Auto-dialing Alarm System for Power, Temperature, and Humidity Monitoring

EE344: Electronic Design Lab
Group TUE-01

Abhiruchi Kotamkar (2100700043)
Shiv Bharuka (210070079)
Shlesh Gholap (210070080)
Shobhit Maheshwari (210070081)

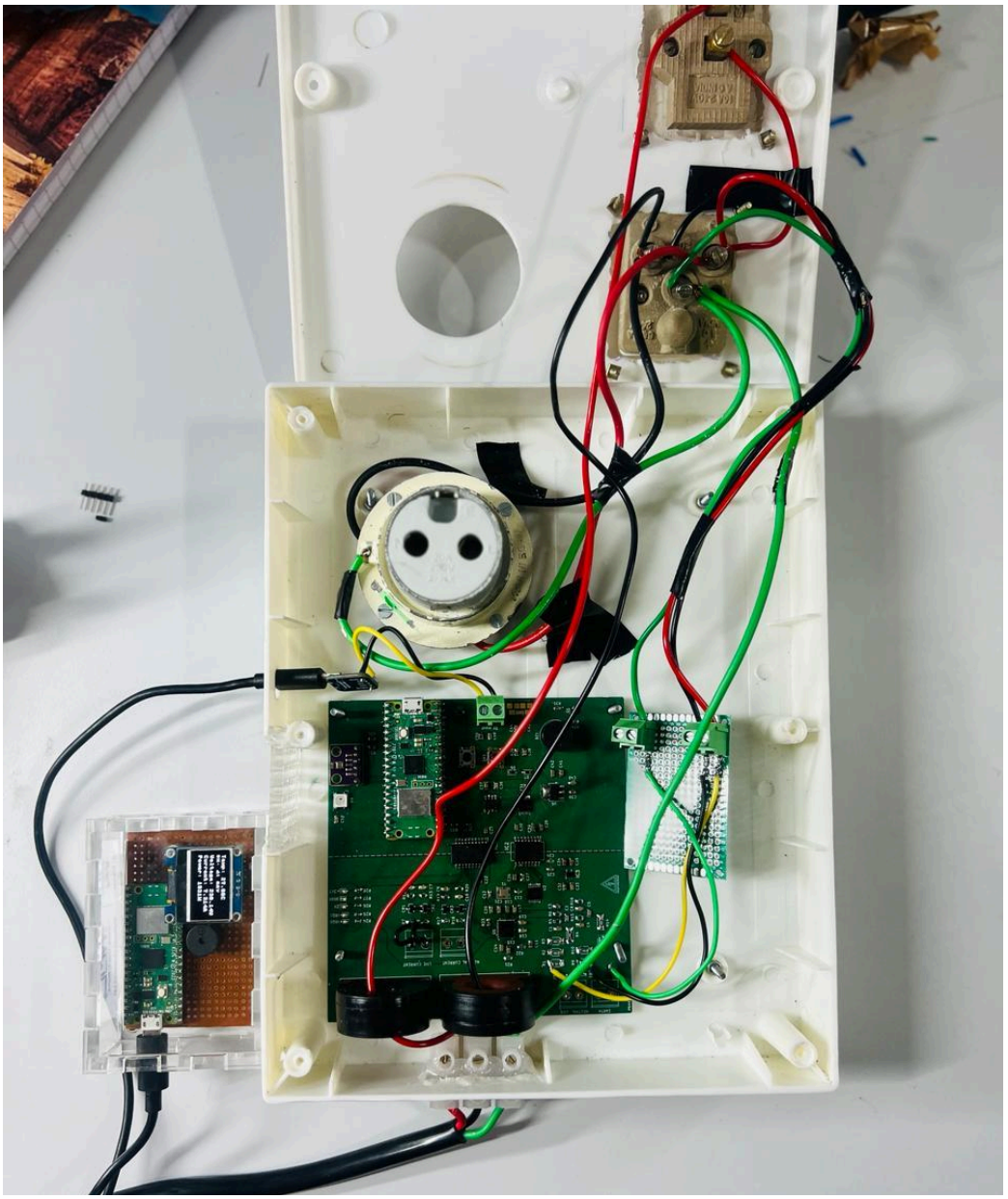


Table of contents

Features and Overview

- Continuous health monitoring and alarm system for high-power appliances such as Air Conditioners, Humidity Regulators, Lab equipment, and all appliances up to 20kW.
- Machine health monitoring by analyzing harmonic distortions in voltage waveform
- Data such as power consumption, local temperature and humidity, and voltage waveform are uploaded to a remote database accessible from anywhere.
- Multi-unit integration over LAN in a setting such as a laboratory or factory with the help of a central unit that coordinates data logging.
- Alarm system with push notifications, email, and call-based alarms on user-defined thresholds for all monitored values provided by the central unit

