

Netzanalysator

Christoph Bellmann

22.05.2025

Es soll ein Gerät am Netz gemessen werden. Netzurückwirkungen sollen berücksichtigt werden.

2. Team

- Softwareentwickler
Programmierung in c, web
- Projektmanager
OpenProject, Gantt-Diagramm
- Qualitätsmanagement
- Marketing

4. Anforderungen

Allgemein Anforderungen: Internetverbindung

Tabelle 1: Was soll gemessen werden:

| Größe | Wert [min-max] | Einheit |
|-----------|----------------|---------|
| Spannung: | 0 - 300 | V |
| Strom: | 0 - 5 | A |
| Frequenz: | 0 - 200 | kHz |

Details in der Spezifikation.

5. Spezifikation

Ein Gerät verursacht Netzurückwirkungen. Es ist festzustellen, ob die Höhe der Oberwellen noch innerhalb der Norm liegt.

5.1 Idee und Lösung

Analoges Frontend für AC Spannung + Strom, digital erfassen mit dem eingebauten ADC von einem ESP32 Mikrocontroller..

5.1.1 Elektronik

1) Schaltung Messung AC Spannung:

- Trenntrafo
- Operationsverstärker

2) Schaltung Messung AC Strom:

- Hall-Effekt Spule
- Operationsverstärker

5.1.2 Software

Basierend auf ESP32.

Hardware-Limiterungen:

- Prozessor: 240 Mhz Dualcore
- Speicher
 - Flash: 4 MB oder mehr/weniger
 - RAM: 4 MB oder mehr/weniger

Elektronik

Analoges Interface für die Messung:

Für die Spannungsmessung:

Isolierender Transformator und Operationsverstärker:

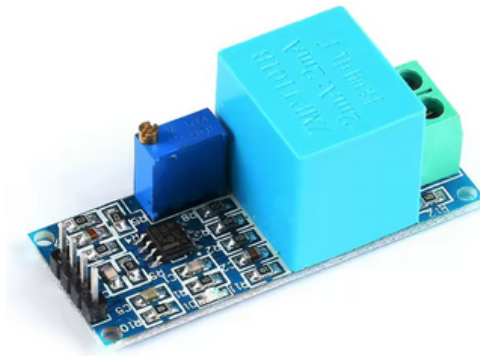


Abbildung 1: ZMPT101B

Für die Strommessung:

Berührungsfrei mit Messwandler, auch mit opAmp:

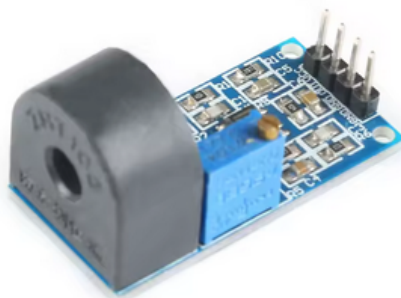


Abbildung 2: ZMCT103C

Power Analyzer / Speicher-Oszilloskop

Ein ESP32 soll das 230 V AC Netz auf Netzurückwirkungen durch Verbraucher zu überwachen.

Userinterface:

http://< ESP-IP >/ oder Websocket

Es sollen Scope | Harmonics| Recorder geben

Oszilloskop:

- Echtzeit-Wellenformen
- Einstellbare Zeit-/Spannungs-Scale & Trigger
- In Laufzeit zwischen „Interleaved“ und „Single-Channel“ umschaltbar

