THOMAS MORE

Smart Energy Meter (SEM)

Practice Enterprise

2021 - 2022

Sander Speetjens

1 Embedded Software

Version 1.0

Sunday 29th May, 2022 - 21:57

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. I would also like to explicitly thank my father, Tim Speetjens, for helping me develop a MySQL interface written in PHP.

Contents

1 Theory **7**
- 1.1 The Smart Meter . . . 7
  - 1.1.1 Introduction . . . 7
  - 1.1.2 User-Ports . . . 7
  - 1.1.3 Physical connection . . . 7
  - 1.1.4 Internals of the meter . . . 8
- 1.2 TCP/IP and HTTP . . . 10
  - 1.2.1 TCP/IP . . . 10
  - 1.2.2 HTTP . . . 11

2 Project specifications **12**
- 2.1 Components used . . . 12
- 2.2 Proposed Project Specifications . . . 12
- 2.3 Hardware Diagram . . . 13
- 2.4 Block diagram . . . 14
- 2.5 Block diagram . . . 15
- 2.6 Gantt Chart . . . 16

3 The actual build **17**
- 3.1 My Goals . . . 17
- 3.2 Details . . . 18
- 3.3 Faults . . . 19
- 3.4 Learning Curve . . . 20
- 3.5 Code . . . 20
- 3.6 Conclusion . . . 20

4 References **21**

Figure 1: The new ”Smart Meter” from Sagemcom

1 — Theory

1.1 The Smart Meter

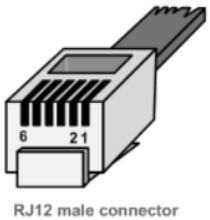
1.1.1 Introduction

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 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.

