

HERKANSING: EMBEDDED MICROCONTROLLER APPLICATIONS

Branddetectie

Jonas Van Hool

Inhoud

1. INLEIDING	5
2. UITGEBREID DOEL	6
2.1. Branddetectie met ESP32 en KY-026:	6
2.2. MQTT Communicatie:	6
2.3. Alarmactivering met Raspberry Pi en KY-006:	6
2.4. Alarmuitschakeling:	6
3. BENODIGDE COMPONENTEN	6
4. BENODIGDE LIBRARY'S	7
4.1. ESP32	7
4.1.1. Wifi.h	7
4.1.2. PubSubClient.h (MQTT)	7
4.2. Raspberry Pi	7
4.2.1. RPI.GPIO	7
4.2.2. Gpiozero	7
4.2.3. Mosquitto	8
5. WERKWIJZE/CODE CONTROLE	9
5.1. ESP32	9
5.1.1. Werking KY-026 (branddetectiemodule)	9
5.1.1.1. Code	10
5.1.2. MQTT	10
5.2. Raspberry Pi	13
5.2.1. Werking KY-006 (passieve buzzer module)	13
5.2.1.1. Code	14
5.2.2. Werking knop	14
5.2.2.1. Code	14
5.2.3. MQTT	15
5.3. Influxdb	17
5.4. GRAFANA	20
5.5. Opstarten bij boot:	20
6. VOLLE CODE	21
6.1. ESP32	21
6.2. Raspberry PI (Python)	23
7. VOLLEDIG AANSLUITSCHEMA	26
7.1. ESP32	26
7.2. Raspberry Pi	27
8. EXTRA BRONNEN:	28
8.1. KY-026	28
8.2. KY-006	28
8.3. Knop	28
8.4. MQTT Arduino	28

8.5. MQTT Raspberry Pi 28

1. Inleiding

Dit project beschrijft de ontwikkeling van een branddetectie- en alarmsysteem met behulp van een ESP32 microcontroller en een Raspberry Pi. Het systeem maakt gebruik van een KY-026 branddetectiemodule om brand te detecteren en een KY-006 passieve buzzermodule om een alarm te activeren. De communicatie tussen de ESP32 en de Raspberry Pi verloopt via het MQTT-protocol.

Bovendien wordt Grafana gebruikt om de status en meldingen van het systeem visueel te monitoren en te analyseren.

2. Uitgebreid Doel

Het doel van dit project is om een betrouwbaar en draadloos branddetectiesysteem te creëren dat in staat is om brand te detecteren en een alarm te activeren. Dit systeem zal continu de omgeving monitoren en bij het detecteren van een brand een waarschuwingssignaal sturen naar een centrale server (Raspberry Pi), die vervolgens een alarm laat afgaan. Het alarm kan handmatig worden uitgeschakeld via een knop, maar alleen als er geen brand meer wordt gedetecteerd.

2.1. Branddetectie met ESP32 en KY-026:

- De KY-026 branddetectiemodule is verbonden met de ESP32.
- De ESP32 leest de gegevens van de KY-026 en bepaalt of er brand is gedetecteerd.
- Bij detectie van brand stuurt de ESP32 een bericht via MQTT naar de broker op de Raspberry Pi.

2.2. MQTT Communicatie:

- De ESP32 is geconfigureerd als een MQTT client en publiceert berichten naar de broker wanneer brand wordt gedetecteerd.
- De Raspberry Pi fungeert als de MQTT broker en ontvangt berichten van de ESP32.

2.3. Alarmactivering met Raspberry Pi en KY-006:

- De Raspberry Pi is verbonden met een KY-006 passieve buzzermodule.
- De Raspberry Pi abonneert zich op het door de ESP32 gepubliceerde branddetectiebericht.
- Bij ontvangst van een branddetectiebericht activeert de Raspberry Pi de buzzer om een alarm af te laten gaan.

2.4. Alarmuitschakeling:

- Een drukknop is aangesloten op de Raspberry Pi.
- Wanneer de knop wordt ingedrukt, stuurt de Raspberry Pi een bericht naar de ESP32 om te controleren of er nog steeds brand wordt gedetecteerd.
- Als er geen brand meer wordt gedetecteerd, stuurt de ESP32 een bevestigingsbericht terug en stopt de Raspberry Pi het alarm.

3. Benodigde Componenten

- ESP32 microcontroller
- KY-026 branddetectiemodule
- Raspberry Pi (met MQTT broker, bijvoorbeeld Mosquitto)
- KY-006 passieve buzzermodule
- Drukknop
- Verbindingskabels
- Voedingsbronnen voor ESP32 en Raspberry Pi

4. Benodigde Library's

4.1. ESP32

4.1.1. Wifi.h

```
#include <WiFi.h>
```

Blynk by Volodymyr Shymanskyy

1.3.2 installed

Build a smartphone app for your project in minutes! It supports WiFi, Ethernet, Cellular connectivity. Works with over 400 boards like...
[More info](#)

1.3.2



REMOVE

4.1.2. PubSubClient.h (MQTT)

```
#include <PubSubClient.h>
```

PubSubClient by Nick O'Leary
<nick.oleary@gmail.com>

2.8 installed

A client library for MQTT messaging. MQTT is a lightweight messaging protocol ideal for small devices. This library allows you to send and receive...
[More info](#)

4.2. Raspberry Pi

4.2.1. RPI.GPIO

Om gebruik te kunnen maken van de buzzer aangesloten op de Raspberry Pi, hebben we de rpi.gpio library nodig.

```
jonaspi@jonaspi:~$ sudo apt-get install rpi.gpio
```

4.2.2. Gpiozero

<https://gpiozero.readthedocs.io/en/stable/installing.html>

```
jonaspi@jonaspi:~/oefeningen/herkansing$ sudo apt install python3-gpiozero
```

4.2.3. Mosquitto

<https://raspberrypi.tilburgs.com/mosquitto-mqtt/>

```
pi@raspberry:~ $ sudo apt update  
pi@raspberry:~ $ sudo apt install -y mosquitto mosquitto-clients
```

