THOMAS MORE

BASIS EMBEDDED SYSTEMS

Brievenbus openingsmelding

Ewout Doms

20XX-20XX

Inhoud

Inhoudsopgave

1. Beschrijving

Dit project integreert een ESP32 met een lichtafhankelijke weerstand (LDR) in een brievenbus. Wanneer de bovenkant van de brievenbus wordt geopend en er licht binnenvalt, detecteert de LDR dit en stuurt het een signaal via MQTT naar een Raspberry Pi. De Raspberry Pi activeert vervolgens een buzzer voor een auditieve melding en stuurt tegelijkertijd een bericht naar de Telegram-app, waardoor een melding op de smartphone wordt weergegeven. Bovendien wordt het aantal keren dat de brievenbus is geopend bijgehouden en weergegeven op zowel de Grafana-website als in de Telegram-app.

Om de brievenbus te openen en de post te verwijderen, is een knop aan de Raspberry Pi bevestigd. Wanneer deze knop wordt ingedrukt, stuurt de Raspberry Pi een signaal naar de ESP32, die op zijn beurt een servo-motor activeert om het slot te openen. Nadat de post is verwijderd, kan de knop opnieuw worden ingedrukt om het slot te sluiten. Deze actie reset ook de telling van het aantal keren dat de brievenbus is geopend, waardoor het systeem gereed is voor de volgende keer.

2. Hardwarde

2.1 ESP32

De ESP32 is een veelzijdige microcontrollerchip die zowel Wi-Fi als Bluetooth-functionaliteit biedt, waardoor het ideaal is voor IoT-toepassingen zoals huisautomatisering.

In dit project fungeert de ESP32 als het centrale zenuwstelsel. Het is uitgerust met een LDR (Light Dependent Resistor) die veranderingen in licht detecteert wanneer de brievenbus wordt geopend. Deze verandering wordt omgezet in een signaal dat via MQTT naar de Raspberry Pi wordt gestuurd, waardoor de verdere acties worden geactiveerd. Het stuurt dan ook een bericht naar de telegram app voor te laten weten hoe vaak de brievenbus is geopend. Het ontvangt ook signalen van de Raspberry Pi. Wanneer bijvoorbeeld de knop op de Raspberry Pi wordt ingedrukt, stuurt de ESP32 een signaal om de servo motor te activeren en het slot te openen of te sluiten.

2.2 Lichtafhankelijke weerstand (LDR)

De LDR is een sensor die de veranderingen in lichtniveaus detecteert wanneer de bovenkant van de brievenbus wordt geopend. Het is verbonden met de ESP32 en speelt een cruciale rol bij het initiëren van het detectieproces en het doorgeven van het signaal naar de Raspberry Pi.

2.3 Servo motor

De servo-motor is verantwoordelijk voor het openen en sluiten van het slot van de brievenbus. Wanneer de knop aan de Raspberry Pi wordt ingedrukt, stuurt de Raspberry Pi een signaal naar de ESP32, die de servo-motor activeert om het slot te openen. Een tweede druk op de knop zal het slot sluiten. Deze servo-motor actie wordt aangestuurd door de ESP32 en regelt fysiek de toegang tot de brievenbus.

2.4 Raspberry Pi:

De Raspberry Pi fungeert als de centrale hub van het systeem. Het ontvangt MQTT-berichten van de ESP32 en initieert vervolgens verschillende acties op basis van deze berichten. Dit omvat het activeren van de buzzer voor een auditieve melding en het bijhouden van het aantal keren dat de brievenbus is geopend weergegeven op de grafana-website. Net zoals het ontvangen van de signalen van de knop, stuurt het dan een melding naar de esp32 voor verdere acties.

2.5 Buzzer

De buzzer maakt een geluid die door de Raspberry Pi wordt geactiveerd wanneer de LDR, verbonden met de ESP32, een verandering in licht detecteert.

