**Academiejaar: 2021 / 2022**

**IOT Datacom project**

**Verslaggever: Lukas Verschraegen**

**Medestudent:** **Jannes Breusegem**

**Klas: 2ELO**

**Lector: Piet Coussens**

**Inhoudstafel:**

[1 Inleiding 3](#_Toc99793917)

[1.1 Onderwerp 3](#_Toc99793918)

[2 Module’s 3](#_Toc99793919)

[2.1 TTGO ESP32 module 3](#_Toc99793920)

[2.2 BMP280 3](#_Toc99793921)

[2.3 Anemometer 4](#_Toc99793922)

[3 Open source 4](#_Toc99793923)

[4 Blokschema 5](#_Toc99793924)

[5 Eagle schema 5](#_Toc99793925)

[6 I2C 6](#_Toc99793926)

[6.1 Datasheet 6](#_Toc99793927)

[6.2 Uitleg 6](#_Toc99793928)

[6.3 Metingen 7](#_Toc99793929)

[6.4 Debuggen 7](#_Toc99793930)

[6.5 Code 8](#_Toc99793931)

[6.6 Resultaten 8](#_Toc99793932)

[7 OLED 9](#_Toc99793933)

[7.1 Uitleg 9](#_Toc99793934)

[7.2 Metingen 9](#_Toc99793935)

[7.3 Debuggen 10](#_Toc99793936)

[7.4 Code 10](#_Toc99793937)

[7.5 Resultaten 10](#_Toc99793938)

[8 LoRa 11](#_Toc99793939)

[8.1 Uitleg 11](#_Toc99793940)

[8.2 Metingen 11](#_Toc99793941)

[8.3 Debuggen 12](#_Toc99793942)

[8.4 Code 13](#_Toc99793943)

[9 Windmeter 15](#_Toc99793944)

[9.1 Uitleg 15](#_Toc99793945)

[9.2 Metingen 16](#_Toc99793946)

[9.3 Debuggen 17](#_Toc99793947)

[9.4 Code 17](#_Toc99793948)

[9.5 Resultaten 18](#_Toc99793949)

# Inleiding

## 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.

# Module’s

## TTGO ESP32 module

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

Info:

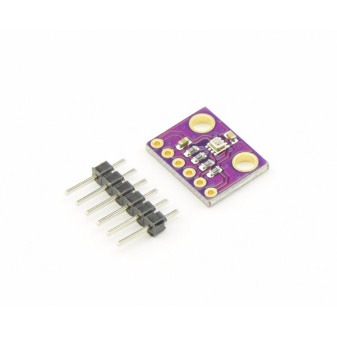
* ESP32 chip (240Mhz dual core processor)
* Flashgeheugen: 4MB
* SX1276 LoRa chip
* 0.96 inch OLED
* 1x LoRa antenne 868MHz

## 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.



## Anemometer

Afbeelding met meter

Automatisch gegenereerde beschrijving<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

