Academiejaar: 2021 / 2022

IOT Datacom project

Verslaggever: Lukas Verschraegen

Medestudent: Jannes Breusegem

Klas: 2ELO

Lector: Piet Coussens

Inhoudstafel:

1 Inleiding

1.1 Onderwerp

Dit project hebben we gekozen omdat het weer en het klimaat nog steeds actuele onderwerpen zijn. Met dit weerstation zouden we mensen willen aansporen om te besparen; bijvoorbeeld de chauffage wat lager zetten, het huis eens verluchten enz.

Om dit project te realiseren gebruiken we TTGO ESP32 module's en om de module te programmeren gebruiken we de programmeertaal micropython. Bij dit project zullen we de temperatuur en de windsnelheid meten. De opgenomen data zullen via de TTGO ESP32 module's verwerkt worden via draadloze communicatie, dit gebeurt a.d.h.v. LoRa. De andere ESP32 ontvangt de data via het LoRa protocol en stuurt deze door naar een oled scherm dat is ingebouwd in de ESP32 module.
De data van de temperatuur zullen uitgelezen worden aan de hand van een BMP280 module en de anemometer zal uitgelezen worden a.d.h.v. RS485 communicatie.

2 Module's

2.1 TTGO ESP32 module

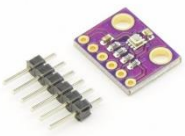
https://www.tinytronics.nl/shop/nl/development-boards/microcontroller-boards/met-lora/lilygo-ttgo-lora32-868mhz-esp32

Info:

- ESP32 chip (240Mhz dual core processor)
- Flashgeheugen: 4MB
- SX1276 LoRa chip
- 0.96 inch OLED
- 1x LoRa antenne 868MHz

2.2 BMP280

https://www.tinytronics.nl/shop/nl/sensoren/lucht/druk/bmp280-digitale-barometer-druk-sensor-module



Info:

- I2C en SPI ondersteuning.
- ±1 hPa en ±1.0°C nauwkeurigheid.
- Werkt zowel op 3.3V als 5V.

2.3 Anemometer

https://nl.aliexpress.com/item/32957564379.html?spm=a2g0o.search0302.0.0.515a54e4oRAwV8&algo_pvid=bc8eee63-2fe3-495b-9a78-9b5117af2291&algo_exp_id=bc8eee63-2fe3-495b-9a78-9b5117af2291-0

Info:

- Signaal output :RS485
- Ingangsspanning: DC12-DC24V
- Responstijd: < 5 S
- Transmissie afstand: > 1km
- Meetbereik: 0 ~ 30 m/s
- ± 1m/s nauwkeurig
- Baud rate: 9600

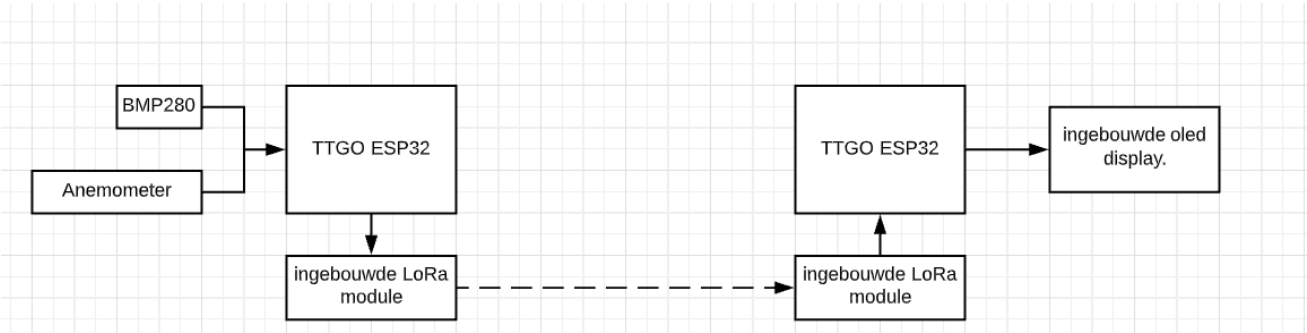
3 Open source

Ons project is te volgen via onderstaande github/gitlab pagina.
In deze files beschrijven we delen van de code. De volledige code vindt u terug in de mappen main en libraries.
https://github.com/LukasVerschraegen/project-datacommunicatie
https://gitlab.com/jannes.breusegem/project-datacommunicatie