2.6 Knop:

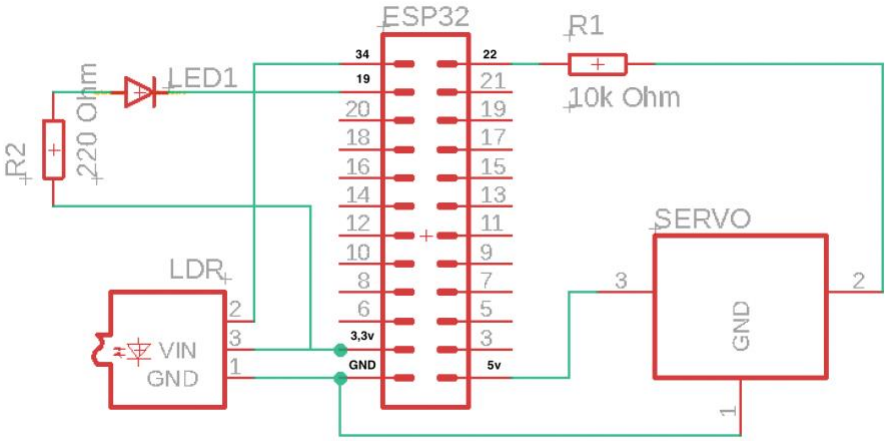
De knop is bevestigd aan de Raspberry Pi en wordt gebruikt om de brievenbus te openen en te sluiten. Wanneer de gebruiker de knop indrukt, stuurt de Raspberry Pi een signaal naar de ESP32, die vervolgens een servo-motor activeert om het slot van de brievenbus te openen. Na het verwijderen van de post kan de knop opnieuw worden ingedrukt om het slot te sluiten. Deze actie reset ook de telling van het aantal keren dat de brievenbus is geopend, waardoor het systeem klaar is voor de volgende keer.

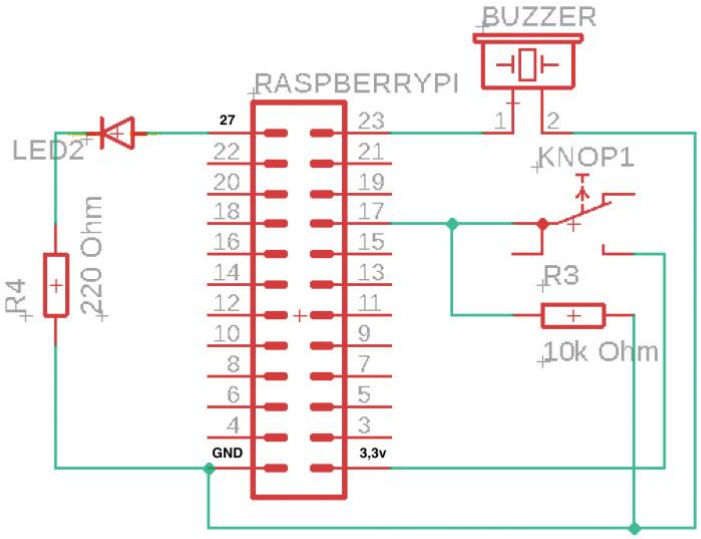
2.7 2 LED's:

Je hebt ook twee LED's: één verbonden met de Raspberry Pi en de andere met de ESP32. Ze lichten op wanneer de knop wordt ingedrukt om aan te geven dat het slot van de servo motor open staat. Zodra de knop opnieuw wordt ingedrukt, gaan de LED's uit en sluit het slot weer.

3. Schema

Dit aansluitschema geeft weer hoe alles verbonden is met elkaar en welke pinnen er gebruikt worden van de ESP32 en de Raspberry pi.





4. Codering

4.1 Code van esp32

- Deze code wacht tot de LDR een signaal geeft dat de brievenbus is geopend.
    - Stuurt naar de Raspberry Pi een melding voor verdere instructies
    - Stuurt een bericht naar de “Telegram” app
    - Telt op hoe vaak de klep is opengegaan.
- Wacht ook op melding van Raspberry Pi. Als het de juiste melding is, gaat het slot open of dicht en begint de telling opnieuw

