Smart Energy Meter (SEM)

Practice Enterprise

2021 - 2022

Sander Speetjens

1 Embedded Software

Version 1.0

Saturday 27th August, 2022
- 16:47

sander.speetjens@gmail.com

Preface

I'm Sander Speetjens, a first year student Electronics Embedded Software at Thomas More - Campus De Nayer.

This bundle will be a summary of all of the information that I gathered over the course of half a semester about my project Smart Energy Meter (SEM) or making my digital electricity meter more smart with a separate observation station. I had chosen this subject, because recently our energy supplier installed one of their new energy meters at home and I wanted to know our energy consumption. The system had to be reliable in the long term, but also consume little energy. This was quite challenging in terms of software, because there was a design constrained that we couldn't use any pre-written libraries or platforms like Arduino. With this practice enterprise, I want to demonstrate that I can apply my knowledge gained in secondary school and university to both the Electronic and Mathematical parts. Moreover, this was the perfect moment to focus on my favourite subjects, namely the combination of hardware, software and mathematics.

Because I could not have achieved all this without help from others, I would like to thank a few people. First and foremost, I would like to thank my teachers for assisting me with the technical and mathematical aspects of my practice entry. I would also like to thank my friends who have worked together with me to make this possible. Finally, I would also like to thank my parents, who have always supported me and helped me review my assignments.

Contents

Figure 1: The new "Smart Meter" from Sagemcom

1 — Theory

1.1 The Smart Meter

1.1.1 Introduction

The digital meters for electricity and gas were introduced in Flanders on 1 July 2019. Since then, they have replaced the old, mechanical meters that are no longer in circulation. Fluvius automatically installs them at new housing developments or renovations where the electricity connection is renewed. In addition, Fluvius launched a major geographical roll-out this spring to install such meters at every household and small business in Flanders. This will be organised by region, town or municipality. The roll-out aims at a complete conversion by 2029, with the main acceleration in the next three years. By the end of 2024, 80% of homes should have digital meters.

1.1.2 What is a Smart Meter

A smart meter is an electronic device that records information such as consumption of electric energy, voltage levels, current, and power factor. Smart meters communicate the information to the consumer for greater clarity of consumption behavior, and electricity suppliers for system monitoring and customer billing. Smart meters typically record energy near real-time, and report regularly, short intervals throughout the day. Smart meters enable two-way communication between the meter and the central system. Such an advanced metering infrastructure (AMI) differs from automatic meter reading (AMR) in that it enables two-way communication between the meter and the supplier. Communications from the meter to the network may be wireless, or via fixed wired connections such as power line carrier (PLC). Wireless communication options in common use include cellular communications, Wi-Fi (readily available), wireless ad hoc networks over Wi-Fi, wireless mesh networks, low power long-range wireless (LoRa), Wize (high radio penetration rate, open, using the frequency 169 MHz) ZigBee (low power, low data rate wireless), and Wi-SUN (Smart Utility Networks). `https://en.wikipedia.org/wiki/Smart_meter`

1.1.3 Objective

How to read the required data from the smart meter with my microcontroller. Wich protocol does it use and how do I translate the data to useable variables.

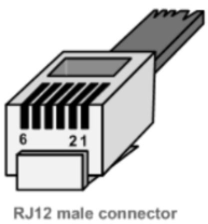
1.1.4 Expectation

I think that the meter will use a simple protocol like RS232 because it only uses 1 data signal and one ground for one-way comunication. I also expect that the meter will send out the data in clear text because it's easier to decipher then a custom data format.

1.1.5 User-Ports

These digital meters contain 2 User ports, which you are able to read you're consumption/production statistics from. The P1 port follows the DSMR 5 standard of the Dutch Smart Meter extended with the e-Mucs specification. The S1 port provides a limited possibility of data and is going to be removed from the newer meters and is not going to be used in this project.

1.1.6 Physical connection



Pin	Signal
1	5V power supply
2	Data request (5V)
3	Data GND
4	Not connected
5	Data (open drain)
6	GND