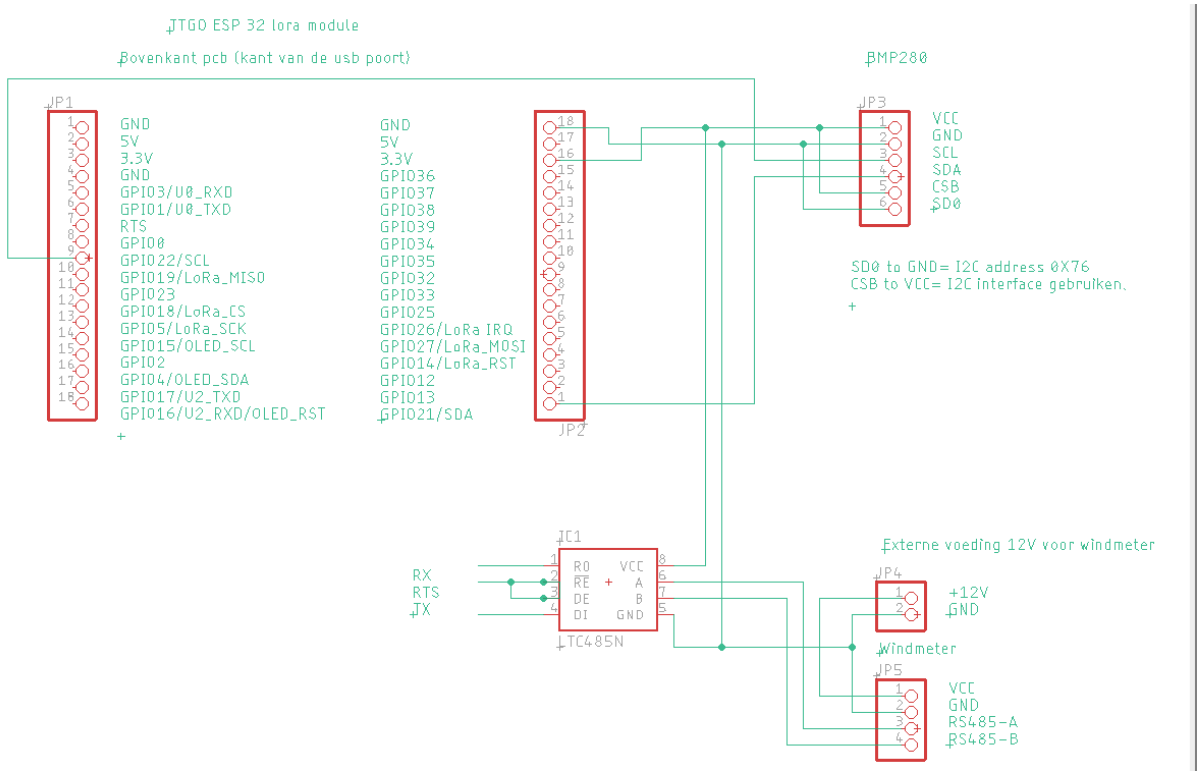
4 Blokschema

In dit blokschema zien we duidelijk de functionaliteit van de opstelling.

- De BMP280 en anemometer sturen data door naar de ESP32.
- De ESP32 zal gebruik maken van de ingebouwde LoRa module en zal op zijn beurt de data doorsturen naar de andere ESP32.
- De ontvangen data worden doorgestuurd naar de interne OLED display.



5 Eagle schema



6 I2C

6.1 Datasheet

https://cdn-shop.adafruit.com/datasheets/BST-BMP280-DS001-11.pdf

6.2 Uitleg

Zoals we weten, heeft ieder I2C protocol een slave en een master. In dit geval is de ESP module de master en is de sensor de slave.
Elk apparaat heeft een uniek adres nodig.

