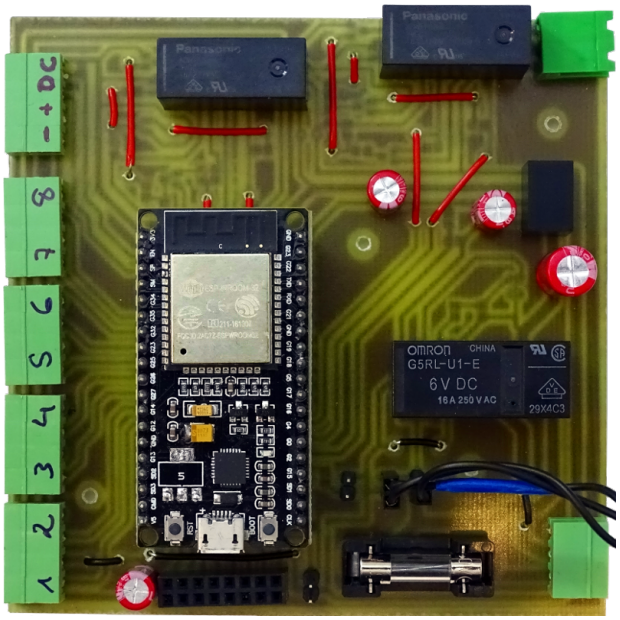
# Dedicated Sensor Array




28/1/2018

Max Hackinger & Marc Schäfer

# 1 Introduction

The Dedicated Sensor Array is in essence a system for measuring and collecting environmental data. The implementation is generic so the System can work with a variety of different sensor types and measure a large spectrum of environmental parameters.

The System is composed of two parts, one central server and several dedicated sensor stations. The sensor stations are battery powered. It measures environmental parameters, evaluates the raw data and sends the values to the server over WiFi. The server receives, collects and makes the data accessible for additional applications.

The technical requirements for the server are a JVM and WiFi connectivity. The WiFi connectivity can either be a direct Hotspot created by the system running the JVM or the system can be part of a network with a WiFi bridge. The sensor station requires an ESP-WROOM-32 Module with an Espressif ESP32 MCU.

The sensor station has a low power design to offer a long runtime. They provide several, different ADC for the measurements, a potential free relay for triggering external events and a voltage supervisor to protect the battery against deep discharge. Furthermore, the power supply of the sensor station is extendable by a solar charging module.

For the moment there are three environmental parameters that are measurable: temperature, light intensity and magnetic field strength. For monitoring of the runtime the battery voltage can also be measured.

The server can handle multiple connections from several sensor stations. It combines and stores all sensor data received from every sensor station in a database table.

# 2 Execution & Technical Details

The two parts, the server and the sensor station, were respectively assigned to one team member. The server was assigned to Max, the sensor station to Marc. Initially, the general framework was determined and interfaces were defined. Subsequently, the two parts were independently developed and successively aligned. This was necessary because of the completely new development environment regarding the ESP32 and the therefore unknown possibilities and obstacles.

The server is written in Java and uses the SQL database Apache Derby. At first startup the database is automatically configured. It listens for TCP/IP connections on a configurable port. A JSON based protocol is used for communication between server and sensor station. The server waits for a JSON frame from the sensor station, processes the frame and responds in turn with a JSON frame, containing an acknowledgement and/or commands for the sensor station.

For the sensor station, a custom PCB was designed and made to connect the sensors and $I^2C$ bus. Additionally, the low dropout regulator (LDO) of the ESP-WROOM-32 dev board was replaced by a high-efficiency DC switching regulator. The sensors and the $I^2C$ bus can be fully turned off to save energy while not acquiring analog signals. The firmware for the ESP32 is written in C++. The sensor station uses different run modes. In the normal run mode, the sensor station measures data, evaluates and stores it to the flash memory. Subsequently, it enters a lower run mode called deep sleep. In this mode, the sensor station uses just a fraction of the energy used in the normal run mode and can wait for a special event or for a specific time to wake up and enters the normal run mode again. After a defined amount of measuring cycles, the sensor station enters the communication mode and turns on WiFi. It connects to the server, sends all the collected data, waits for a response and enters the normal mode again. Received commands get processed and if necessary, the sensor station enters the communication mode again. Otherwise, the sensor station begins a new measuring cycle.