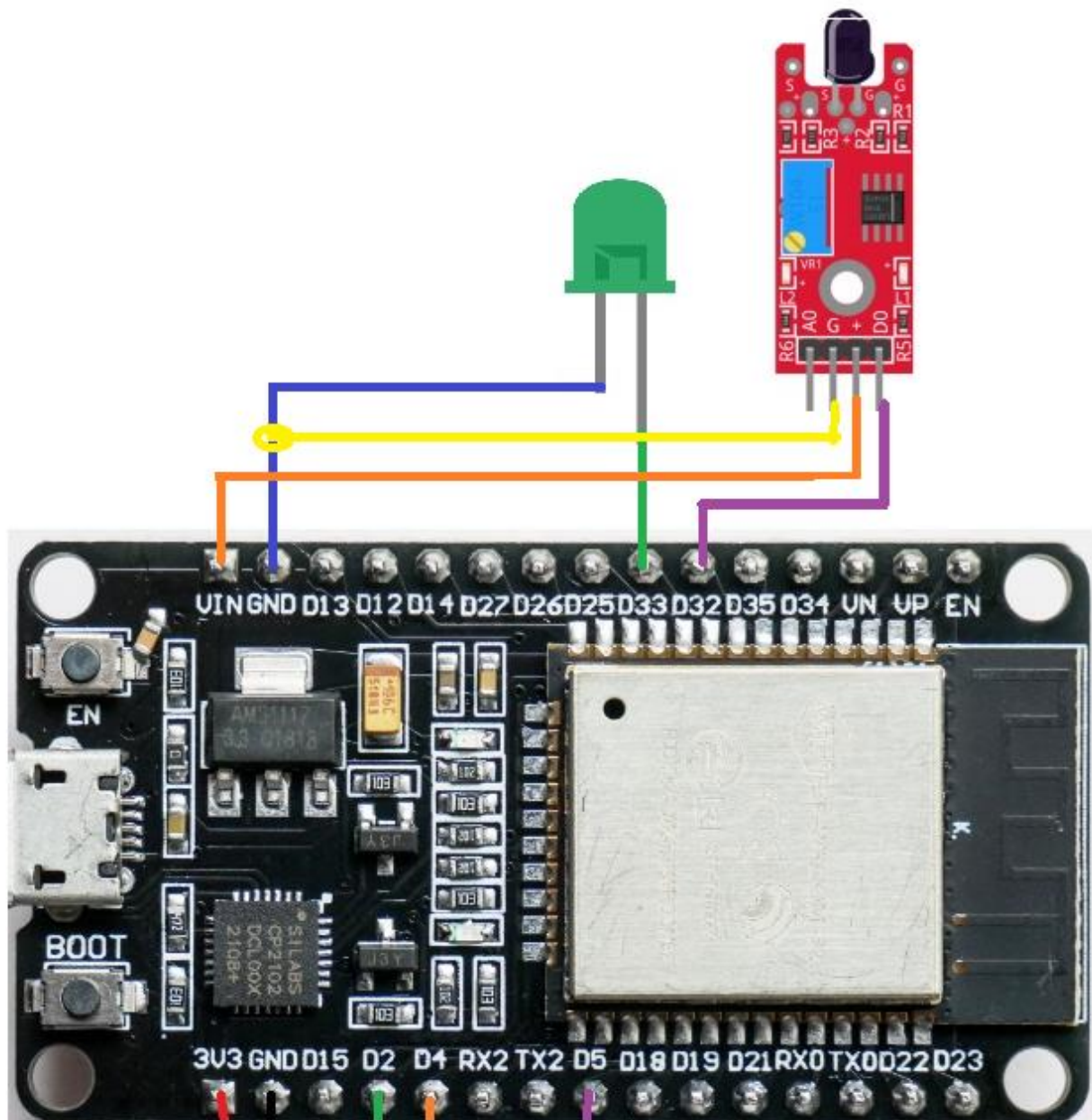
```
pi@raspberry:~ $ sudo systemctl enable mosquitto.service
```


5. Werkwijze/Code controle

5.1. ESP32

5.1.1. Werking KY-026 (branddetectiemodule)

De werking van de KY-026 kunnen we zeer gemakkelijk testen. Dit doen we door een groen lampje te laten branden als er **geen** brand wordt gedetecteerd.



5.1.1.1. Code

```
#define FLAME 32 // connect D0 pin of sensor to this pin
#define LED 33 // pin 33 for LED

void setup(){
  Serial.begin(9600);
  Serial.println("Fire Module Test");
  pinMode(FLAME, INPUT); //define FLAME input pin
  pinMode(LED, OUTPUT); //define LED output pin
}

void loop(){
  int fire = digitalRead(FLAME); // read FLAME sensor
  if(fire == HIGH){
    digitalWrite(LED, HIGH); // set the LED ON
    Serial.println("Fire! Fire!");
  }else{
    digitalWrite(LED, LOW); // Set the LED OFF
  }
  delay(200);
}
```

5.1.2. MQTT

Om het deel mqtt uit te leggen ga ik het opdelen per categorie: “boven void setup, void setup, void loop en void callback”.

BOVEN VOID SETUP

Om te kunnen verbinden met de broker moeten we op hetzelfde netwerk (**SSID** en **Password** aanpassen) als de broker zitten.

De **SSID** is de naam van je netwerk.

Het **Password** is het bijbehorende wachtwoord ervan.

Ook moeten we de instellingen van de server (broker) correct wijzigen.

Hierdoor zal je volgende 3 dingen moeten veranderen: “**mqttServer**, **mqttUser** en **mqttPassword**”.

Bij **mqttServer** kies je het IP-adres van de Raspberry Pi.

Bij **mqttUser** en **mqttPassword** geef je de hostnaam en het bijbehorende wachtwoord van de Raspberry Pi in.

```
#include <WiFi.h>
#include <PubSubClient.h>

// WiFi en MQTT server instellingen
const char* ssid = "Proximus-Home-69D8"; // WiFi SSID
const char* password = "wya7ymxdyzynd"; // WiFi wachtwoord
const char* mqttServer = "192.168.1.43"; // MQTT server IP-adres
const int mqttPort = 1883; // MQTT server poort
const char* mqttUser = "jonaspi"; // MQTT gebruikersnaam
const char* mqttPassword = "jonaspi"; // MQTT wachtwoord
const char* clientID = "ESP32"; // Client ID voor de ESP32

// WiFi en MQTT client initialisatie
WiFiClient espClient;
PubSubClient client(espClient);
```

VOID SETUP

```
// Verbinden met WiFi
WiFi.begin(ssid, password);
while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.println("Verbinden met WiFi...");
}
Serial.println("Verbonden met het WiFi-netwerk");

// Instellen van de MQTT server
client.setServer(mqttServer, mqttPort);
client.setCallback(callback); // Voeg de callback functie toe
```

VOID LOOP

In dit stukje code staat geschreven dat het mqtt berichten blijft verwerken (**client.loop();**)

Vervolgens staat er dat het gaat verbinden met MQTT als deze verbinding nog niet tot stand is gekomen. Als dit wel al het geval is, wordt dit stukje overgeslagen. Kan het geen verbinding maken dan zal deze de foutcode uitprinten op d seriële monitor en vervolgens nog eens proberen verbinding te maken.

!!TIP!! Bij foutmelding; kijk **mqttServer**, **mqttUser** en **mqttPassword** na.