In dit geval is het BMP280-adres 0x76 hexadecimaal. (zie datasheet stukje, dit vind je ook terug op pagina 28)

5.2 I²C Interface

The I²C slave interface is compatible with Philips I²C Specification version 2.1. For detailed timings refer to Table 27. All modes (standard, fast, high speed) are supported. SDA and SCL are not pure open-drain. Both pads contain ESD protection diodes to VDDIO and GND. As the devices does not perform clock stretching, the SCL structure is a high-Z input without drain capability.

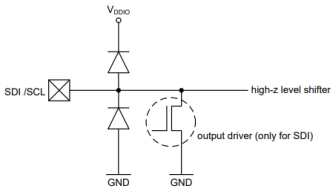


Figure 6: SDI/SCK ESD drawing

The 7-bit device address is 111011x. The 6 MSB bits are fixed. The last bit is changeable by SDO value and can be changed during operation. Connecting SDO to GND results in slave address 1110110 (0x76); connection it to V_{DDIO} results in slave address 1110111 (0x77), which

In dit stukje tekst zien we dus ook direct dat deze module gebruikt wordt als Active state => lijn laag (GND).

SDA is geconnecteerd aan gpio 21 van de ESP

SCL is geconnecteerd aan gpio 22 van de ESP.

Om de data uit te lezen, gebruiken we een library van een BME280:
In deze library kunnen we duidelijk de structuur zien voor het uitlezen. (De datasheet geeft dit ook aan → een vb. a.d.h.v. temperatuur).

Table 18: Memory map

Register Name	Address	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0	Reset state
temp_xlsb	0xFC	temp_xlsb<7:4>				0	0	0	0	0x00
temp_lsb	0xFB	temp_lsb<7:0>								0x00
temp_msb	0xFA	temp_msb<7:0>								0x80

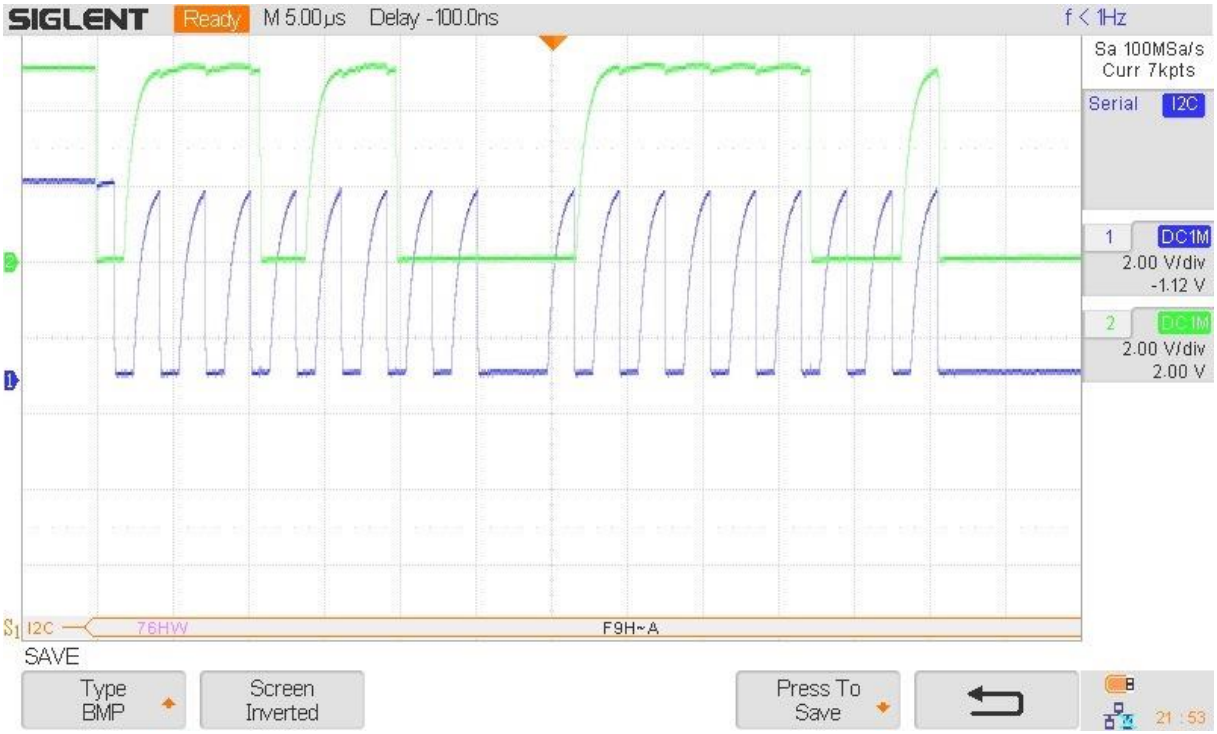
De code in de library toont aan dat de uitgelezen data per 8 bits uitgelezen wordt. Dit zien we doordat er gebitshift wordt naar links met 8 bits per keer. Er wordt 4 keer gebitshift naar rechts, aangezien de 4 nullen van de xlsb moeten weggewerkt worden.
(Dit zijn de ruwe data, er wordt met deze data nog verder gerekend in de library om een float variabele te krijgen)

```
msb = self._device.readU8(BME280_REGISTER_PRESSURE_DATA)
lsb = self._device.readU8(BME280_REGISTER_PRESSURE_DATA + 1)
xlsb = self._device.readU8(BME280_REGISTER_PRESSURE_DATA + 2)
raw = ((msb << 16) | (lsb << 8) | xlsb) >> 4
return raw
```