| Mode: | Deep Sleep | Normal Operation | Communication |
|---|---|---|---|
| Power: | 1-5 mA | 15-20 mA | 50-100 mA |

Table 1: Power consumption in different modes

# 3 Downsizing

The scope of our project needed to be reduced somewhat as dejan had to leave the group because of private reasons. It was originally planned to create a GUI application that

would represent the data that the Sensor Station sent the Server, but this proved to be too challenging for just the two of us. The timing was especially unfortunate, because this happened midway through development, when they basics were defined but the details still had to be implemented and it therefore was too late to change the distribution of the tasks to compensate for the loss. This lead to the project becoming a Dedicated Sensor Array with a flexible server, instead of a Weather Station with a touch screen to control the sensors and visualize the data.

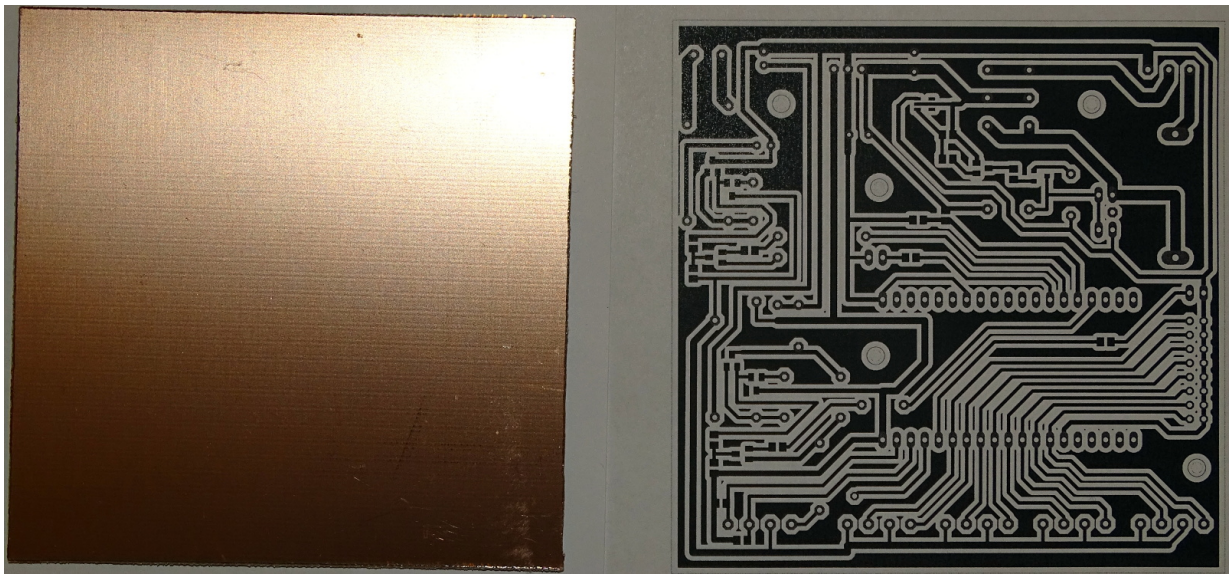
# 4 Conclusion

On the microcontroller side, we discovered a little later then we would have liked that the NodeMCU Firmware that we were planning to use, was simply to resource intensive and buggy for our needs. This was not as big a problem as it could have been through, because we used the original Espressif ESP-IDF in combination with a Arduino plugin.

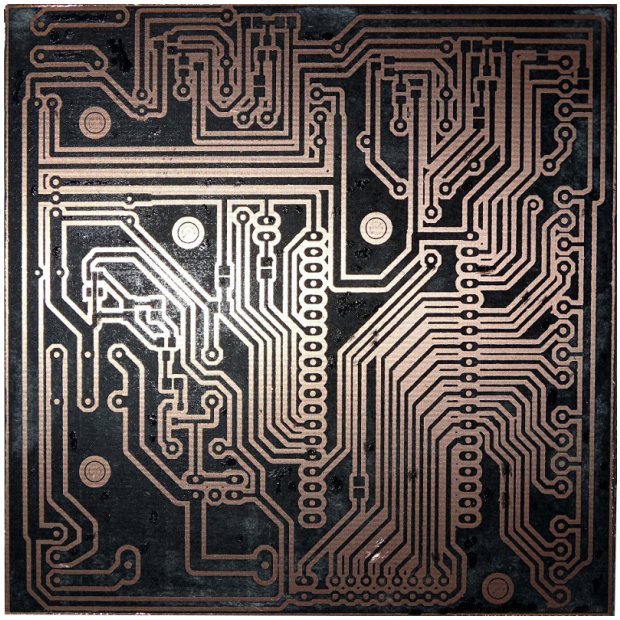
Another surprises was that the built in ADC (Analog to Digital Converter) on the ESP32 was insufficiently sensitive for our use case because of the unusual low input impedance. Thankfully there was an external ADC available which fulfilled the requirements.