Figure 1.1 & Table 1.1: RJ12 connector and the pinout for the P1 port

1.1.7 Internals of the meter

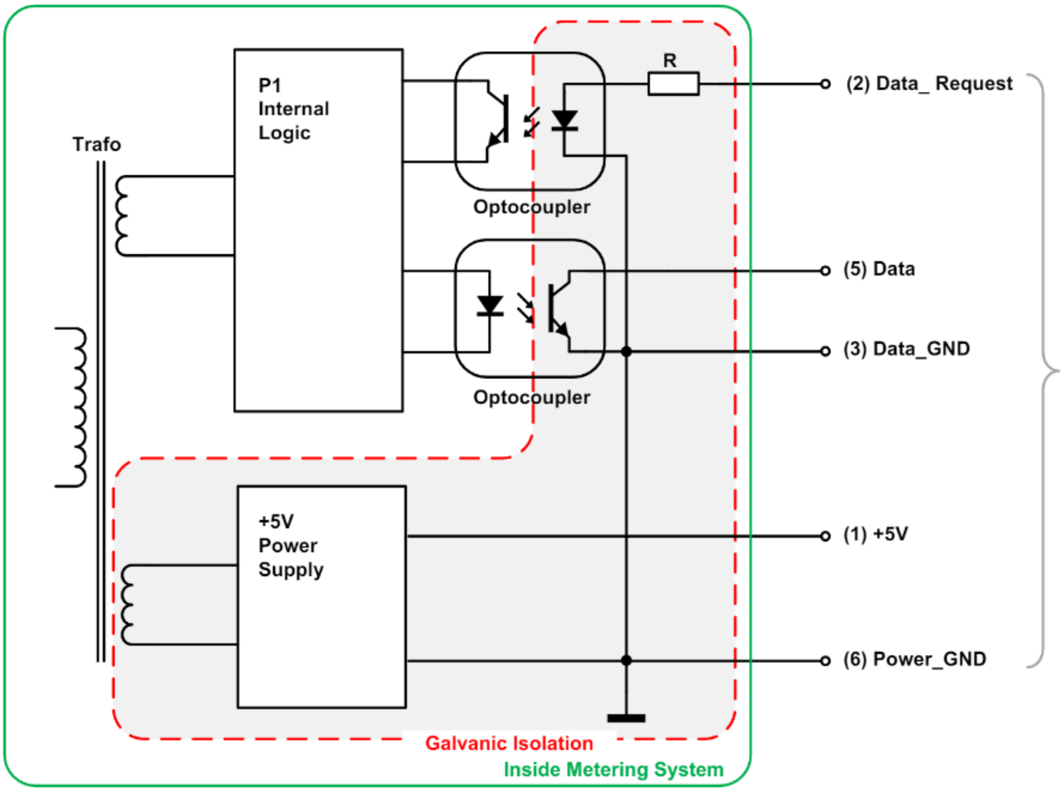


Figure 1.2: The internal workings of the Fluvius Meter

+5V Power supply

The P1 interface provides a stable +5V DC power supply via "+5V" (pin 1) and "GND"(pin 6) lines to provide a connected IoT device with a power source.
U = 5,0 V (max = 5,5 V with I = 0 mA , min = 4,9 V with I = 250 mA)

Data request

The P1 port is activated (will start sending data) by setting "Data request" (pin 2) high (4,0V to 5,5V). While recieving data, this line must be kept high.
Warning: To stop receiving data the "Data request" line must be put in a high impedance mode and must not be connected to the GND or 0V

Data

Here we run into a problem, due to the use of optocouplers, the "Data" (pin 5) line must be designed as an Open Collector output and must be logically inverted or inverted via software before it can be used with IoT devices.
A "Data" line LOW has a voltage of 0,2 V (0 - 1V), HIGH has a voltage provided by a pull-up resistor to the VCC of the microcontroller with a maximum current of 30 mA.

Communication Protocol

The interface must use a fixed transfer speed of 115200 baud.

The Fluvius Smart Meter sends its data to the connected IoT device every single second and the transmission of the entire P1 telegram is completed within 1s.

The format of transmitted data is defined as “8N1”. Namely:

- 1 start bit,
- 8 data bits,
- no parity bit and
- 1 stop bit.

Data readout

The Fluvius Smart Meter transmits the data message, as described below, immediately following the activation through the Request signal.

/	X	X	X	5	Identification	CR	LF	CR	LF	Data	!	CRC	CR	LF

Figure 1.3: Photo of one transmission message

End of transmission

The data transmission is complete after the data message has been transmitted. An acknowledgement signal is not provided for.

Data objects

Go to the DSMR 5 standard and e-Mucs specification for more information. `https://www.netbeheernederland.nl/_upload/Files/Slimme_meter_15_a727fce1f1.pdf` and `https://www.fluvius.be/sites/fluvius/files/2019-12/e-mucs_h_ed_1_3.pdf`

1.2 Ethernet

1.2.1 Introduction

Now we have our data,but we don't have a place to store it. The microcontroller doesn't have enough memory to store all of our data, so we have transfer this data to another system. Here comes Ethernet/ WIFI to the rescue. Because it is used on every modern computer system it becomes super easy to implement, also our microcontroller is specially made to be an Internet of Things (IoT) device. Because it has a lot of features out of the box like WIFI, Ethernet and Bluetooth with it's prepackaged libraries or also known as stacks to make it super straight forward to use.

1.2.2 What is ethernet

This is a computer networking standard for Local Area and Wide Area Networks that is commonly used today.
See the link for more info: `https://en.wikipedia.org/wiki/Ethernet`

1.2.3 Objective

How can we connect our microcontroller to another device over ethernet and with wich protocol are we going to use, that is both easy to use and widly used. In what format are we going to send our data over the air to that device.

1.2.4 Expectations

My background in computers says me that TCP/IP and HTTP are the most commonly used protocols over ethernet.