# 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>

# Blokschema

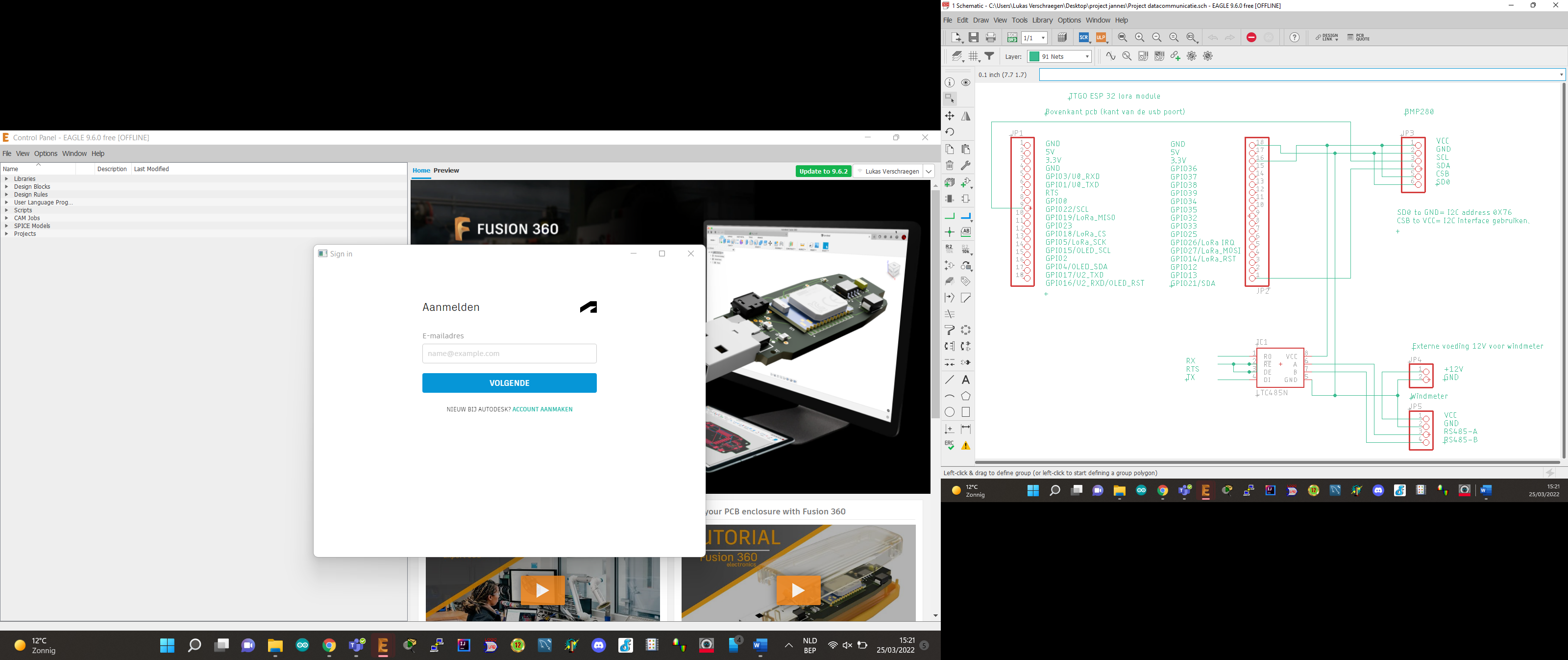
In dit blokschema zien we duidelijk de functionaliteit van de opstelling.

* 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.

Afbeelding met tekst, computer, binnen

Automatisch gegenereerde beschrijving

# Eagle schema



# I2C

## Datasheet

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

## 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)

Afbeelding met tekst, schermafbeelding, monitor, computer

Automatisch gegenereerde beschrijving

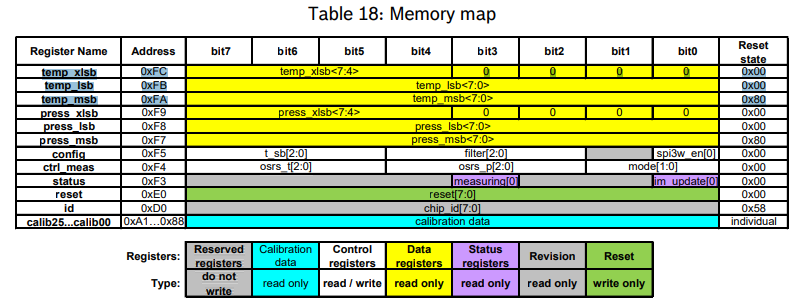
In dit stukje tekst zien we dus ook direct dat deze module gebruikt wordt als Active state => lijn laag (GND).

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).



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

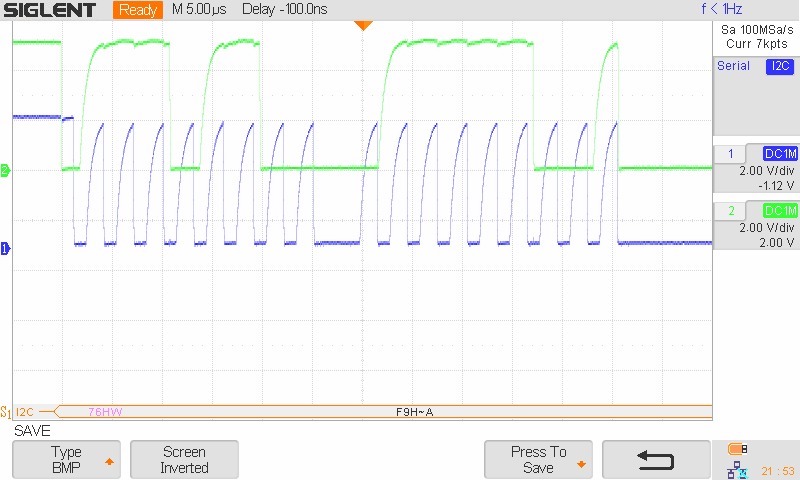
## Metingen

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.