WiFi- en MQTT-configuratie: Dit deel stelt de ESP32 in staat om verbinding te maken met een WiFi-netwerk en met een MQTT-broker om berichten te verzenden en ontvangen. Het bevat ook de inloggegevens voor de WiFi en MQTT-server, evenals de initialisatie van de WiFi-client en de PubSubClient voor MQTT-communicatie.

```
#include <WiFi.h>
#include <PubSubClient.h>
const char* ssid = "BandOfBrothers";
const char* password = "HOMEBARIBAL9";
const char* mqttServer = "192.168.0.242";
const int mqttPort = 1883;
const char* mqttUser = "EwoutDoms";
const char* mqttPassword = "SMOD3344";
WiFiClient espClient;
PubSubClient client(espClient);
```

Telegram-botconfiguratie: Hier wordt een Telegram-bot geconfigureerd voor het verzenden van meldingen naar een specifiek chat-ID. Het maakt gebruik van de UniversalTelegramBot-bibliotheek en stelt de bot in met het BOT-token en het chat-ID. Controleert elke seconde voor nieuwe berichten

```
#include <WiFiClientSecure.h>
#include <UniversalTelegramBot.h>
#define BOTtoken "7009974340:AAET5McU0Lr7IFZE-J-bs-U6rvgCctaCVi4"
#define CHAT_ID "7163672330"
WiFiClientSecure telegramclient;
UniversalTelegramBot bot(BOTtoken, telegramclient);
int botRequestDelay = 1000;
unsigned long lastTimeBotRan;
```

Servo-motorinitialisatie: Dit deel initialiseert de servo-motor, die wordt gebruikt om het slot van de brievenbus te openen en te sluiten.

```
#include <ESP32Servo.h>
Servo myservo;
```

Variabelen en pinneninitialisatie: Dit deel initialiseert variabelen en pinnen die worden gebruikt in de sketch, zoals de LDR-pin, LED-pin en servo-pin. Het stelt ook enkele variabelen in voor het bijhouden van het aantal keren dat de brievenbus is geopend.

```
const int ldr = 34;
const int led = 19;
const int servo = 22;
int y = 0;
int hoi = 1;
```

Seriële communicatie initialiseren:

```
void setup() {
  Serial.begin(115200);
```

Verbinding maken met Wi-Fi: De code stelt de ESP32 in om verbinding te maken met een Wi-Fi-netwerk met de opgegeven SSID en wachtwoord. Het wacht vervolgens tot de verbinding tot stand is gebracht voordat het doorgaat.

```
WiFi.mode(WIFI_STA);
WiFi.begin(ssid, password);
telegramclient.setCACert(TELEGRAM_CERTIFICATE_ROOT);
while (WiFi.status() != WL_CONNECTED) {
  delay(500);
  Serial.println("Zoeken naar wifi");
}
Serial.println("geconecteerd met wifi netwerk");
client.setServer(mqttServer, mqttPort);
```

Verbinding maken met de MQTT-server: Nadat de Wi-Fi-verbinding tot stand is gebracht, wordt er verbinding gemaakt met de MQTT-server met de opgegeven server, poort, gebruikersnaam en wachtwoord. Het wacht ook tot de verbinding tot stand is gebracht voordat het doorgaat.

```
while (!client.connected()) {
  Serial.println("connecteren met MQTT");
  if (client.connect("ESP32Client", mqttUser, mqttPassword)) {
    Serial.println("Geconnecteerd met MQTT");
  } else {
    Serial.print("Gefaald met connecteren met MQTT ");
    Serial.print(client.state());
    delay(2000);
  }
}
```

Bericht verzenden via Telegram: Nadat zowel de Wi-Fi- als de MQTT-verbinding zijn gemaakt, wordt er een bericht verzonden via de Telegram-bot om aan te geven dat het systeem gereed is om te beginnen.

```
// Print ESP32 Local IP Address
Serial.println(WiFi.localIP());
bot.sendMessage(CHAT_ID, "We kunnen beginnen", "");
```

MQTT-client configureren: De MQTT-client wordt geconfigureerd met een callback-functie die wordt aangeroepen wanneer een bericht wordt ontvangen op het onderwerp "brievenbus/slot". Het systeem abonneert zich ook op dit onderwerp om berichten te ontvangen.

```
client.setCallback(callback);
client.subscribe("brievenbus/slot");
```

