweep_param_startV_SWV_id" value=-200 min=-600 max=600>
          </div>
        </div>


        <div class="div mybox6">
          <div class="form-box ms-5 ms-md-0">
            <label class="form-label mb-0">End voltage (mV)</label>
            <input type="number" class="form-control p-0 border border-secondary" name="sweep_param_endV_SWV"
              id="sweep_param_endV_SWV_id" value=200 min=-600 max=600>
          </div>
        </div>


        <div class="div mybox6">
          <div class="form-box ms-5 ms-md-0">
            <label class="form-label mb-0">Pulse ampl. (mV)</label>
            <input type="number" class="form-control p-0 border border-secondary" name="sweep_param_pulseAmp_SWV"
              id="sweep_param_pulseAmp_SWV_id" value=20 min=1 max=200>
          </div>
        </div>


        <div class="div mybox6">
          <div class="form-box ms-5 ms-md-0">
            <label class="form-label mb-0">Step voltage (mV)</label>
            <input type="number" class="form-control p-0 border border-secondary" name="sweep_param_stepV_SWV"
              id="sweep_param_stepV_SWV_id" value=5 min=1 max=1000>
          </div>
        </div>

        <div class="div mybox6">
          <div class="form-box ms-5 ms-md-0">
            <label class="form-label mb-0">Frequency (Hz)</label>
            <input type="number" class="form-control p-0 border border-secondary" name="sweep_param_freq_SWV"
              id="sweep_param_freq_SWV_id" value=10 min=1 max=100000>
          </div>
        </div>


      </div>
      <!-- CV -->
      <div class="col-md p-0">

        <div class="div mybox7 ps-1 pt-1 ">
          <h6>Cyclic voltammetry</h6>
        </div>

        <div class="div mybox6">
          <div class="form-box ms-5 ms-md-0">
            <label class="form-label mb-0"># cycles</label>
            <input type="number" class="form-control p-0 border border-secondary" name="sweep_param_cycles_CV"
              id="sweep_param_startV_CV_id" value=3 min=1>
          </div>
        </div>

        <div class="div mybox6">
          <div class="form-box ms-5 ms-md-0">
            <label class="form-label mb-0">Start voltage (mV)</label>
            <input type="number" class="form-control p-0 border border-secondary" name="sweep_param_startV_CV"
              id="sweep_param_startV_CV_id" value=0 min=-600 max=600>
          </div>
        </div>

        <div class="div mybox6">
          <div class="form-box ms-5 ms-md-0">
            <label class="form-label mb-0">End voltage (mV)</label>
            <input type="number" class="form-control p-0 border border-secondary" name="sweep_param_endV_CV"
              id="sweep_param_endV_CV_id" value=0 min=-600 max=600>
          </div>
        </div>

        <div class="div mybox6">
          <div class="form-box ms-5 ms-md-0">
            <label class="form-label mb-0">Max. voltage (mV)</label>
            <input type="number" class="form-control p-0 border border-secondary" name="sweep_param_vertex1_CV"
              id="sweep_param_vertex1_CV_id" value=100 min=-600 max=600>
          </div>
        </div>

        <div class="div mybox6">
          <div class="form-box ms-5 ms-md-0">
            <label class="form-label mb-0">Min. voltage (mV)</label>
            <input type="number" class="form-control p-0 border border-secondary" name="sweep_param_vertex2_CV"
              id="sweep_param_vertex1_CV_id" value=-100 min=-600 max=600>
          </div>
        </div>

        <div class="div mybox6">
          <div class="form-box ms-5 ms-md-0">
            <label class="form-label mb-0">Step voltage (mV)</label>
            <input type="number" class="form-control p-0 border border-secondary" name="sweep_param_stepV_CV"
```

```
            id="sweep_param_stepV_CV_id" value=5 min=1 max=100>
          </div>
        </div>

        <div class="div mybox6">
          <div class="form-box ms-5 ms-md-0">
            <label class="form-label mb-0">Scan rate (mV/s)</label>
            <input type="number" class="form-control p-0 border border-secondary" name="sweep_param_rate_CV"
              id="sweep_param_rate_CV_id" value=100 min=1 max=10000>
          </div>
        </div>

      </div>
      <!-- IV curve -->









      <div class="col-md p-0">

        <div class="div mybox7 ps-1 pt-1 ">
          <h6>IV Curve</h6>
        </div>

        <div class="div mybox6">
          <div class="form-box ms-5 ms-md-0">
            <label class="form-label mb-0">Start voltage (mV)</label>
            <input type="number" class="form-control p-0 border border-secondary" name="sweep_param_startV_IV"
              id="sweep_param_startV_IV_id" value=-200 min=-600 max=600>
          </div>
        </div>


        <div class="div mybox6">
          <div class="form-box ms-5 ms-md-0">
            <label class="form-label mb-0">End voltage (mV)</label>
            <input type="number" class="form-control p-0 border border-secondary" name="sweep_param_endV_IV"
              id="sweep_param_endV_IV_id" value=200 min=-600 max=600>
          </div>
        </div>


        <div class="div mybox6">
          <div class="form-box ms-5 ms-md-0">
            <label class="form-label mb-0">Delay time (ms)</label>
            <input type="number" class="form-control p-0 border border-secondary"
name="sweep_param_delayTime_ms_IV"
              id="sweep_param_delayTime_ms_IV_id" value=50 min=1 max=1000000>
          </div>
        </div>

        <div class="div mybox6">
          <div class="form-box ms-5 ms-md-0">
            <label class="form-label mb-0"># of points</label>
            <input type="number" class="form-control p-0 border border-secondary" name="sweep_param_numPoints_IV"
              id="sweep_param_numPoints_IV_id" value=701 min=1 max=6000>
          </div>
        </div>
        <!-- NOISE -->


        <div class="div mybox7 ps-1 pt-1 ">
```

```
          <h6>DC Noise</h6>
        </div>

        <div class="div mybox6">
          <div class="form-box ms-5 ms-md-0">
            <label class="form-label mb-0">Bias voltage (mV)</label>
            <input type="number" class="form-control p-0 border border-secondary"
name="sweep_param_biasV_noisetest"
              id="sweep_param_biasV_noisetest_id" value=100 min=-600 max=600>
          </div>
        </div>

        <div class="div mybox6">
          <div class="form-box ms-5 ms-md-0">
            <label class="form-label mb-0"># of points</label>
            <input type="number" class="form-control p-0 border border-secondary"
name="sweep_param_numPoints_noisetest"
              id="sweep_param_numPoints_noisetest_id" value=100 min=1 max=100000>
          </div>
        </div>

        <div class="div mybox6">
          <div class="form-box ms-5 ms-md-0">
            <label class="form-label mb-0">Delay time (ms)</label>
            <input type="number" class="form-control p-0 border border-secondary"
              name="sweep_param_delayTime_ms_noisetest" id="sweep_param_delayTime_ms_noisetest_id" value=50 min=1
              max=1000000>
          </div>
        </div>





      </div>





      <!-- NPV -->
      <div class="col-md p-0">

        <div class="div mybox7 ps-1 pt-1 ">
          <h6>Normal Pulse Voltammetry</h6>
        </div>







        <div class="div mybox6">
          <div class="form-box ms-5 ms-md-0">
            <label class="form-label mb-0">Start voltage (mV)</label>
            <input type="number" class="form-control p-0 border border-secondary" name="sweep_param_startV_NPV"
              id="sweep_param_startV_NPV_id" value=-200 min=-600 max=600>
          </div>
        </div>



        <div class="div mybox6">
          <div class="form-box ms-5 ms-md-0">
            <label class="form-label mb-0">Pulse amp. (mV)</label>
            <input type="number" class="form-control p-0 border border-secondary" name="sweep_param_pulseAmp_NPV"
              id="sweep_param_pulseAmp_NPV_id" value=20 min=1 max=1000>
          </div>
        </div>



        <div class="div mybox6">
          <div class="form-box ms-5 ms-md-0">
            <label class="form-label mb-0">Pulse width (ms)</label>
            <input type="number" class="form-control p-0 border border-secondary" name="sweep_param_width_NPV"
              id="sweep_param_width_NPV_id" value=50 min=1 max=1000>
          </div>
        </div>


        <div class="div mybox6">
          <div class="form-box ms-5 ms-md-0">
            <label class="form-label mb-0">End voltage (mV)</label>
            <input type="number" class="form-control p-0 border border-secondary" name="sweep_param_endV_NPV"
              id="sweep_param_endV_NPV_id" value=200 min=-600 max=600>
          </div>
        </div>




        <div class="div mybox6">
          <div class="form-box ms-5 ms-md-0">
            <label class="form-label mb-0">Pulse period (ms)</label>
            <input type="number" class="form-control p-0 border border-secondary" name="sweep_param_period_NPV"
              id="sweep_param_period_NPV_id" value=200 min=1 max=1000>
          </div>
        </div>


        <div class="div mybox6">
          <div class="form-box ms-5 ms-md-0">
            <label class="form-label mb-0">Quiet time (ms)</label>
            <input type="number" class="form-control p-0 border border-secondary" name="sweep_param_quietTime_NPV"
              id="sweep_param_quietTime_NPV_id" value=1000 min=1 max=100000>
          </div>
        </div>


        <div class="div mybox7 ps-1 pt-1 ">
          <h6>Calibration</h6>
        </div>

        <div class="div mybox6">
          <div class="form-box ms-5 ms-md-0">
            <label class="form-label mb-0">Delay time (ms)</label>
            <input type="number" class="form-control p-0 border border-secondary"
name="sweep_param_delayTime_ms_CAL"
              id="sweep_param_delayTime_ms_CAL_id" value=50 min=1 max=1000000>
          </div>
        </div>


      </div>






      <!-- CA -->
      <div class="col-md p-0">

        <div class="div mybox7 ps-1 pt-1 ">
          <h6>Chronoamperometry</h6>
        </div>

        <div class="div mybox6">
          <div class="form-box ms-5 ms-md-0">
            <label class="form-label mb-0">Initial voltage (mV)</label>
            <input type="number" class="form-control p-0 border border-secondary" name="sweep_param_pre_stepV_CA"
              id="sweep_param_pre_stepV_CA_id" value=50 min=1 max=1000>
          </div>
        </div>


        <div class="div mybox6">
          <div class="form-box ms-5 ms-md-0">
            <label class="form-label mb-0">Quiet time (ms)</label>
            <input type="number" class="form-control p-0 border border-secondary" name="sweep_param_quietTime_CA"
              id="sweep_param_quietTime_CA_id" value=2000 min=1 max=100000>
          </div>
        </div>




        <div class="div mybox6">
          <div class="form-box ms-5 ms-md-0">
            <label class="form-label mb-0">1st step volt. (mV)</label>
            <input type="number" class="form-control p-0 border border-secondary" name="sweep_param_V1_CA"
              id="sweep_param_V1_CA_id" value=50 min=-600 max=600>
          </div>
        </div>