1.1.3 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

1.1.4 Internals of the meter

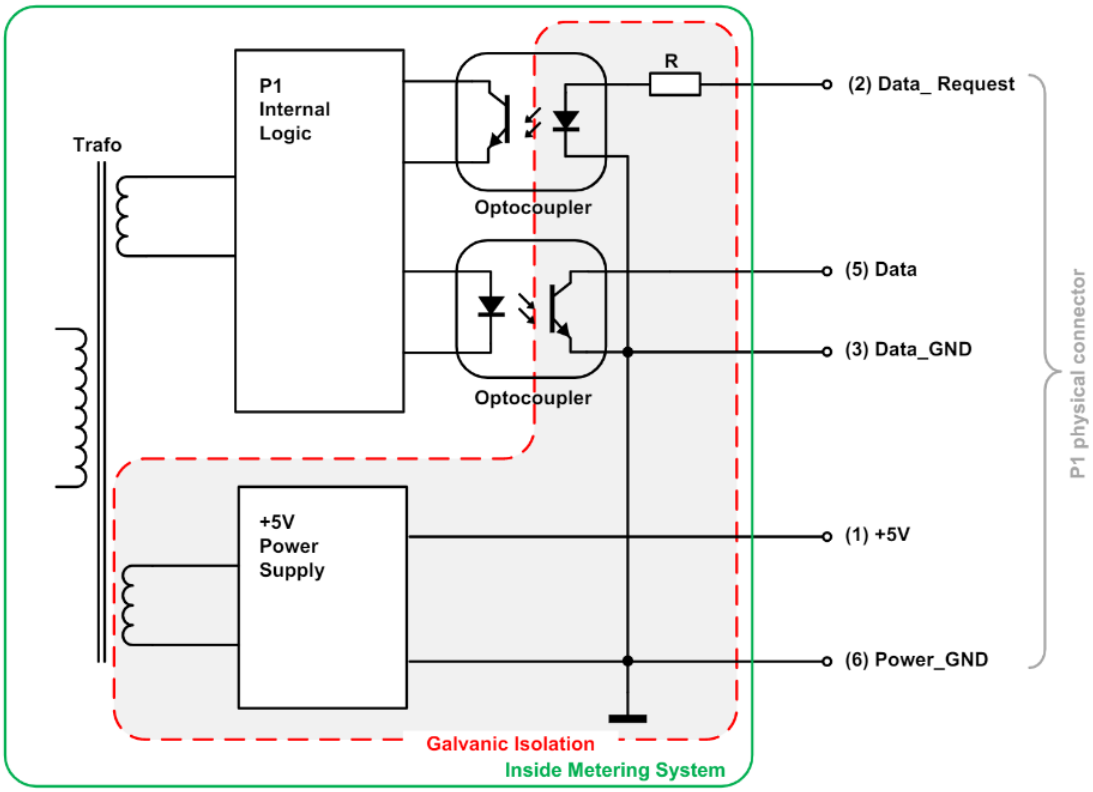


Figure 1.2: The internal workings of the Fluvius Meter

+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

Transfert speed and character formatting 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. `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 TCP/IP and HTTP

1.2.1 TCP/IP

The OSI Model we just looked at is just a reference/logical model. It was designed to describe the functions of the communication system by dividing the communication procedure into smaller and simpler components. But when we talk about the TCP/IP model, it was designed and developed by 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/`