Pin-modus instellen + Servo-initialisatie: De pinnen voor de LDR (lichtafhankelijke weerstand), LED en servo worden ingesteld op respectievelijk invoer, uitvoer en servo-uitgang. Dit initialiseert de pinnen voor het gebruik in de rest van het programma. De servo motor wordt geïnitialiseerd voor gebruik in het programma, waarbij de beginpositie op 0 graden wordt ingesteld.

```
pinMode(ldr, INPUT);
pinMode(led, OUTPUT);
myservo.attach(servo);
myservo.write(0);
```

Client.loop(): Word aangeroepen om de MQTT-client te laten luisteren naar inkomende berichten en eventuele uitgaande berichten te verzenden.

```
void loop() {
  client.loop();
```

Analoge waarde lezen: De analoge waarde van de LDR (lichtafhankelijke weerstand) wordt gelezen met `analogRead(ldr)` en opgeslagen in de variabele `ldrwaarde`. Deze waarde geeft de intensiteit van het licht weer die de LDR detecteert.

```
int ldrwaarde = analogRead(ldr);
Serial.print("ldr waarde: ");
Serial.println(ldrwaarde);
```

Brievenbusstatus controleren: Als de gemeten lichtintensiteit (`ldrwaarde`) onder of gelijk is aan 500, wordt de brievenbus als geopend beschouwd. Er wordt een teller verhoogd, een bericht verzonden via de Telegram-bot, een MQTT-bericht gepubliceerd en relevante informatie naar de seriële monitor afgedrukt.

```
if (ldrwaarde <= 500) {
  y++;
  delay(10000);
  bot.sendMessage(CHAT_ID, "Brievenbus " + String(y) + " keer geopend", "");
  Serial.println("Brievenbus " + String(y) + " keer geopend");
  String ldrtekst = String(y);
  client.publish("brievenbus/huis/open", ldrtekst.c_str());
  Serial.println("Brievenbus geopend");
}
```

Brievenbusstatus controleren: Als de gemeten lichtintensiteit (`ldrwaarde`) boven 500 is, wordt de brievenbus als gesloten beschouwd. Een MQTT-bericht wordt gepubliceerd en relevante informatie wordt afgedrukt naar de seriële monitor.

```
if (ldrwaarde > 500) {
  String ldrtekst = String(0);
  client.publish("brievenbus/huis/dicht", ldrtekst.c_str());
  Serial.println("Brievenbus gesloten");
}
```

Er wordt een korte pauze van 2 seconden ingelast met `delay(2000)` om niet de hele tijd te moeten runnen.

```
  Serial.println(" ");
  delay(2000);
}
```

Functiedefinitie: Dit is de definitie van de functie `callback`, die wordt aangeroepen wanneer een MQTT-bericht wordt ontvangen.
Variabelen initialiseren: Een lege string `bericht` wordt gemaakt om het ontvangen bericht op te slaan.

```
void callback(char* topic, byte* payload, unsigned int length) {
  String bericht = "";
```

Controleren op het onderwerp: Het onderwerp van het ontvangen MQTT-bericht wordt gecontroleerd. Als het onderwerp overeenkomt met "brievenbus/slot", wordt de rest van de code uitgevoerd.

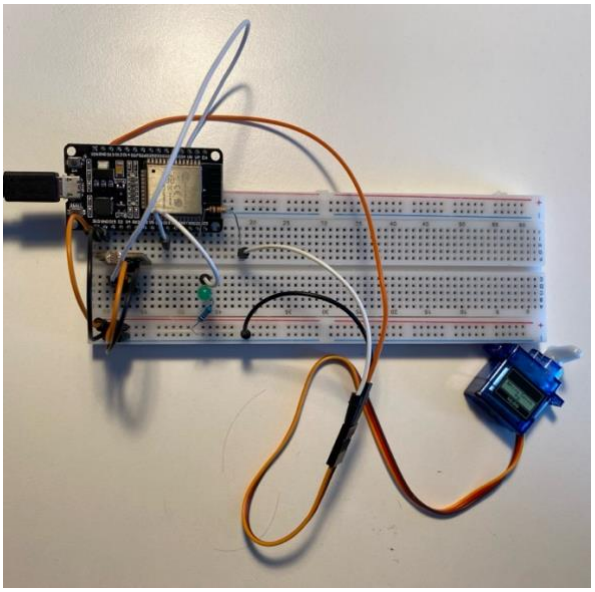
```
if (String(topic) == "brievenbus/slot") {
  for (int i = 0; i < length; i++) {
    bericht += (char)payload[i];
  }
```

