Power Line Monitor

CAUTION. This project involves voltages that can kill you. Be careful.
The author takes no responsibility for injury or death.

1. This is a simple project to continuously monitor a residential 120 V_{RMS} 60 Hz power line (see §9 for other systems). After construction, the unit is plugged into any convenient outlet and can remain unattended indefinitely. It connects to your home wifi network and all command and status reporting is done from any browser on the same network. Each occurrence of abnormal line voltage or frequency is automatically recorded, or a fresh measurement can be taken any time on demand. The unit includes its own battery which allows it to continue functioning for well over a day without commercial power.

2. The unit consists of the following components. Links are provided to the specific products I purchased but substitutions are certainly possible.

   2.1 Adafruit ESP32 Feather - https://www.adafruit.com/product/3405

   2.2 Adafruit Lipo battery 2500 mAh - https://www.adafruit.com/product/328

   2.3 Sensor module based on ZMPT101B – https://www.amazon.com/dp/B07D1WP884

      2.3.1 helpful links: https://www.electroschematics.com/voltage-sensor

      2.3.2 https://solarduino.com/how-to-measure-ac-voltage-with-arduino

   2.4 USB wall wart supply – https://www.amazon.com//dp/B07P41X3L3

   2.5 Short USB A to micro USB cable – https://www.amazon.com/dp/B013G4EAEI

   2.6 Project box – https://www.amazon.com/dp/B07BPPKF2C

3. Connect the modules together according to Figure 1. Note although the L and N terminals on the sensor module appear to be isolated, I took care to connect the fat spade on the power plug to the N terminal.

4. After carefully inspecting your work, plug the unit in. Temporarily connect an oscilloscope to sensor pins OUT and GND. You want to see a sine wave roughly 400-500 mV_{P-P} centered at about 1.6 volts. Adjust the pot for the largest P-P voltage that shows *no sign of compression* at either the top or bottom of the wave. Any distortion will interfere with the frequency determination of the waveform. It is not necessary to strive for the largest possible P-P voltage, the ESP32 ADC has plenty of precision to measure even a few mV voltage change, it is more important to have a clean waveform.

5. Temporarily connect the ESP32 Feather micro USB port to your host computer. Load the following software on your computer and build the project:

5.1 Install Arduino IDE from https://www.arduino.cc/en/software.

5.2 Follow the instructions at https://github.com/espressif/arduino-esp32 to install the ESP32 board support packages. When finished, be sure to select:

Tools → Board → ESP32 Arduino (in sketchbook) → Adafruit ESP32 Feather

5.3 Open IDE *Tools → Manage Libraries* and install the following libraries:

5.3.1 NTPClient V3.2.1

5.3.2 TimerInterrupt_Generic 1.13.0

5.4 Install my project zip file from https://clearskyinstitute.com/ham/PowerLineMonitor

5.5 Open the file **PowerLineMonitor.ino** and edit the **ssid**[] and **password**[] strings to match your WiFi login credentials.

5.6 Open the file **html.cpp** and edit the **tz** variable to match you local time zone offset.

5.7 Build and load the project onto your ESP32. After several seconds, if all is well, the LED will blink once per second. Different blink counts indicate status as follows.

Count	Meaning
1	Normal operation. Blinks at the beginning of each 1 second data set
2	Successful calibration
3	Not connected to mains or not adjusted properly -- review step 4
4	WiFi login failed -- review step 5.5
5	NTP time service failure -- retries automatically until successful
6	* Out of memory
7	Remote update failed -- resume current version
8	* Restart requested by operator
9	* Successful upload

*Fatal conditions that cause a reboot are saved to EEPROM and reported in the System report as Reboot flag.

5.8 After loading the ESP32 software and testing the web connection (see next section), reconnect the Feather micro USB port to the power supply and plug in the battery.

6. Use a web browser to open http://PowerLineMonitor.local. You should see a single page similar to Figure 2. The page contains a row of buttons; a table of events; and two graphs of either voltage and frequency over time, or details of one event.