        <div class="div mybox6">
          <div class="form-box ms-5 ms-md-0">
            <label class="form-label mb-0">1st step time (mS)</label>
            <input type="number" class="form-control p-0 border border-secondary" name="sweep_param_t1_CA"
              id="sweep_param_t1_CA_id" value=2000 min=1 max=100000>
          </div>
        </div>

        <div class="div mybox6">
          <div class="form-box ms-5 ms-md-0">
            <label class="form-label mb-0">2nd step volt. (mV)</label>
            <input type="number" class="form-control p-0 border border-secondary" name="sweep_param_V2_CA"
              id="sweep_param_V2_CA_id" value=50 min=-600 max=600>
          </div>
        </div>


        <div class="div mybox6">
          <div class="form-box ms-5 ms-md-0">
            <label class="form-label mb-0">2nd step time (ms)</label>
            <input type="number" class="form-control p-0 border border-secondary" name="sweep_param_t2_CA"
              id="sweep_param_t2_CA_id" value=2000 min=1 max=100000>
          </div>
        </div>



        <div class="div mybox6">
          <div class="form-box ms-5 ms-md-0">
            <label class="form-label mb-0"># of samples per step</label>
            <input type="number" class="form-control p-0 border border-secondary" name="sweep_param_samples_CA"
              id="sweep_param_samples_CA_id" value=100 min=1 max=1666>
          </div>
        </div>
```

```
    </div>



    <!-- XYZ -->
    <!-- <div class="col-md p-0">

      <div class="div mybox7 ">
        <h6>XYZ</h6>
      </div>

      <div class="div mybox6">
        <div class="form-box ms-5 ms-md-0">
          <label class="form-label mb-0">xyz</label>
          <input type="number" class="form-control p-0 border border-secondary" name="sweep_param_xyz_CV"
            id="sweep_param_xyz_CV_id" value=xyz min=xyz max=xyz>
        </div>
      </div>

      <div class="div mybox6">
        <div class="form-box ms-5 ms-md-0">
          <label class="form-label mb-0">xyz</label>
          <input type="number" class="form-control p-0 border border-secondary" name="sweep_param_xyz_CV"
            id="sweep_param_xyz_CV_id" value=xyz min=xyz max=xyz>
        </div>
      </div>

      <div class="div mybox6">
        <div class="form-box ms-5 ms-md-0">
          <label class="form-label mb-0">xyz</label>
          <input type="number" class="form-control p-0 border border-secondary" name="sweep_param_xyz_CV"
            id="sweep_param_xyz_CV_id" value=xyz min=xyz max=xyz>
        </div>
      </div>

      <div class="div mybox6">
        <div class="form-box ms-5 ms-md-0">
          <label class="form-label mb-0">xyz</label>
          <input type="number" class="form-control p-0 border border-secondary" name="sweep_param_xyz_CV"
            id="sweep_param_xyz_CV_id" value=xyz min=xyz max=xyz>
        </div>
      </div>

      <div class="div mybox6">
        <div class="form-box ms-5 ms-md-0">
          <label class="form-label mb-0">xyz</label>
          <input type="number" class="form-control p-0 border border-secondary" name="sweep_param_xyz_CV"
            id="sweep_param_xyz_CV_id" value=xyz min=xyz max=xyz>
        </div>
      </div>

      <div class="div mybox6">
        <div class="form-box ms-5 ms-md-0">
          <label class="form-label mb-0">xyz</label>
          <input type="number" class="form-control p-0 border border-secondary" name="sweep_param_xyz_CV"
            id="sweep_param_xyz_CV_id" value=xyz min=xyz max=xyz>
        </div>
      </div>

      <div class="div mybox6">
        <div class="form-box ms-5 ms-md-0">
          <label class="form-label mb-0">xyz</label>
          <input type="number" class="form-control p-0 border border-secondary" name="sweep_param_xyz_CV"
```

```
              id="sweep_param_xyz_CV_id" value=xyz min=xyz max=xyz>
          </div>
        </div>

      </div> -->


    </div>
    </form>
  </div>


  <!-- Page footer -->
  <!-- <footer class="container text-center myfooter fs-6 fw-lighter pt-3 fst-italic"> -->
  <footer class="container text-center  fs-6 fw-lighter pt-3 fst-italic">
    <p>NanoStat Rev. 3.5 @ BurkeLab 2021</p>
  </footer>


  <!-- Bootstrap JS -->
  <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.0-beta3/dist/js/bootstrap.bundle.min.js"
    integrity="sha384-JEW9xMcG8R+pH31jmWH6WWP0WintQrMb4s7ZOdauHnUtxwoG2vI5DkLtS3qm9Ekf"
    crossorigin="anonymous"></script>
</body>
```

```html
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>NanoStat
  </title>
  <!-- Bootstrap CSS -->
  <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.0-beta3/dist/css/bootstrap.min.css" rel="stylesheet"
    integrity="sha384-eOJMYsd53ii+scO/bJGFsiCZc+5NDVN2yr8+0RDqr0Ql0h+rP48ckxlpbzKgwra6" crossorigin="anonymous">
  <script src='https://cdn.jsdelivr.net/npm/chart.js@3.1.1/dist/chart.min.js'></script>
  <script type="text/javascript" src="websocketcode.js"></script>
  <script type="text/javascript" src="button_handle_JS.js"></script>
  <script src="https://cdn.plot.ly/plotly-latest.min.js"></script>
  <!-- <link rel="stylesheet" href="menubarstyle.css"> -->
  <!-- <link rel="stylesheet" href="CSS-Reset.css"> -->
  <!-- Google font: -->
  <link rel="preconnect" href="https://fonts.gstatic.com">
  <link href="https://fonts.googleapis.com/css2?family=Leckerli+One&display=swap" rel="stylesheet">
  <!-- MDB Bootstrap CSS -->
  <!-- Font Awesome -->
  <link href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/5.15.1/css/all.min.css" rel="stylesheet" />
  <!-- Google Fonts -->
  <link href="https://fonts.googleapis.com/css?family=Roboto:300,400,500,700&display=swap" rel="stylesheet" />
  <!-- MDB -->
  <!-- <link href="https://cdnjs.cloudflare.com/ajax/libs/mdb-ui-kit/3.3.0/mdb.min.css" rel="stylesheet" /> -->
  <!-- Custom style: -->
  <link rel="stylesheet" href="nanostatstyle.css">
</head>


<!-- <body style="background-color: lightblue;"> -->

<body>

  <!-- Navigation bar -->
  <nav class="navbar navbar-expand-md navbar-dark bg-primary">
    <div class="container">
      <!-- <nav class="navbar navbar-expand-md navbar-light"> -->
      <!-- Brand icon: -->
      <a href="#" class="navbar-brand" style="font-family: 'Leckerli One', cursive;">NanoStat</a>
      <!-- Hamburger button: -->
      <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target="#toggleMobileMenu"
        aria-controls="toggleMobileMenu" aria-expanded="false" aria-table="Toggle navigation">
        <span class="navbar-toggler-icon"></span>
      </button>
      <!-- Menu items: -->
      <div class="collapse navbar-collapse" id="toggleMobileMenu">
        <ul class="navbar-nav ms-auto text-center">
          <li>
            <a class="nav-link" href="index.html">Sweep</a>
          </li>
          <li>
            <a class="nav-link" href="setup.html">Setup</a>
          </li>
          <li>
            <a class="nav-link" href="panel.html">Panel</a>
          </li>
          <li>
            <a class="nav-link" href="download.html">Download</a>
          </li>

          <li class="nav-item dropdown">
            <a href="#" class="nav-link dropdown-toggle" id="navbarDropdown" role="button" data-bs-
toggle="dropdown"
```

```
              aria-expanded="false">System</a>
            <ul class="dropdown-menu" aria-labeledby="navbarDropdown">
              <li><a href="files.html" class="dropdown-item">Files</a></li>
              <li><a href="wifi.html" class="dropdown-item">Wifi</a></li>
              <li><a href="firmware.html" class="dropdown-item">Firmware</a></li>
              <li><a href="reboot.html" class="dropdown-item">Reboot</a></li>

            </ul>

          </li>



          <li>
            <a class="nav-link" href="help.html">Help</a>
          </li>
          <li>
            <a class="nav-link" href="about.html">About</a>
          </li>
        </ul>
      </div>
  </nav>

  <!-- #0D6EFD is color of Bootstrap menubar "primary" -->
  <!--  hsl(216, 98%, 52%) -->

  <!-- Page title -->
  <!-- <header class="container text-center mybox1"> -->
    <header class="container text-center ">
      <h1>Setup</h1>
      <h2>Sweep settings</h2>
    </header>


  <!-- Page footer -->
  <!-- <footer class="container text-center myfooter fs-6 fw-lighter pt-3 fst-italic"> -->
    <footer class="container text-center  fs-6 fw-lighter pt-3 fst-italic">
      <p>NanoStat Rev. 3.5 @ BurkeLab 2021</p>
    </footer>

  <!-- Bootstrap JS -->
  <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.0-beta3/dist/js/bootstrap.bundle.min.js"
    integrity="sha384-JEW9xMcG8R+pH31jmWH6WWP0WintQrMb4s7ZOdauHnUtxwoG2vI5DkLtS3qm9Ekf"
    crossorigin="anonymous"></script>
  <!-- MDB Bootstrap JS -->
  <!-- MDB -->
  <!-- <script type="text/javascript" src="https://cdnjs.cloudflare.com/ajax/libs/mdb-ui-kit/3.3.0/mdb.min.js"></-
script> -->
</body>
```

```javascript
// Javascript code to set up a websocket for the LMP91000 control panel.


var m_websocket;
var m_url_JS = "ws://nanostat.local:81/";
var button;
var canvas;
var context;
var maxDataPoints = 100;
var sum = 0;
var avg_current = 0;
var std_dev_current =0;
var total_number_of_points_recieved_so_far = 0;

// initialize plotly plot

// var trace_scope = {
//     x: [],
//     y: [],
//     mode: 'markers',
//     type: 'scatter'
// };
// var data_scope = [trace_scope];
// Plotly.newPlot('plotly-scope', data_scope);

var trace_scope_current = {
    x: [],
    y: [],
    mode: 'markers',
    type: 'scatter',
    name: "Current"
};
var trace_scope_voltage = {
    x: [],
    y: [],
    mode: 'markers',
    type: 'scatter',
    yaxis: 'y2',
    name: "Voltage"
};

var data_IVvsTime_scope = [trace_scope_current, trace_scope_voltage];
var m_2yaxis_layout = {
    margin: {
        l: 50,
        r: 5,
        b: 50,
        t: 1,
        pad: 4
    },
    xaxis: {
        title: { text: 'Time (ms)' }
    },
    yaxis: { title: 'Current (microA)' },
    yaxis2: {
        title: 'Voltage (mV)',
        titlefont: { color: 'rgb(148, 103, 189)' },
        tickfont: { color: 'rgb(148, 103, 189)' },
        overlaying: 'y',
        side: 'right'
    }
};