In de lijn: **client.subscribe("alarm/alarm/reset");**
Gaat de ESP32 zich abonneren op topic **alarm/alarm/reset**

```
client.loop();
while (!client.connected()) {
  Serial.println("Verbinden met MQTT...");
  if (client.connect("ESP32Client", mqttUser, mqttPassword)) {
    Serial.println("Verbonden");
    client.subscribe("alarm/alarm/reset"); // Abonneer op het reset topic
  } else {
    Serial.print("Verbinding mislukt, status: ");
    Serial.print(client.state());
    delay(2000);
  }
}
```

In deze lijn gaat de ESP32 het bericht “vuur” naar de topic “alarm/alarm/trigger” sturen.

```
| | client.publish("alarm/alarm/trigger", "vuur");
```

VOID CALLBACK

Als er een bericht wordt **gepublished** waar wij op zijn gesubscribed (**void loop**) dan kunnen we dit bericht ontvangen:

Hieronder gaan we 1st op de Seriële Monitor de topic printen (nuttig bij gebruik van meerdere topics) En vervolgens het bericht. Het bericht wordt in een **for loop** ontleed en karakter per karakter weergegeven.

```
// Callback functie om berichten te verwerken die binnenkomen via MQTT
void callback(char* topic, byte* message, unsigned int length) {
  Serial.print("Bericht ontvangen op topic: ");
  Serial.print(topic);
  Serial.print(". Bericht: ");
  String messageString;

  for (int i = 0; i < length; i++) {
    Serial.print((char)message[i]);
    messageString += (char)message[i];
  }
  Serial.println();
}
```

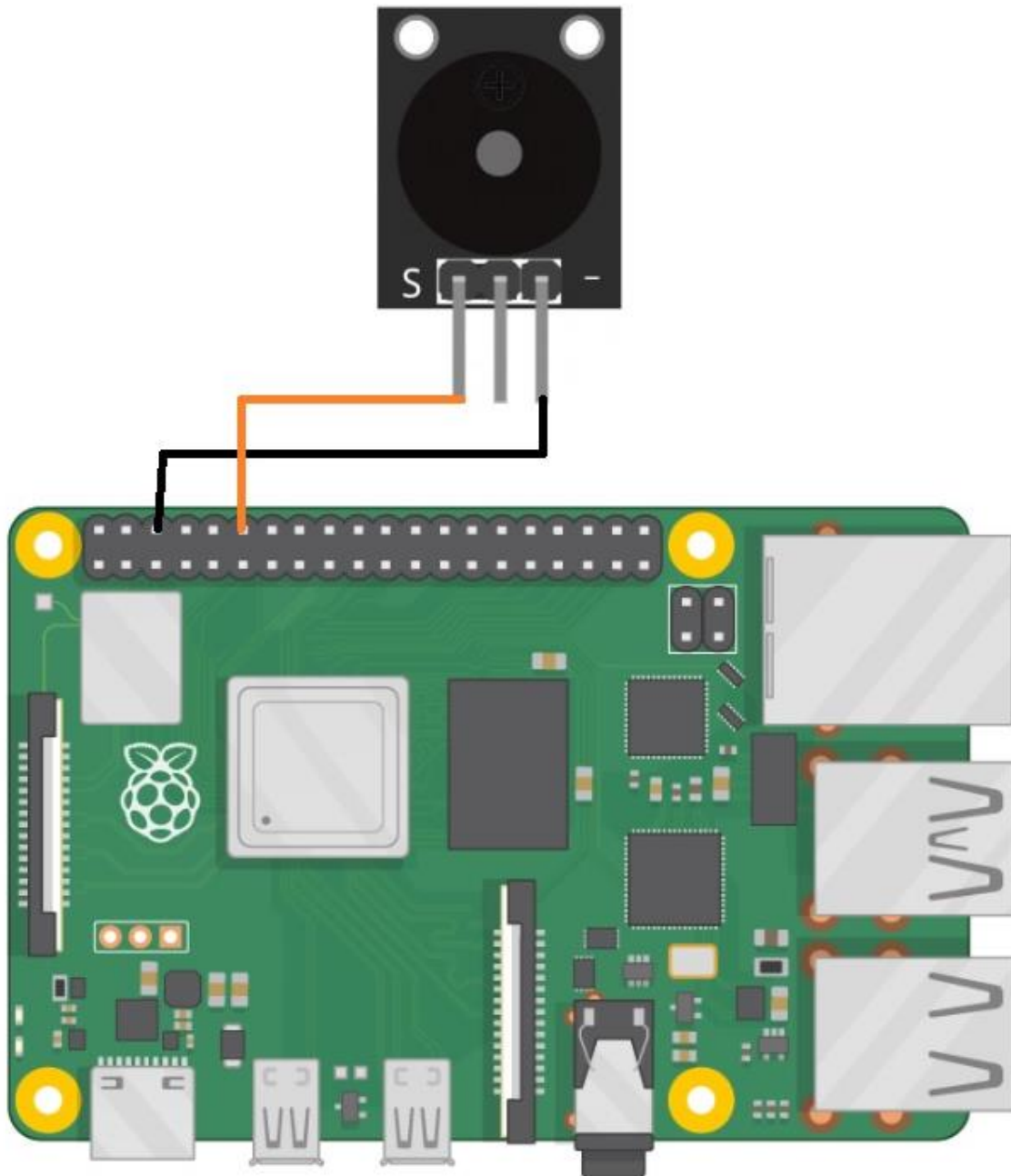
Vervolgens kunnen we ook acties uitvoeren aan de hand van wat we hebben binnengekregen. Zoals een LED aan zetten bij bericht 1 en de LED uit bij bericht 0 naar topic “alarm/alarm/reset”

```
if (String(topic) == "alarm/alarm/reset") {
  if (messageString == "0") {
    digitalWrite(LED, LOW);
  }
  if (messageString == "1") {
    digitalWrite(LED, HIGH);
  }
}
```

5.2. Raspberry Pi

5.2.1. Werking KY-006 (passieve buzzer module)

In deze code staat beschreven hoe we het signaal geleidelijk aan verhogen (zoals in brandalarm) door de pwm (frequentie) geleidelijk aan, aan te passen.



