



RELIP SMARTY MODBUS

Feil Charel



MARCH 31, 2023

BTS-IOT

Lycée des Arts et Métiers
Real Life Project 2022/2023

1, rue de l'école,
Bissen L-7768

RELIP

Supervisors:

THIERRY BERNARD

JEAN DAUBENFELD

ROBERT FISCH

GILLES GARDULA

GAETAN HOLDERBEKE

GUY WEILER

Table of contents

1	Description of the task	2
2	Introduction.....	3
2.1	Step by step description.....	3
3	Main Part.....	4
3.1	List of material/ electronics	8
3.2	Logical schema of my infrastructure	10
3.3	Electronic circuit schema	12
3.4	Pictures of the final product	15
3.5	Testing table	4
3.6	Description of firmware	17
3.7	Retrospective analysis.....	21
3.8	Market analysis	21
4	Executive summary	22
5	Bibliography and list of sources	23
6	Executive summary	7
7	Acknowledgements.....	24
8	Declaration of autonomy	25

1 Description of the task

The project given by our teachers consisted of 2 main parts.

Firstly, we needed to build a smarty simulator. We needed to do this as a convenience and safety measure for the class. This would enable us to work on our main projects the whole time without needing to wait for another person to be finished using the only smarty meter that we have. To build this we had a limited timeframe in order to being able to finish the main project in time. By the end of the given deadline, the smarty simulator needed to be working 100% and we needed to make a report on how and what we did.

Secondly, the main project was to build a smarty reader, which can decrypt the data received from the smarty simulator/meter and send it out via Modbus TCP and Modbus RTU. Furthermore, our product needed to be easily configurable by the end user.

2 Introduction

2.1 Step by step description

2.1.1 *Smarty Simulator*

My simulator simply sends out data when it's connected to a power source of 5 Volts. However, for it to be able to send the data, you need to remove the only jumper on the board while bootup, because the connected pin is needed for the boot sequence. Afterwards, you put the jumper back on and the simulator will send data when the required triggers arrive. These are that a request is being sent to the simulator and that a non-blocking delay of 10 seconds has been passed. Then it will send data to the receiver on the other end. The simulator only sends pre configured data, which is hardcoded into the microcontroller. This is good to check reliability and be sure that all the data received by the reader is the data sent.

2.1.2 *Smarty Modbus*

My smarty Modbus device consists of three parts.

The configuration, which at first startup or after a reset is being shown on the access point at the IP address <http://192.168.168.168> , which was given by our teachers. In this configuration, the end user can firstly decide whether they want to use RTU (RS485), TCP/IP (over ethernet) or WIFI (also TCP/IP, but named differently to reduce confusion). These are three different pages where the end user can enter all the necessary details for a successful configuration and operation.

The encryption key needs to be entered on all of the configuration pages.

In the RTU configuration page, the end user needs to enter the bitrate for the Modbus line, the slave address and the configuration for the transmission, so the Stop bit and parity.

In the TCP/IP configuration page, the end user needs to enter the IP addresses for the device itself, the gateway, subnet and DNS.

The WIFI configuration page is nearly the same as the TCP/IP one, only that the end user needs to enter the WIFI SSID and password additionally to the IP addresses asked.

After having entered all the data for the selected transmission mode, the end user needs to press the "Submit" button and the data is being saved on the device and it restarts.

The Access point reopens after bootup, so that the end user has the ability to reset the device if needed. If it isn't necessary, the end user needs to wait 2 minutes and then the main program for the transmission via the selected Modbus is being started.

3 Main Part

3.1 Project progress

3.1.1 Smarty Simulator

3.1.1.1 Initial thoughts

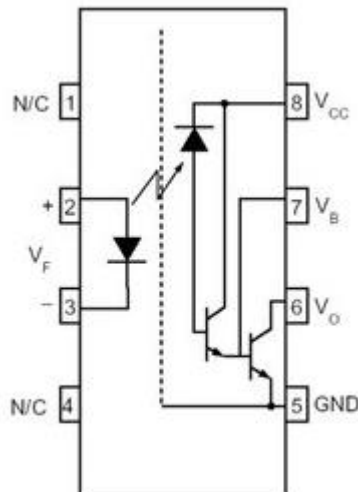
One of the first things I did when we got the assignment and deadline for the simulator was to research how the optocouplers we needed to use work. Then I needed to firstly draw a functioning circuit and solder it in place on a circuit board given by our teachers. (Note to avoid confusion: the circuit wasn't etched on the board)

At the beginning of this task, I thought that this could go rather fast if all things go well and I could focus earlier on the main part of the project.

3.1.1.2 Research

I mainly researched how the optocouplers worked, because I've never worked with these chips and I wanted and needed to understand them before using them.

During my research, I found this fitting picture:



As shown in the picture, the input and output side are isolated (indicated by the dotted line). The input signal is transmitted via an LED on the input side to a light sensitive diode, which then activates the transistors to put out the signal on the output pin.

Then I needed to research the datasheet for the optocouplers and to calculate the needed resistors accordingly.

MAXIMUM RATINGS

	CHARACTERISTIC	SYMBOL	RATING	UNIT
LED	Forward Current	I_F	20	mA
	Pulse Forward Current (Note 1)	I_{FP}	40	mA
	Reverse Voltage	V_R	5	V
DETECTOR	Output Current	I_O	50	mA
	Output Voltage	V_O	7	V
	Supply Voltage (1 minute Maximum)	V_{CC}	7	V
	Enable Input Voltage (Not to exceed V_{CC} by More than 500mV)	V_{EH}	5.5	V
	Output Collector Power Dissipation	P_O	85	mW
	Operating Temperature Range	T_{opr}	0~70	°C
	Storage Temperature Range	T_{stg}	-55~125	°C
	Lead Solder Temperature (10s) (Note 2)	T_{sol}	260	°C

3.1.1.3 Calculations

The operating voltage for the internal LEDs is written as 1,7V and the maximum current is 20mA. I took 10mA as the forward current to not overstress the LED. The calculations needed to be done twice, once for the 5V side, from the RJ12 port and once for 3,3V for the ESP8266 side.