6.3 Metingen

Bij het meten van de bmp met de scoop ondervonden we een probleem.
Als we de meetklemmen van de scoop aan de bmp hingen, crashte het programma.
We stelden vast dat dit probleem zich voordeed wanneer we het programma geüpload hadden terwijl we aan het meten waren.

Het probleem bleek een boot pin te zijn. Als we het programma uploadden en maten tegelijkertijd, liep het programma vast. Eens het programma geüpload was, ondervonden we geen problemen meer.



6.4 Debuggen

De BMP kunnen we uitlezen a.d.h.v. een normale ESP32 met een library. Dit ging uiteindelijk niet met de TTGO ESP32. Het alternatief was om de temperatuur op te vragen door het I2C adres te gebruiken.
De voorbeeldcode met library die we eerst gebruikten kwam van volgende website:
https://github.com/dafvid/micropython-bmp280

De oplossing voor het probleem was het gebruiken van een BME280 library, dit is een chip uit dezelfde familie. Het enige verschil tussen de 2 is dat de BME280 ook humidity kan lezen.

6.5 Code

Controle voor het I2C adres:

```
from machine import I2C, Pin

i2c = I2C(0, sda=Pin(21), scl=Pin(22))
print(i2c.scan())
```

Dit is de correcte code voor het uitlezen van de BMP280.
Met als library: https://randomnerdtutorials.com/micropython-bme280-esp32-esp8266/

```
from machine import Pin, I2C
from time import sleep
import BME280

i2c = I2C(0,scl=Pin(22), sda=Pin(21))

while True:
  bme = BME280.BME280(i2c=i2c)
  temperature = bme.temperature
  pressure = bme.pressure

  print('Temperature is: ', temperature)
  print('Pressure is: ', pressure)
  sleep(10)
```

6.6 Resultaten

Het resultaat van de controle voor het I2C adres gaf ons: [118]
Dit betekent dat dit hexadecimaal 0x76 is en komt dus overeen met de datasheet.

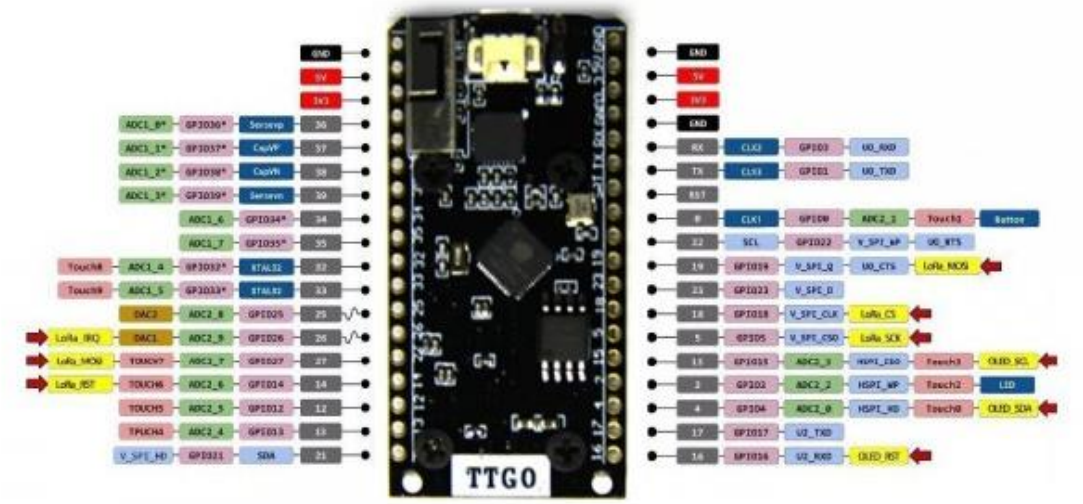
De oplossing gaf uiteindelijk het volgende als resultaat:

```
Temperature is:  20.68C
Pressure is:  1031.70hPa
```