Plotly.newPlot('plotly-scope-2yaxis', data_IVvsTime_scope, m_2yaxis_layout, { scrollZoom: true, editable: true,
responsive: true });
// Plotly.newPlot('plotly-scope-2yaxis', data_IVvsTime_scope);
// This is called when the page finishes loading
function init() {

    // Assign page elements to variables
    button = document.getElementById("toggleButton");
    canvas = document.getElementById("led");

    // Draw circle in canvas
    // context = canvas.getContext("2d");
    // context.arc(25, 25, 15, 0, Math.PI * 2, false);
    // context.lineWidth = 3;
    // context.strokeStyle = "black";
    // context.stroke();
    // context.fillStyle = "black";
    // context.fill();

    wsConnect(m_url_JS);   // Connect to WebSocket server


}

// Call this to connect to the WebSocket server
function wsConnect(m_url_JS) {

    // Connect to WebSocket server
    m_url_JS = "ws://" + window.location.hostname + ":81/"
    m_websocket = new WebSocket(m_url_JS);

    // Assign callbacks
    m_websocket.onopen = function (evt) { onOpen(evt) };
    m_websocket.onclose = function (evt) { onClose(evt) };
    m_websocket.onmessage = function (evt) { onMessage(evt) };
    m_websocket.onerror = function (evt) { onError(evt) };

}

// Called when a WebSocket connection is established with the server
function onOpen(evt) {

    // Log connection state
    console.log("Connected");

    // initialize values to front panel browswer:
    doSend("{\"change_cell_voltage_to\":" + document.getElementById("cell_voltage_id").value + "}");
    doSend("{\"change_num_readings_to_average_per_point_to\":" +
document.getElementById("num_readings_to_average_per_point_id").value + "}");
    doSend("{\"change_lmpGain_to\":" + document.getElementById("lmpGain_id").value + "}");
    doSend("{\"change_control_panel_is_active_to\":false}");
    doSend("{\"change_delay_between_points_ms_to\":" + document.getElementById("delay_between_points_ms_id").value
+ "}");

}

// Called when the WebSocket connection is closed
function onClose(evt) {

    // Log disconnection state
    console.log("Disconnected");



    // Try to reconnect after a few seconds
    setTimeout(function () { wsConnect(m_url_JS) }, 2000);
}



// Called when a message is received from the server
function onMessage(evt) {

    // Print out our received message
    // console.log("Received: " + evt.data);
    var m_json_obj = JSON.parse(evt.data);
    // console.log(m_json_obj);
    if ('is_sweeping' in m_json_obj) {// changin in is sweeping status, update indicator on browswer page...
        // do something
        if (m_json_obj.is_sweeping == true) { // mode is sweeping
            // do something
            document.getElementById('sweep_mode_id').innerHTML = "<span style=\"color:red\">SWEEPING</span>";
            console.log("need to update indicator to true...");
        }
        if (m_json_obj.is_sweeping == false) { // mode is sweeping
            // do something
            document.getElementById('sweep_mode_id').innerHTML = "<span style=\"color:green\">IDLE</span>";
            console.log("need to update indicator to false...");
        }
    };

    if ('dacVout' in m_json_obj) {// update dacVout and percentage indicators on browswer page...
        // do something
        var m_dacVout = m_json_obj.dacVout;
        var m_percentage = m_json_obj.percentage;
        document.getElementById('m_dacVout_id').innerHTML = m_dacVout;
        document.getElementById('m_percentage_id').innerHTML = m_percentage;
    };

    if ('volts' in m_json_obj) {// next time point, parse and add to graph
        var m_voltage_point = m_json_obj.volts;
        var m_current_point = m_json_obj.amps;
        var m_time_point = m_json_obj.time;
        var m_analog_read_avg_bits = m_json_obj.analog_read_avg_bits;
        var m_analog_read_avg_mV = m_json_obj.analog_read_avg_mV;
        document.getElementById('analog_read_avg_bits_id').innerHTML = m_analog_read_avg_bits;
        document.getElementById('analog_read_avg_mV_id').innerHTML = m_analog_read_avg_mV;

        if (trace_scope_current.x.length > maxDataPoints) {
            trace_scope_current.x.shift();
            trace_scope_current.y.shift();
            trace_scope_voltage.x.shift();
            trace_scope_voltage.y.shift();
        }
        trace_scope_current.x.push(m_time_point);
        trace_scope_current.y.push(m_current_point);
        trace_scope_voltage.x.push(m_time_point);
        trace_scope_voltage.y.push(m_voltage_point);
        var data_IVvsTime_scope = [trace_scope_current, trace_scope_voltage];
        Plotly.newPlot('plotly-scope-2yaxis', data_IVvsTime_scope, m_2yaxis_layout, { scrollZoom: true, editable:
true, responsive: true });
        total_number_of_points_recieved_so_far++;
        update_average_current();
        // if (Number.isInteger(total_number_of_points_recieved_so_far/100)) { // update every 100 points
        //     update_average_current();
        // };
    };




}
```

```
// Called when a WebSocket error occurs
function onError(evt) {
    console.log("ERROR: " + evt.data);
}

// Sends a message to the server (and prints it to the console)
function doSend(message) {
    console.log("Sending: " + message);
    m_websocket.send(message);
}


// Called whenever the HTML button is pressed
function onPress() {
    doSend("Button pressed. Sending this message over websocket from html page to ESP32 websocket server....");
}
// Call the init function as soon as the page loads
window.addEventListener("load", init, false);

// Set listeners to send UI inputs to ESP32:
document.getElementById("cell_voltage_id").addEventListener("change", respond_to_cell_voltage_input_change);

function respond_to_cell_voltage_input_change() { // tell main.cpp to change cell voltage
    // console.log("cell voltage input changed to");
    // console.log(document.getElementById("cell_voltage_id").value);
    // send message via websockets to server
    var cell_voltage_JSON_command;
    cell_voltage_JSON_command = "{\"change_cell_voltage_to\":" + document.getElementById("cell_voltage_id").value
+ "}";
    // console.log("Prototype JSON string to send=");
    // console.log(cell_voltage_JSON_command);
    doSend(cell_voltage_JSON_command);
}

document.getElementById("num_readings_to_average_per_point_id").addEventListener("change",
respond_to_num_readings_to_average_per_point_change);

function respond_to_num_readings_to_average_per_point_change() { // tell main.cpp to change num readings per point
    var cell_voltage_JSON_command;
    cell_voltage_JSON_command = "{\"change_num_readings_to_average_per_point_to\":" +
document.getElementById("num_readings_to_average_per_point_id").value + "}";
    doSend(cell_voltage_JSON_command);
}

document.getElementById("delay_between_points_ms_id").addEventListener("change",
respond_to_delay_between_points_ms_change);

function respond_to_delay_between_points_ms_change() { // tell main.cpp to change num readings per point
    var cell_voltage_JSON_command;
    cell_voltage_JSON_command = "{\"change_delay_between_points_ms_to\":" +
document.getElementById("delay_between_points_ms_id").value + "}";
    doSend(cell_voltage_JSON_command);
}

document.getElementById("lmpGain_id").addEventListener("change", respond_to_lmpGain_change);

function respond_to_lmpGain_change() { // tell main.cpp to change num readings per point
    var cell_voltage_JSON_command;
    cell_voltage_JSON_command = "{\"change_lmpGain_to\":" + document.getElementById("lmpGain_id").value + "}";
    doSend(cell_voltage_JSON_command);
}

document.getElementById("m_control_panel_is_active_id").addEventListener("change",
respond_to_m_control_panel_is_active_id_change);
```

```
function respond_to_m_control_panel_is_active_id_change() {
    var control_panel_is_active_JSON_command;
    control_panel_is_active_JSON_command = "{\"change_control_panel_is_active_to\":true}";
    doSend(control_panel_is_active_JSON_command);
    // also send all the parameters for the biasing: 1) TIA gain 2) cell voltage 3) # readings to avg 4) delay...
    respond_to_delay_between_points_ms_change();
    respond_to_num_readings_to_average_per_point_change();
    respond_to_lmpGain_change()
    respond_to_cell_voltage_input_change();
}


document.getElementById("m_control_panel_is_inactive_id").addEventListener("change",
respond_to_m_control_panel_is_inactive_id_change);

function respond_to_m_control_panel_is_inactive_id_change() {
    var control_panel_is_active_JSON_command;
    control_panel_is_active_JSON_command = "{\"change_control_panel_is_active_to\":false}";
    doSend(control_panel_is_active_JSON_command);
}


document.getElementById("max_number_of_points_in_browser_id").addEventListener("change",
respond_to_max_number_of_points_in_browser_id_change);

function respond_to_max_number_of_points_in_browser_id_change() {
    maxDataPoints = document.getElementById("max_number_of_points_in_browser_id").value;
    // console.log("*******************");
    // console.log(maxDataPoints);
    // console.log(trace_scope.x.length);
    // if (trace_scope.x.length > maxDataPoints) {
    //     var num_points_to_delete;
    //     num_points_to_delete = trace_scope.x.length - maxDataPoints;
    //     for (i = 0; i < num_points_to_delete; i++) {
    //         trace_scope.x.shift();
    //         trace_scope.y.shift();
    //         // console.log(i);
    //     }
    // }
    // // console.log(trace_scope.x.length);
    // console.log("*******************");



    if (trace_scope_current.x.length > maxDataPoints) {
        var num_points_to_delete;
        num_points_to_delete = trace_scope_current.x.length - maxDataPoints;
        for (i = 0; i < num_points_to_delete; i++) {
            trace_scope_current.x.shift();
            trace_scope_current.y.shift();
            trace_scope_voltage.x.shift();
            trace_scope_voltage.y.shift();
            // console.log(i);
        }
    }








}
```

```
function removeData() {
    dataPlot.data.labels.shift();
    dataPlot.data.datasets[0].data.shift();
}
function addData(label, data) {
    if (dataPlot.data.labels.length > maxDataPoints) removeData();
    dataPlot.data.labels.push(label);
    dataPlot.data.datasets[0].data.push(data);
    dataPlot.update();
}


// PSUEDO CODE:

function update_average_current() {
    avg_current = 0;
    sum = 0;
    // trace_scope_current.y = [2,3,4,1,...]
    trace_scope_current.y.forEach(add_item_to_sum);

    avg_current = (1 / trace_scope_current.y.length) * sum
    document.getElementById("AVG_current_ID").innerHTML = avg_current;
    // console.log("updated avg current is =",avg_current);