The calculations: $\frac{5V-1.7V}{10mA} = 330\Omega$ | $\frac{3.3V-1.7V}{10mA} = 160\Omega$

3.1.1.4 Drawing the schematic

Then I researched examples on how to connect an optocoupler and found out that the output pin needs to be pulled up to 5V, so VCC (in most cases) or to the wanted voltage. In our case, the side going to the RJ12 port needed to be pulled up to 5V and the side going to the ESP8266 needed to be pulled up to 3,3V. In the beginning, I had both sides pulled up to 5V, which I thankfully realized in time with some help. Before putting all the components together, I drew a schematic in Eagle and asked to verify the circuit before I built it.

3.1.2 Smarty Modbus

3.1.2.1 Initial thoughts

One of the first things I thought when asked to make this product was that I had no idea how Modbus worked and how to decrypt the received data from the smarty. We needed to do this already in an exercise during our course, but I didn't finish it because I didn't understand it that well that time. The decryption would become my main focus at the beginning of the project, because this seemed to be the biggest hurdle for myself at that time.

For the RS485, I knew that we needed to use a similar chip than the one we used in an exercise during our course, so I didn't worry about that too much.

3.1.2.2 First tests

The first thing I wanted to get working was the receiving of the data and the decryption, because as I stated before, these were my main concerns at the beginning. To save some time later, I already used the RS485 chip as my serial port for debugging, which was no issue at all.

It took me some time to receive all the data correctly, but eventually I got it. Then I concentrated on the decryption, which previously gave me some headaches, but in this project it all came together, and I finally understood how to use the given libraries and could interpret the data correctly.

The next thing I tested was the Ethernet module, for that I needed the Ethernet library and simply connect the module to the ESP8266. To make sure that the module would connect to the network, I took an example code from the library and tested the connection and if it would automatically reconnect if the cable was pulled out and plugged back in.

After that, I tested the Modbus RTU with the RS485 chip by simply adding some registers and adding random values into these registers. I had some issues with this, because the software I used for reading out these registers often gave me an error that the request timed out or an invalid data request.

3.1.2.3 Managing the project

My main focus of the project was the code, because we needed to deliver something that works, and the code and the circuit are the most important pieces to this.

This means that I dedicated most of my time to the code and making sure everything would work as I intended. This was more work than I initially anticipated, but I managed to finish in time. This resulted in the design lacking a bit of finishing, but overall, I'm happy with my time management of this project.

3.1.2.4 Code

My code is far bigger than I anticipated and I'm certain that I could save a lot of space if I were to optimize some functions or write it differently, but it works the way I intended it. It was my main focus, so I dedicated all my time to this, that it works and not so much on the case. I had a lot of problems at the beginning to receive the data correctly and to decrypt the message, but I managed to do it successfully in a few days. Then I needed to concentrate on where and how the values from the smarty meter would be saved.

This took a lot more time than I anticipated, I made a lookup table consisting of the OBIS codes received from the smarty meter, their description, the register where it begins to save into, how many registers are being used for that value and a multiplication value to make the values the same unit as the ones from the SOCOMEC, which all of us based ourselves on. This took so long because many values from the smarty weren't read by the SOCOMEC and therefore not saved in their registers. This led to a lot of confusion whether we needed to save these in random registers or not at all and some separated values from the smarty were only one register in the SOCOMEC, which meant that we needed to make a conversion. There was a lot of confusion around where to save everything, but in the end, I think everyone managed to use the same registers for everything and has their data in the same units.

3.1.2.5 Making the PCB

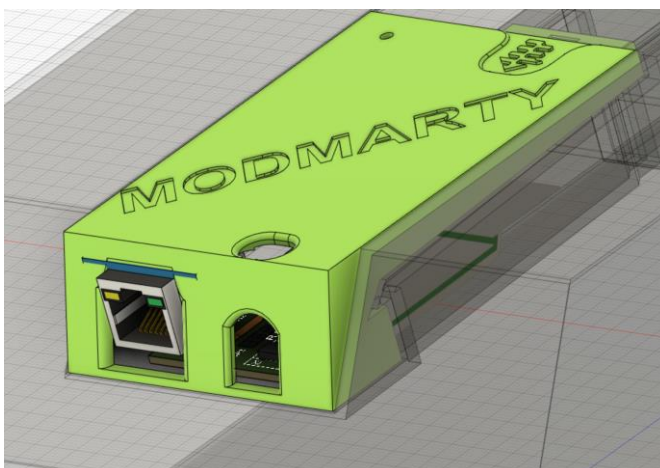
The PCB design was done only in a few days, because of the time constraints to get the code working properly and the availability of the room to make the PCB itself. The routing was a bit difficult, but in the end I managed to get everything routed correctly, except the one error I already mentioned in the [circuits](#) which still persists in my [PCB](#). It still works, but this could explain some errors while reading from the RS485.