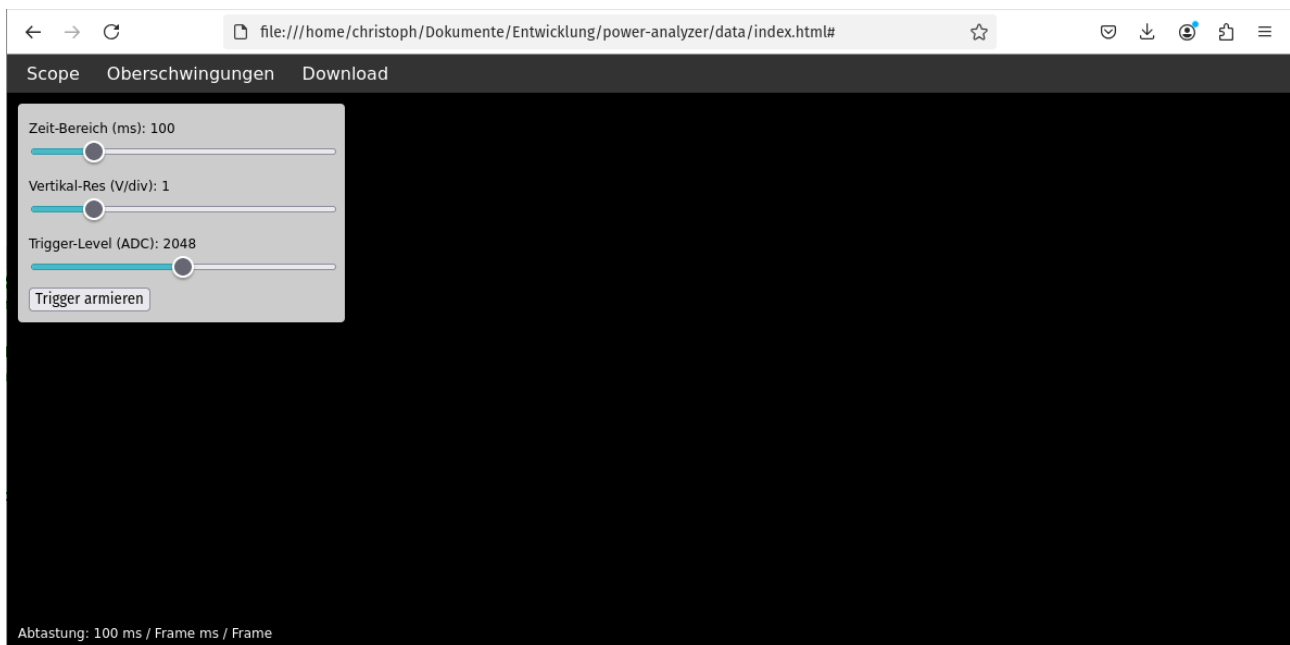


Abbildung 3: Scope-Screenshot-22052025

Analyse der Oberschwingungen:

- DC-Anteil
- Harmonische
- THD

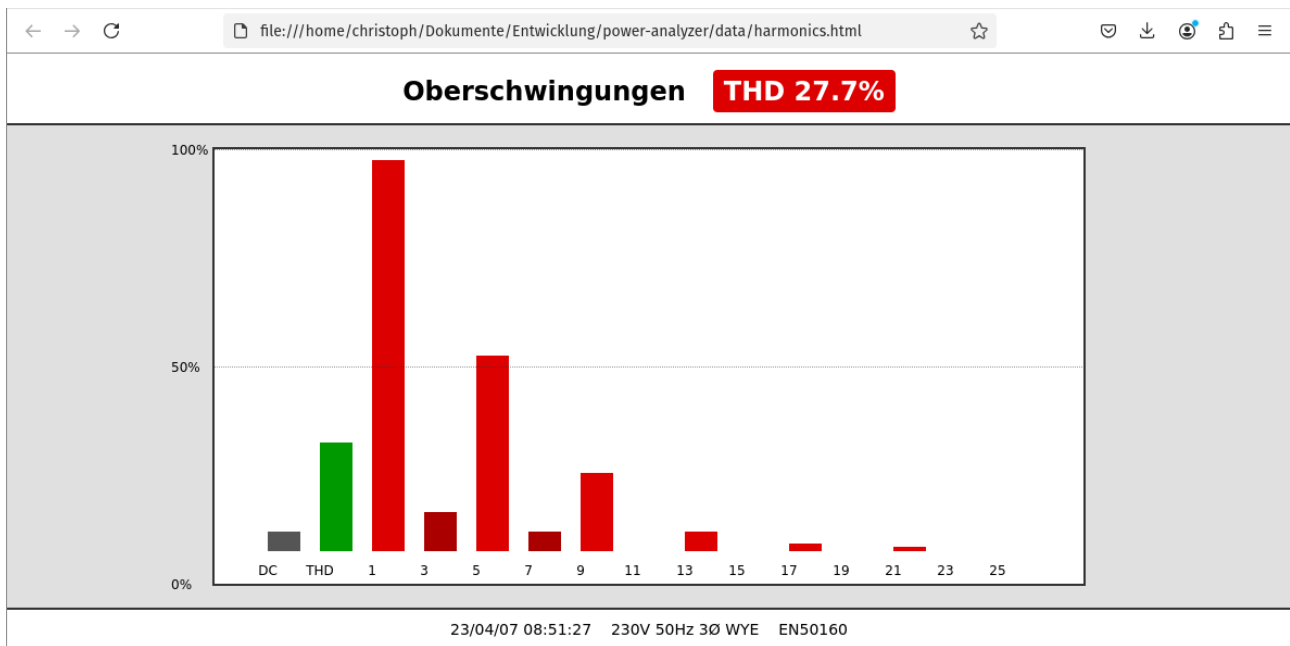


Abbildung 4: Harmonics-Screenshot-22052025

Download des Speichers (anders ausgedrückt: Aufnahme):

Klick Download -> speichert data.bin zur Auswertung z.B. in python.

Logs:

```
1 - ADC_FFT, HARMONICS, RECORDER -> ESP_LOG_INFO
2 - httpd_ws, httpd_txrx -> ESP_LOG_DEBUG
```

Ordnerstruktur

```
1 .
2 |-- data/
3 | |-- index.html # Oszilloskop-Frontend
4 | |-- harmonics.html # Oberschwingungs-Frontend
5 |-- include/
6 | |-- adc_fft.h
7 | |-- config.h
8 | |-- harmonics.h
9 | |-- recorder.h
10 | |-- webserver.h
11 |-- src/
12 | |-- main.c
13 | |-- adc_fft.c
14 | |-- harmonics.c
15 | |-- recorder.c
16 | |-- webserver.c
17 |-- README.md
```

-

Echtzeit-Oszilloskop (Spannung + Strom)

- **Oberschwingungs-/THD-Analyse** (DC + 1. bis 40. Harmonische)
 - **harmonics.c / harmonics.h**
Berechnet über FFT: DC-Anteil, 1. - 40. Harmonische und THD in % für Spannung und Strom.
- **Download** der letzten Messungen (interleaved Voltage, Current)
- **recorder.c / recorder.h**
Stellt über [/download](#) einen Binär-Stream bereit:
 1. Header: `uint32_t sample_rate, num_samples`
 2. Daten: `float32`-Paare (Voltage, Current)

Webserver-Komponente

- **webserver.c / webserver.h**
Startet HTTP- und WebSocket-Server, mountet SPIFFS und registriert alle Routen:

| Route | Methode | Beschreibung |
|------------------------------|---------|---|
| / | GET | <code>index.html</code> (Scope) |
| <code>/harmonics.html</code> | GET | <code>harmonics.html</code> |
| <code>/harmonics</code> | GET | JSON mit DC, Harmonischen & THD |
| <code>/ws_scope</code> | WS | Live-Daten für Oszilloskop (V, I) |
| <code>/ws_harmonics</code> | WS | Live-Daten für Oberschwingungen |
| <code>/download</code> | GET | Binär-Dump der letzten 1024 Samples (V,I) |

Installation & Build

1. Vorbereitung

- ESP-IDF installieren und Umgebung aktivieren
- Projekt-Ordner öffnen

2. SPIFFS vorbereiten

```
“bash idf.py spiffs-flash
```

Puffer-Daten des Oszilloskop

Die Zeit, die ein Puffer mit `FFT_SIZE` Samples bei einer Abtastrate von **200 kSPS** füllt, ist entscheidend für den horizontalen Maßstab eines Oszilloskops.