## 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.

## 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)

## 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

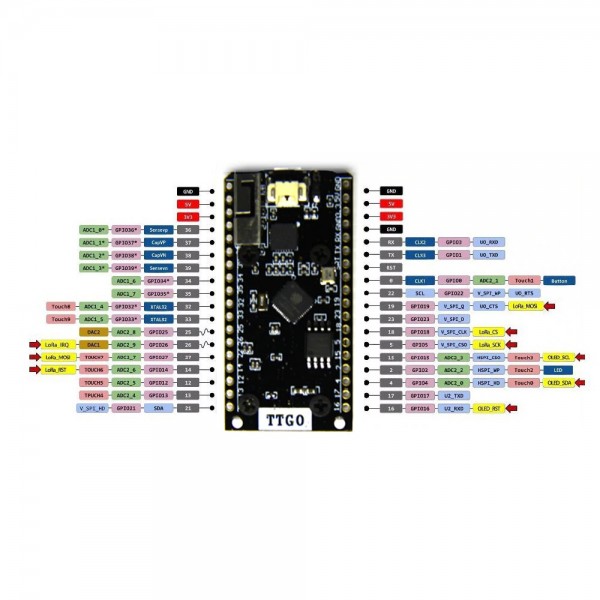
Pressure is: 1031.70hPa

# OLED

## 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.



## Metingen

Hier zien we dat het I2C adres 0x3C is.



## 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.

## 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()

## 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()

Afbeelding met tekst, elektronica, circuit

Automatisch gegenereerde beschrijving

# LoRa

## 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.

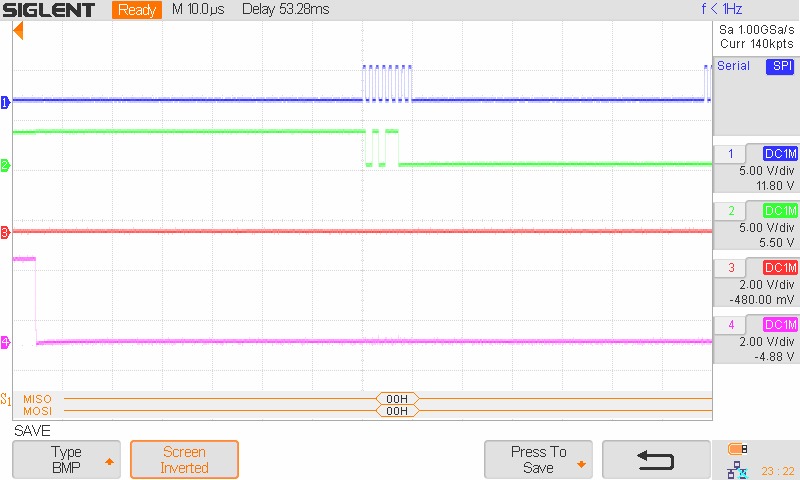
De gebruikte LoRa module is de SX1276, deze module werkt met het ISP protocol.

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

## Metingen

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





## 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.

## 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)

# Windmeter

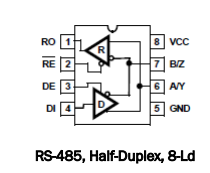
## Uitleg

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).



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.

Afbeelding met meter, apparaat

Automatisch gegenereerde beschrijving

Er zijn verschillende communicatie parameters:



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.





## Metingen



## 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>

Afbeelding met tekst, binnen, computer, schermafbeelding

Automatisch gegenereerde beschrijving

## 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)

## Resultaten

De ontvangen data verstuurd via micropython.