3.1.2.6 Designing the case

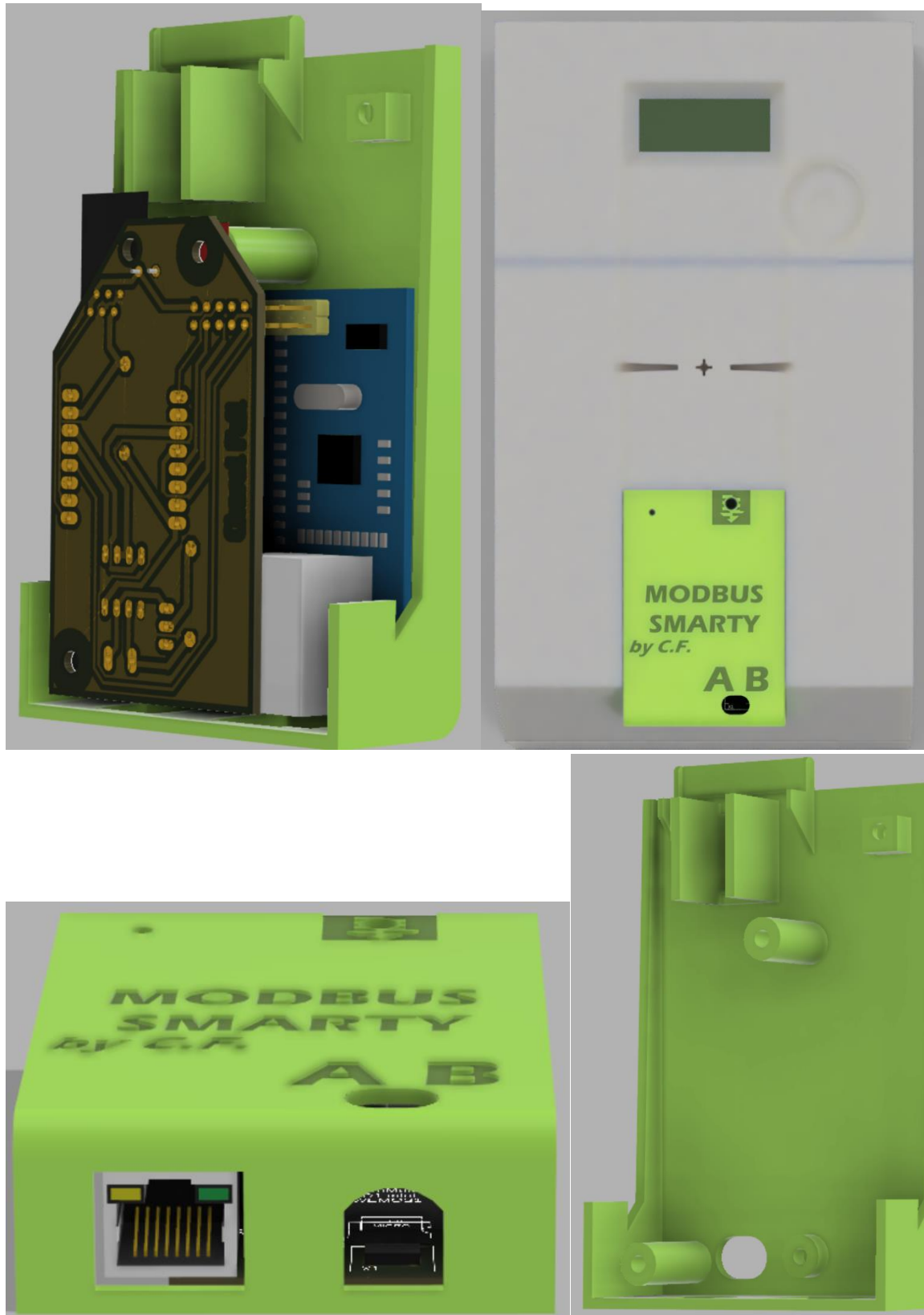
The case design was one of the last steps I made for this project, because it was obvious from the task given that the product needs to be working at the end and not necessarily need to look the best.

I based it all off of my PCB and designed the case around it so that it would hold the PCB in place, fit into the smarty and that the ports are accessible from the front.

First idea:



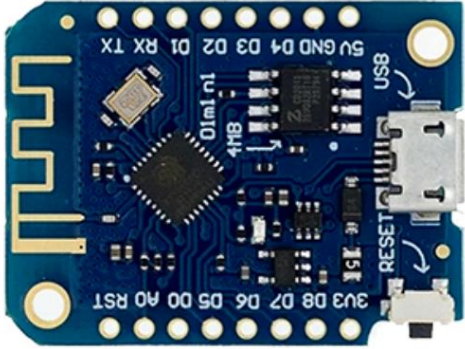
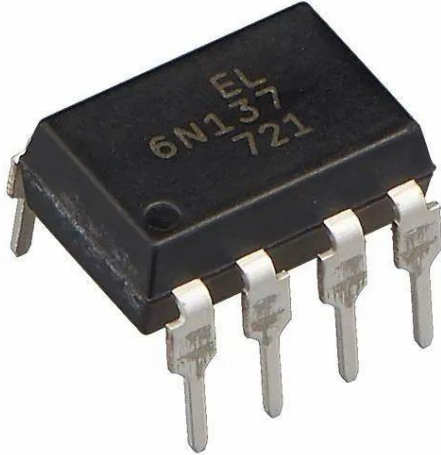


Final Design:



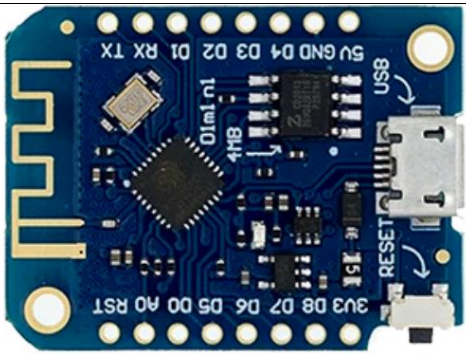

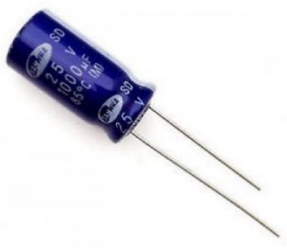
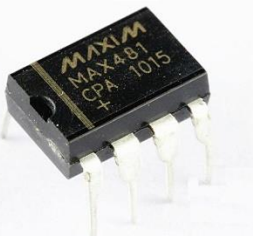

In my final design, I also implemented a screw hole on the upper left corner to screw the device in place. I also made some flanges to keep the cable in place on the bottom.