File organization

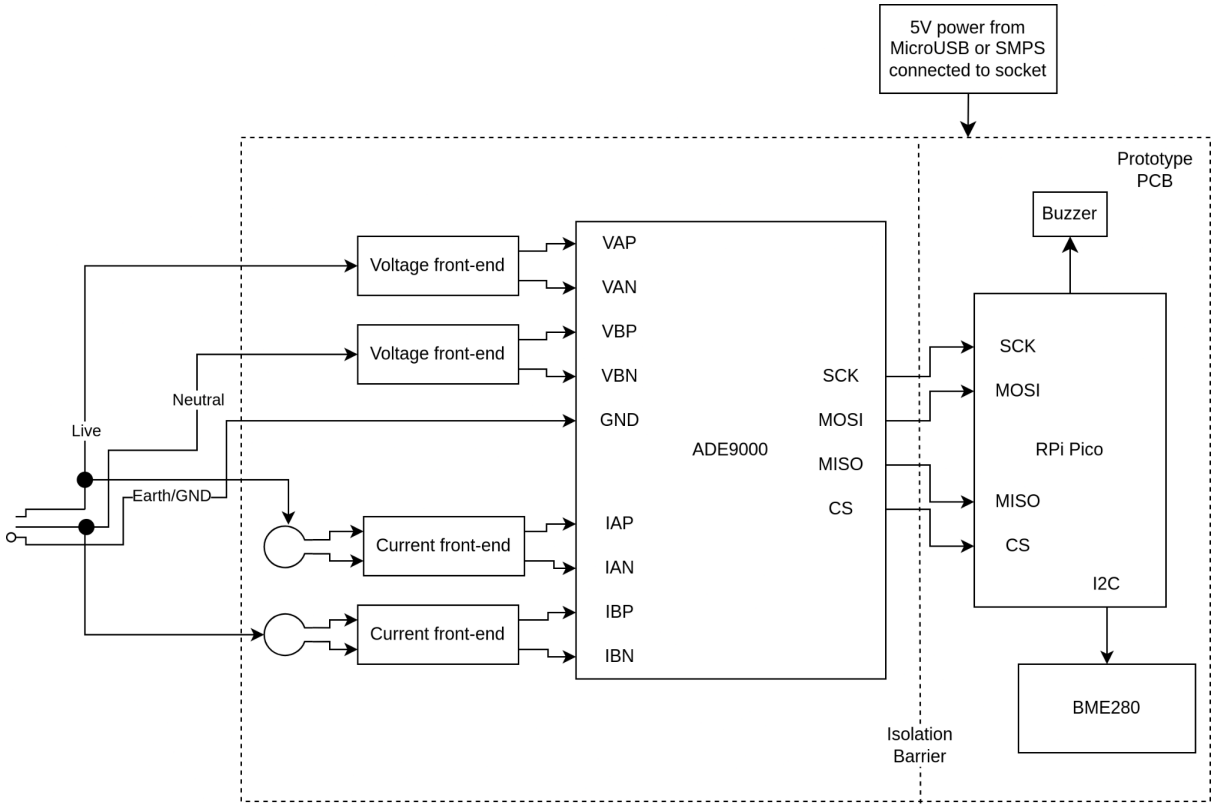
The following folders will be present in the root directory of the project:

- **`energy_meter`**: Contains KiCad design files for the sensing unit PCB
- **`Software`**: Contains all firmware used in the project
    - **`RP2040_code`**: Source code for the firmware of the sensing unit written using the RP2040 C/C++ SDK. The code may be built using CMake or the uf2 file in the build subdirectory may be flashed directly to the RPi Pico W.
    - **`arduino_code`**: Source code for the sensing unit based on arduino. Based on code provided by Analog Devices for the EVAL-ACE9000 SHIELDZ board.
    - **`central_code`**: Source code for the central unit written in MicroPython
    - Other python files are for setting up a TCP client to test data logging
- **`CAD`**: Contains files for the enclosures of the sensing unit and central unit
- **`Test`**: Contains test data from a long-term test using an Air Conditioner
- **`central_unit`**: Contains KiCad design files for a central unit with a SIM800L module. This is not used in the final prototype but may be useful for future iterations
- **`Website`**: Contains Code for a locally hosted website to view real-time data from the sensing units
- README file
- Documentation file (this file)

This documentation may be changed. Find the live version here:
https://docs.google.com/document/d/e/2PACX-1vSk20U0GbC1EArys-i2jhloWY60GNWYztmFV4l-aTbJ1UrM4m5yABL2bQwfzQ4_mtzRKHXNJLR0H1ZW/pub

Hardware documentation

Sensing unit



- Based on the ADE9000 Power Monitoring IC and the RPI Pico W Micro-controller
- Design is referenced from the design of the EVAL-ADE9000 SHIELDZ evaluation board with significant changes
- By default, the circuit will allow the ADE9000 to start in PowerMode 3 which
- allows SPI communication. Deep sleep can be toggled by the MCU

Changes from the EVAL-ADE9000 SHIELDZ

- Only two current channels and two voltage channels (Live and Neutral) and a connector for the high powered side ground connection (Earth).
- We have added screw terminals to connect voltage sources. Along with the current transformer input, we have added screw terminals for current input as well
- The RaspberryPi Pico W is used instead of an Arduino Zero as both are 32 bit (ARM Cortex M0+) MCUs.
- We do not need the EEPROM as the flash memory of the RPi Pico is enough (2MB)
- The board requires 5V power. This can be drawn from a microUSB port or an SMPS or any equivalent 5V DC source. There are screw terminals and a microUSB connector for these connections

Isolation circuitry

- The high-power circuit (ADE9000 and input channels) must be electrically isolated from the low-power logic circuitry. However, signals must be exchanged between them for the MCU to interpret data from the ADE9000
- An SPI isolator (ADuM4151) is used to transmit SPI signals (MOSI, MISO, SCLK,
- SS/CS)
- The ADE9000 does not receive power from the socket but instead from the power supply in the logic circuit. This power is fed using a power isolator (ADuM6404) at 5V. Some logic signals have to be level shifted to 5V and then back to 3.3V for transmission which is done by the SN74LV4T125PWR buffer.

Voltage and current channel calculations

We are measuring power consumption of 1-Phase 3-Wire Residential sockets. According to the ADE9000 Technical Reference Manual, this requires two voltage channels and two current channels (one for the live wire and one for the neutral wire. The earth wire is used as ground for the high power circuit).

Burden resistors are used in current channels to prevent the inrush of current. A diode configuration similar to a bridge rectifier is used to allow for AC current. The voltage input is scaled down using a voltage divider circuit made of high-power resistors. Both the channels produce a differential input to the ADE9000

- Voltage divider:
    - We have four 200kOhm resistors in series (instead of one resistor to manage heat dissipation) and a 1kOhm resistor across which the voltage is measured.
    - This results in a gain of 1/(800+1) = 0.0012.
    - A nominal voltage of 240 V rms is expected. The ADC will receive 0.299 V rms = 42.7% of full-scale reading.
- Current channel burden resistor:
    - We are using a 1:2500 current transformer.
    - The expected load for a three-star 2-ton AC is 7-8 A rms. The input to the ADE9000 must not exceed 0.707 V rms. Let us assume a typical sustained current of 10 A rms.
    - Thus for the burden resistor we must have (10/2500)*2R < 0.707 or R < 88 ohm.
    - We choose R = 20 for a nominal reading of 0.16 V rms = 25% of the full scale reading. The 24-bit ADC provides a good resolution (4.21e-8 V) for a small reading as well.