Communication in the group worked well, even though we mostly meet remotely over Skype. However the project could have probably profited from a more regular frequency of meetings, for example twice a week. This might have made it clearer earlier on in the project that one of our members was overwhelmed and needed more support.

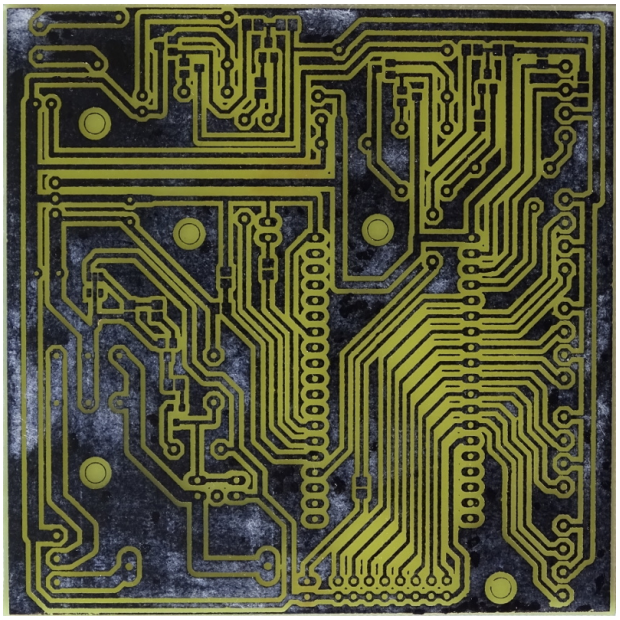
# 5 Pictures



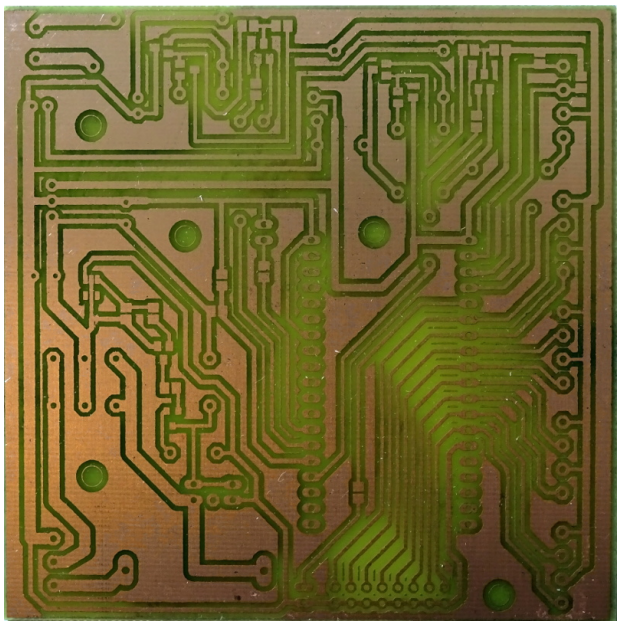
(a) Raw board (left) and printed circuit layout (left)