3.2 List of material/ electronics

3.2.1 Smarty Simulator

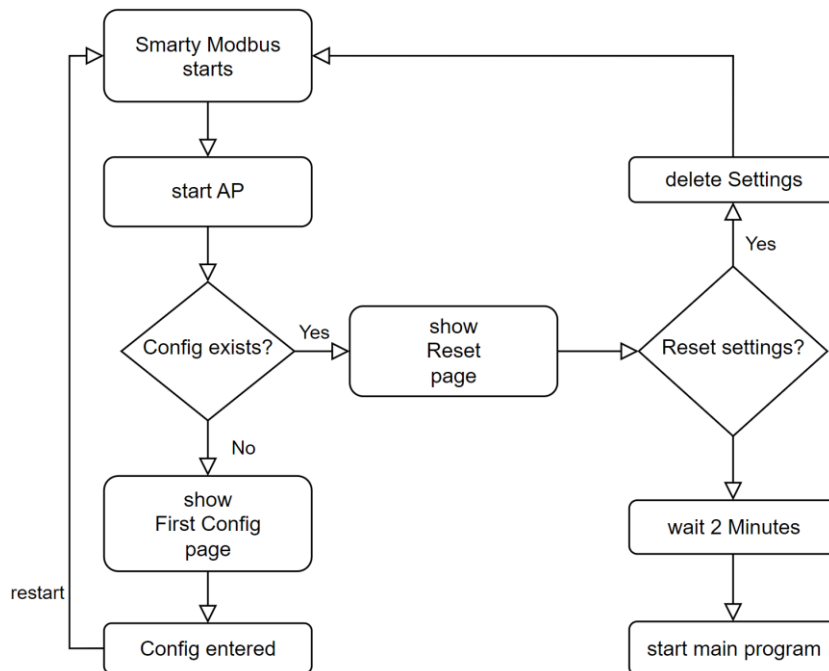
	ESP8266
	Optocoupler 2x
	Resistor 3x
	RJ12 Connector

3.2.2 Smarty Modbus

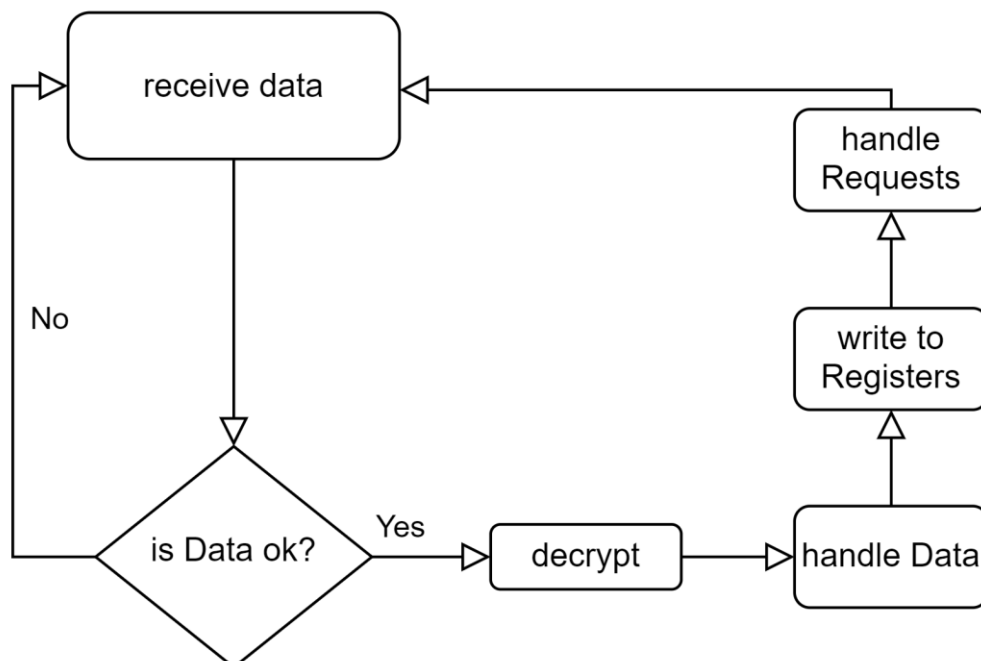
	ESP8266
	Resistor 2x
	1000uF Capacitor
	MAX485 CPA
	Ethernet Module W5500