   At first there will be just one initial persistent Calibration event in the table. Each new event will add a row to the table until there are 50 events, after which the oldest events will be overwritten. Click Plot to see the details of one event. The buttons work as follows:

   Timeline: update the event table and timeline, *the page does not update by itself*

   Measure: measure the line now and add the result as a new table entry marked M

   Upload: install a new firmware file such as `PowerLineMonitor.ino.bin`

   System: show a table of internal state and information

   Calibrate: acquire new calibration, renormalize existing timeline and discard events because their triggering criteria are no longer valid.

7. **Theory of operation.** The sensor module uses a transformer to isolate and step down the line voltage. This is fed to an analog circuit that shifts the mean sine wave value to half the supply voltage with a gain adjustment. This is read by a 12 bit ADC input at 620 Hz to form one sample set. This is analyzed for overall mean and RMS. It is also broken into ten shorter segments to find the minimum and maximum RMS of each 100 ms subinterval.

   The single dominant frequency component is determined using a binary search for maximum quadrature correlation to achieve a resolution of 0.005 Hz. A straight DFT would require a sample duration over 3 minutes to achieve this resolution. I also explored Goertzel, chirp-DFT and timing the zero crossings but this approach worked best for this specific application. Frequency measurement relies solely on the microprocessor oscillator period.

   The first sample after booting is used for calibration. The software assumes this is taken while the supply voltage is exactly 120 V_{RMS} to establish the ADC scale reference and exactly 60 Hz to establish the frequency offset. Calibration is repeated until the values lie within expected norms. All subsequent voltage reports are relative to this initial calibration unless a new calibration is performed using Calibrate which again assumes the input are ideal.

Once calibration succeeds, sampling continues forever in a double-buffered manner for continuous coverage without gaps. A sample set is logged as an error event if:

7.1 its best fit frequency differs by more than 0.1 from 60 Hz;

7.2 its overall RMS voltage differs by more than 6 V from the calibration value;

7.3 its overall Peak-to-Peak voltage differs by more than 14 V from calibration;

7.4 it contains a surge 6 V above or sag 6 V below 120 V in any 100 ms subinterval;

7.5 or it contains a spike of at least 6 V in any subinterval while otherwise dead.

Logging is disabled after 10 consecutive events in order not to fill the log during a brownout. Logging resumes after a sample set is again within range or a manual measurement is taken from the web page. In addition to error events, the overall RMS and frequency from every 60th sample set (*ie*, once per minute) are stored in a separate trend circular queue containing 1500 elements to produce a running history timeline spanning up to the previous 25 hours.

8. The web page is just a veneer over several RESTful commands. These may be issued directly at any time from a browser or with command line tools such as *curl* or *wget* to save the raw data if desired. Let me know if you prefer there be a way to download the log files via the web page. Note the files record time in UTC using UNIX seconds; only the web page reports time in nice local hours and minutes. To get a list of available commands send the following command:

```
PowerLineMonitor.local/help
```

8.1 To upload a new firmware file with *curl* use this command:

```
curl --data-binary @PowerLineMonitor.ino.bin PowerLineMonitor.local/upload
```

9. Miscellaneous notes:

9.1 To use for 240 V and/or 50 Hz lines, edit `LINE_V_CAL`, `LINE_HZ_CAL`, `LINE_V_TOL` and `LINE_HZ_TOL` in PowerLineMonitor.h as required.

9.2 Do not be tempted to use an ESP8266. It’s ADC is only 10 bits and is quite noisy.

9.3 I had to purchase two sensor modules to get one that worked. The first just flat-lined at 1.6 V regardless of the pot setting.

9.4 The line voltage at boot, and any subsequent recalibration, is assumed to be exactly `LINE_V_CAL` so if already marginally high or low it can cause false events later.

I enjoy using this device. It is interesting to watch the power company load leveling behavior and the frequency chugging of the great generators. I am also surprised just how often the power spikes and how really ugly it can be when service cuts out and slams back on. I hope you also enjoy using your Power Line Monitor.