7 OLED

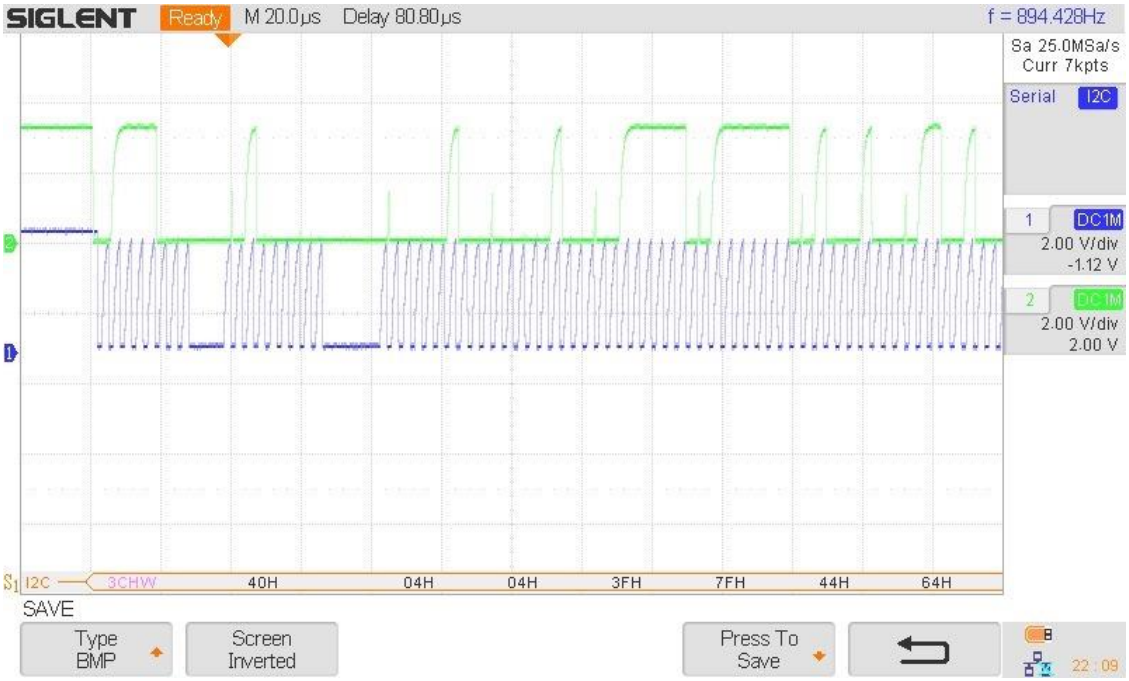
7.1 Uitleg

Het builtin OLED display werkt ook op een I2C protocol, maar zit op andere pinnen. In onderstaand schema kunt u dat zien. (oled_SCL=pin 15) (oled_SDA=pin 4) (oled_RST=pin 16). Het OLED display heeft een resolutie van 128 x 64.



7.2 Metingen

Hier zien we dat het I2C adres 0x3C is.



7.3 Debuggen

We konden niet direct data sturen naar de ingebouwde OLED display. We baseerden ons steeds op voorbeelden online waarbij een extern display was aangesloten.
Dit wilde dus niet werken. Het grote verschil met het ingebouwde display is dat er een OLED_RST pin verbonden is.
De oplossing van dit probleem hebben we gevonden door de RST pin een ‘1’ te sturen.