3.3 Logical schema of my infrastructure

3.3.1 Access Point loop



3.3.2 Main program (simplified)

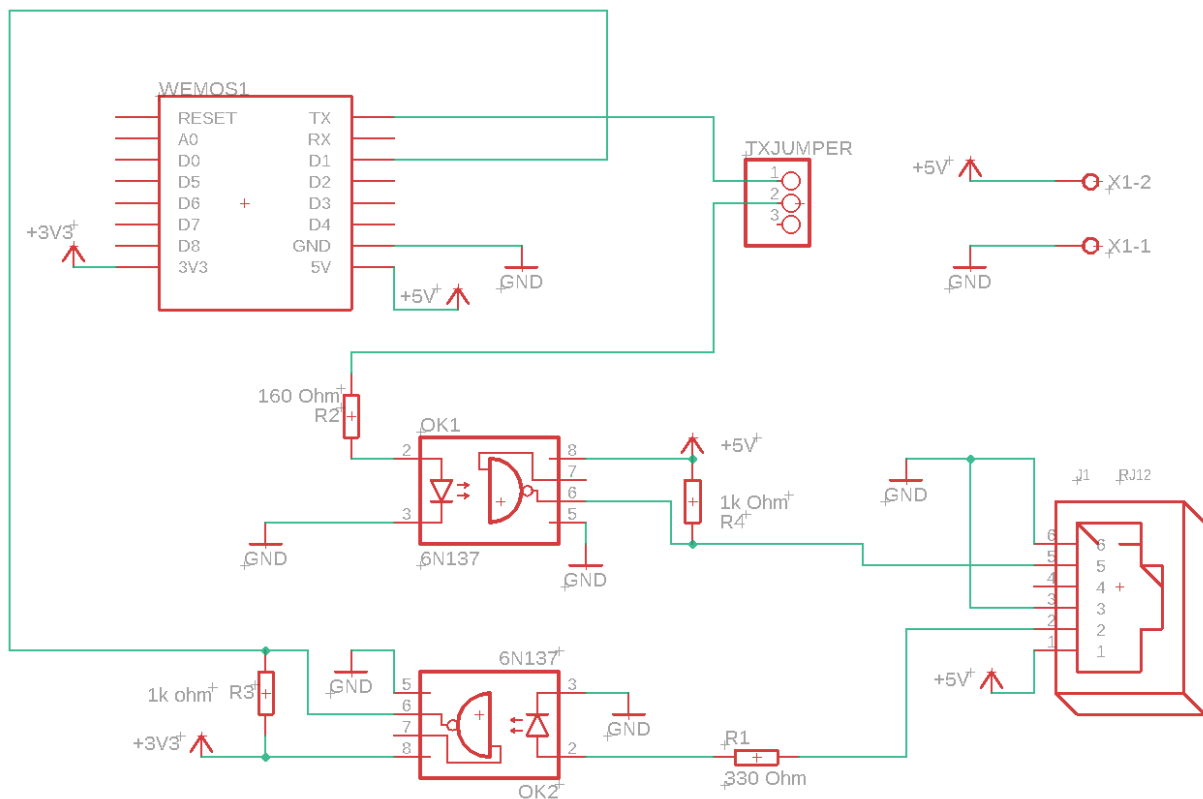


3.3.3 Lookup Table

	Register	numreg	multiplier	*/
1	/*OBIS			
2	{*1-3:0.2.8"	50007, 1, 1}	//software version of smarty needs to be read X/10	
3	{*0-0:42.0.0"	50042, 16, 1}	//characters in hex	
4	{*1-0:1.8.0"	50770, 2, 1}	//kWh	
5	{*1-0:2.8.0"	50776, 2, 1}	//kWh	
6	{*1-0:3.8.0"	50772, 2, 1}	//kvarh	
7	{*1-0:4.8.0"	50778, 2, 1}	//kvarh	
8	{*1-0:1.7.0"	50536, 2, 100}	//kW	
9	{*1-0:2.7.0"	50538, 2, 100}	//kW	
10	{*1-0:3.7.0"	50538, 2, 100}	//kvarh	
11	{*1-0:4.7.0"	50538, 2, 100}	//kvarh	
12	{*1-0:9.7.0"	50017, 1, 100}	//kVa placeholder register maybe both added in 50542	
13	{*1-0:10.7.0"	50018, 1, 100}	//kVa placeholder register	
14	{*1-0:32.7.0"	50520, 2, 100}	//V	
15	{*1-0:52.7.0"	50522, 2, 100}	//V	
16	{*1-0:72.7.0"	50524, 2, 100}	//V	
17	{*1-0:31.7.0"	50528, 2, 1000}	//A	
18	{*1-0:51.7.0"	50530, 2, 1000}	//A	
19	{*1-0:71.7.0"	50532, 2, 1000}	//A	
20	{*1-0:21.7.0"	50544, 2, 100}	//kW	
21	{*1-0:22.7.0"	50544, 2, 100}	//kW	
22	{*1-0:41.7.0"	50546, 2, 100}	//kW	
23	{*1-0:42.7.0"	50546, 2, 100}	//kW	
24	{*1-0:61.7.0"	50548, 2, 100}	//kW	
25	{*1-0:62.7.0"	50548, 2, 100}	//kW	
26	{*1-0:23.7.0"	50550, 2, 100}	//kvar	
27	{*1-0:24.7.0"	50550, 2, 100}	//kvar	
28	{*1-0:43.7.0"	50552, 2, 100}	//kvar	
29	{*1-0:44.7.0"	50552, 2, 100}	//kvar	
30	{*1-0:63.7.0"	50554, 2, 100}	//kvar	
31	{*1-0:64.7.0"	50554, 2, 100}	//kvar	