73, Elwood, WBØOEW, ecdowney@clearskyinstitute.com

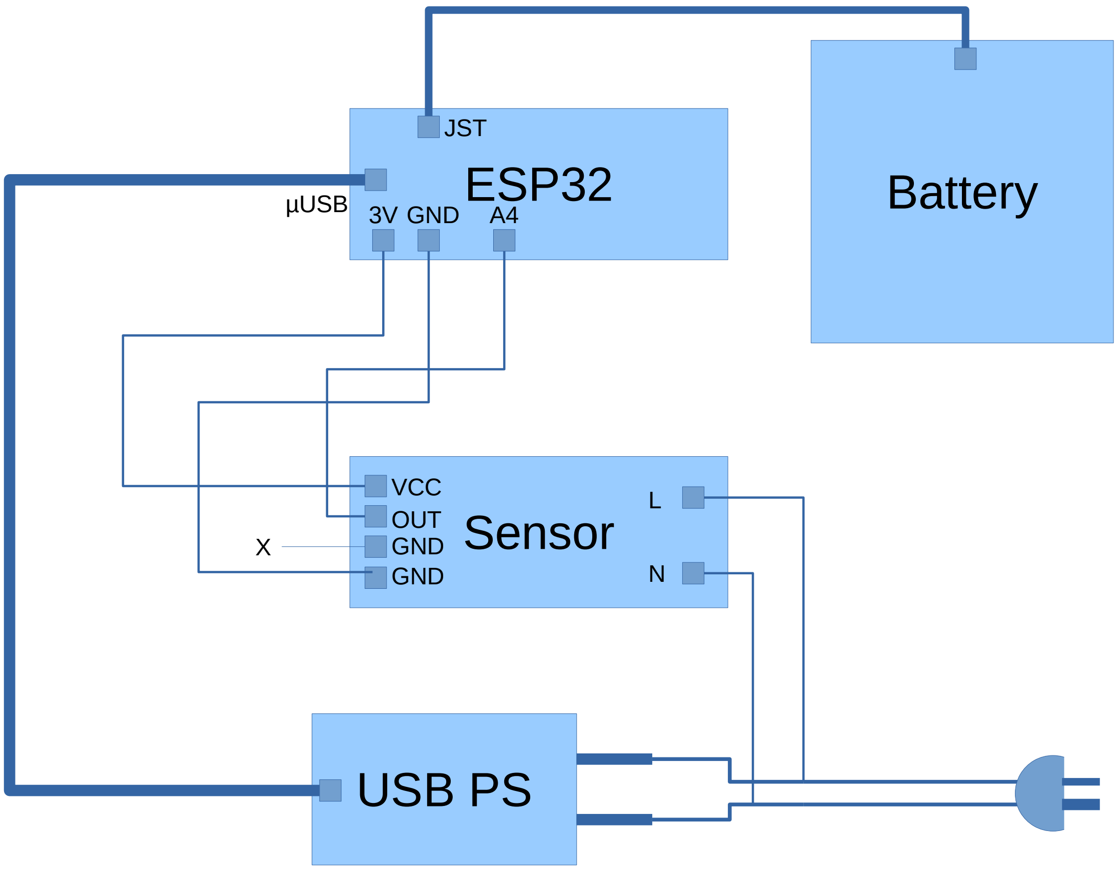
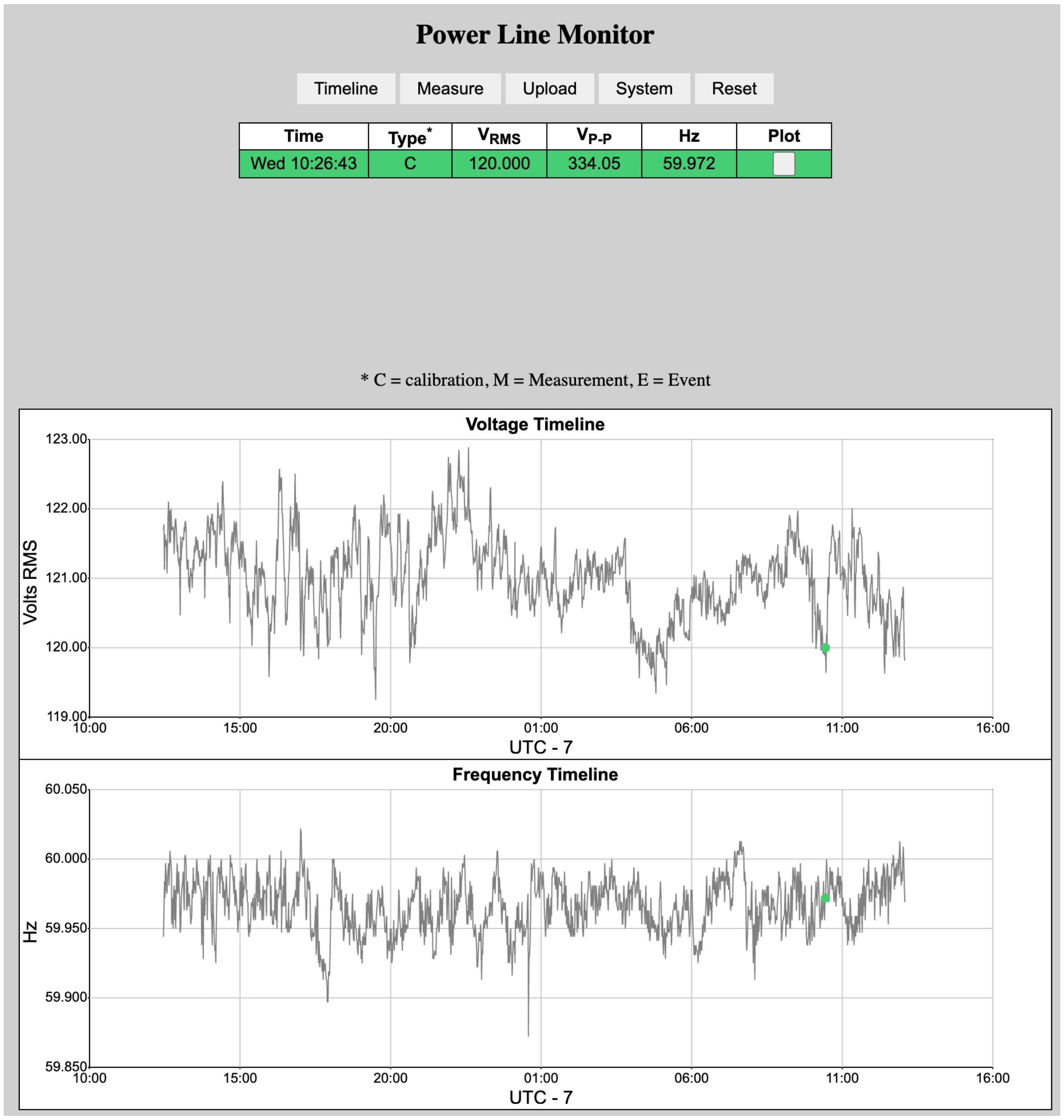