7.4 Code

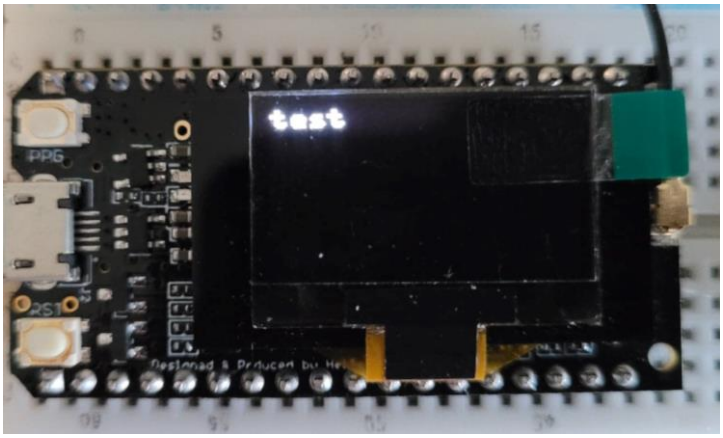
De code die we gebruiken, omvat een library:
https://github.com/micropython/micropython/blob/master/drivers/display/ssd1306.py

```
from machine import Pin,I2C
import ssd1306

p16 = Pin(16, Pin.OUT)
p16.value(1) # reset scherm

i2c = I2C(0,scl=Pin(15), sda=Pin(4)) # let op de pinnen

oled = ssd1306.SSD1306_I2C(128, 64, i2c)
oled.fill(0)
oled.text('test',0,0,1)
oled.show()
```

7.5 Resultaten

Het resultaat is dus dat we de tekst “test” in de linker bovenhoek zien verschijnen.
Stuur zowel de grootte van het scherm en de I2C data door naar de library.
oled = ssd1306.SSD1306_I2C(128, 64, i2c)

De achtergrondkleur wordt bepaald door oled.fill(0) de 0 staat voor kleur zwart.
De plaats en tekst was bepaald door oled.text('test',0,0,1), de 1 staat voor kleur wit.
Om het wel degelijk te displayen, gebruiken we oled.show()



8 LoRa

8.1 Uitleg

LoRa is een draadloze modulatietechniek die is afgeleid van Chirp Spread Spectrum (CSS)-technologie. Het codeert informatie op radiogolven met behulp van Chirp pulsen, vergelijkbaar met de manier waarop dolfijnen en vleermuizen communiceren. De LoRa gemoduleerde transmissie is robuust tegen storingen en kan over grote afstanden worden ontvangen.

De gekozen LoRa module werkt op een vrije frequentie van 868 MHz, op een andere frequentie uitzenden is strafbaar. In andere delen van de wereld zijn er andere vrije frequenties om op uit te zenden. Het is dus noodzakelijk om je goed te informeren welke de vrije frequentie is op de plaats van gebruik van de LoRa module.

De gebruikte LoRa module is de SX1276, deze module werkt met het ISP protocol.

(pin 26 = MOSI) (pin 19 = MISO) (pin 18 = CS) (pin 5 = SCK)

8.2 Metingen

Niettegenstaande dat onze opstelling werkte, konden we dit niet direct afleiden uit de scoop beelden.

8.3 Debuggen

De LoRa module werkte niet direct. In het begin hadden we ons gebaseerd op een library van op het internet waarin helaas wat foutjes zaten. De andere library die we daarna probeerden bleek wel correct te zijn en werkte direct.