1.2.5 TCP/IP and HTTP

What is TCP/IP

The TCP/IP model was designed and developed by the US Department of Defense (DoD) in 1960s and is based on standard protocols. It stands for Transmission Control Protocol/Internet Protocol. The TCP/IP model is a concise version of the OSI model. It contains four layers, unlike seven layers in the OSI model.

1. Network Access Layer

   This layer corresponds to the combination of Data Link Layer and Physical Layer of the OSI model. It looks out for hardware addressing and the protocols present in this layer allows for the physical transmission of data. ARP is a protocol of the Internet layer, but there is a conflict about declaring it as such or as a Network access layer protocol. It is described as residing in layer 3, being encapsulated by layer 2 protocols.

2. Internet Layer

This layer parallels the functions of OSI's Network layer. It defines the protocols which are responsible for logical transmission of data over the entire network. The main protocols residing at this layer are :

- IP – stands for Internet Protocol and it is responsible for delivering packets from the source host to the destination host by looking at the IP addresses in the packet headers. IP has 2 versions: IPv4 and IPv6. IPv4 is the one that most of the websites are using currently. But IPv6 is growing as the number of IPv4 addresses are limited in number when compared to the number of users.
- ICMP – stands for Internet Control Message Protocol. It is encapsulated within IP datagrams and is responsible for providing hosts with information about network problems.
- ARP – stands for Address Resolution Protocol. Its job is to find the hardware address of a host from a known IP address. ARP has several types: Reverse ARP, Proxy ARP, Gratuitous ARP and Inverse ARP.

3. Host-to-Host Layer

   This layer is analogous to the transport layer of the OSI model. It is responsible for end-to-end communication and error-free delivery of data. It shields the upper-layer applications from the complexities of data. The two main protocols present in this layer are :

   - Transmission Control Protocol (TCP) – It is known to provide reliable and error-free communication between end systems. It performs sequencing and segmentation of data. It also has acknowledgment feature and controls the flow of the data through flow control mechanism. It is a very effective protocol but has a lot of overhead due to such features. Increased overhead leads to increased cost.
   - User Datagram Protocol (UDP) – On the other hand does not provide any such features. It is the go-to protocol if your application does not require reliable transport as it is very cost-effective. Unlike TCP, which is connection-oriented protocol, UDP is connectionless.

for more info go to
`https://www.geeksforgeeks.org/tcp-ip-model/`

HTTP

Hypertext Transfer Protocol (HTTP) is an application-layer protocol for transmitting hypermedia documents, such as HTML. It was designed for communication between web browsers and web servers, but it can also be used for other purposes. HTTP follows a classical client-server model, with a client opening a connection to make a request, then waiting until it receives a response. HTTP is a stateless protocol, meaning that the server does not keep any data (state) between two requests.

Methods HTTP defines a set of request methods to indicate the desired action to be performed for a given resource. Although they can also be nouns, these request methods are sometimes referred to as HTTP verbs. Each of them implements a different semantic, but some common features are shared by a group of them: e.g. a request method can be safe, idempotent, or cacheable. These are the most used once:

- GET: The GET method requests a representation of the specified resource. Requests using GET should only retrieve data.
- POST: The POST method submits an entity to the specified resource, often causing a change in state or side effects on the server.
- PUT: The PUT method replaces all current representations of the target resource with the request payload.
- DELETE: The DELETE method deletes the specified resource.

for more info over http go to
`https://developer.mozilla.org/en-US/docs/Web/HTTP`

1.2.6 Implementation

We will use the HTTP and TCP/IP stack for handeling data. And we will use the HTTP Post method to send the data over the stack.

1.3 Containers, Webservers, Databases and Graphing

1.3.1 Introduction

At this point we have the data send over the network and we need a way to receive it and store it in a safe place. Therefor we need to design the receiving and of the system on a PC, this handles the data storage side of the whole project.

1.3.2 Objective

- We have to handle the incomming HTTP Post request and put it in a database.
- We have to choose which webserver we are going to use with which language.
- We have to choose the database software.
- We have to choose the graphing software.
- We have to make it work a lot of PC's therefor it will have to be containerized via Docker.

1.3.3 Software selection

I'm going to use Free Open Source Software (FOSS) for this project. For the database I will use MariaDB, for the webserver I'm using apache2 and I'm using PHP for server side scripting. For graphing I will use Grafana.
And for the containers I'm using Podman. This is a free and opensource alternative to Docker, developed by Red Hat.

1.3.4 Realization

For the realization I created a Pod, this is a way to combine multiple containers and let them talk to each other. We have to connect the ports that the pod uses to our system and give them permanent storage on our disk in the form of shared volumes.
Then we have to set up the database, tables in the database, create users with specific permissions, create the php script and make a front end on Grafana.