5.2.1.1. Code

```
jonaspi@jonaspi: ~/oefeningen/herkansing
GNU nano 7.2
import RPi.GPIO as GPIO
import time

# Gebruik de BCM pin-nummering
GPIO.setmode(GPIO.BCM)

# Definieer de GPIO pin waar de I/O pin van de buzzer is aangesloten
buzzer_pin = 18

# Stel de GPIO pin in als output
GPIO.setup(buzzer_pin, GPIO.OUT)

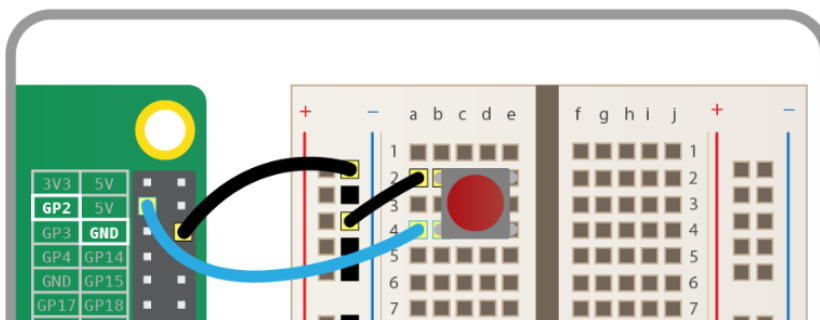
# Start PWM op de buzzer pin met een beginfrequentie
pwm = GPIO.PWM(buzzer_pin, 440) # 440 Hz is de frequentie van de A noot

# Start PWM met een duty cycle van 50%
pwm.start(50)

try:
    while True:
        # Verander de frequentie om de toonhoogte van de buzzer te veranderen
        for freq in range(500, 1000, 1):
            pwm.ChangeFrequency(freq)
            time.sleep(0.01)
            print("up")
except KeyboardInterrupt:
    # Stop PWM en maak GPIO schoon bij beëindiging van het programma
    pwm.stop()
    GPIO.cleanup()
```

5.2.2. Werking knop

Als je de knop hebt ingedrukt, wordt er **you pressed me** geprint.



5.2.2.1. Code

```
jonaspi@jonaspi: ~/oefeningen/herkar
GNU nano 7.2
from gpiozero import Button
button = Button(2)

button.wait_for_press()
print('you pressed me')
```

5.2.3. MQTT

Voor volledig en correct te kunnen bespreken, ga ik het opdelen in volgende onderdelen: "initialisatie, def on_connect, def on_message, verbinding instellingen, publish, try/except".

INITIALISATIE

Als eerste moet je de **paho.mqtt.client** library importeren.

Vervolgens kies je de topics waarop je wilt abonneren of op gaat pubblisen. (**alarm = ...**, **reset = ...**)

```
import paho.mqtt.client as mqtt
alarm = "alarm/alarm/trigger" # Topic om op te abonneren voor alarmactivatie
reset = "alarm/alarm/reset" # Topic om op te abonneren voor alarmreset
```

Vervolgens ga je de instellingen van de broker invullen, deze komen overeen met de instellingen die je voor de ESP32 (5.1.2 mqtt, boven void setup) hebt ingegeven.

Wijzig ook hier **mqtt_broker**, **mqtt_user** en **mqtt_password** naar de juiste instellingen.

```
mqtt_broker = "192.168.1.43"
mqtt_port = 1883
mqtt_user = "jonaspi"
mqtt_password = "jonaspi"
```

DEF ON_CONNECT

In de **def on_connect** gaan we bepalen op welke topics we ons abonneren.

Dit doen we met de **client.subscribe(...)**.

De lijn **print("verbonden met code: " + str(rc))** geeft de verbindingstatus met de broker weer.

```
def on_connect(client, userdata, flags, rc, properties=None):
    print("Verbonden met code: " + str(rc)) # Toont de verbindingstatus met de MQTT broker
    client.subscribe(alarm) # Abonneer op het alarm topic
    client.subscribe(reset) # Abonneer op het reset topic
```

DEF_ON MESSAGE

In de **def on_message(client, userdata, msg)**: gaan we de ontvangen berichten op de geabonneerde topics bekijken.

```
def on_message(client, userdata, msg):
```

Je ontvangt de **topic** en **payload**(bericht) via **msg.topic** en **msg.payload**.

Bij de **msg.payload** maak ik er **msg.payload.decode("utf-8")** van aangezien alle binnengekomen karakters dan ook correct worden vervangen.

Vervolgens printen we het ontvangen **topic** en **payload** uit.

```
    topic = msg.topic
    payload = msg.payload.decode("utf-8")

    print(f"Ontvangen bericht: Topic: {topic}, Payload: {payload}") # T
```


Vervolgens kan je aan de hand van de binnengekomen berichten of topics acties uitvoeren. In onderstaande code gaan we dus 1st kijken of het **topic** gelijk is aan dat van **alarm** ("alarm/alarm/alarm").

Indien dit het geval is gaan we kijken wat het ontvangen **bericht** is dat bij deze topic hoort.

Is het ontvangen bericht **vuur**, dan gaan we **FIIIIIIIREEEEE** printen.

Is het ontvangen bericht **geen vuur**, dan gaan we **Geen vuur, alarm uit** printen.

```
if topic == alarm:
    if payload == "vuur":
        print("FIIIIIIIREEEEEEEEEEEE") # Toont bericht bij alarmactivatie
    elif payload == "geen vuur":
        print("Geen vuur, alarm uit") # Toont bericht bij deactivering van het alarm
```

V ERBINDINGS INSTELLINGEN

Deze instellingen zorgen ervoor dat je correct kan verbinden met de MQTT broker.

Het kan zijn dat je bij **mqttc** het volgende schrijft: **mqttc = mqtt.Client()**

```
mqttc = mqtt.Client(mqtt.CallbackAPIVersion.VERSION2)
mqttc.on_connect = on_connect
mqttc.on_message = on_message
mqttc.username_pw_set(mqtt_user, mqtt_password)
mqttc.connect(mqtt_broker, mqtt_port, 60) # Verbind met MQTT broker
```

PUBLISH

Met **mqttc.publish(topic, message)** kan je een gekozen **bericht** op een gekozen **topic** publiceren.

In mijn geval ga ik dus het bericht **1** naar het topic **reset** (alarm/alarm/reset) sturen.

```
mqttc.publish(reset, "1") # Publiceer bericht om alarm te resetten
```

TRY/EXCEPT

De try gaat de daarin beschreven code 1 keer doorlopen. Daarom staat er nog eens de voorwaarde **while True**: dit is zodat de code constant geloopt wordt.

Hierin plaatsen we dan ook de **mqttc.loop()** deze lijn zorgt ervoor dat alles in de hierboven hoofdstukjes in verband met mqtt worden uitgevoerd.

```
try:
    while True:
        mqttc.loop() # Blijf MQTT berichten verwerken
```

Dit gebeurt totdat de **except** voorwaarde wordt bereikt (de **KeyboardInterrupt**).