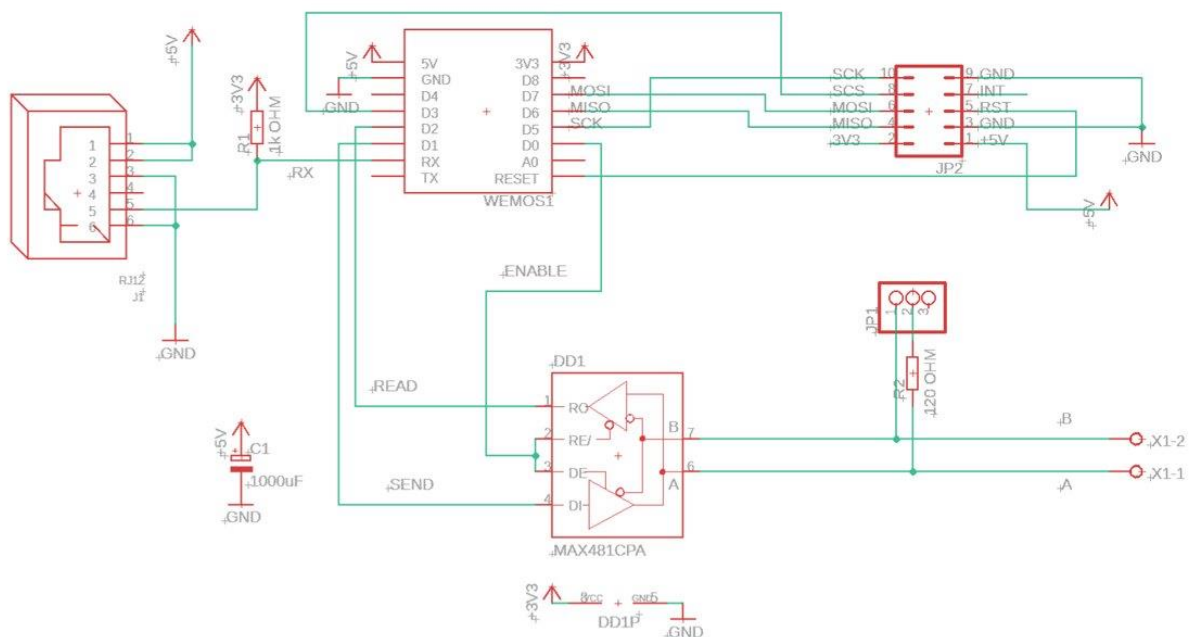
3.4 Electronic circuit schema

3.4.1 Smarty Simulator

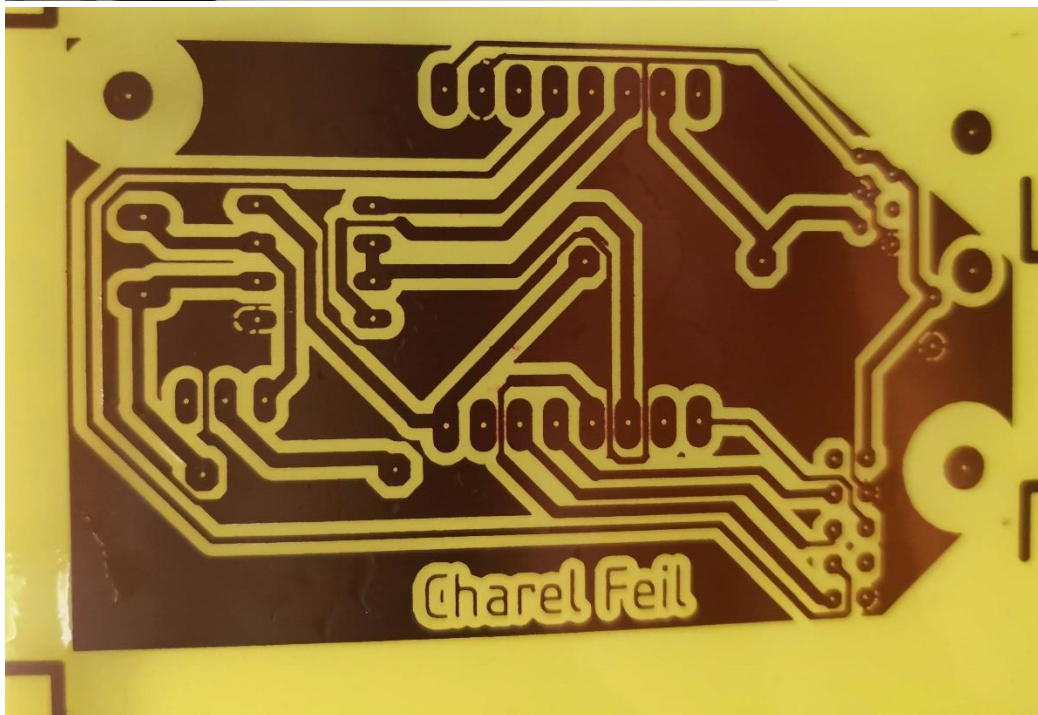
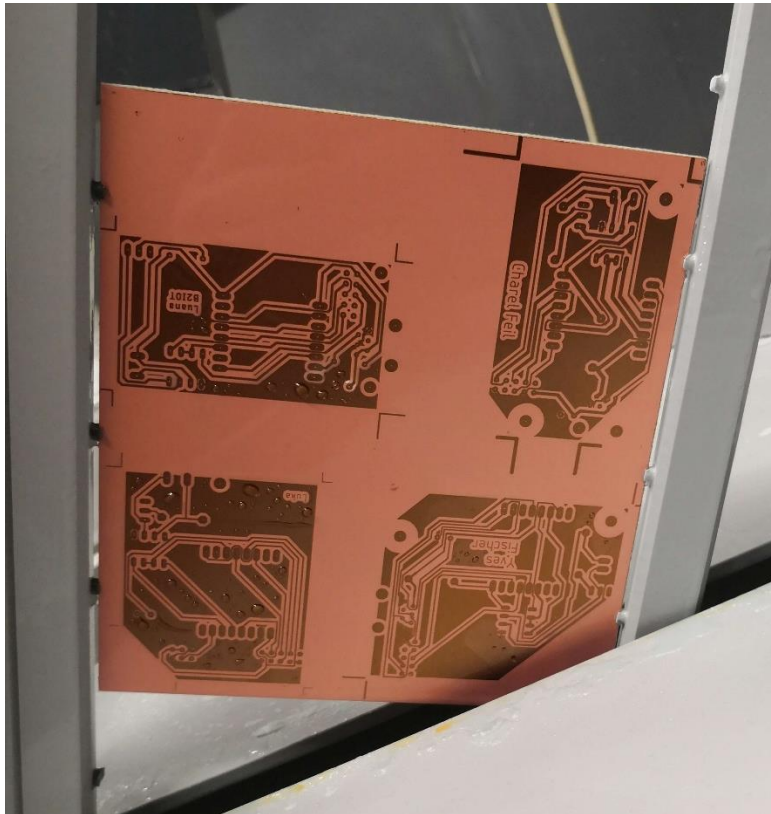