8.4 Code

https://github.com/jpuk/simple-lora-esp32-test/tree/master/t-beam

main.py

```
import LoRaReceiver
from config import *
from machine import Pin, SPI
from sx127x import SX127x

device_spi = SPI(baudrate = 10000000,
        polarity = 0, phase = 0, bits = 8, firstbit = SPI.MSB,
        sck = Pin(device_config['sck'], Pin.OUT, Pin.PULL_DOWN),
        mosi = Pin(device_config['mosi'], Pin.OUT, Pin.PULL_UP),
        miso = Pin(device_config['miso'], Pin.IN, Pin.PULL_UP))

lora = SX127x(device_spi, pins=device_config,
parameters=lora_parameters)

#example = 'sender' #uncomment voor lora te gebruiken als zender
example = 'receiver' #uncomment voor lora te gebruiken als receiver

if __name__ == '__main__':
    if example == 'sender':
      LoRaSender.send(lora)
    if example == 'receiver':
      LoRaReceiver.receive(lora)
```

LoRaReceiver.py

```
from machine import Pin,I2C
import ssd1306
p16 = Pin(16, Pin.OUT)
p16.value(1) # resets screen
i2c = I2C(1,scl=Pin(15), sda=Pin(4), freq=400000)
oled = ssd1306.SSD1306_I2C(128, 64, i2c)

def receive(lora):
    print("LoRa Receiver")
    counter = 0
    oled.fill(0)
    payload = ['','','']
    #display = Display()
```

```
    while True:
        if lora.received_packet():
            if counter==3:
              counter=0
            lora.blink_led()
            print('something here')
            payload[counter] = lora.read_payload()
            print(payload[counter])
            oled.fill(0)
            oled.text(payload[0],0,0,1)
            oled.text(payload[1],0,20,1)
            oled.text(payload[2],0,40,1)
            oled.show()
            print(counter)
            counter += 1
```

LoRaSender.py

```
from time import sleep
from machine import Pin, I2C
import BME280
i2c = I2C(scl=Pin(22), sda=Pin(21), freq=10000)

def send(lora):
    counter = 0
    print("LoRa Sender")
    while True:
        bme = BME280.BME280(i2c=i2c)
        temp = bme.temperature
        hum = bme.humidity
        pres = bme.pressure
        payload = 'Temp: {0}'.format(temp)
        payload2 = 'Hum: {0}'.format(hum)
        payload3 = 'Pres: {0}'.format(pres)
        print("Sending packet: \n{}\n".format(payload))
        sleep(2)
        lora.println(payload2)
        sleep(2)
        lora.println(payload3)
        counter += 1
        sleep(2)
```

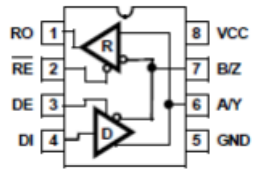
9 Windmeter

9.1 Uitleg

De windmeter werkt op het RS485 protocol, dit heeft als grote voordeel dat we grotere afstanden dan RS-232 kunnen afleggen; tot 1200 meter.
De snelheden van het versturen van data zijn ook een pak hoger: tot 35 Mbps@12 m.

De RS485 is een half duplex systeem, dit wil dus zeggen dat je data in beide richtingen kan sturen, maar niet op hetzelfde moment.

De modbus IC driver die we gebruiken is om te communiceren en moet specifiek voor half duplex zijn. (om de inquiry frame en answer frame te laten afwisselen).



RS-485, Half-Duplex, 8-Ld

Zoals we weten, heeft een RS485 zeker 2 draden nodig om signalen tegengesteld te sturen, een differentieel signaal dus. Dit wordt gebruikt om ruis te verminderen.
Wij hebben 4 draden omdat de windmeter 12 Volt en een GND nodig heeft.

De twee communicatiedraden bij onze windmeter zijn ten eerste de gele kabel, hij stelt het inverterende signaal voor en ten tweede de blauwe kabel, die stelt het positieve signaal voor.
De bruine kabel is + 12V en de zwarte kabel is GND.
Aangezien de gele en de blauwe kabel een differentieel signaal vormen, weten we dat ze een twisted pair vormen.

Er zijn verschillende communicatie parameters:

PARAMETERS	CONTENT
Code	8-bit binary
Data bits	8 bit
Parity bit	No
Stop bit	1 bit
Error checking	CRC (redundant loop code)
Baud rate	2400 bps/ 4800 bps/ 9600 bps can be set factory defaults to 9600 bps

Om data te kunnen krijgen sturen we eerst een inquiry frame.
Als dit frame correct ontvangen is, krijgen we een answer frame terug.
In de handleiding die bij de windmeter zat, zien we duidelijk welke twee frames dit zijn.