Als dit gebeurt, dan gaan we de mqtt verbinding verbreken.

```
except KeyboardInterrupt:
    mqttc.disconnect() # Verbreek de verbinding met MQTT broker bij KeyboardInterrupt
```


5.3. Influxdb

Om gebruik te maken van influxdb(-clients) moeten we deze 1st nog installeren:

```
SERRE@SERRE:~/serre $ sudo apt-get install influxdb
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following packages were automatically installed and are no longer required:
  libraspberrypi0 libwpe-1.0-1 libwpebackend-fdo-1.0-1
Use 'sudo apt autoremove' to remove them.
The following NEW packages will be installed:
  influxdb
0 upgraded, 1 newly installed, 0 to remove and 0 not upgraded.
Need to get 4,483 kB of archives.
After this operation, 18.0 MB of additional disk space will be used.
Get:1 http://deb.debian.org/debian bookworm/main arm64 influxdb arm64 1.6.7~rc0-1+b13 [4,483 kB]
Fetched 4,483 kB in 2s (1,993 kB/s)
Selecting previously unselected package influxdb.
(Reading database ... 147733 files and directories currently installed.)
Preparing to unpack .../influxdb_1.6.7~rc0-1+b13_arm64.deb ...
Unpacking influxdb (1.6.7~rc0-1+b13) ...
Setting up influxdb (1.6.7~rc0-1+b13) ...
Adding system user `influxdb' (UID 113) ...
Adding new user `influxdb' (UID 113) with group `nogroup' ...
Not creating home directory `/var/lib/influxdb'.
Adding group `influxdb' (GID 123) ...
Done.
Adding user `influxdb' to group `influxdb' ...
Done.
Created symlink /etc/systemd/system/influxd.service → /lib/systemd/system/influxdb.service.
Created symlink /etc/systemd/system/multi-user.target.wants/influxdb.service → /lib/systemd/system/influxdb.service.
Processing triggers for man-db (2.11.2-2) ...

SERRE@SERRE:~/serre $ sudo apt-get install influxdb-client
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following packages were automatically installed and are no longer required:
  libraspberrypi0 libwpe-1.0-1 libwpebackend-fdo-1.0-1
Use 'sudo apt autoremove' to remove them.
The following NEW packages will be installed:
  influxdb-client
0 upgraded, 1 newly installed, 0 to remove and 0 not upgraded.
Need to get 2,034 kB of archives.
After this operation, 6,828 kB of additional disk space will be used.
Get:1 http://deb.debian.org/debian bookworm/main arm64 influxdb-client arm64 1.6.7~rc0-1+b13 [2,034 kB]
Fetched 2,034 kB in 1s (2,076 kB/s)
Selecting previously unselected package influxdb-client.
(Reading database ... 147748 files and directories currently installed.)
Preparing to unpack .../influxdb-client_1.6.7~rc0-1+b13_arm64.deb ...
Unpacking influxdb-client (1.6.7~rc0-1+b13) ...
Setting up influxdb-client (1.6.7~rc0-1+b13) ...
Processing triggers for man-db (2.11.2-2) ...
```

Vervolgens moeten we controleren of de status actief is.

Indien dit niet het geval is, pas dan in onderstaande lijn het volgende aan: status → enable. En controleer daarna nogmaals.

```
SERRE@SERRE:~/serre $ sudo systemctl status influxdb
● influxdb.service - InfluxDB is an open-source, distributed, time series database
   Loaded: loaded (/lib/systemd/system/influxdb.service; enabled; preset: enabled)
   Active: active (running) since Wed 2024-05-29 10:07:16 CEST; 3min 50s ago
     Docs: man:influxd(1)
    Main PID: 31041 (influxd)
      Tasks: 9 (limit: 8732)
         CPU: 266ms
```

Als influxdb actief is dan gaan we volgend bestand wijzigen.

```
SERRE@SERRE:~/serre $ sudo nano /etc/influxdb/influxdb.conf
```

Haal de # voor enabled = true bij http weg. Om deze lijn te activeren.

```
[http]
# Determines whether HTTP endpoint is enabled.
enabled = true

# The bind address used by the HTTP service.
```

Vervolgens kan je je eigen database aanmaken.

```
jonaspi@jonaspi:~ $ influx
Connected to http://localhost:8086 version 1.6.7~rc0
InfluxDB shell version: 1.6.7~rc0
> CREATE DATABASE Brandalarm
>
> CREATE USER Jonas with password 'Jonas'
> GRANT ALL ON Brandalarm to Jonas
> exit
```

- CREATE DATABASE “naam_database” → kies je een naam voor de database
- CREATE USER “gebruikersnaam” with password “wachtwoord” → kies gebruiker met wachtwoord voor database. **Belangrijk voor grafana.**
- GRANT ALL ON “naam_database” TO “gebruikersnaam”
- Exit → Verlaat influx

Vervolgens maken we een brug aan.

```
SERRE@SERRE:~/serre $ nano bridge.py
```

En vul je onderstaande code in:

PAS AAN: influxdb_(adress, user, password, database) en MQTT_(adress, user, password). Bij

MQTT_(topic, regex) pas je het 1^{ste} woord aan.

```
import re
from typing import NamedTuple

import paho.mqtt.client as mqtt
from influxdb import InfluxDBClient

INFLUXDB_ADDRESS = '192.168.1.34'
INFLUXDB_USER = 'jonaspi'
INFLUXDB_PASSWORD = 'jonaspi'
INFLUXDB_DATABASE = 'Brandalarm'

MQTT_ADDRESS = '192.168.1.34'
MQTT_USER = 'jonaspi'
MQTT_PASSWORD = 'jonaspi'
MQTT_TOPIC = 'alarm/+/+'
MQTT_REGEX = 'alarm/([^\s]+)/([^\s/]+)'
MQTT_CLIENT_ID = 'MQTTInfluxDBBridge'

influxdb_client = InfluxDBClient(INFLUXDB_ADDRESS, 8086, INFLUXDB_USER, INFLUXDB_PASSWORD, None)

class SensorData(NamedTuple):
    location: str
    measurement: str
    value: float

def on_connect(client, userdata, flags, rc):
    """ The callback for when the client receives a CONNACK response from the server."""
    print('Connected with result code ' + str(rc))
    client.subscribe(MQTT_TOPIC)

def _parse_mqtt_message(topic, payload):
    match = re.match(MQTT_REGEX, topic)
    if match:
        location = match.group(1)
        measurement = match.group(2)
        if measurement == 'status':
            return None
        return SensorData(location, measurement, str(payload))
    else:
        return None

def _send_sensor_data_to_influxdb(sensor_data):
    json_body = [
        {
            'measurement': sensor_data.measurement,
            'tags': {
                'location': sensor_data.location
            },
            'fields': {
                'value': sensor_data.value
            }
        }
    ]
    influxdb_client.write_points(json_body)

def on_message(client, userdata, msg):
    """The callback for when a PUBLISH message is received from the server."""
    print(msg.topic + ' ' + str(msg.payload))
    sensor_data = _parse_mqtt_message(msg.topic, msg.payload.decode('utf-8'))
    if sensor_data is not None:
        _send_sensor_data_to_influxdb(sensor_data)
```

```
def _init_influxdb_database():
    databases = influxdb_client.get_list_database()
    if len(list(filter(lambda x: x['name'] == INFLUXDB_DATABASE, databases))) == 0:
        influxdb_client.create_database(INFLUXDB_DATABASE)
    influxdb_client.switch_database(INFLUXDB_DATABASE)

def main():
    _init_influxdb_database()

    mqtt_client = mqtt.Client(MQTT_CLIENT_ID)
    mqtt_client.username_pw_set(MQTT_USER, MQTT_PASSWORD)
    mqtt_client.on_connect = on_connect
    mqtt_client.on_message = on_message

    mqtt_client.connect(MQTT_ADDRESS, 1883)
    mqtt_client.loop_forever()

if __name__ == '__main__':
    print('MQTT to InfluxDB bridge')
    main()
```