2 — Project specifications

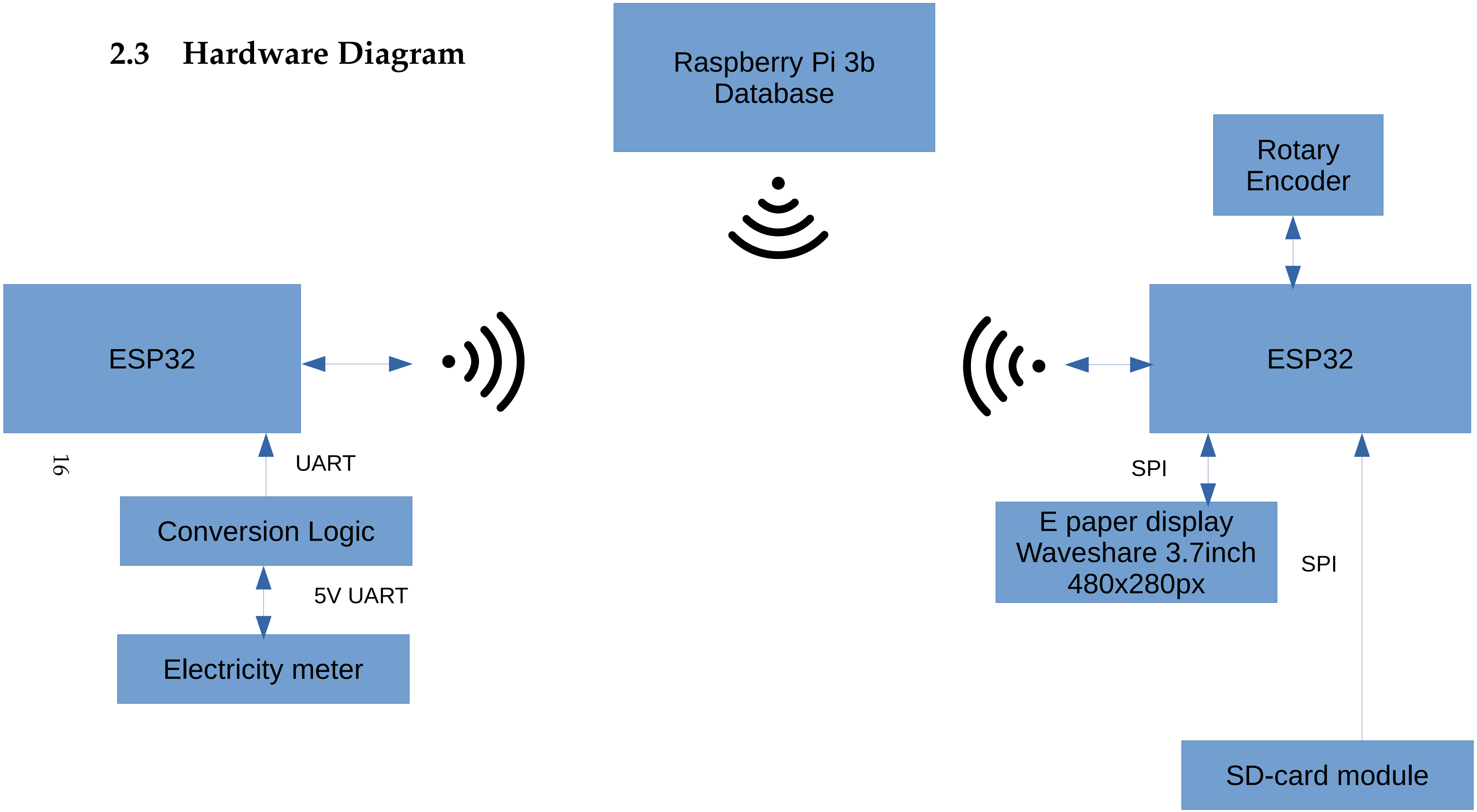
2.1 Components used

- ESP32-WROOM (€5)
- Raspberry Pi/ PC (Free)
- extra components (€20)