    // now calculate standard deviatoin:
    sum=0;
    trace_scope_current.y.forEach(add_difference_between_item_and_average_squared_to_sum);
    std_dev_current=Math.sqrt(sum/trace_scope_current.y.length);
    document.getElementById("STD_DEV_current_ID").innerHTML = std_dev_current;


}

function add_item_to_sum(item) {
    sum += item;
}

function add_difference_between_item_and_average_squared_to_sum(item) {
    sum += (avg_current-item)*(avg_current-item);
}
```

```
// Javascript code to set up a websocket.

var m_url_JS = "ws://nanostat.local:81/";
//var m_url_JS = "ws://192.168.1.44:81/";
// var url = "ws://192.168.4.1:1337/";

var m_websocket;
var m_canvas_JS;
var context;
var dataPlot;
var maxDataPoints = 20; // max points in browser cache
var new_binary_data_is_incoming = false; // if true, reset counters, will recieve 3 binary messages with arrays
for current voltage time
var amps_array_has_been_received = false;
var volts_array_has_been_received = false;
var time_array_has_been_received = false;
var amps_array = []; // these have to be global and filled one by one, assume browswer has infinite processing
power and memory
var volts_array = []; // these have to be global and filled one by one, assume browswer has infinite processing
power and memory
var time_array = []; // these have to be global and filled one by one, assume browswer has infinite processing
power and memory

// This is called when the page finishes loading
function init() {

    // Assign page elements to variables
    m_canvas_JS = document.getElementById("m_canvas");

    // create chart:

    // dataPlot = new Chart(document.getElementById("m_canvas"), {
    //     type: 'line',
    //     data: {
    //         labels: [],
    //         datasets: [{
    //             data: [],
    //             label: "Temperature (C)",
    //             borderColor: "#3e95cd",
    //             fill: false
    //         }]
    //     },
    //     options: {
    //         scales: {
    //             y: {
    //                 beginAtZero: true
    //             }
    //         }
    //     }
    // });

    // Connect to WebSocket server
    wsConnect(m_url_JS);
}

// Call this to connect to the WebSocket server
function wsConnect(m_url_JS) {

    // Connect to WebSocket server
    m_url_JS = "ws://" + window.location.hostname + ":81/"
    // console.log(m_url_JS);
    m_websocket = new WebSocket(m_url_JS);
    m_websocket.binaryType = "arraybuffer";

    // Assign callbacks
    m_websocket.onopen = function (evt) { onOpen(evt) };
    m_websocket.onclose = function (evt) { onClose(evt) };
    m_websocket.onmessage = function (evt) { onMessage(evt) };
    m_websocket.onerror = function (evt) { onError(evt) };

}

// Called when a WebSocket connection is established with the server
function onOpen(evt) {

    // Log connection state
    console.log("Connected");

    // Enable button
    // button.disabled = false;

    // Get the current state of the LED
    // doSend("getLEDState");
}

// Called when the WebSocket connection is closed
function onClose(evt) {

    // Log disconnection state
    console.log("Disconnected");

    // Disable button
    // button.disabled = true;

    // Try to reconnect after a few seconds
    setTimeout(function () { wsConnect(m_url_JS) }, 2000);
}

// remove excess data from plot
function removeData() {
    dataPlot.data.labels.shift();
    dataPlot.data.datasets[0].data.shift();
}

// add data to plot (through chart object push method...)
function addData(label, data) {
    if (dataPlot.data.labels.length > maxDataPoints) removeData();
    dataPlot.data.labels.push(label);
    dataPlot.data.datasets[0].data.push(data);
    dataPlot.update();
}




// Called when a message is received from the server
function onMessage(evt) {
    console.log("onMessage called");


    if (typeof (evt.data) == "object") { // payload is binary, an ArrayBuffer
        console.log("OBJECT! parsing....");
        const view = new DataView(evt.data);
        var m_num_points;
        if (amps_array_has_been_received == false) {// this is the amps array
            console.log("this is the amps array");
            m_num_points = evt.data.byteLength / 4;
            console.log(" m_num_points=evt.data.byteLength/4=");
            console.log(m_num_points);
            amps_array.splice(0, amps_array.length); // empty old amps array
            for (i = 0; i < m_num_points; i++) {
                // console.log(i, view.getFloat32(i * 4, true));
                amps_array.push(view.getFloat32(i * 4, true)); // Add element to array
            }
            amps_array_has_been_received = true;
            return;
        }
        if ((amps_array_has_been_received == true) & (volts_array_has_been_received == false)) {// this is the
volts array
            console.log("this is the volts array");
            m_num_points = evt.data.byteLength / 2;
            console.log(" m_num_points=evt.data.byteLength/2=");
            console.log(m_num_points);
            volts_array.splice(0, volts_array.length); // empty old amps array
            for (i = 0; i < m_num_points; i++) {
                // console.log(i, view.getInt16(i * 2, true));
                volts_array.push(view.getInt16(i * 2, true)); // Add element to array
            }
            volts_array_has_been_received = true;
            return;
        }
        if ((amps_array_has_been_received == true) & (volts_array_has_been_received == true)) {// this is the time
array
            console.log("this is the time array");
            m_num_points = evt.data.byteLength / 4;
            console.log(" m_num_points=evt.data.byteLength/4=");
            console.log(m_num_points);
            time_array.splice(0, time_array.length); // empty old amps array
            for (i = 0; i < m_num_points; i++) {
                // console.log(i, view.getInt32(i * 4, true));
                time_array.push(view.getInt32(i * 4, true)); // Add element to array
            }
            time_array_has_been_received = true;
            new_binary_data_is_incoming = false;
            console.log("Received all 3 arrays binary websocket (I,V,t):");
            console.log(volts_array);
            console.log(amps_array);
            console.log(time_array);
            //  call the graph function now...
            plotGlobalArrays();
            return;

        }


    }


    if (typeof (evt.data) == "string") { // payload is string (JSON probably)
        console.log("STRING! parsing....");


        // Print out our received message
        console.log("Received: " + evt.data);
        var m_json_obj = JSON.parse(evt.data);
        //    console.log(m_json_obj);
        if ('expect_binary_data' in m_json_obj) {// changin in is sweeping status, update indicator on browswer
page...
            // do something
            if (m_json_obj.expect_binary_data == true) { // expect binary data
                // set expect data, set amps volts others to invalid, erase them
                new_binary_data_is_incoming = true; // if true, reset counters, will recieve 3 binary messages
with arrays for current voltage time
                amps_array_has_been_received = false;
                volts_array_has_been_received = false;
                time_array_has_been_received = false;
                // erase arrays (not sure how in Javascript just yet.... xyz)
                // amps_array; // these have to be global and filled one by one, assume browswer has infinite
processing power and memory
                // current_array; // these have to be global and filled one by one, assume browswer has infinite
processing power and memory
```

```
                // time_array; // these have to be global and filled one by one, assume browswer has infinite
processing power and memory
                console.log("(m_json_obj.expect_binary_data == true recieved...");
            }
            if (m_json_obj.expect_binary_data == false) { // mode is sweeping
                // do something
                // actually do nothing, just log to console
                console.log("m_json_obj.expect_binary_data == false recieved...");
            }
        };

        if ('is_sweeping' in m_json_obj) {// changin in is sweeping status, update indicator on browswer page...
            // do something
            if (m_json_obj.is_sweeping == true) { // mode is sweeping
                // do something
                // document.getElementById('sweep_mode_id').innerHTML = "SWEEPING";
                document.getElementById('sweep_mode_id').innerHTML = "<span style=\"color:red\">SWEEPING</span>";
                console.log("need to update indicator to true...");
            }
            if (m_json_obj.is_sweeping == false) { // mode is sweeping
                // do something
                document.getElementById('sweep_mode_id').innerHTML = "<span style=\"color:green\">IDLE</span>";
                console.log("need to update indicator to false...");
            }
        };



        if ('Voltage' in m_json_obj) {// this is the voltamagram (sent as string in JSON over websocket), parse
and plot it...
            var m_voltage_array = m_json_obj.Voltage;
            var m_current_array = m_json_obj.Current;
            var m_time_array = m_json_obj.Time;
            console.log(m_voltage_array);
            console.log(m_current_array);
            console.log(m_time_array);


            var trace_IV = {
                x: m_voltage_array,
                y: m_current_array,
                mode: 'markers',
                type: 'scatter'
            };
            var data_IV = [trace_IV];

            var m_IV_layout = {
                // title: 'IV Curve',
                showlegend: false,
                margin: {
                    l: 50,
                    r: 5,
                    b: 50,
                    t: 1,
                    pad: 4
                },
                // title: {
                //     text:'Plot Title',
                //     font: {
                //       family: 'Courier New, monospace',
                //       size: 24
                //     },
                //     xref: 'paper',
                //     x: 0.05,
                //   },
                xaxis: {
                    title: { text: 'Voltage (mV)' }
                },
                yaxis: {
                    title: { text: 'Current (microA)' }
                }
            };


            // Plotly.newPlot('plotly-IV', data_IV, m_IV_layout, {scrollZoom: true}, {editable: true},
{responsive: true});
            Plotly.newPlot('plotly-IV', data_IV, m_IV_layout, { scrollZoom: true, editable: true, responsive:
true });


            var trace_IvsTime = {
                x: m_time_array,
                y: m_current_array,
                mode: 'markers',
                type: 'scatter',
                name: "Current"
            };
            var trace_VvsTime = {
                x: m_time_array,
                y: m_voltage_array,
                mode: 'markers',
                yaxis: 'y2',
                type: 'scatter',
                name: "Voltage"
            };

            var data_IVvsTime = [trace_IvsTime, trace_VvsTime];


            var m_2yaxis_layout = {
                margin: {
                    l: 50,
                    r: 5,
                    b: 50,
                    t: 1,
                    pad: 4
                },
                // title: {
                //     text:'Plot Title',
                //     font: {
                //       family: 'Courier New, monospace',
                //       size: 24
                //     },
                //     xref: 'paper',
                //     x: 0.05,
                //   },
                xaxis: {
                    title: { text: 'Time (ms)' }
                },
                yaxis: { title: 'Current (microA)' },
                yaxis2: {
                    title: 'Voltage (mV)',
                    titlefont: { color: 'rgb(148, 103, 189)' },
                    tickfont: { color: 'rgb(148, 103, 189)' },
                    overlaying: 'y',
                    side: 'right'
                }
            };

            Plotly.newPlot('plotly-IvsTime', data_IVvsTime, m_2yaxis_layout, { scrollZoom: true, editable: true,
responsive: true });
            // Since data is fresh, might as well force a download for user....
            // if (download button is on) // not yet impelmented
            // xxxyyyzzz       <li><a href="downloadfile">Download</a></li>
```