1.2.2 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`

2 — Project specifications

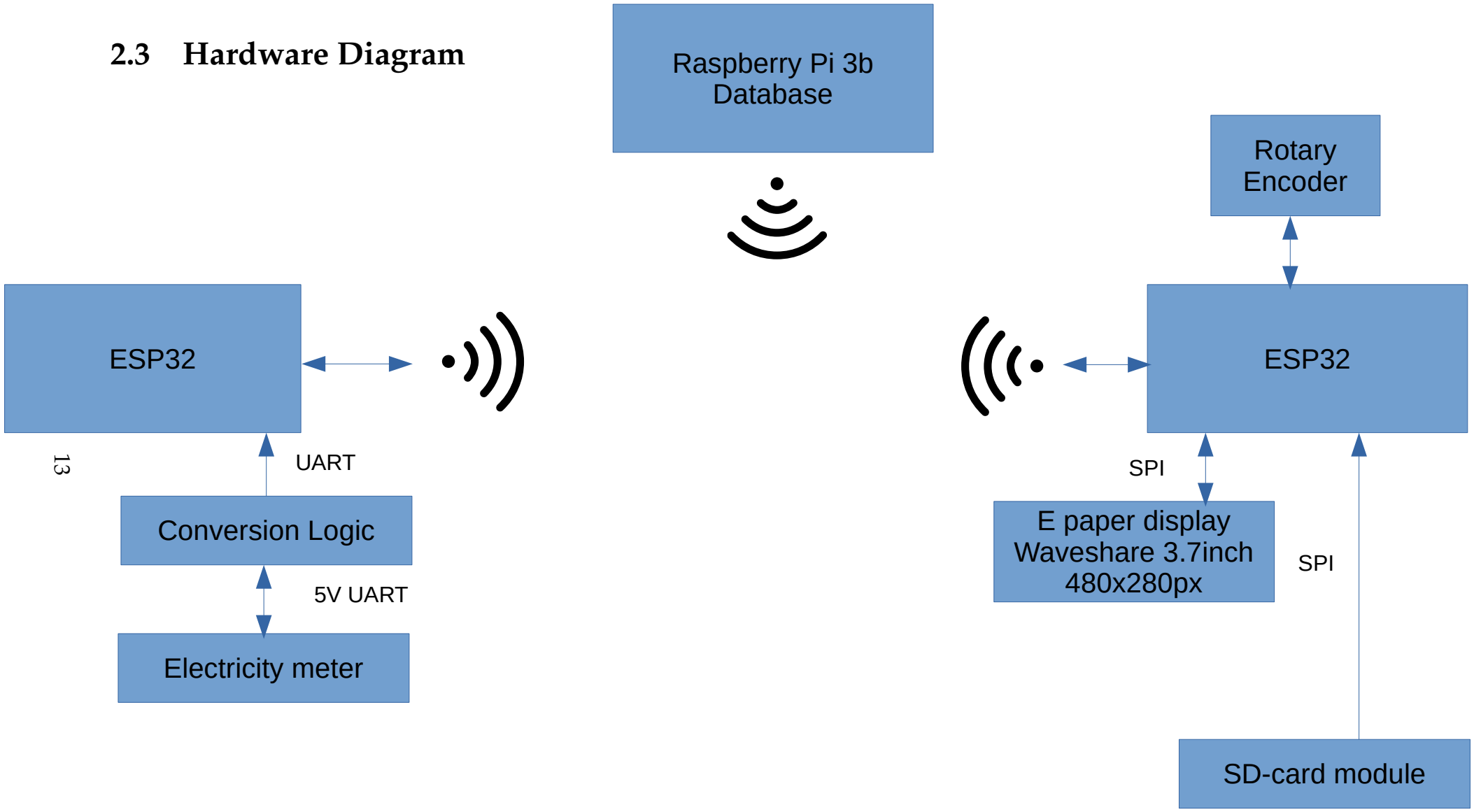
2.1 Components used

- ESP32-WROOM (€5)
- Raspberry Pi/ PC (Free)
- Waveshare e-paper display 3.7 inch 480x280px (€42,13)
- extra components (€20)