2.2 Proposed Project Specifications

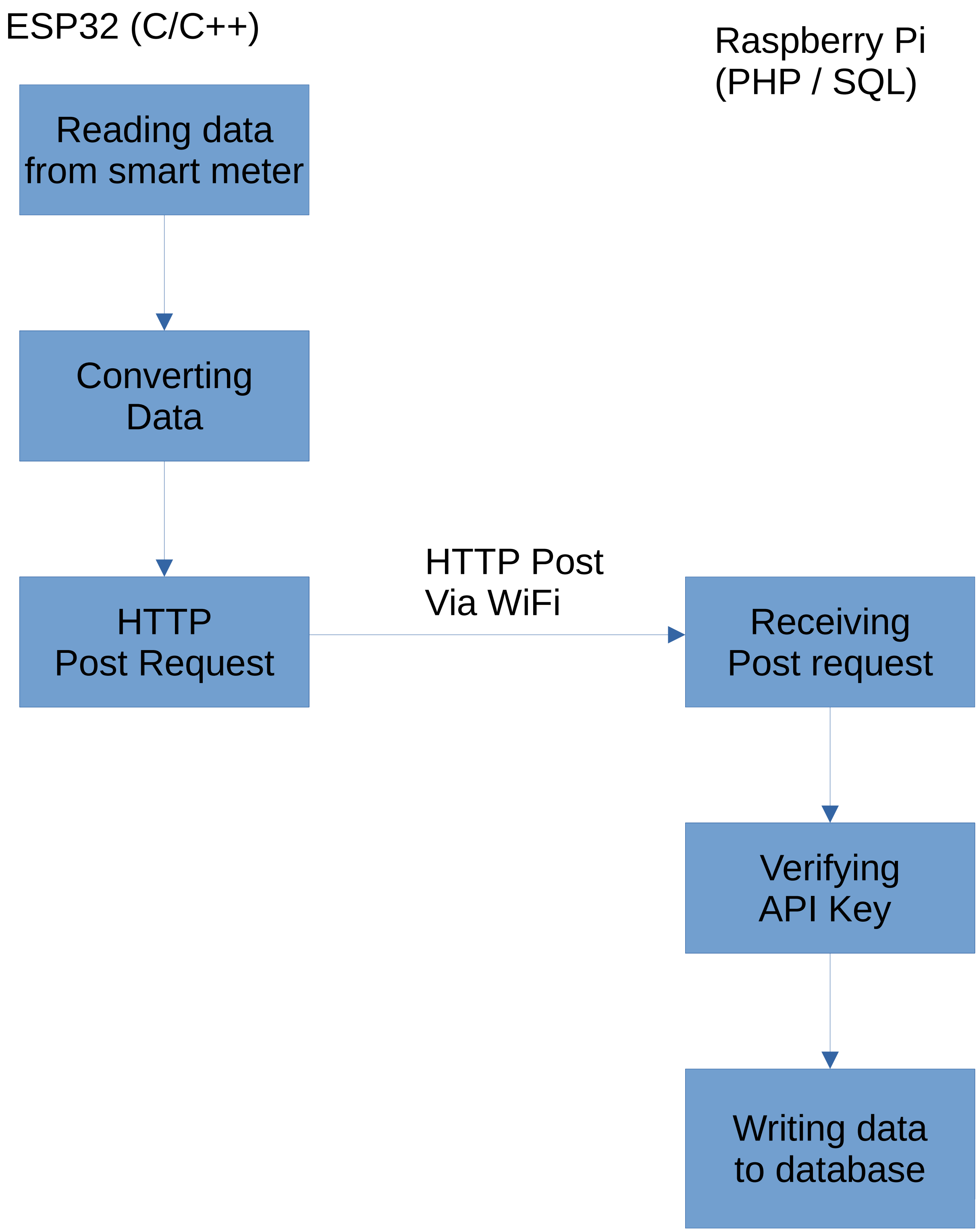
The project is composed out of 2 parts: The sensor, the database with a web page that contains the graphs.
The sensor side of the project contains all of the hardware that converts the data from the digital energy meter and sends that data to the database over WIFI once every couple of minutes (1-5 min).
The database stores all of the records in a table. The data is received and send via a web-interface written in PHP and can be received via http requests. There will also be a dashboard that can be accessed via a web browser. But this is not important for this project.
The Grafana web interface is a way to interface to all kinds of datasources to combine them in one interface.
All of the parts are powered by a 5V wall adapter.