Help Write Out Where Is Cut Execute Location Undo
Exit Read File Replace Paste Justify Go To Line Redo

Nu kan je deze code runnen: (Merk op dat de ESP ook aan moet staan)

```
SERRE@SERRE:~/serre $ python bridge.py
MQTT to InfluxDB bridge
Connected with result code 0
serre/serre/humdht11 b'48.00'
serre/serre/tempdht11 b'25.00'
serre/serre/ds18b20 b'24.63'
serre/serre/bodemvochtsensor b'0.00'
serre/serre/ultrasone b'96.82'
serre/serre/ldr b'0.00'
```

5.4. GRAFANA

Volg het stappenplan op onderstaande website stap voor stap bij installatie GRAFANA:

<https://grafana.com/tutorials/install-grafana-on-raspberry-pi/>

Als dit volledig is gelukt, kan je browsen naar `http://<ip address>:3000` inloggen met gebruikersnaam/wachtwoord admin. Vervolgens kan je deze aanpassen naar eigen keuze.

5.5. Opstarten bij boot:

In onderstaande video wordt dit uitgebreid uitgelegd hoe je dit moet verwezenlijken.

<https://www.youtube.com/watch?v=meeKv3DY9ps>

6. Volle Code

6.1. ESP32

```
#include<WiFi.h>
#include<PubSubClient.h>

// WiFi en MQTT server instellingen
constchar* ssid = "Proximus-Home-69D8"; // WiFi SSID
constchar* password = "wya7ymxdyzynd"; // WiFi wachtwoord
constchar* mqttServer = "192.168.1.43"; // MQTT server IP-adres
constintmqttPort = 1883; // MQTT server poort
constchar* mqttUser = "jonaspi"; // MQTT gebruikersnaam
constchar* mqttPassword = "jonaspi"; // MQTT wachtwoord
constchar* clientID = "ESP32"; // Client ID voor de ESP32

// WiFi en MQTT client initialisatie
WiFiClientespClient;
PubSubClientclient(espClient);

#defineFLAME32 // Aansluiting DO pin van sensor op deze pin
#defineLED33 // Pin voor LED

int status = HIGH; // Globale statusvariabele

voidsetup(){
    Serial.begin(9600);
    pinMode(FLAME, INPUT); // Definieer FLAME input pin
    pinMode(LED, OUTPUT); // Definieer LED output pin

    // Verbinden met WiFi
    WiFi.begin(ssid, password);
    while(WiFi.status() != WL_CONNECTED){
        delay(500);
        Serial.println("Verbinden met WiFi...");
    }
    Serial.println("Verbonden met het WiFi-netwerk");

    // Instellen van de MQTT server
    client.setServer(mqttServer, mqttPort);
    client.setCallback(callback); // Voeg de callback functie toe
}

voidloop(){
    client.loop();
    while(!client.connected()){
        Serial.println("Verbinden met MQTT...");
```

```
if(client.connect("ESP32Client", mqttUser, mqttPassword)){
    Serial.println("Verbonden");
    client.subscribe("alarm/alarm/reset"); // Abonneer op het reset topic
}else{
    Serial.print("Verbinding mislukt, status: ");
    Serial.print(client.state());
    delay(2000);
}
}

int fire = digitalRead(FLAME); // Lees de FLAME sensor
// Als er vuur wordt gedetecteerd gaat de noodverlichtings-LED aan en wordt
er gepubliceerd dat het alarm aan moet.
if(fire == HIGH && status == HIGH){
    digitalWrite(LED, HIGH); // Zet de LED aan
    Serial.println("Vuur! Vuur!");
    client.publish("alarm/alarm/trigger", "vuur"); // Publiceer bericht
"vuur" naar het topic alarm/alarm/trigger
    status = LOW; // Update de status naar LOW om aan te geven dat het alarm
is geactiveerd
}
}

// Callback functie om berichten te verwerken die binnenkomen via MQTT
void callback(char*topic, byte*message, unsigned int length){
    Serial.print("Bericht ontvangen op topic: ");
    Serial.print(topic);
    Serial.print(". Bericht: ");
    String messageString;

    for(int i = 0; i < length; i++){
        Serial.print((char)message[i]);
        messageString += (char)message[i];
    }
    Serial.println();
    // Als een bericht is ontvangen op het topic alarm/alarm/reset, controleer
of het bericht "1" is.
    // Verander de status terug naar HIGH
    if(String(topic) == "alarm/alarm/reset"){
        Serial.print("Reset ontvangen: ");
        if(messageString == "1" && digitalRead(FLAME) == LOW){
            Serial.println("BINGO");
            client.publish("alarm/alarm/trigger", "geen vuur"); // Stuur bericht
"geen vuur"
            status = HIGH; // Update de globale statusvariabele
            digitalWrite(LED, LOW); // Zet de LED uit
        }
    }
}
}
```