Instructie verwerken: De ontvangen instructie wordt gecontroleerd. Als de instructie "Servo 180" is, wordt de servo naar de open positie (180 graden) gedraaid, wordt het LED-lampje ingeschakeld en wordt een bericht afgedrukt naar de seriële monitor.

```
if (bericht == "Servo 180") {
  myservo.write(180);
  digitalWrite(led, HIGH);
  Serial.println("Slot: open");
```

Andere instructie verwerken: Als de instructie "Servo 0" is, wordt de servo naar de gesloten positie (0 graden) gedraaid, wordt het LED-lampje uitgeschakeld, wordt een herstart van de teller (`y`) uitgevoerd en wordt een bericht verzonden via de Telegram-bot en MQTT.

```
} else if (bericht == "Servo 0") {
  myservo.write(0);
  digitalWrite(led, LOW);
  Serial.println("Slot: gesloten");

  y = 0;
  bot.sendMessage(CHAT_ID, "Herstart telling van openen brievenbus: " +
  String(y), "");
  Serial.println("Herstart telling van openen brievenbus: " + String(y));

  String ldrtekst = String(0);
  client.publish("brievenbus/huis/open", ldrtekst.c_str());
}
```



```
#include <WiFi.h>
#include <PubSubClient.h>
const char* ssid = "BandOfBrothers";
const char* password = "HOMEBARIBAL9";
const char* mqttServer = "192.168.0.242";
const int mqttPort = 1883;
const char* mqttUser = "EwoutDoms";
const char* mqttPassword = "SMOD3344";
WiFiClient espClient;
PubSubClient client(espClient);

//////////////////////////////////////////////////

#include <WiFiClientSecure.h>
#include <UniversalTelegramBot.h>
#define BOTtoken "7009974340:AAET5McU0Lr7IFZE-J-bs-U6rvgCctaCVi4"
#define CHAT_ID "7163672330"
WiFiClientSecure telegramclient;
UniversalTelegramBot bot(BOTtoken, telegramclient);
int botRequestDelay = 1000;
unsigned long lastTimeBotRan;

//////////////////////////////////////////////////

#include <ESP32Servo.h>
Servo myservo;

//////////////////////////////////////////////////

const int ldr = 34;
const int led = 19;
const int servo = 22;
int y = 0;
int hoi = 1;


//------------------------------------------------------------
void setup() {
  Serial.begin(115200);

  //////////////////////////////////////////////////
```

```
  WiFi.mode(WIFI_STA);
  WiFi.begin(ssid, password);
  telegramclient.setCACert(TELEGRAM_CERTIFICATE_ROOT);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.println("Zoeken naar wifi");
  }
  Serial.println("geconecteerd met wifi netwerk");
  client.setServer(mqttServer, mqttPort);

  while (!client.connected()) {
    Serial.println("connecteren met MQTT");
    if (client.connect("ESP32Client", mqttUser, mqttPassword)) {
      Serial.println("Geconnecteerd met MQTT");
    } else {
      Serial.print("Gefaald met connecteren met MQTT ");
      Serial.print(client.state());
      delay(2000);
    }
  }

  // Print ESP32 Local IP Address
  Serial.println(WiFi.localIP());
  bot.sendMessage(CHAT_ID, "We kunnen beginnen", "");

  ////////////////////////////////////////////////

  client.setCallback(callback);
  client.subscribe("brievenbus/slot");

  ////////////////////////////////////////////////

  pinMode(ldr, INPUT);
  pinMode(led, OUTPUT);
  myservo.attach(servo);
  myservo.write(0);
}


//--------------------------------------------------------------
void loop() {
  client.loop();
```

```
  ////////////////////////////////////////////////////

  int ldrwaarde = analogRead(ldr);
  Serial.print("ldr waarde: ");
  Serial.println(ldrwaarde);

  if (ldrwaarde <= 500) {
    y++;
    delay(10000);
    bot.sendMessage(CHAT_ID, "Brievenbus " + String(y) + " keer geopend", "");
    Serial.println("Brievenbus " + String(y) + " keer geopend");
    String ldrtekst = String(y);
    client.publish("brievenbus/huis/open", ldrtekst.c_str());
    Serial.println("Brievenbus geopend");
  }

  if (ldrwaarde > 500) {
    String ldrtekst = String(0);
    client.publish("brievenbus/huis/dicht", ldrtekst.c_str());
    Serial.println("Brievenbus gesloten");
  }

  ////////////////////////////////////////////////////

  Serial.println(" ");
  delay(2000);
}


//////////////////////////////////////////////////////

void callback(char* topic, byte* payload, unsigned int length) {
  String bericht = "";

  if (String(topic) == "brievenbus/slot") {
    for (int i = 0; i < length; i++) {
      bericht += (char)payload[i];
    }
    if (bericht == "Servo 180") {
      myservo.write(180);
      digitalWrite(led, HIGH);
      Serial.println("Slot: open");
```

```
    } else if (bericht == "Servo 0") {
      myservo.write(0);
      digitalWrite(led, LOW);
      Serial.println("Slot: gesloten");

      y = 0;
      bot.sendMessage(CHAT_ID, "Herstart telling van openen brievenbus: " + String(y), "");
      Serial.println("Herstart telling van openen brievenbus: " + String(y));

      String ldrtekst = String(0);
      client.publish("brievenbus/huis/open", ldrtekst.c_str());
    }
  }
}
```