2.3 Hardware Diagram

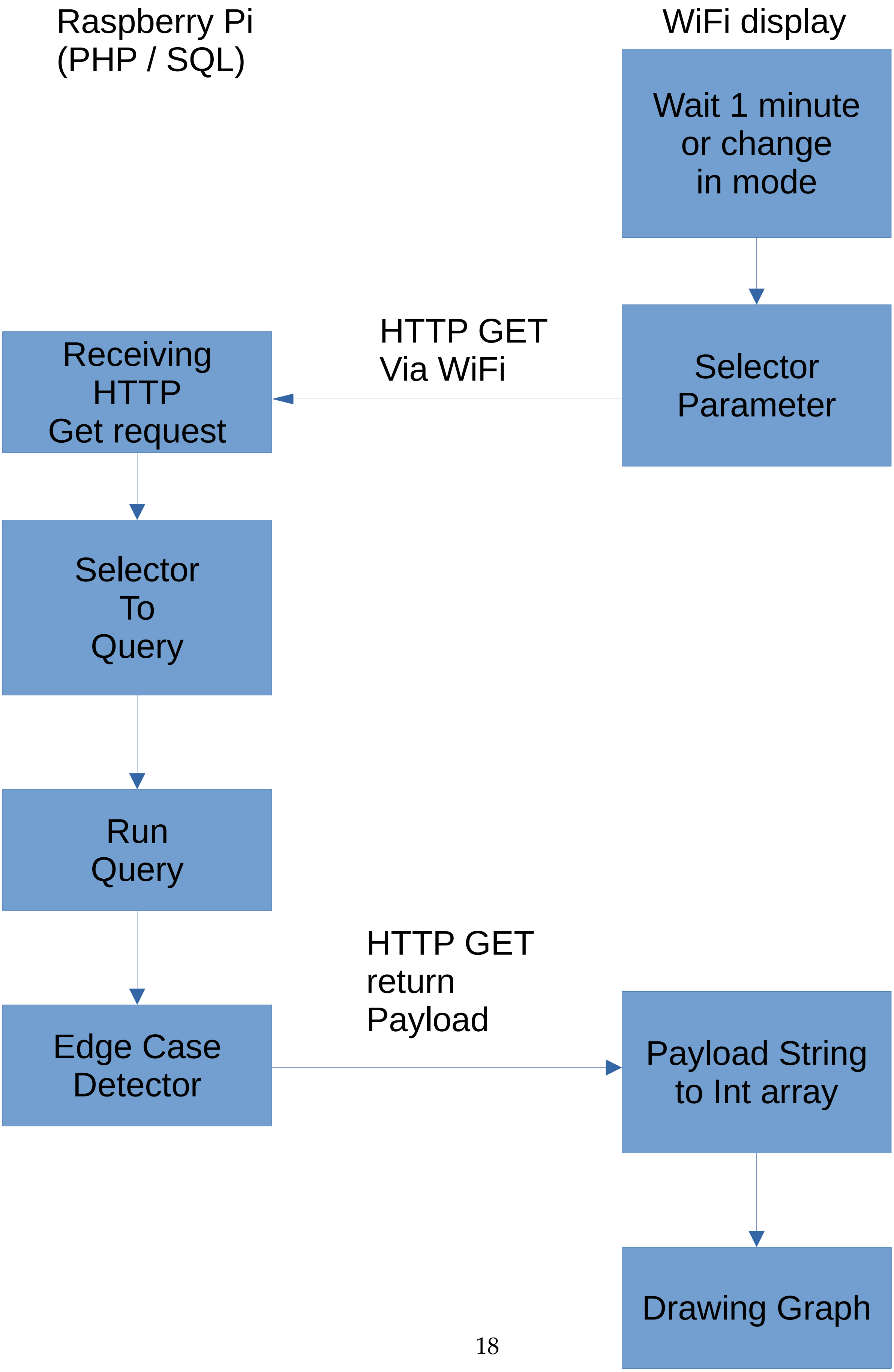


Sander Speetjens 1 (E)ES

2.4 Block diagram



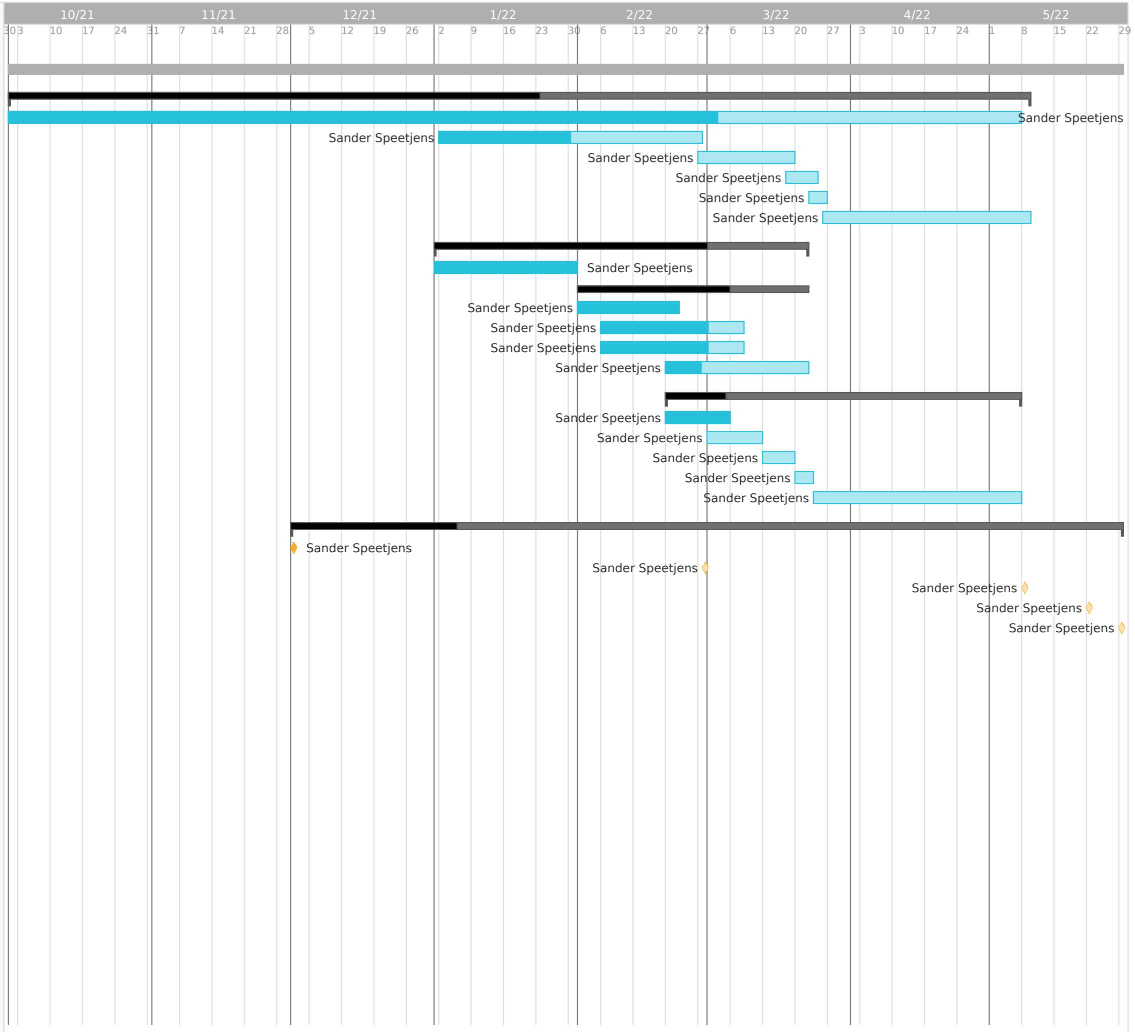
2.5 Block diagram



2.6 Gantt Chart

teamgantt
Created with Free Edition

Task	Hours	Progress
Practice Enterprice	**0h**	**52%**
Measuring (KW/h meter)	**0h**	**52%**
Research HAN Port	0	70%
Design Schematic	0	50%
Design PCB	0	0%
Buy PCB	0	0%
Testing and assembly (Possible redes...	0	0%
Software	0	0%
Database (Container)	**0h**	**73%**
Setup Pod for LAMP stack	0	100%
Setup database	**0h**	**66%**
tables	0	100%
receive script	0	75%
send script	0	75%
formats	0	25%
Observation (Epaper display)	**0h**	**17%**
Design Schematic	0	100%
Design PCB	0	0%
Buy PCB	0	0%
Testing and assembly (Possible redes...	0	0%
Software	0	0%
Milestones	**0h**	**20%**
Source Main Parts	0	100%
Test individual components	0	0%
Test PHP scripts	0	0%
Test Graphing	0	0%
Test Sending	0	0%



3 — The actual build

3.1 My Goals

- Sensor Reached