Dauer in ms

- **Abtastrate** $f_s = \text{SAMPLE_RATE} = 200\,000 \text{ Samples/s}$
- **Puffergröße** $N = \text{FFT_SIZE} = 1024 \text{ Samples}$

$$T_{\text{ms}} = \frac{N}{f_s} = \frac{1024}{200\,000} \times 1000 = 5,12 \text{ ms}$$

Bedeutung für das Oszilloskop:

Ein Paar aus Spannung und Strom wird in nur **5,12 ms** abgebildet – das entspricht einer horizontalen

Zeitauflösung von ca. **5 ms/div** (bei 10 Divisionen).

Ringpuffer-Länge

Speichert man z. B. **10** Puffer hintereinander, deckt der Ringpuffer ab:

$$10 \times 5,12 \text{ ms} = 51,2 \text{ ms}$$

So erhält man eine „Vergangenheit“ von knapp **50 ms** für Scroll-Back.

Extraktion zur Auswertung der Daten

Damit die Rohdaten nach dem Download bequem mit Python (z. B. NumPy) eingelesen werden können, speichern wir:

1. **Header** (im Binärformat):

- `uint32_t sample_rate` (z. B. 200000)
- `uint32_t num_samples` (z. B. 1024)

2. **Daten** als **float32, interleaved** ($V_0, I_0, V_1, I_1, \dots$)

Auf der PC-Seite lädt man sie dann z. B. mit:

```
1 import numpy as np
2
3 with open('data.bin', 'rb') as f:
4     sr = np.fromfile(f, dtype=np.uint32, count=1)[0]
5     n = np.fromfile(f, dtype=np.uint32, count=1)[0]
6     data = np.fromfile(f, dtype=np.float32).reshape(-1, 2)
7     voltage = data[:, 0]
8     current = data[:, 1]
9     t = np.arange(n) / sr
```

„ADC DMA“ steht für Analog-Digital-Wandler mit Direct Memory Access. In kurz:

1. ADC (Analog-Digital-Wandler)

Er wandelt kontinuierlich oder einmalig eine analoge Spannung in digitale Samples um (z. B. 0...4095 für eine 12-Bit-Auflösung).

2. DMA (Direct Memory Access) Ein Hardware-Controller, der Daten zwischen Peripherie und RAM überträgt, ohne den Hauptprozessor (CPU) für jeden einzelnen Wert zu beschäftigen.

Warum ADC + DMA?

- Hohe Abtastraten

Mit DMA kann der ADC Datenpakete (Frames) mit sehr hoher Geschwindigkeit in RAM schreiben (z. B. 200 kS/s), ohne dass die CPU bei jedem Sample eingreifen muss.

1 CPU-Entlastung

Die CPU wird entlastet und kann sich um FFT-Berechnungen, Web-Server-Tasks oder andere Anwendungen kümmern, während der DMA die Daten im Hintergrund puffert.

- Ringpuffer / Blockweise Verarbeitung

```
1 Typischerweise konfiguriert man den DMA so, dass er immer eine bestimmte
  Frame-Größe (z. B. 1024 Samples) in einen Ring- oder Ping-Pong-Puffer
  schreibt. Wenn ein Frame voll ist, wird per Interrupt oder Task-
  Benachrichtigung signalisiert: "Puffer ist voll -- hier stehen die nä
  chsten 1024 Samples bereit."
```

Typischer Ablauf auf einem ESP32

1. Konfiguration

- Im `adc_continuous_config_t` gibt man an, mit welcher Abtastrate gearbeitet werden soll und ob Single-Channel oder Interleaved-Modus (zwei Kanäle) benutzt wird.
- Im DMA-Handle (`adc_continuous_handle_t`) legt man fest, wie groß der Puffer sein soll.

2. Starten

- `adc_continuous_start(adc_handle)` aktiviert ADC + DMA.
- Der DMA-Controller beginnt, ADC-Daten in den Puffer zu schreiben.

3. Lesen

- Per `adc_continuous_read(...)` holt man blockweise alle neuen Samples aus dem DMA-Puffer.
- Danach können die Daten (z. B. Spannung und Strom) in einem FFT-Task ausgewertet oder über WebSocket verschickt werden.

4. Ideen

Ausblick.

5. Zitate und Quellen