6.2. Raspberry PI (Python)

```
from gpiozero import Button
button = Button(2) # Initialiseer een knop op pin 2

import RPi.GPIO as GPIO
GPIO.setmode(GPIO.BCM)
buzzer_pin = 18
GPIO.setup(buzzer_pin, GPIO.OUT)
pwm = GPIO.PWM(buzzer_pin, 0.1) # Initialiseer PWM op pin 18 met frequentie 0.1 Hz
pwm.start(50)
pwm.stop() # Zet de buzzer uit bij het starten

import time

import paho.mqtt.client as mqtt
alarm = "alarm/alarm/trigger" # Topic om op te abonneren voor alarmactivatie
reset = "alarm/alarm/reset" # Topic om op te abonneren voor alarmreset

mqtt_broker = "192.168.1.43"
mqtt_port = 1883
mqtt_user = "jonaspi"
mqtt_password = "jonaspi"

alarm_active = False # Variabele om de status van het alarm bij te houden

def on_connect(client, userdata, flags, rc, properties=None):
    print("Verbonden met code: " + str(rc)) # Toont de verbindingstatus met de MQTT broker
    client.subscribe(alarm) # Abonneer op het alarm topic
    client.subscribe(reset) # Abonneer op het reset topic

def on_message(client, userdata, msg):
    global alarm_active
    topic = msg.topic
    payload = msg.payload.decode("utf-8")

    print(f"Ontvangen bericht: Topic: {topic}, Payload: {payload}") # Toont ontvangen berichten

    if topic == alarm:
        if payload == "vuur":
            print("FIIIIIIIREEEEEEEEEEEE") # Toont bericht bij alarmactivatie
            alarm_active = True
            pwm.start(50) # Start de buzzer met 50% duty cycle
        elif payload == "geen vuur":
            print("Geen vuur, alarm uit") # Toont bericht bij deactivering van het alarm
            alarm_active = False
            pwm.stop() # Stop de buzzer

    mqttc = mqtt.Client(mqtt.CallbackAPIVersion.VERSION2)
    mqttc.on_connect = on_connect
    mqttc.on_message = on_message
    mqttc.username_pw_set(mqtt_user, mqtt_password)
    mqttc.connect(mqtt_broker, mqtt_port, 60) # Verbind met MQTT broker

def check_button():
    if button.is_pressed:
        print("Resetknop ingedrukt") # Toont bericht bij indrukken van de resetknop
        mqttc.publish(reset, "1") # Publiceer bericht om alarm te resetten
```

```
try:
    while True:
        mqttc.loop() # Blijf MQTT berichten verwerken
        check_button() # Controleer of de resetknop is ingedrukt
        if alarm_active:
            for freq in range(500, 900, 1):
                if not alarm_active:
                    break
                pwm.ChangeFrequency(freq) # Verander frequentie van de buzzer bij alarmactivering (500-900Hz)
            time.sleep(0.01)
        else:
            time.sleep(0.1) # Wacht tussen lussen
```

```
except KeyboardInterrupt:
    mqttc.disconnect() # Verbreek de verbinding met MQTT broker bij KeyboardInterrupt
    pwm.stop() # Zet de buzzer uit
    GPIO.cleanup() # Maak GPIO pinnen schoon
```

```
jonaspi@jonaspi: ~/oefeningen/herkansing
GNU nano 7.2 full-c
from gpiozero import Button
button = Button(2) # Initialiseer een knop op pin 2

import RPi.GPIO as GPIO
GPIO.setmode(GPIO.BCM)
buzzer_pin = 18
GPIO.setup(buzzer_pin, GPIO.OUT)
pwm = GPIO.PWM(buzzer_pin, 0.1) # Initialiseer PWM op pin 18 met frequentie 0.1 Hz

import time

import paho.mqtt.client as mqtt
alarm = "alarm/alarm/trigger" # Topic om op te abonneren voor alarmactivatie
reset = "alarm/alarm/reset" # Topic om op te abonneren voor alarmreset

mqtt_broker = "192.168.1.43"
mqtt_port = 1883
mqtt_user = "jonaspi"
mqtt_password = "jonaspi"

alarm_active = False # Variabele om de status van het alarm bij te houden

def on_connect(client, userdata, flags, rc, properties=None):
    print("Verbonden met code: " + str(rc)) # Toont de verbindingstatus met de MQTT broker
    client.subscribe(alarm) # Abonneer op het alarm topic
    client.subscribe(reset) # Abonneer op het reset topic

def on_message(client, userdata, msg):
    global alarm_active
    topic = msg.topic
    payload = msg.payload.decode("utf-8")

    print(f"Ontvangen bericht: Topic: {topic}, Payload: {payload}") # Toont ontvangen berichten

    if topic == alarm:
        if payload == "vuur":
            print("FIIIIIIIREEEEEEEEE") # Toont bericht bij alarmactivatie
            alarm_active = True
            pwm.start(50) # Start de buzzer met 50% duty cycle
        elif payload == "geen vuur":
            print("Geen vuur, alarm uit") # Toont bericht bij deactivering van het alarm
            alarm_active = False
            pwm.stop() # Stop de buzzer

mqttc = mqtt.Client(mqtt.CallbackAPIVersion.VERSION2)
mqttc.on_connect = on_connect
mqttc.on_message = on_message
mqttc.username_pw_set(mqtt_user, mqtt_password)
mqttc.connect(mqtt_broker, mqtt_port, 60) # Verbind met MQTT broker

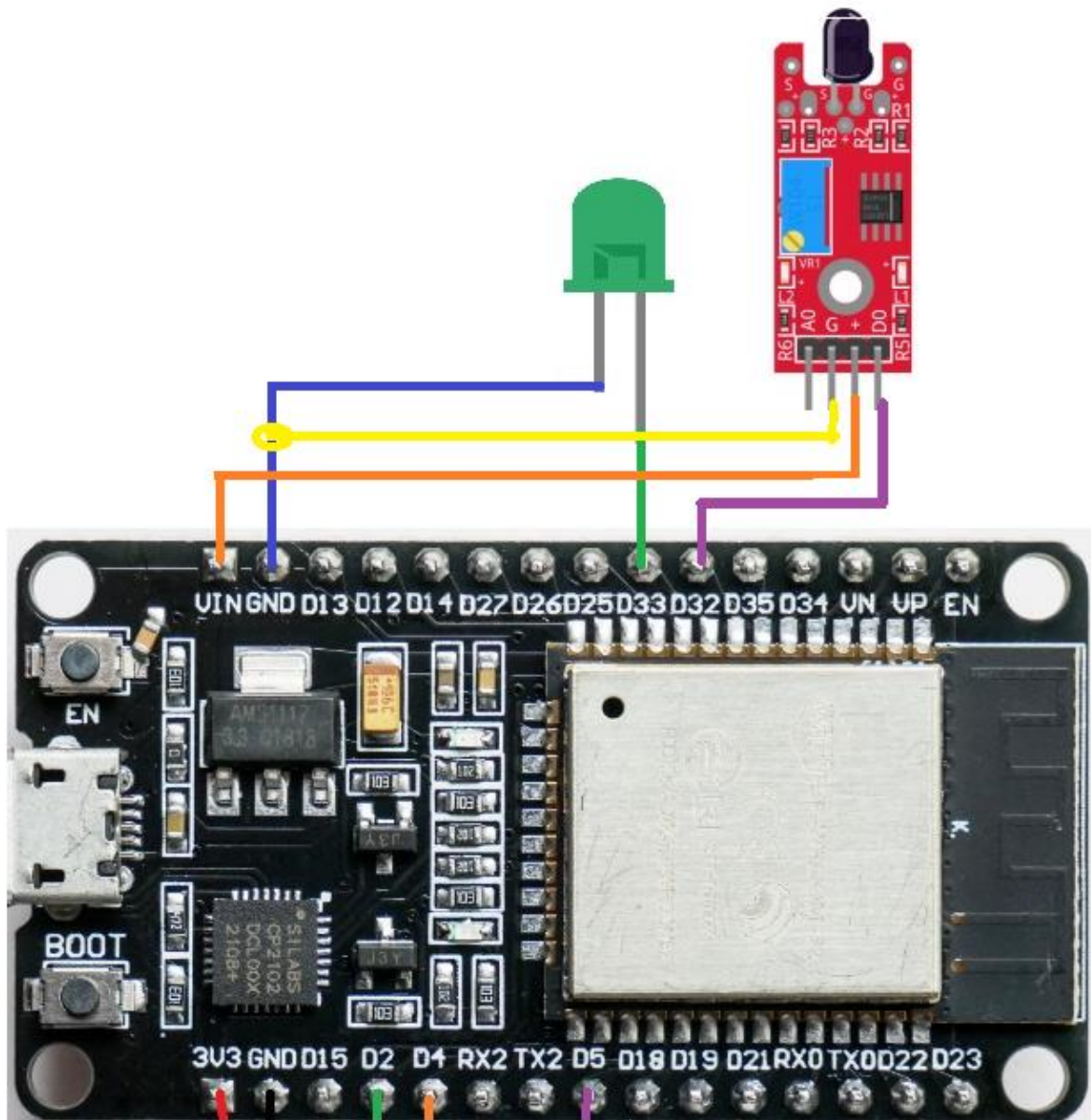
def check_button():
    if button.is_pressed:
        print("Resetknop ingedrukt") # Toont bericht bij indrukken van de resetknop
        mqttc.publish(reset, "1") # Publiceer bericht om alarm te resetten
```