```
            // window.open("http://nanostat.local/downloadfile");
            window.open("/downloadfile");

        };






    }

}

function addData(label, data) {
    if (dataPlot.data.labels.length > maxDataPoints) removeData();
    dataPlot.data.labels.push(label);
    dataPlot.data.datasets[0].data.push(data);
    dataPlot.update();
}

// Called when a WebSocket error occurs
function onError(evt) {
    console.log("ERROR: " + evt.data);
}

// Sends a message to the server (and prints it to the console)
function doSend(message) {
    console.log("Sending: " + message);
    websocket.send(message);
}

// Slider calls this to set data rate:
function sendDataRate() {
    var dataRate = document.getElementById("dataRateSlider").value;
    m_websocket.send(dataRate);
    dataRate = 1.0 * dataRate;
    document.getElementById("dataRateLabel").innerHTML = "Rate: " + dataRate.toFixed(2) + "Hz";
}


function plotGlobalArrays() {// plot global arrays that have been filled by the websocket data received as binary

    // console.log(volts_array);
    // console.log(amps_array);
    // console.log(time_array);

    var trace_IV = {
        x: volts_array,
        y: amps_array,
        mode: 'markers',
        type: 'scatter'
    };
    var data_IV = [trace_IV];

    var m_IV_layout = {
        // title: 'IV Curve',
        showlegend: false,
        margin: {
            l: 50,
            r: 5,
            b: 50,
            t: 1,
            pad: 4
        },
        xaxis: {
            title: { text: 'Voltage (mV)' }
        },
        yaxis: {
            title: { text: 'Current (microA)' }
        }
    };

    Plotly.newPlot('plotly-IV', data_IV, m_IV_layout, { scrollZoom: true, editable: true, responsive: true });

    var trace_IvsTime = {
        x: time_array,
        y: amps_array,
        mode: 'markers',
        type: 'scatter',
        name: "Current"
    };
    var trace_VvsTime = {
        x: time_array,
        y: volts_array,
        mode: 'markers',
        yaxis: 'y2',
        type: 'scatter',
        name: "Voltage"
    };

    var data_IVvsTime = [trace_IvsTime, trace_VvsTime];

    var m_2yaxis_layout = {
        margin: {
            l: 50,
            r: 5,
            b: 50,
            t: 1,
            pad: 4
        },

        xaxis: {
            title: { text: 'Time (ms)' }
        },
        yaxis: { title: 'Current (microA)' },
        yaxis2: {
            title: 'Voltage (mV)',
            titlefont: { color: 'rgb(148, 103, 189)' },
            tickfont: { color: 'rgb(148, 103, 189)' },
            overlaying: 'y',
            side: 'right'
        }
    };

    Plotly.newPlot('plotly-IvsTime', data_IVvsTime, m_2yaxis_layout, { scrollZoom: true, editable: true,
responsive: true });
    // Since data is fresh, might as well force a download for user....
    // if (download button is on) // not yet impelmented
    // xxxyyyzzz       <li><a href="downloadfile">Download</a></li>
    // window.open("http://nanostat.local/downloadfile");
    window.open("/downloadfile");





}


// Call the init function as soon as the page loads
window.addEventListener("load", init, false);
```

```html
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>NanoStat
  </title>
  <!-- Bootstrap CSS -->
  <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.0-beta3/dist/css/bootstrap.min.css" rel="stylesheet"
    integrity="sha384-eOJMYsd53ii+scO/bJGFsiCZc+5NDVN2yr8+0RDqr0Ql0h+rP48ckxlpbzKgwra6" crossorigin="anonymous">
  <script src='https://cdn.jsdelivr.net/npm/chart.js@3.1.1/dist/chart.min.js'></script>
  <!-- <script type="text/javascript" src="websocketcode.js"></script> -->
  <!-- <script type="text/javascript" src="button_handle_JS.js"></script> -->
  <!-- Wifi handling JS -->
  <script type="text/javascript" src="wifimanager.js"></script>
  <script src="https://cdn.plot.ly/plotly-latest.min.js"></script>
  <!-- <link rel="stylesheet" href="menubarstyle.css"> -->
  <!-- <link rel="stylesheet" href="CSS-Reset.css"> -->
  <!-- Google font: -->
  <link rel="preconnect" href="https://fonts.gstatic.com">
  <link href="https://fonts.googleapis.com/css2?family=Leckerli+One&display=swap" rel="stylesheet">
  <!-- Custom style: -->
  <link rel="stylesheet" href="nanostatstyle.css">
</head>


<body>

  <!-- Navigation bar -->
  <nav class="navbar navbar-expand-md navbar-dark bg-primary">
    <div class="container">
      <!-- <nav class="navbar navbar-expand-md navbar-light"> -->
      <!-- Brand icon: -->
      <a href="#" class="navbar-brand" style="font-family: 'Leckerli One', cursive;">NanoStat</a>
      <!-- Hamburger button: -->
      <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target="#toggleMobileMenu"
        aria-controls="toggleMobileMenu" aria-expanded="false" aria-table="Toggle navigation">
        <span class="navbar-toggler-icon"></span>
      </button>
      <!-- Menu items: -->
      <div class="collapse navbar-collapse" id="toggleMobileMenu">
        <ul class="navbar-nav ms-auto text-center">
          <li>
            <a class="nav-link" href="index.html">Sweep</a>
          </li>
          <li>
            <a class="nav-link" href="setup.html">Setup</a>
          </li>
          <li>
            <a class="nav-link" href="panel.html">Panel</a>
          </li>
          <li>
            <a class="nav-link" href="download.html">Download</a>
          </li>

          <li class="nav-item dropdown">
            <a href="#" class="nav-link dropdown-toggle active" id="navbarDropdown" role="button"
              data-bs-toggle="dropdown" aria-expanded="false">System</a>
            <ul class="dropdown-menu" aria-labeledby="navbarDropdown">
              <li><a href="files.html" class="dropdown-item">Files</a></li>
              <li><a href="wifi.html" class="dropdown-item active">Wifi</a></li>
              <li><a href="firmware.html" class="dropdown-item">Firmware</a></li>
              <li><a href="reboot.html" class="dropdown-item">Reboot</a></li>

            </ul>
          </li>



          <li>
            <a class="nav-link" href="help.html">Help</a>
          </li>
          <li>
            <a class="nav-link" href="about.html">About</a>
          </li>
        </ul>
      </div>
  </nav>

  <!-- Page title -->
  <!-- <header class="container text-center mybox1"> -->
  <header class="container text-center ">
    <h1>Wifi</h1>
    <h2>NanoStat Wifi Management Portal</h2>
  </header>

  <main class="container mybox1">

    <form action="saveSecret" method="POST">

      <div class="row">
        <div class="col-sm ">
          <div class="form-box ms-5 ms-md-0">
            <label class="form-label mb-0">SSID 1</label>
            <input type="text" class="form-control p-0 border border-secondary" name="ssid1" id="ssid1"
maxlength="32"
              size="16">
          </div>
          <div class="form-box ms-5 ms-md-0">
            <label class="form-label mb-0">Password 1</label>
            <input type="text" class="form-control p-0 border border-secondary" name="pass1" id="pass1"
maxlength="32"
              size="16">
          </div>
        </div>
        <div class="col-sm ">
          <div class="form-box ms-5 ms-md-0">
            <label class="form-label mb-0">SSID 2</label>
            <input type="text" class="form-control p-0 border border-secondary" name="ssid2" id="ssid2"
maxlength="32"
              size="16">
          </div>
          <div class="form-box ms-5 ms-md-0">
            <label class="form-label mb-0">Password 2</label>
            <input type="text" class="form-control p-0 border border-secondary" name="pass2" id="pass2"
maxlength="32"
              size="16">
          </div>
        </div>
        <div class="col-sm ">
          <div class="form-box ms-5 ms-md-0">
            <label class="form-label mb-0">SSID 3</label>
            <input type="text" class="form-control p-0 border border-secondary" name="ssid3" id="ssid3"
maxlength="32"
              size="16">
          </div>
          <div class="form-box ms-5 ms-md-0">
            <label class="form-label mb-0">Password 3</label>
            <input type="text" class="form-control p-0 border border-secondary" name="pass3" id="pass3"
maxlength="32"
              size="16">
          </div>
        </div>

      </div>

      <!-- Clear form -->
      <!-- <input type="reset" value="Clear" /> -->
      <!-- <input type="submit" value="Save" /> -->
      <button type="reset" class="btn btn-primary my-2">Clear</button>
      <button type="submit" class="btn btn-primary my-2">Save</button>

    </form>


    <!-- <div class="form-box ms-5 ms-md-0">
      <label class="form-label mb-0">xyz</label>
      <input type="text" class="form-control p-0 border border-secondary" name="xyz" id="xyz_id" maxlength="32"
size="16">
    </div> -->

    <!-- Load wifi credentials from NanoStat to display. -->
    <!-- <script>
      getSecretJson();
    </script> -->

  </main>

  <!-- Wifi scan -->


  <div class="container mybox2">
    <button class="btn btn-primary my-2" onClick="getWifiScanJson()">Wifi Scan</button>
    <p>Click twice.</p>
    <div id="wifilist"></div>

  </div>


  <!-- <footer class="container text-center myfooter fs-6 fw-lighter pt-3 fst-italic"> -->
  <footer class="container text-center  fs-6 fw-lighter pt-3 fst-italic">
    <p>NanoStat Rev. 3.5 @ BurkeLab 2021</p>
  </footer>

  <!-- Bootstrap JS -->
  <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.0-beta3/dist/js/bootstrap.bundle.min.js"
    integrity="sha384-JEW9xMcG8R+pH31jmWH6WWP0WintQrMb4s7ZOdauHnUtxwoG2vI5DkLtS3qm9Ekf"
    crossorigin="anonymous"></script>
</body>
```

```html
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>NanoStat
  </title>
  <!-- Bootstrap CSS -->
  <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.0-beta3/dist/css/bootstrap.min.css" rel="stylesheet"
    integrity="sha384-eOJMYsd53ii+scO/bJGFsiCZc+5NDVN2yr8+0RDqr0Ql0h+rP48ckxlpbzKgwra6" crossorigin="anonymous">
  <script src='https://cdn.jsdelivr.net/npm/chart.js@3.1.1/dist/chart.min.js'></script>
  <!-- <script type="text/javascript" src="websocketcode.js"></script> -->
  <!-- <script type="text/javascript" src="button_handle_JS.js"></script> -->
  <!-- Wifi handling JS -->
  <script type="text/javascript" src="wifimanager.js"></script>
  <script src="https://cdn.plot.ly/plotly-latest.min.js"></script>
  <!-- <link rel="stylesheet" href="menubarstyle.css"> -->
  <!-- <link rel="stylesheet" href="CSS-Reset.css"> -->
  <!-- Google font: -->
  <link rel="preconnect" href="https://fonts.gstatic.com">
  <link href="https://fonts.googleapis.com/css2?family=Leckerli+One&display=swap" rel="stylesheet">
  <!-- Custom style: -->
  <link rel="stylesheet" href="nanostatstyle.css">
</head>