  I'm able to read a mock message from a meter, decode it and send it to a database via an HTTP post request. The only part that didn't work was the PCB.
- Database Reached
  I'm able to receive post and get requests and give or receive the required data back.
- Display data via grafana Reached
  I'm able to show data in graphs on a web interface.
- Learn about the ESP32 and it's IDE Reached
  I'm able to initialize, build, upload and monitor projects via Microsoft Visual Studio Code. And I'm able to understand the documentation from Espressif.
- Better understand how to use LaTEX for creating documents Reached
  This whole document is written in LaTEX with the exception of the blockdiagrams.

3.2 Details

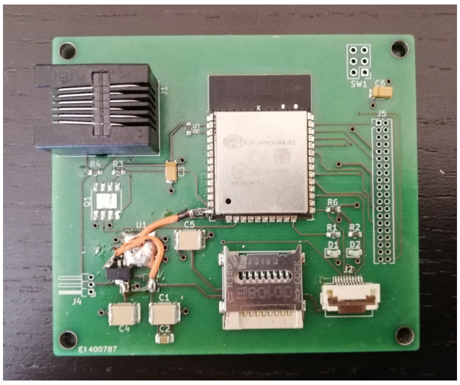


Figure 3.1: What my PCB looks like

- ESP32
  The esp32 is a 32 bit dual core LX6 RISC V Microcontroller from Xtensa. Build for the purpose of IoT applications. It has 30 GPIO pins (6 are reserved for the SPI Flash), onboard UART, SPI, I2C and PWM. It also has 2-4MBytes of onboard flash.