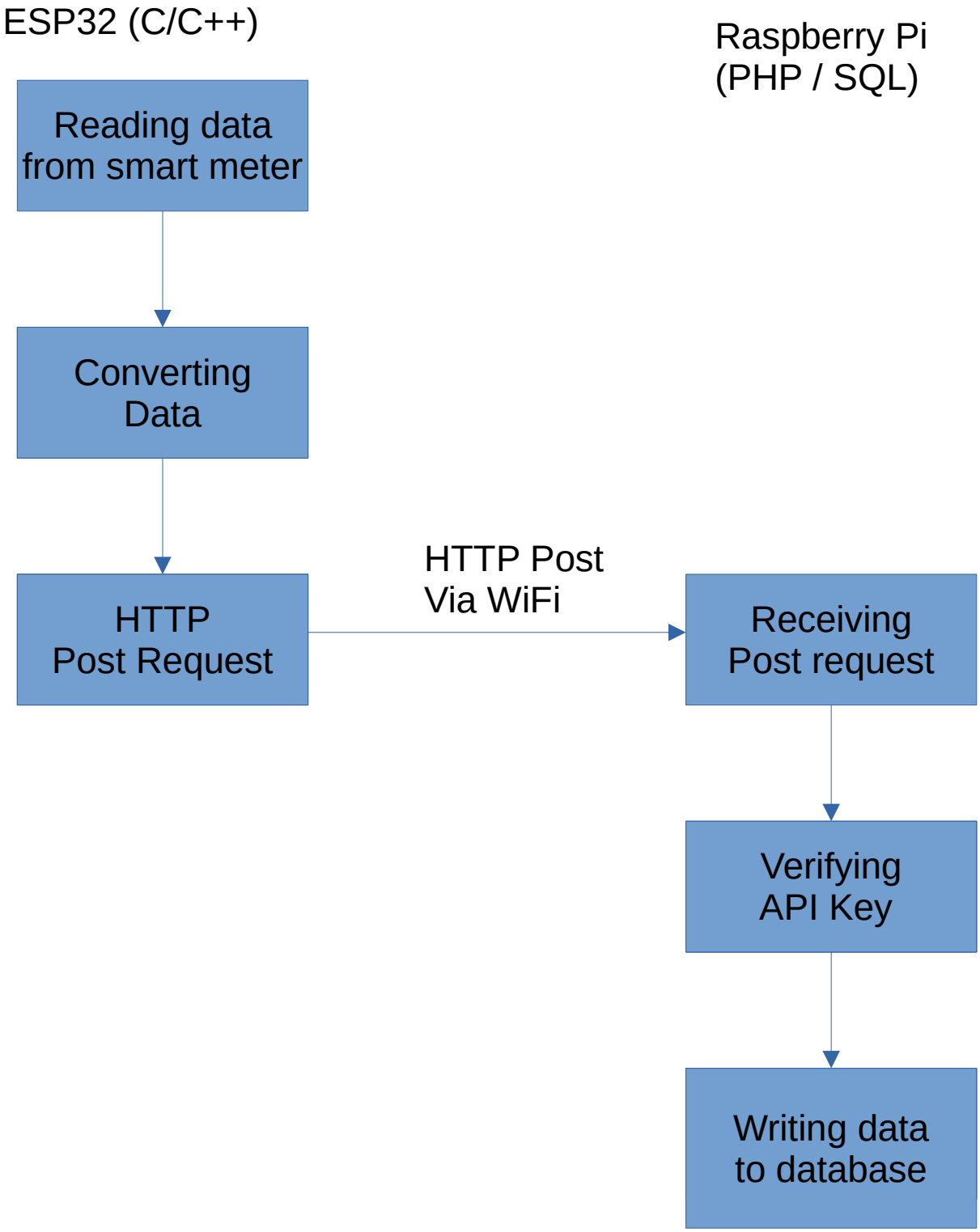
2.2 Proposed Project Specifications

The project is composed out of 3 parts: The sensor, the database and an observation unit.
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 display is the most important part of this project because it is the only visible part and has to be eye catching. It contains a graphical e-paper display to display the graph, a rotary encoder to select the data format (day, week, month or year) and you are able to ask for a specific format as example date format ”week, -2” that will be two weeks prior to now.
All of the parts are powered by a 5V wall adapter.