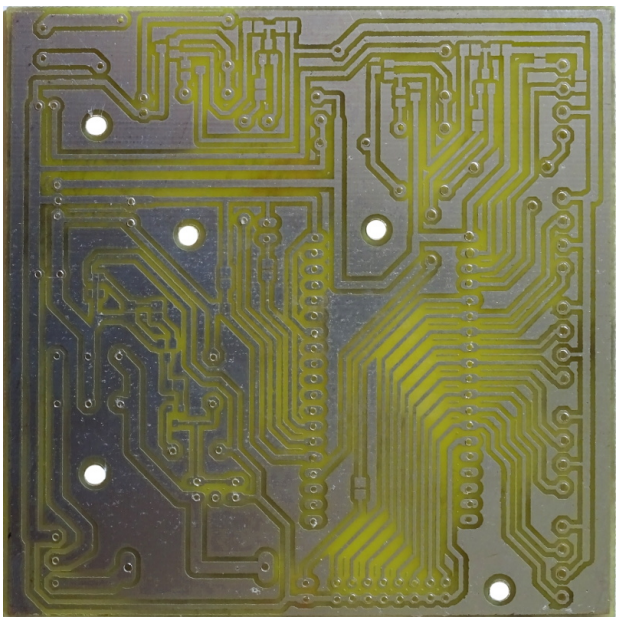
(b) Layout transferred to the board



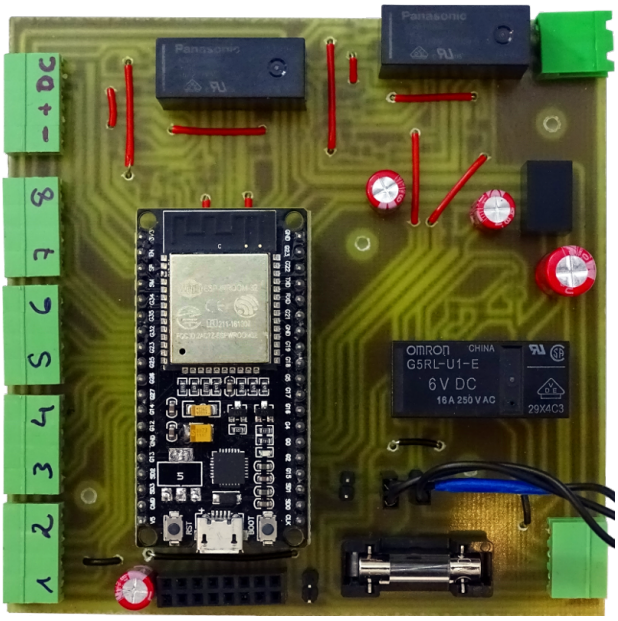
(c) Circuits etched into the board



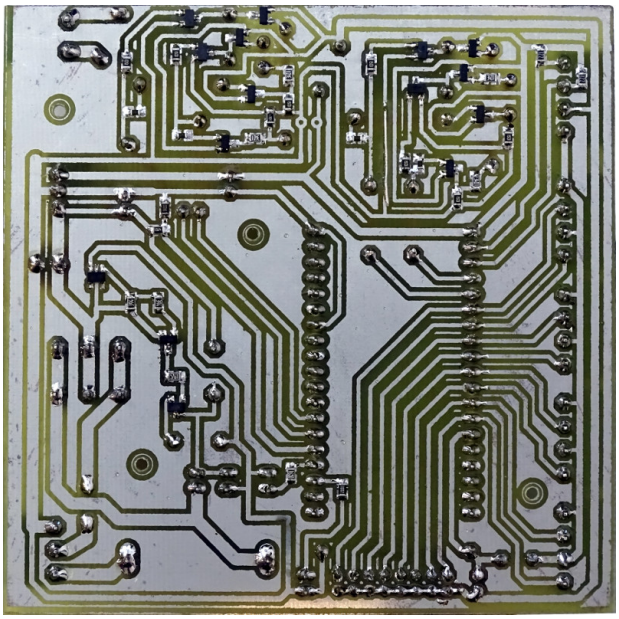
(a) PCB cleaned



(b) Plated with tin




(c) Assembled PCB (top)



(d) Assembled PCB (bottom)

# References

[1] Espressif, ESP IDF, https://github.com/espressif/esp-idf.git

[2] Espressif, ESP IDF API, https://esp-idf.readthedocs.io/en/latest/index.html

[3] Espressif, Xtensa Toolchain,
https://www.espressif.com/en/products/hardware/esp32/resources

[4] Espressif, ESP Arduino ESP32, https://github.com/espressif/arduino-esp32.git

[5] iggr, MKFSS, https://github.com/igrr/mkspiffs.git

[6] bblanchon, Arduino JSON, https://github.com/bblanchon/ArduinoJson.git

[7] Adafruit, Adafruit ADS1X15, https://github.com/adafruit/Adafruit_ADS1X15.git

[8] Apachce, Derby, https://db.apache.org/derby/

[9] Google, gson, https://github.com/google/gson