Other peripherals

- BME280 sensor for temperature and humidity measurements
- RGB Led (WS2812B) for state indication
- Buzzer to act as a local alarm

Troubleshooting

Recommended testing sequence is:

- Check voltage input at low power side: 5V

- Check voltage output of AMS1117: 3.3V
- Check voltage input to power isolator: 5V
- Check voltage output of power isolator: 5V
- Check voltage output of linear regulator at high-powered side: 3.3V
- Check reference voltages generated by ADE9000:
    - AVDDOUT: 1.9V
    - DVDDOUT: 1.7V
    - REF: 1.25V
- Verify IRQ1 is low and IRQ0 is high through test pads
- Verify CS signal is high through test pad
- Verify SPI communication by polling the PART_ID register of the ADE9000

In the final prototype, we are using a zero-PCB with screw terminals and ferrite beads as input voltage channels due to a short circuit that happened during mains voltage testing which resulted in the voltage tracks getting damaged. This is not needed in any other sensing unit.

Central unit

Implements the alarm system and acts as a bridge between multiple sensing units and the real-time interface for device health monitoring. Currently it is made by soldering modules on a perforated board or zero-PCB.

Modules

- RPi Pico W
- 0.96 Inch OLED display
- Buzzer

Software documentation

Sensing unit

ADE9000 driver

The driver is adapted from the Arduino code provided by Analog Devices (https://www.analog.com/en/resources/evaluation-hardware-and-software/evaluation-boards-kits/eval-ade9000-shield.html#eb-relatedsoftware) but optimized to work well with the C/C++ SDK of the RP2040. The register map is referenced from the datasheet. Wrapper functions for SPI communication are provided to make reading from registers in the ADE9000 easier. SPI burst read functions to read coherent sampled waveform data and decoherent sampled waveform data are also available. Function-wise documentation can be found in the code files.

BME280 driver

The driver is built using the official C API for the BME280 provided by Bosch (https://github.com/boschsensortec/BME280_SensorAPI). I2C communication is used. All default settings are used as provided by Bosch.

Wireless communication

The lwIP library creates a lightweight TCP server with one port (https://www.nongnu.org/lwip/2_1_x/group__tcp__raw.html). This continuously transmits data in the following form:

- Data from register polling such as Power, Temperature, and Humidity (60B)
- Waveform buffer1 (512B)
- Waveform buffer2 (512B)

The waveform buffers are to be concatenated to get ~4 cycles of voltage readings.
The server automatically restarts if the connection with a client is broken, which allows the central unit to poll multiple servers in quick succession. The sensing unit sends data on a specific IP address and Port which is required to be defined in the central unit.

EEPROM in Flash

A customized linker script (`memmap_custom.ld` in `Software/RP2040_code`) is used to reserve 16kB in the flash memory of the RPi Pico as non-volatile EEPROM used to store configuration data for the ADE9000. This eliminates the need for a separate EEPROM, which is used in the Analog Devices Shield

Central unit

The complete implementation of the Central Unit is available in the repository at (`Software/central_unit.py`). The data received (60B + 512B + 512B) from the sensing unit via the TCP server gets unpacked and stored using the following decoding scheme:

- Time: Received as unsigned integer (first 4B)
- VArms, VBrms, IArms, IBrms: Voltage and current readings (Each 4B float)
- PowerA, PowerB: Power readings (Each 4B float)

- PfA, PfB: Power Factor reading (Each 4B float)
- freqA, freqB: Frequency readings (Each 4B float)
- Temperature, Pressure, Humidity: Environmental sensor readings received from BME280 (Each 4B float)
- emergency: Unsigned integer (4B) whose each bit represents a trigger in any of the defined parameters, providing 32 emergency interceptions.
- waveform_buffer: Stores the last 512 short integers (2B) representing waveform data.

OLED Display

The temperature, relative humidity, voltage RMS, current RMS, and power readings are displayed on the 0.96-inch OLED display with a resolution of 128x64 pixels. Establishes communication with an SSD1306 OLED display over an I2C bus. The display used runs on the SSD1306 OLED driver. The MicroPython driver to display data on the display was taken from (https://github.com/stlehmann/micropython-ssd1306). Optionally displays a full-screen alert message, overriding the sensor data display in case any of the pre-defined thresholds are crossed.

Buzzer

A buzzer is connected to a general-purpose input/output (GPIO) pin on the Raspberry Pi Pico. It specifically utilizes Pico's PWM (Pulse Width Modulation) hardware capabilities for volume control. The buzzer goes off if any of the pre-defined thresholds are crossed.