3.4.2 *Smarty Modbus*

3.4.2.1 Circuit

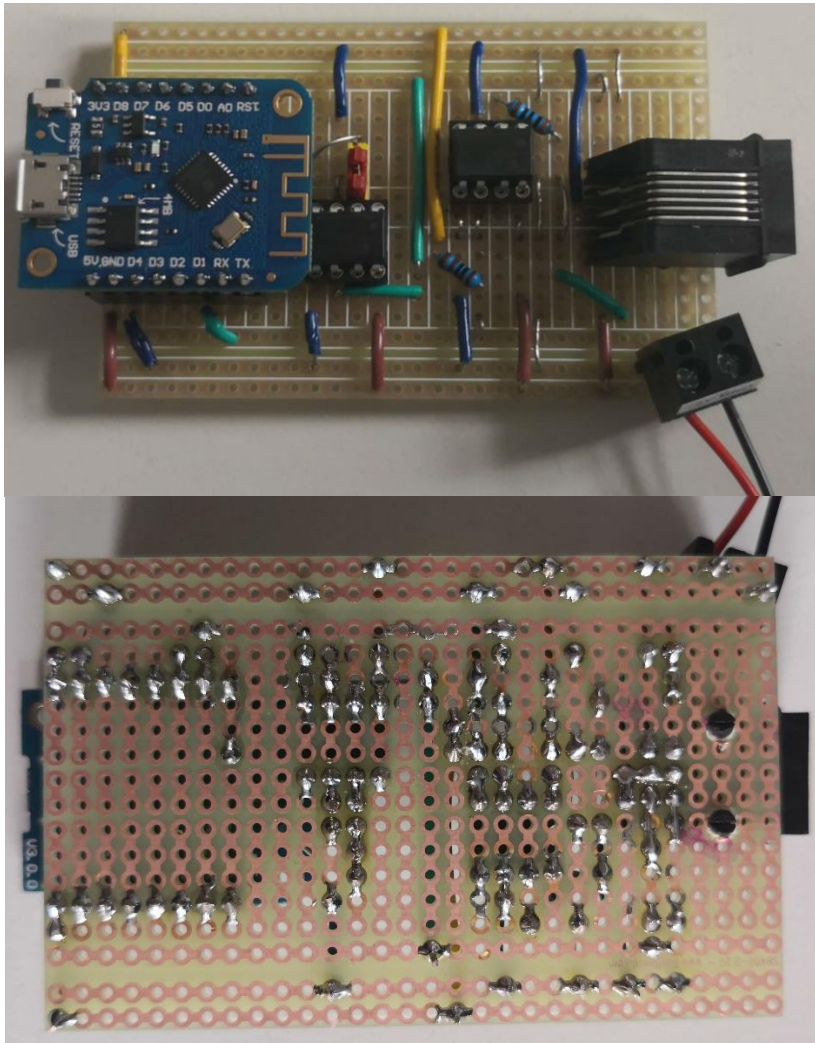


In this schematic you can see that the RS485 chip is being connected to 3,3V instead of the 5V written in the datasheet. This is an error, I thought the chip worked the same

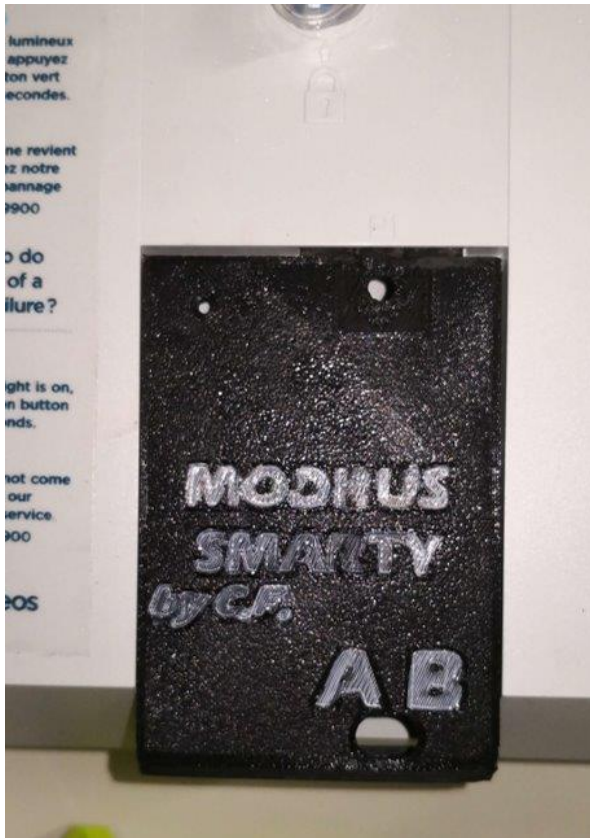


3.5 Pictures of the final product

3.5.1 *Smarty Simulator*



3.5.2 Smarty Modbus



3.6 Description of firmware

You can find the flowcharts explaining the code roughly [here](#).