<body>

  <!-- Navigation bar -->


  <!-- Page title -->
  <!-- <header class="container text-center mybox1"> -->
  <header class="container text-center ">
    <h1>Wifi</h1>
    <h2>NanoStat Wifi Management Portal</h2>
    <p>Save Wifi credentials to enable system reboot, connection to Wifi AP, and full functionality of the
NanoStat.</p>
  </header>

  <main class="container mybox1">

    <form action="saveSecret" method="POST">

      <div class="row">
        <div class="col-sm ">
          <div class="form-box ms-5 ms-md-0">
            <label class="form-label mb-0">SSID 1</label><br>
            <input type="text" class="form-control p-0 border border-secondary" name="ssid1" id="ssid1"
maxlength="32"
              size="16">
          </div>
          <div class="form-box ms-5 ms-md-0">
            <label class="form-label mb-0">Password 1</label><br>
            <input type="text" class="form-control p-0 border border-secondary" name="pass1" id="pass1"
maxlength="32"
              size="16">
          </div>
        </div>
        <div class="col-sm ">
          <div class="form-box ms-5 ms-md-0">
            <label class="form-label mb-0">SSID 2</label><br>
            <input type="text" class="form-control p-0 border border-secondary" name="ssid2" id="ssid2"
maxlength="32"
              size="16">
          </div>
          <div class="form-box ms-5 ms-md-0">
            <label class="form-label mb-0">Password 2</label><br>
            <input type="text" class="form-control p-0 border border-secondary" name="pass2" id="pass2"
maxlength="32"
              size="16">
          </div>
        </div>
        <div class="col-sm ">
          <div class="form-box ms-5 ms-md-0">
            <label class="form-label mb-0">SSID 3</label><br>
            <input type="text" class="form-control p-0 border border-secondary" name="ssid3" id="ssid3"
maxlength="32"
              size="16">
          </div>
          <div class="form-box ms-5 ms-md-0">
            <label class="form-label mb-0">Password 3</label><br>
            <input type="text" class="form-control p-0 border border-secondary" name="pass3" id="pass3"
maxlength="32"
              size="16">
          </div>
        </div>

      </div><br>

      <!-- Clear form -->
      <!-- <input type="reset" value="Clear" /> -->
      <!-- <input type="submit" value="Save" /> -->
      <button type="reset" class="btn btn-primary my-2">Clear</button>
      <button type="submit" class="btn btn-primary my-2">Save</button>

    </form><br>


    <!-- <div class="form-box ms-5 ms-md-0">
      <label class="form-label mb-0">xyz</label>
      <input type="text" class="form-control p-0 border border-secondary" name="xyz" id="xyz_id" maxlength="32"
size="16">
    </div> -->

    <!-- Load wifi credentials from NanoStat to display. -->
    <!-- <script>
      getSecretJson();
    </script> -->

  </main>

  <!-- Wifi scan -->


  <div class="container mybox2">
    <button class="btn btn-primary my-2" onClick="getWifiScanJson()">Wifi Scan</button>
    <p>Click twice.</p>
    <div id="wifilist"></div>

  </div>


  <!-- <footer class="container text-center myfooter fs-6 fw-lighter pt-3 fst-italic"> -->
  <footer class="container text-center  fs-6 fw-lighter pt-3 fst-italic">
    <p>NanoStat Rev. 3.5 @ BurkeLab 2021</p>
  </footer>

  <!-- Bootstrap JS -->
  <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.0-beta3/dist/js/bootstrap.bundle.min.js"
    integrity="sha384-JEW9xMcG8R+pH31jmWH6WWP0WintQrMb4s7ZOdauHnUtxwoG2vI5DkLtS3qm9Ekf"
    crossorigin="anonymous"></script>
```

```
</body>
```

```
function getWifiScanJson() {
    var xhttp = new XMLHttpRequest();
    xhttp.onreadystatechange = function () {
        if (this.readyState == 4 && this.status == 200) {
            var obj = JSON.parse(this.responseText);
            if (obj.scan_result.length) {
                var htmlSrc = '<ul class="list-group">';
                for (var i = 0; i < obj.scan_result.length; i++) {
                    htmlSrc += '<li class="list-group-item"><strong>' + obj.scan_result[i].SSID + '</strong> ' +
obj.scan_result[i].RSSI + ' dBm</li>';
                }
                htmlSrc += '</ul>';
                document.getElementById("wifilist").innerHTML = htmlSrc;
            }
            console.log(obj);
        }
    };
    xhttp.open("GET", "wifiScan.json", true);
    xhttp.send();
}


function getSecretJson() {
    var xhttp = new XMLHttpRequest();
    xhttp.onreadystatechange = function () {
        if (this.readyState == 4 && this.status == 200) {
            //document.getElementById("LEDState").innerHTML =
            var obj = JSON.parse(this.responseText);
            for (var i = 1; i < 4; i++) {
                document.getElementById("ssid" + i).value = obj['ssid' + i];
                document.getElementById("pass" + i).value = obj['pass' + i];
            }
            console.log(obj);
        }
    };
    xhttp.open("GET", "secrets.json", true);
    xhttp.send();
}


function showPassword(id) {
    var x = document.getElementById(id);
    if (x.type === "password") {
        x.type = "text";
    } else {
        x.type = "password";
    }
}


$('form').submit(function (e) {
  e.preventDefault();
  var form = $('#upload_form')[0];
  var data = new FormData(form);
  $.ajax({
    url: '/edit',
    type: 'POST',
    data: data,
    contentType: false,
    processData: false,
    xhr: function () {
      var xhr = new window.XMLHttpRequest();
      xhr.upload.addEventListener('progress', function (evt) {
        if (evt.lengthComputable) {
          var per = evt.loaded / evt.total;
          $('#prg').html('progress: ' + Math.round(per * 100) + '%');
        }
      }, false);
      return xhr;
    },
    success: function (d, s) {
      console.log('success!');
      getFileList();
    },
    error: function (a, b, c) {
    }
  });
});



function deleteFile(fileName) {
  var xhttp = new XMLHttpRequest();
  xhttp.onreadystatechange = function () {
    if (this.readyState == 4 && this.status == 200) {
      //var obj =  JSON.parse();
      //console.log(this.responseText);
      getFileList();
    }
  };
  if (confirm('Are you sure you want to delete the file ?')) {
    xhttp.open("DELETE", "/edit?file=/" + fileName, true);
    xhttp.send();
  }


}

function getFileList() {
  console.log('starting getFileList!');
  var xhttp = new XMLHttpRequest();
  xhttp.onreadystatechange = function () {
    if (this.readyState == 4 && this.status == 200) {
      var obj = JSON.parse(this.responseText);