- Buzzer Pin: BUZZER_PIN (default: 15) specifies the GPIO pin connected to the buzzer.
- Sound Configuration: BUZZER_FREQUENCY (default: 500 Hz) sets the desired frequency of the buzzer sound. BUZZER_DURATION (default: 1 second) defines how long the buzzer will sound. BUZZER_DUTY_CYCLE (default: 100%) controls the buzzer's volume (when the signal is on within a cycle).
- Functions:
    - play_buzzer(): Plays a buzzer sound with the configured parameters, using PWM for volume control.
    - stop_buzzer(pin_num=15): Stops the buzzer sound, regardless of the play duration set in play_buzzer.

MongoDB

The data(1084B) is received once every 10 seconds and is stored on an online database management service called MongoDB. MongoDB is a non-SQL-based service and mainly stored data as JSON files. MongoDB allows downloads of the data stored as CSV or JSON files. It also allows API calls to receive the data by running a simple Python script (`Software/Mongodb_datascrap.py`). Two databases were created on the MongoDB dashboard, one of which stores the waveform data and the other of which stores the rest of the sensor readings. Each data point received is sent to the respective databases using API calls before receiving the next data point.
The main reason for this slow sampling rate is that the uploads to the MongoDB database using API calls take a few seconds. This sampling rate can be improved using a more powerful microcontroller as a central unit or a self-hosted server(local or cloud-based). The API calls to MongoDB took less than a second when made from a Windows PC.

Website

The data received on MongoDB is then used to create time-labeled interactive plots which are deployed on an HTML page (`Website/index.html`) that is hosted on GitHub pages. The plots are updated every minute, and the plots for the last 10 minutes are displayed, including temperature, relative humidity, voltage RMS, current RMS, and power readings. The website can also be accessed from phones and provides all the functionalities.
An additional aspect of the project was to analyze appliance health, which is done by looking at the voltage waveforms received at every data point. The data that was collected on the Waveform database of MongoDB is used to make voltage waveform plots along with a periodogram to identify the dominant voltage frequencies (`Test/Waveform_plot.png`). A bad appliance's health correlates with the frequency in the periodogram. Identifying frequencies other than 50 Hz in the periodogram hints at problems with the appliance's health. A Python script (`Software/waveform_plot.py`) runs on a local server (can be scaled to an online host); it scraps the waveform data from the MongoDB database and stores the plots of waveform and periodogram assuming a sampling rate for voltage waveform as 8000, locally. The local server is then used to host the plot created on an HTML page (`Website/waveform.html`). Note that these plots can only update (with a refresh every 5 seconds) when a local server runs. An example of such a plot is shown on the website.

Push Notifications and Mailing

One of the most important components of the project is alerting the user which is done by using a software called ntfy.sh, which is an HTTP-based Publisher-Subscriber notification service. Initially, users need to subscribe to a topic (we used EDLProjectTUE01) they want to receive the notifications. Users can connect to the topic from any device (phone, iPad, laptop) that supports ntfy.sh without the required to provide a phone number. If any thresholds get crossed, instantaneous and customized push notifications and emails are published on the topic and sent to the user's email ID depending on the parameters that are not in the expected range. The central unit makes HTTP requests to connect to the service and send an alert message. The paid version of this service can even make calls that use a text-to-speech algorithm to convert the message into a robotic voice and make calls to specific users.

Possible future work

- Improvement of waveform capture using decoherent sampling. Currently, coherent sampling is used with resampled data
- Providing power to the logic circuit through the mains itself (may use an SMPS)
- Multi-unit integration beyond three sensor units using a sensor mesh (may use Bluetooth instead of LAN). This would require operation without a central unit and uploading data directly to the cloud, making this a proper “IoT” device
- Analysis of long-term power consumption and waveform data to detect deterioration in device health
- Shrinking of form factor and re-design of outer case for ease of mounting

Additional references

- Project github: https://github.com/Ru0110/EE344
- Project drive folder (contains BOM and other relevant files): https://drive.google.com/drive/folders/13c8EIVbcSlq_PN7hLz76V1K4JDQLavGz?usp=sharing
- The website that displays plots made: https://shleshgholap.github.io/EDL-Project/