When my device first boots up, it starts an Access Point (from hereon AP) under the name "Charel AP" (to avoid confusion in class), if the product were to be released to the public, I would change the name to a more appropriate one. After the AP is started, the device looks if a configuration has already been saved on it. If it hasn't found one, then it continues to show the configuration pages to the user under the IP Address <http://192.168.168.168>. I made 3 different pages for RTU, TCP/IP and TCP/IP over WIFI, the starting page is RTU, so the user already sees some input and not only 3 closed tabs are being shown. The encryption key always needs to be entered, but The first configuration page of RTU looks like this:

First configuration

The screenshot shows a web interface for the RTU configuration page. At the top, there are three tabs: RTU (selected), TCP/IP, and WIFI. Below the tabs, the form contains the following fields:

- Encryption Key: A text input field.
- Modbus Bitrate: A dropdown menu with the value 115200 selected.
- Modbus Slave Address: A text input field.
- Modbus Stop Bit: A dropdown menu with the value 1 selected.
- Modbus Parity: A dropdown menu with the value NONE selected.
- Submit: A blue button at the bottom.

Here you can see that the user needs to enter all the necessary data for the Modbus that a successful transmission of data can be made. For RTU, the important fields for the data transmission would be the Bitrate, stop bit and parity, the slave address is for the Modbus to be able to identify which device it is.

The first configuration page of TCP/IP looks like this:

First configuration

RTU	TCP/IP	WIFI
-----	--------	------

Encryption Key

Modbus IP Address

Modbus Gateway IP

Modbus Subnet

Modbus DNS

For TCP/IP the end user needs to enter the IP address of the device, the gateway and DNS IP addresses and the subnet mask of the network. These are necessary to ensure a good connection to the network with a static IP.

The first configuration page of the WIFI page looks like this:

First configuration

RTU TCP/IP **WIFI**

Encryption Key

WiFi SSID

WiFi Password

Modbus WiFi IP Address

xxx.xxx.xxx.xxx

Modbus WiFi Gateway IP

xxx.xxx.xxx.xxx

Modbus WiFi Subnet

xxx.xxx.xxx.xxx

Submit

Here the end user needs to enter the same things as in the TCP/IP tab, but additionally the Wifi SSID and password.

On these pages, the encryption key only allows for 32 characters and is required to be filled out. All of the necessary input fields have a required tag in the code, this means that the user gets an error when something isn't filled out or filled out in the wrong format.

Length error for the encryption key:



Verlängere diesen Text auf mindestens 32 Zeichen. Derzeit verwendest du 4 Zeichen.

Necessary field not filled out error:



Fülle dieses Feld aus.

When everything is filled out correctly, the user needs to press the “Submit” button and the device restarts.

After it's booted up again, it starts the AP again and checks if there exists a configuration and now it does. This means that the Reset page is being shown, so that if any values need to be changed or any reconfiguration needs to be done, the user can reset the device and start over.

The reset page looks like this:

Reconfiguration

Do you want to RESET all the settings?

RESET

The user needs to press the reset button to reset and restart the device.

If the user doesn't want to reset it, they need to wait 2 minutes for the AP to close. Then the device will start with Modbus.

Finally, when everything is configured correctly, the device will read the data received from the smarty meter and decrypt it.

Then the device will retrieve the values from the data by searching for the correct OBIS codes, extracting the number or characters, make calculations for the special registers and convert all of them into the right units, converting them into the correct format (16bit unsigned int) and writing them to the registers. This is done with my [lookup table](#).

Some values were too big for 16bit unsigned integers, that's why I needed to make a conversion from a 32bit unsigned integer to 2 16bit numbers by first masking the first 16 bits and then the last 16 bits.

Then the values are available in the selected Modbus and can be read.

3.7 Retrospective analysis

Overall, I am pretty happy how my project turned out. This time, I managed my time a lot better than in the projects before. I struggled a bit with the code, but I managed to make it all work together.

For this project, we had a clear guideline and description of the task, which helped me a lot in completing it. Furthermore, the RELIP has a real function and use afterwards. This makes a project a lot more interesting and more appealing to work on.

3.8 Market analysis

There is no other device like this on the market, except a complete electrical meter which directly sends the data via Modbus. These are a lot more expensive than our solution if you want the electrical data from the smarty via Modbus.

I think this would be a high demanded project for big companies like "Batiment Public".

4 Executive summary

The device is easily configurable by the user and can be reset if needed.

It also receives the data from the smarty reader and decrypts it.

Then it puts it in the Modbus registers, and it can be read by the user via the selected Modbus interface.

5 Bibliography and list of sources

1. <https://github.com/emelianov/modbus-esp8266>
2. <https://www.weigu.lu/microcontroller/modbus/index.html>
3. https://www.weigu.lu/tutorials/sensors2bus/04_encryption/index.html
4. <https://www.weigu.lu/microcontroller/smartyReader/index.html>
5. <https://smarty.creos.net/wp-content/uploads/DutchSmartMeterRequirements.pdf>
6. <https://media.bluestonepim.com/50098aaa-684d-40aa-945d-038656764549/beb48000-69ec-449a-8cf1-30056ac36c6c/k8gc14BCCzRT4S9XPtMeBnpHe/APNGQdFVfFmb3x7po9SC1Mce8.pdf>
7. https://www.luxmetering.lu/pdf/SPEC%20-%20E-Meter_P1_specification_20210308.pdf

6 Acknowledgements

Special thanks to Yves Fischer, Luka Theisen and Luana Do Vale Daniel for helping me all around the project.

7 Declaration of autonomy

I certify hereby that I have written the thesis and report independently without the help of other aids than those explicitly stated.

Name: Charel Feil

Place and Date: Bissen, Luxembourg 31.03.2023

Declaration: I certify hereby that I have written the thesis and report independently without the help of other aids than those explicitly stated.

Signature: 