Figure 1: Power line monitor module wiring



Power Line Monitor

Time	Type*	V_{RMS}	V_{P-P}	Hz	Plot
Wed 10:26:43	C	120.000	334.05	59.972	☐

* C = calibration, M = Measurement, E = Event

Figure 2: Web page

Revision History

1.01	add subinterval tests
1.02	tweak colors
1.03	mark each event on Timeline
1.04	add Clear button; reduce voltage tolerance from 10 to 5; only draw events that fit within timeline range; widen plots
1.05	change tolerance to 7 V to allow for NERC standard; expand timeline to include all events; replace Clear with ReCalibrate
1.06	change sample rate to 620 Hz to resync sampling phase every 100 ms; fix frequency posting issue.
1.07	improve peak frequency algorithm; don't ack successful Resets; events remain in trend data after Reset; add more System stats.
1.08	another tweak to frequency algorithm; change button label from Refresh to Timeline.
1.09	add library §5.3.3; fix URL in §6; another tweak to frequency algorithm.
1.10	fix 59.9 Hz bug; improve network reliability; add tooltips to buttons
1.11	much faster Timeline drawing; Reset now renormalizes both voltage and frequency
1.12	don't plot events that occur before the timeline starts but leave them in the table
1.13	two day timeline; better wifi signal recovery
1.14	add more stats to System page; move web page to port 80; auto refresh the web page every 60 seconds; add Restart button; another tweak to wifi recovery; change voltage tolerance to match USA NEC as 120 or ± 6 V_{RMS}
1.15	also mention `LINE_V_TOL` and `LINE_HZ_TOL`; perform frequency determination in corrected coordinate system, not raw; add fast blink for each remote command received; add Reboot flag in System report.
1.16	stability improvements.
1.17	add WiFi events; collect data while WiFi down; expand acceptable range