      if (obj.length) {
        // var htmlSrc = '<ul>';
        var htmlSrc = '<ul class="list-group">';
        for (var i = 0; i < obj.length; i++) {
          if (obj[i].type == 'file') {
            // htmlSrc += '<li>' + obj[i].name + ' <a href="#" onclick="deleteFile(\'' + obj[i].name + '\')">X</-
a></li>';
            htmlSrc += '<li class="list-group-item">' + obj[i].name + ' <a href="#" onclick="deleteFile(\'' +
obj[i].name + '\')">X</a></li>';
          }
        }
        htmlSrc += '</ul>';
        document.getElementById("filelist").innerHTML = htmlSrc;
      }
      console.log(obj);
      // console.log(htmlSrc);
    }
  };
  xhttp.open("GET", "/list?dir=/", true);
  xhttp.send();
}
```

8. Supplemental Information

8.1. List of files

Materials contained in supplemental information for publication:

- Circuit schematics.
- 3d printed case renderings.

Materials contained in supplemental information for review purposes only:

- Source code for firmware, website code (HTML5, JavaScript), and PlatformIO configuration files for integration with the Microsoft Visual Studio Code IDE as one large pdf file.

Materials contained in github repository https://github.com/PeterJBurke/Nanostat (repository is private and will be made public after publication):

- Circuit schematics.
- Fusion 360 source files as well as 3d printable stl files.
- Source code for firmware, website code (HTML5, JavaScript), and PlatformIO configuration files for integration with the Microsoft Visual Studio Code IDE
- Bill of materials, pick and place file, that can be used by any PCB factory.
- Gerber files for PCB etching/hole pattern/solder pattern/etc. that can be used by any PCB factory.
- CAD files in EASY EDA format for PCB.

The code is provided for review purposes as a pdf file, but is more useful as a github repository for users after publication. The CAD files, BOM, pick and place files, and Gerber files are not needed for review, but are useful for users after publication who want to produce these boards and use them in their labs.

8.2. Calibration

Fig. 11 A shows the ADC read voltage vs. the DAC set voltage of the ESP32 when they are connected directly. Clearly there is an offset, non-unity slope, and non-linearity. In Fig. 11 B, the actual voltage of the DAC measured with a multimeter vs. the set voltage is shown. The DAC is well behaved with slope of unity within 5 percent and less than 50 mV offset. Clearly, the ADC is the dominant error. In order to account for this, we developed the following calibration routine.

The LMP91000 allows positive and negative cell bias. In the schematic (Fig. 9 from data sheet), the positive terminal of the TIA is the "virtual ground" for the cell, and is 0.5 of Vref. In the NanoStat, Vref is the output of the ESP32 DAC. Vout of the LMP91000 is tied to the ADC. The cell current should in principle be calculated by measuring the voltage at point C1 and C2 and taking the difference, i.e. WE current = (VC2-VC1)/RTIA. In the NanoStat, only VC2 is measured. VC1 is assumed to be 0.5 of Vref. However, given the offset and slope errors of the ADC, this assumption gives errounous values for the measured current.

In order to calibrate the slope and offset of the ADC, we assume the DAC is perfect, and disconnect the WE. Under this condition, no current flows through the WE and therefore VC1=VC2 = 0.5 Vref. We vary Vref by varying VDAC (=Vref), and record VADC (which is V2=V1=Vout=0.5 Vref). This gives an offset and slope calibration in situ for the (non-ideal) ADC. The linearity of the ADC is not corrected for in this fashion. In practice the linearity is not severe until the ADC reads above 3 V. The LMP91000 can bias the cell up to 20 percent of the DAC output of the ESP32. The ESP32 ADC is very non-linear above 3 V, and cannot go above 3.3 V which is the power supply voltage. Therefore the cell voltage is limited to 3 V * 20 percent = 600 mV, in either direction.

The calibration shows the 10kohm test resistor (Fig. 4) gives the correct slope, linearity, and offset once calibrated.

According to the datasheet, the LMP91000 requires Vref>1.5 V. The data sheet does not specify the tolerance for this, but it could be checked if a future user was concerned about this. So to be conservative, we set it to 1.52 V minimum to ensure this. At this voltage, the cell can be 0 percent, 1 percent, etc. of this Vref. At the 1 percent, this is a minimum voltage of 15 mV on the cell. At 0 percent, it is zero volts on the cell. Therefore, it is not possible to bias between 0 and 15 mV, unless one pushes Vref to very close to 1.5 V, which we choose not to do. The NanoStat firmware during a sweep sets the voltage to the closest of 0 or 15 mV depending on the sweep status and rate.

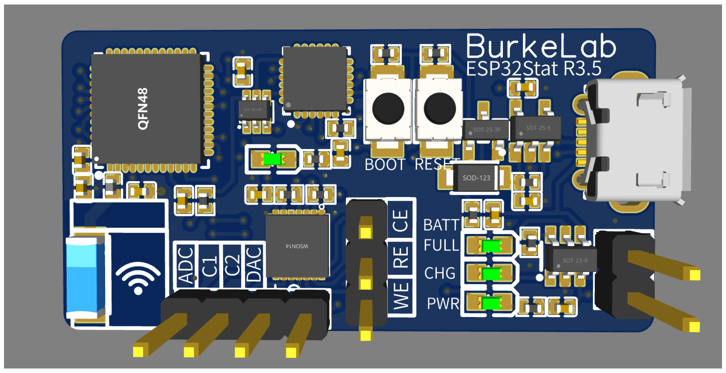


Figure 7: Circuit board rendering. Test ports are provided for access to the ADC, DAC, and C1/C2 which are either side of the TIA resistor. A USB connector is for the first firmware download and to charge the battery. LEDs indicate the battery charging status (charging or complete).

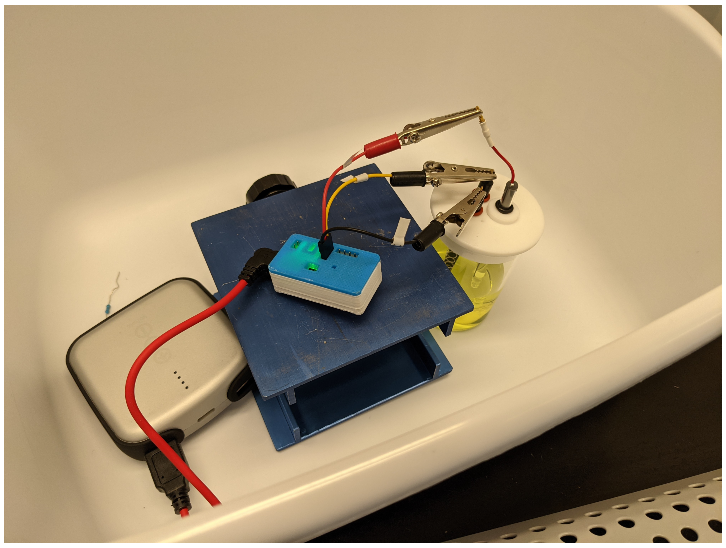


Figure 8: The NanoStat in action. An external USB connector battery , popular among cell phone users, (left) is used. The electrochemical cell is at right. The blue stand is for support only is is not required.

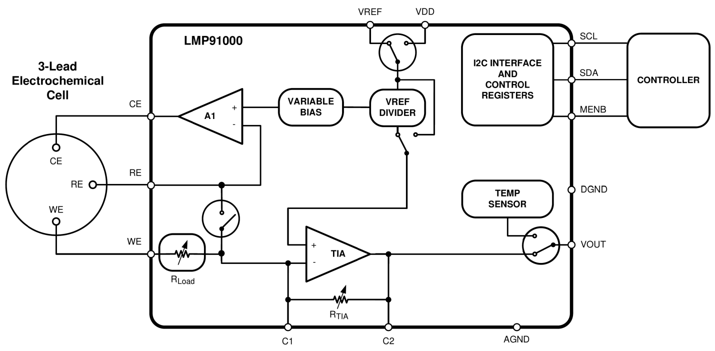
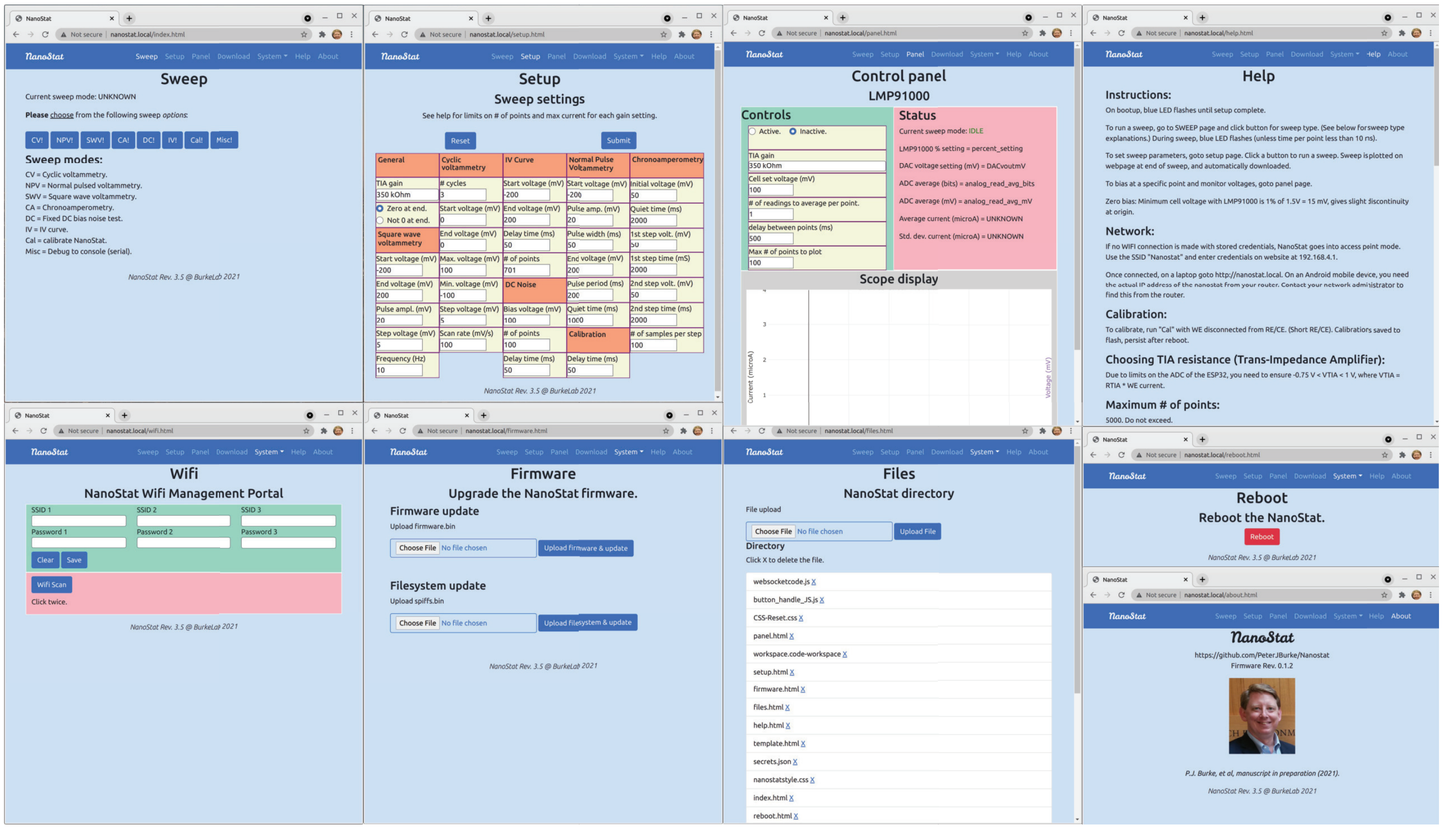


Figure 9: LMP91000 schematic, from the data sheet.



NanoStat — Not secure | nanostat.local/index.html

NanoStat Sweep Setup Panel Download System Help About

Sweep

Current sweep mode: UNKNOWN