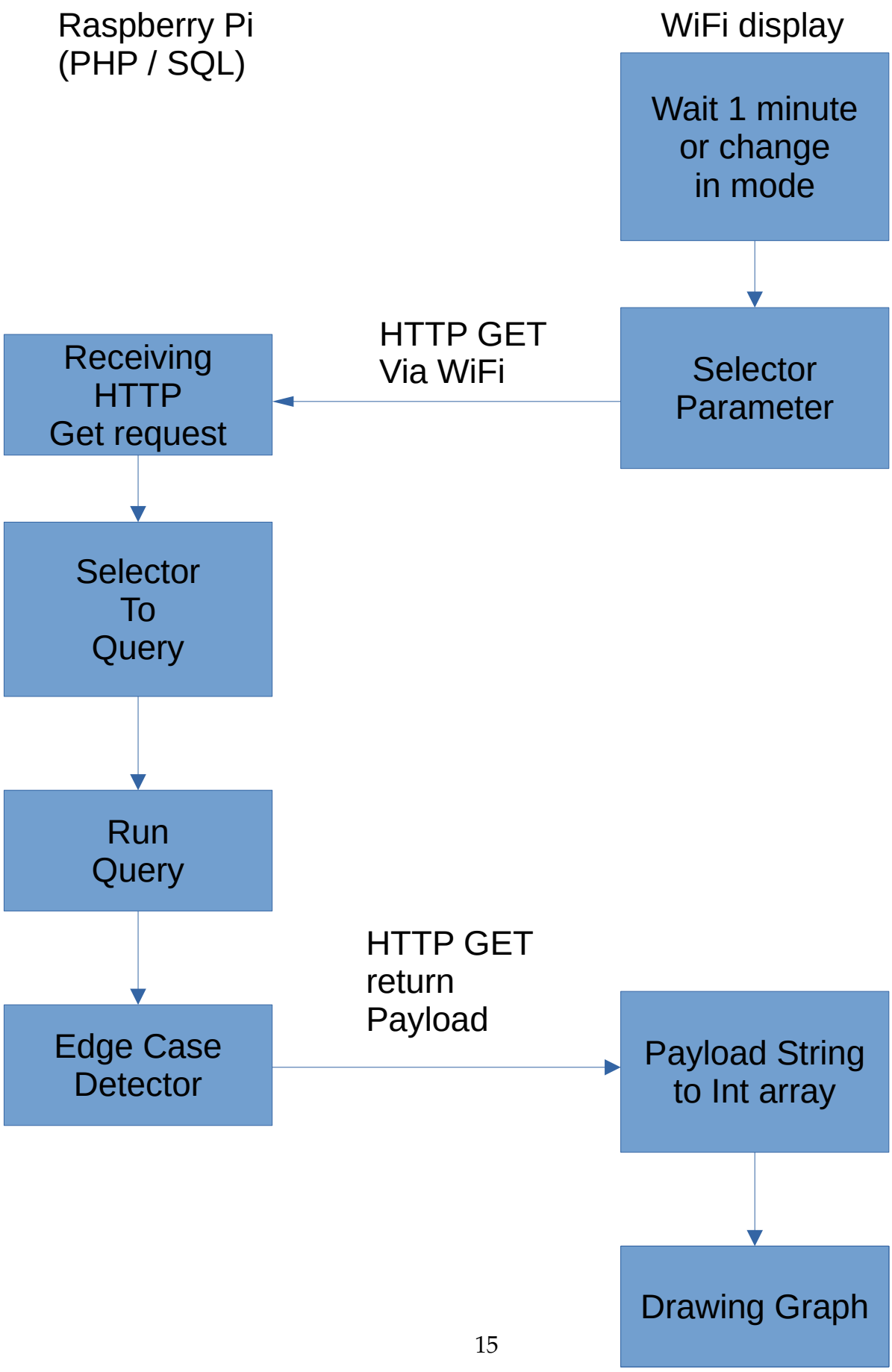
2.3 Hardware Diagram



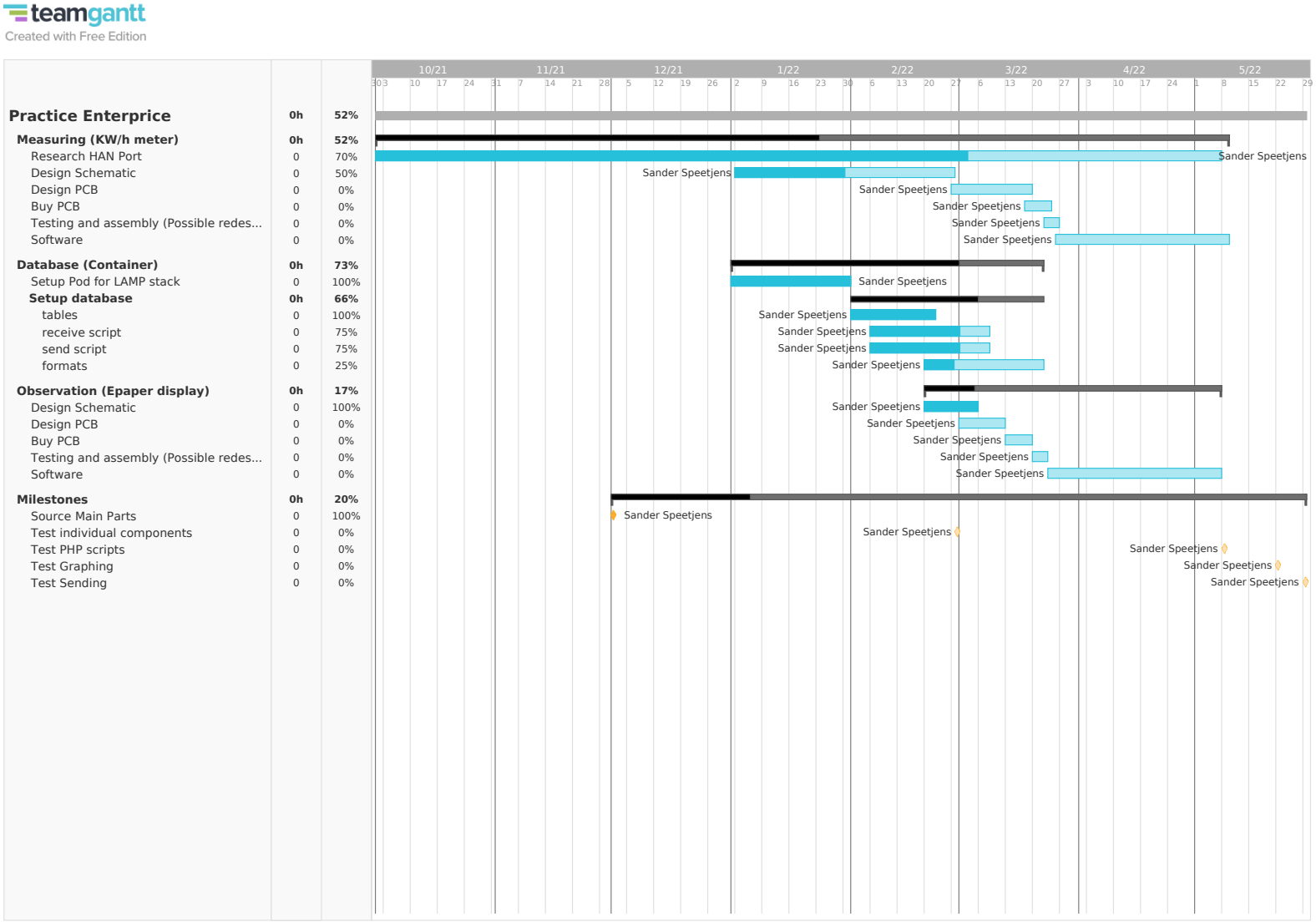
2.4 Block diagram



2.5 Block diagram



2.6 Gantt Chart



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 — Partly Reached
  (only receiving and processing data)
  Due to a lack of time, I only implemented the HTTP Get, data processing steps and high-level graphing functions.
- 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

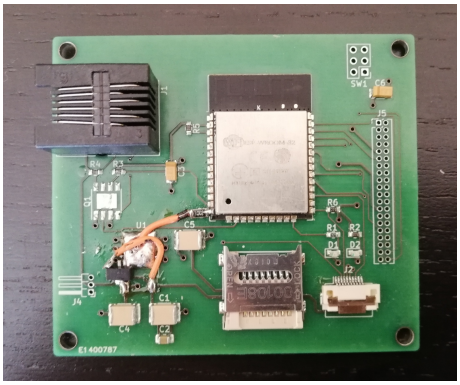


Figure 3.1: What my PCB looks like

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. That's why I only finished the Sensor and Database side of the project. I ran out of time to write code and create documentation for my code.

- 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, so we can just pull it up to 3V3 and we have an inverted UART signal that can be inverted via software. (uart_set_line_inverse) For the enable line we need a tri-state buffer and a level shifter. (ESP (3V3) →Logic Shift (5V) →tri-stat buff (5V tri) →enable pin)

- 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

- Display
  I wanted to use a WaveShare Epaper dislay (3.7 inch, 4 colours), but I hadn't enough time to do anything with it. It is accesible via SPI with 2 extra control signals (command input or data and busy output)

- Rotary Encoder
  I wanted to use a rotary encoder with a switch to control select what data to show. I wanted to create a seporate task to check te rotary switch and report back to the main display task.

3.3 Faults

1. PCB
   I messed a lot up on my PCB, from ordering the right components to using the wrong footprint or even not buying the necessary components.

   - I installed the wrong voltage regulators 5V and 1.2V which destroyed one of my ESP32's and my UART programmer.
   - I bought the wrong type of Mosfets P-type instead of N-type
   - I forgot that the Meter request pin is Tri-state and didn't add the buffer for it
   - I switched around the RX and TX of the programming connector
   - The footprint of the voltage regulator and the dislay connector where the wrong size
   - I forgot to connect the Chip Select line of the Dislay

2. Software

   - My stacksize in my readMeter task was too small and caused Exceptions
   - I used a pin that was internally used for the SPI flash memory
   - I put my uart_set_line_inverse before I attached the driver ...
   - I declared some variables as static in my "uart.c", but that caused my program to only work once
   - I thought that I could give the uart_read_bytes function a pointer to one byte, but it expected an array because it uses the strings library (strncpy)
   - I used some variables inside of statements and they weren't accesible anymore afther it moved out of the clause
   - I forgot to include TLS libs, they where necessary for the http functions to work proparly, even if I didn't use HTTPS

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 on the 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. Although I was only able to finish 2 of the 3 parts, I still think I did a good job in the time I had.

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