Inquiry Frame

Address Code	Function Code	Start Address	Data Length	CRC_L	CRC_H
0x01	0x03	0x00 0x16	0x00 0x01	0x65	0xCE

Answer Frames

(For example, reading a wind speed of 2.3m/s)

Address Code	Function Code	The Number Of Valid Bytes	Wind Speed Value	CRC_L	CRC_H
0x01	0x03	0x02	0x00 0x17	0xF8	0x4A

Wind speed:

0017 H (hexadecimal) = 23 => Wind Speed = 2.3m/s

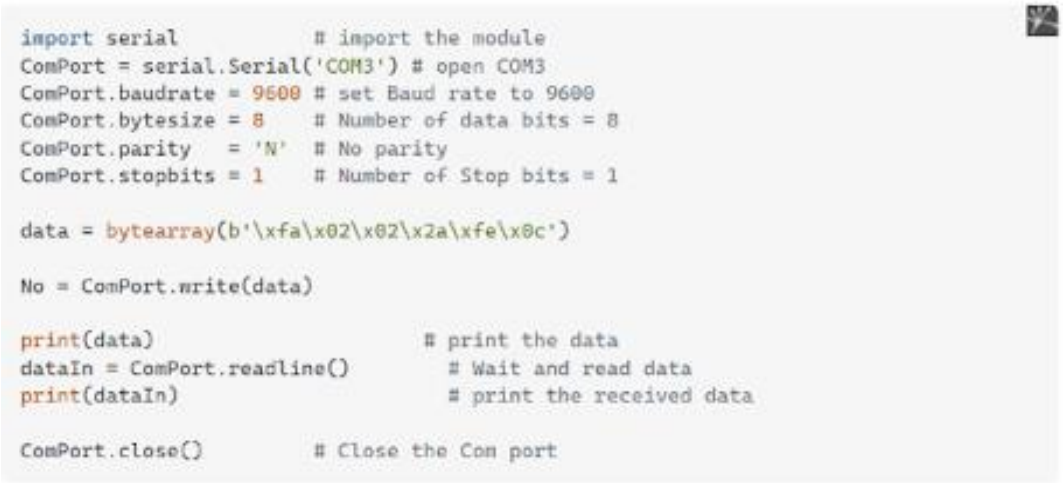
9.2 Metingen

9.3 Debuggen

https://create.arduino.cc/projecthub/philippedc/arduino-esp8266-rs485-modbus-anemometer-45f1d8

Door eerst de arduino code te proberen die we hadden gevonden via bovenstaande link, konden we perfect de windmeter uitlezen.

Om dit om te zetten, hadden we een code gevonden voor micropython dat we kunnen aanpassen, via volgende link.
https://stackoverflow.com/questions/66438228/rs485-communication-with-python



```
import serial            # import the module
ComPort = serial.Serial('COM3') # open COM3
ComPort.baudrate = 9600 # set Baud rate to 9600
ComPort.bytesize = 8    # Number of data bits = 8
ComPort.parity   = 'N'  # No parity
ComPort.stopbits = 1    # Number of Stop bits = 1

data = bytearray(b'\xfa\x02\x02\x2a\xfe\x0c')

No = ComPort.write(data)

print(data)                      # print the data
dataIn = ComPort.readline()        # Wait and read data
print(dataIn)                      # print the received data

ComPort.close()          # Close the Com port
```

9.4 Code

```
import time
from machine import UART,Pin

RTS_pin=Pin(4,Pin.OUT)

ComPort = UART(2,9600)        # init with given baudrate
ComPort.init(9600, bits=8, parity=None, stop=1, timeout=1000)

data = bytes([0x01, 0x03, 0x00, 0x16, 0x00, 0x01, 0x65, 0xCE])

RTS_pin.value(1)
time.sleep(0.2)
print(ComPort.write(data))
time.sleep(0.2)
print(data)
RTS_pin.value(0)
time.sleep(0.2)              # print the data
while not ComPort.any():
   time.sleep(0.2)
dataIn =  ComPort.read(8)       # Wait and read data
print(dataIn)                # print the received data
time.sleep(0.2)
```

9.5 Resultaten

De ontvangen data verstuurd via micropython.