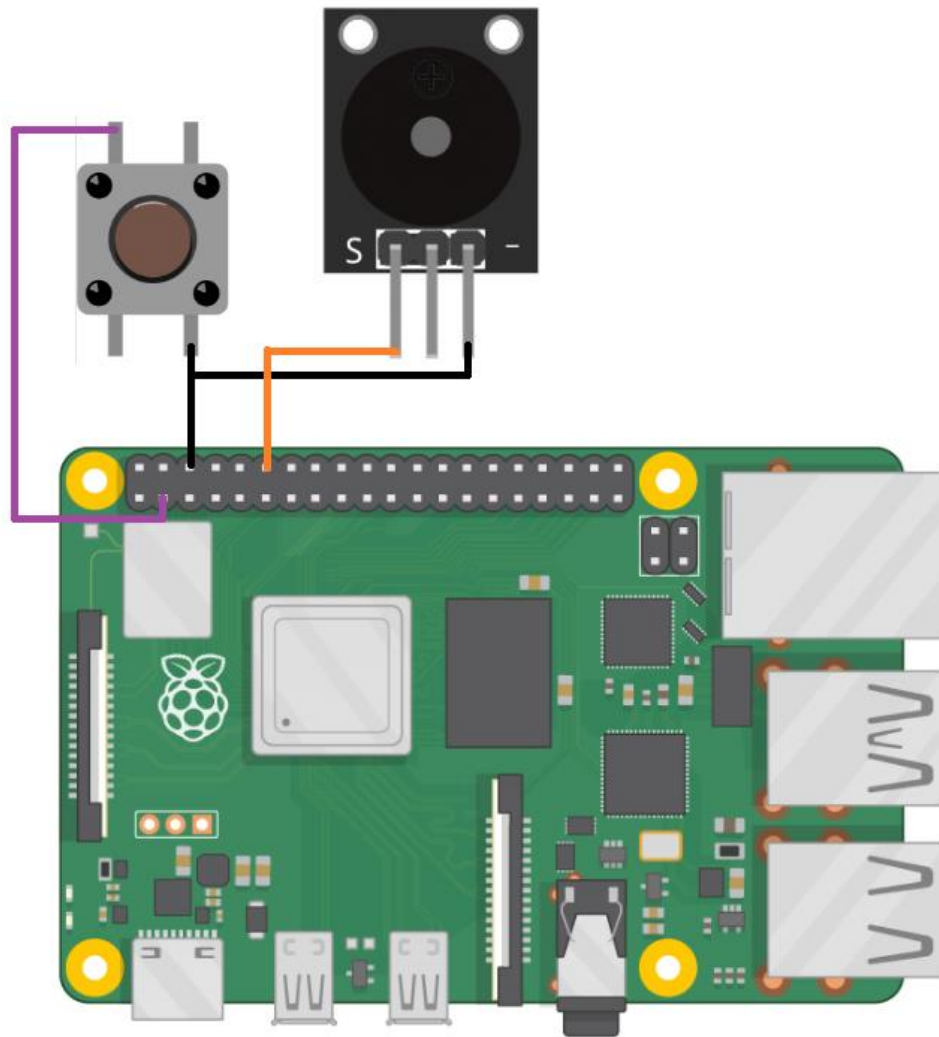
```
try:
    while True:
        mqttc.loop() # Blijf MQTT berichten verwerken
        check_button() # Controleer of de resetknop is ingedrukt
        if alarm_active:
            for freq in range(500, 900, 1):
                if not alarm_active:
                    break
                pwm.ChangeFrequency(freq) # Verander frequentie van de buzzer bij alarmactivering
                time.sleep(0.01)
            else:
                time.sleep(0.1) # Wacht tussen lussen
except KeyboardInterrupt:
    mqttc.disconnect() # Verbreek de verbinding met MQTT broker bij KeyboardInterrupt
    pwm.stop() # Zet de buzzer uit
    GPIO.cleanup() # Maak GPIO pinnen schoon
```

7. Volledig aansluitschema

7.1. ESP32



7.2. Raspberry Pi



8. Extra bronnen:

8.1. KY-026

<https://www.youtube.com/watch?v=wU7xZ9-qLI8>

8.2. KY-006

<https://www.thegeekpub.com/wiki/sensor-wiki-ky-006-passive-piezo-buzzer-module/>

8.3. Knop

<https://projects.raspberrypi.org/en/projects/physical-computing/5>

8.4. MQTT Arduino

<https://randomnerdtutorials.com/esp32-mqtt-publish-subscribe-arduino-ide/>

8.5. MQTT Raspberry Pi

<https://www.emqx.com/en/blog/how-to-use-mqtt-in-python>



CONTACT

Naam | Functie
xxx.xxx@thomasmore.be
Tel. + 32 xx xx xx xx

VOLG ONS

www.thomasmore.be
fb.com/ThomasMoreBE
#WeAreMore

THOMAS
MORE