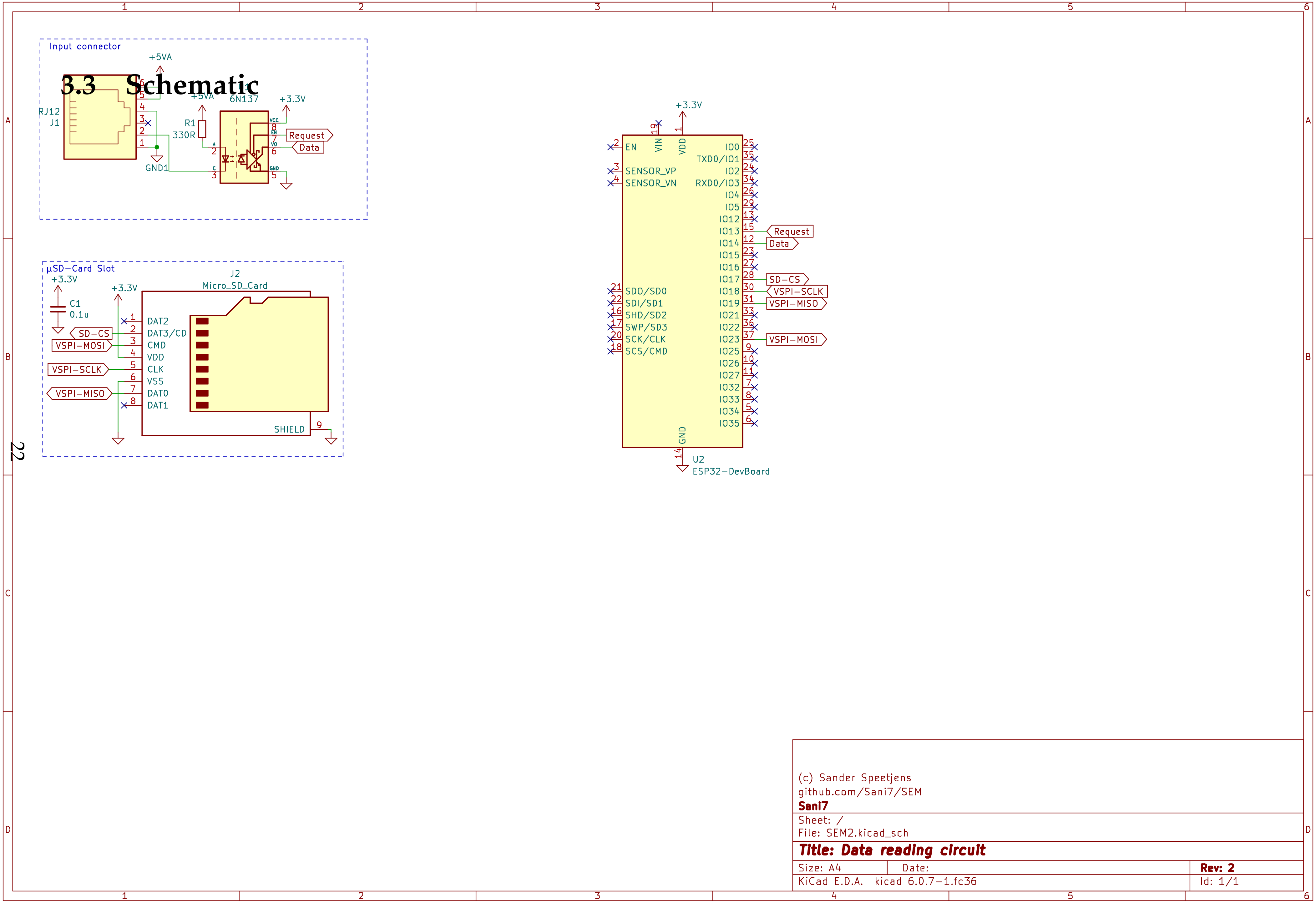
- Conversion Logic
  The logic is pretty easy, the data pin is of type Open Drain, but we have to isolate the meter from the microcontroller. For enabling and disabling I'm using the enable pin on the opto-isolator instead of the enable pin of the meter. That pin is tied to 5V, always enabled.

- SD card
  The SD-Card is connected to the VSPI pins, it has a FAT filesystem and contains one file config.txt which contains the necesary info to connect to wifi, the server ip, port, uri and API key. The code for the SD card is untested because I wasn't able to connect one to the ESP32. In the format: configName=configText\n I'm currently working on a way to set the wifi credentials and other config via a web interface on the esp32 running as a acces point.

- Display
  This is implemented on the computer.

- PCB
  I gave up on my PCB after one month of troubleshooting , but I eventually figured out what the issue was. An overvoltage most likely destroyed both my ESP32 and my programmer. And I only began writing code 2 weeks before the deadline because I was too focussed on trying to find the hardware mistake.
  Afther the first presentation I created a small circuit using wires and components and soldered a new circuit together.

3.3 Schematic



3.4 Learning Curve

There is a big difference between using a microcontroller like the XC888, ATMEGA32U4 (What I previously used) and the ESP32

- CPU architecture: intel 8051 assembly <=> ATMEL RISC Core <=> Xtensa LX6 RISC V dual core processor
- IO: Almost every IO function is available on evey pin ex every pin can be a UART pin or PWM etc
- Task based instead of one flow with interrupts
- The documentation is complicated, you have to know where to look for something
- Almost impossible to program in Assembly because it has so much functionallity
- On board WiFi, BL, BLE
- TCP/IP stack via included libraries
- ... (more them I'll ever need)

3.5 Code

For the code go to `http://www.github.com/Sani7/SEM/tree/main/Code` I'm not going to list my code in this document, there is just to much of it and it will become a mess. And you have a nice blockdiagram of it.

3.6 Conclusion

I'm happy whith what I'v learned so far. It could have been better, but everybody says that when looking back to a project.

4 — References

https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/
was my best friend and guide.
And thanks to the people at netbeheernederland and Fluvius who made the specification available:
https://www.netbeheernederland.nl/_upload/Files/Slimme_meter_15_a727fce1f1.pdf
and
https://www.fluvius.be/sites/fluvius/files/2019-12/e-mucs_h_ed_1_3.pdf