Please choose from the following sweep *options*:

CV! NPV! SWV! CA! DC! IV! Cal! Misc!

Sweep modes:

CV = Cyclic voltammetry.
NPV = Normal pulsed voltammetry.
SWV = Square wave voltammetry.
CA = Chronoamperometry.
DC = Fixed DC bias noise test.
IV = IV curve.
Cal = calibrate NanoStat.
Misc = Debug to console (serial).

NanoStat Rev. 3.5 @ BurkeLab 2021

NanoStat — Not secure | nanostat.local/setup.html

Setup

Sweep settings

See help for limits on # of points and max current for each gain setting.

Reset Submit

General	Cyclic voltammetry	IV Curve	Normal Pulse Voltammetry	Chronoamperometry
TIA gain: 350 kOhm	# cycles: 3	Start voltage (mV): -200	Start voltage (mV): -200	Initial voltage (mV): 50
● Zero at end. ○ Not 0 at end.	Start voltage (mV): 0	End voltage (mV): 200	Pulse amp. (mV): 20	Quiet time (ms): 2000
Square wave voltammetry	End voltage (mV): 0	Delay time (ms): 50	Pulse width (ms): 50	1st step volt. (mV): 50
Start voltage (mV): -200	Max. voltage (mV): 100	# of points: 701	End voltage (mV): 200	1st step time (mS): 2000
End voltage (mV): 200	Min. voltage (mV): -100	**DC Noise**	Pulse period (ms): 200	2nd step volt. (mV): 50
Pulse ampl. (mV): 20	Step voltage (mV): 5	Bias voltage (mV): 100	Quiet time (ms): 1000	2nd step time (ms): 2000
Step voltage (mV): 5	Scan rate (mV/s): 100	# of points: 100	**Calibration**	# of samples per step: 100
Frequency (Hz): 10		Delay time (ms): 50	Delay time (ms): 50	

NanoStat Rev. 3.5 @ BurkeLab 2021

NanoStat — Not secure | nanostat.local/panel.html

Control panel

LMP91000

Controls

○ Active. ● Inactive.
TIA gain: 350 kOhm
Cell set voltage (mV): 100
# of readings to average per point.: 1
delay between points (ms): 500
Max # of points to plot: 100

Status

Current sweep mode: IDLE
LMP91000 % setting = percent_setting
DAC voltage setting (mV) = DACvoutmV
ADC average (bits) = analog_read_avg_bits
ADC average (mV) = analog_read_avg_mV
Average current (microA) = UNKNOWN
Std. dev. current (microA) = UNKNOWN

Scope display

NanoStat — Not secure | nanostat.local/help.html

Help

Instructions:

On bootup, blue LED flashes until setup complete.

To run a sweep, go to SWEEP page and click button for sweep type. (See below forsweep type explanations.) During sweep, blue LED flashes (unless time per point less than 10 ms).

To set sweep parameters, goto setup page. Click a button to run a sweep. Sweep isplotted on webpage at end of sweep, and automatically downloaded.

To bias at a specific point and monitor voltages, goto panel page.

Zero bias: Minimum cell voltage with LMP91000 is 1% of 1.5V = 15 mV, gives slight discontinuity at origin.

Network:

If no WIFI connection is made with stored credentials, NanoStat goes into access point mode. Use the SSID "Nanostat" and enter credentials on website at 192.168.4.1.

Once connected, on a laptop goto http://nanostat.local. On an Android mobile device, you need the actual IP address of the nanostat from your router. Contact your network administrator to find this from the router.

Calibration:

To calibrate, run "Cal" with WE disconnected from RE/CE. (Short RE/CE). Calibrations saved to flash, persist after reboot.

Choosing TIA resistance (Trans-Impedance Amplifier):

Due to limits on the ADC of the ESP32, you need to ensure -0.75 V < VTIA < 1 V, where VTIA = RTIA * WE current.

Maximum # of points:

5000. Do not exceed.

NanoStat — Not secure | nanostat.local/wifi.html

Wifi

NanoStat Wifi Management Portal

SSID 1 SSID 2 SSID 3
Password 1 Password 2 Password 3
Clear Save

Wifi Scan
Click twice.

NanoStat Rev. 3.5 @ BurkeLab 2021

NanoStat — Not secure | nanostat.local/firmware.html

Firmware

Upgrade the NanoStat firmware.

Firmware update

Upload firmware.bin

Choose File No file chosen Upload firmware & update

Filesystem update

Upload spiffs.bin

Choose File No file chosen Upload filesystem & update

NanoStat Rev. 3.5 @ BurkeLab 2021

NanoStat — Not secure | nanostat.local/files.html

Files

NanoStat directory

File upload

Choose File No file chosen Upload File

Directory

Click X to delete the file.

websocketcode.js X
button_handle_JS.js X
CSS-Reset.css X
panel.html X
workspace.code-workspace X
setup.html X
firmware.html X
files.html X
help.html X
template.html X
secrets.json X
nanostatstyle.css X
index.html X
reboot.html X

NanoStat — Not secure | nanostat.local/reboot.html

Reboot

Reboot the NanoStat.

Reboot

NanoStat Rev. 3.5 @ BurkeLab 2021

NanoStat — Not secure | nanostat.local/about.html

NanoStat

https://github.com/PeterJBurke/Nanostat
Firmware Rev. 0.1.2

P.J. Burke, et al, manuscript in preparation (2021).

NanoStat Rev. 3.5 @ BurkeLab 2021

Figure 10: User interface in web browser. Note that no special software is needed to access and control the NanoStat. Only a web browser is needed; all the control and graphical plotting software is contained in the JavaScript and HTML5 code which are intimately synchronized with the on board firmware to control and record the analog signals and report them back to the website over WiFi using WebSocket technology.

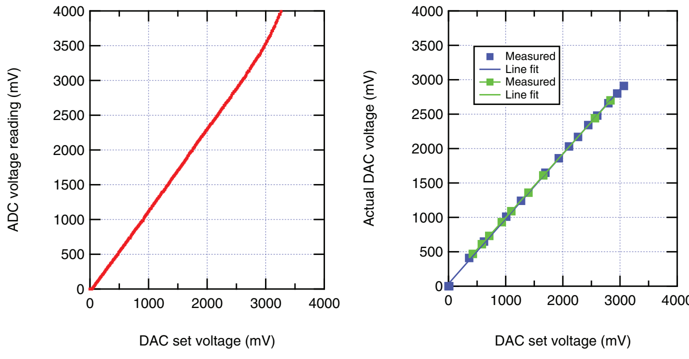


Figure 11: Left: ADC read voltage vs. DAC set voltage. Right: Actual voltage vs. DAC set voltage.

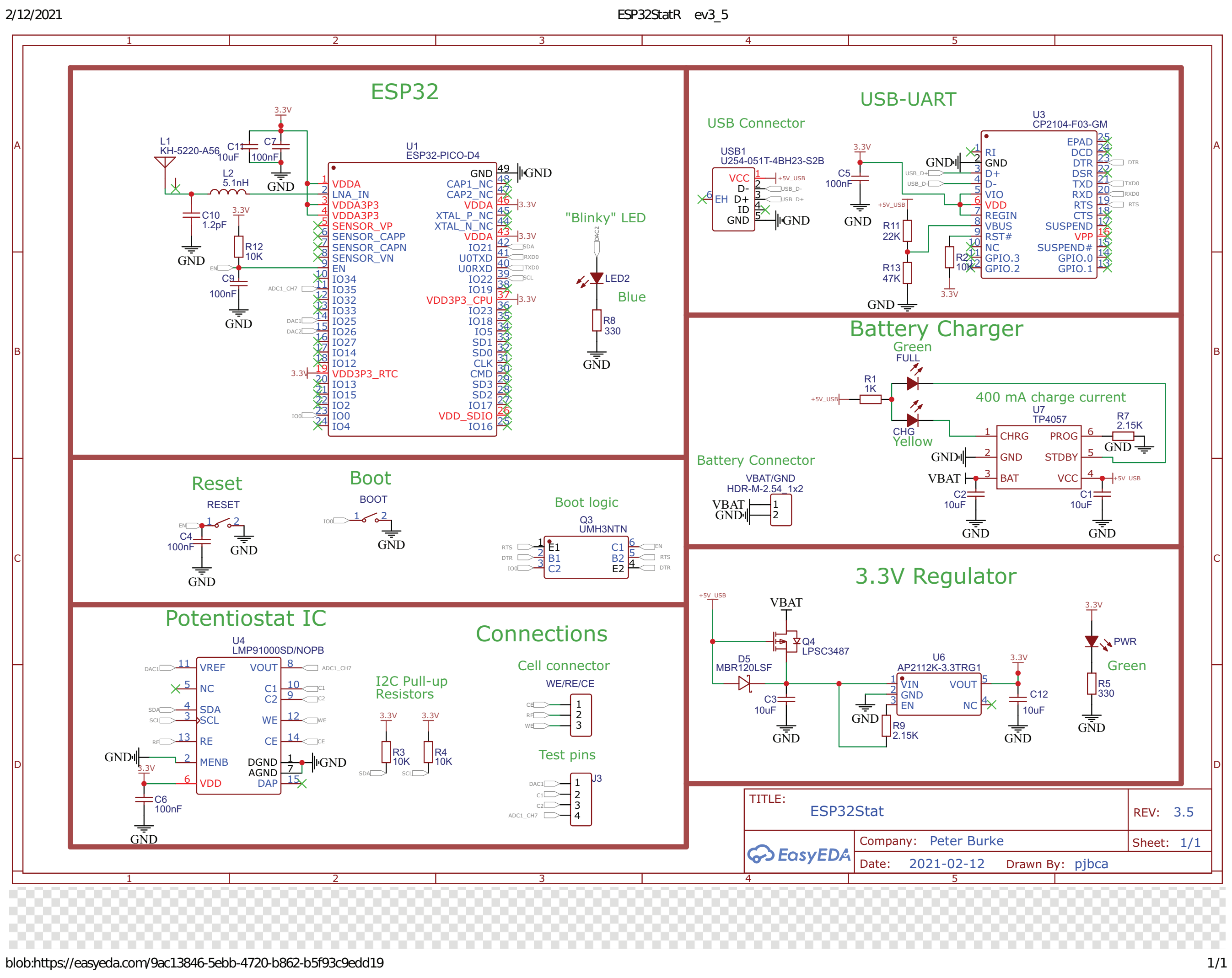


Figure 12: Circuit schematic.

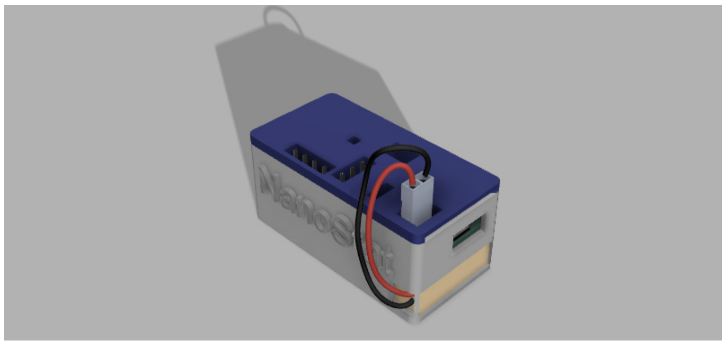


Figure 13: Fusion 360 computer rendering of 3d printed case design with LiPo battery slot. The LiPo battery slides in the bottom (shown inserted).

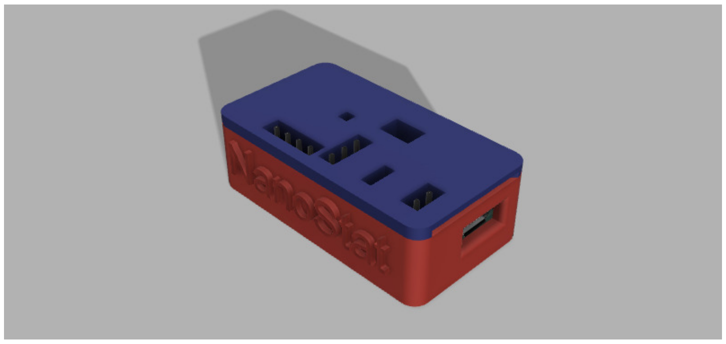


Figure 14: Fusion 360 computer rendering of 3d printed case design without LiPo battery slot.

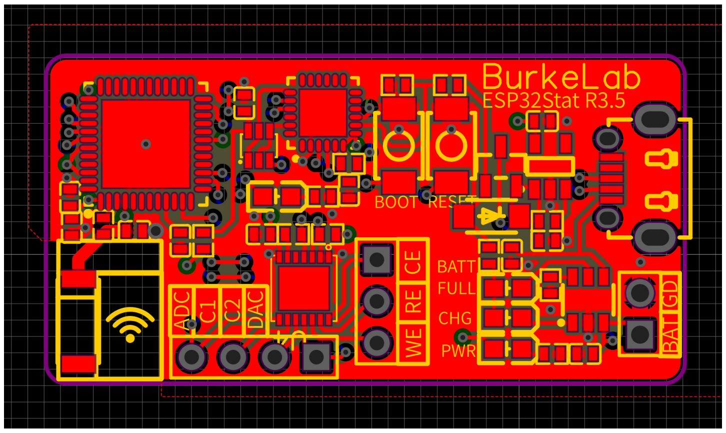


Figure 15: Trace layout (top layer).