4.2 1ste code van raspberry pi

- Wacht op melding van ESP32. Als het een melding krijgt, laat het een buzzer afgaan.
- Wacht op signaal van knop, stuurt dan melding naar ESP32 voor verdere instructies.

Importeer bibliotheken: Begint met het importeren van de nodige bibliotheken zoals RPi.GPIO, gpiozero, paho.mqtt.client, time en subprocess om GPIO-pinnen te bedienen, MQTT-communicatie mogelijk te maken, en externe processen uit te voeren.

```
import RPi.GPIO as GPIO
from gpiozero import Buzzer, Button, LED
import paho.mqtt.client as mqtt
import time
from time import sleep
import subprocess  # Voor "servo 180" te versturen
```

2. **GPIO-initialisatie**: GPIO-pinnen worden geïnitialiseerd voor het aansturen van de zoemer, knop en LED.

```
buzzer = Buzzer(23) # (= GPIO 23, ...)
knop1 = Button(17,pull_up=0)
toggle = True
led = LED(27)
```

MQTT-configuratie: De variabelen voor de MQTT-brokerconfiguratie worden ingesteld, waaronder het IP-adres van de broker, poortnummer, gebruikersnaam en wachtwoord.

```
mqtt_broker = "192.168.0.242"
mqtt_port = 1883
mqtt_user = "EwoutDoms"
mqtt_password = "SMOD3344"
mqtt_topic_prefix = "brievenbus/"
mqtt_topic = mqtt_topic_prefix + "button_press"
```

Connectie- en berichthandlers: Functies worden gedefinieerd om te reageren op MQTT-connectiegebeurtenissen en inkomende berichten van de broker. Als de brievenbus wordt geopend, wordt de zoemer ingeschakeld en vice versa.

```
def on_connect(client, userdata, flags, rc):
        print("Geconnecteerd ")
        client.subscribe([(mqtt_topic_prefix + "huis/open", 0),
                (mqtt_topic_prefix + "huis/dicht", 0),])

def on_message(client, userdata, msg):
        topic = msg.topic
        if topic.endswith("huis/open"):
                buzzer.on()
                print('Brievenbus geopend!!!')
        elif topic.endswith("huis/dicht"):
                buzzer.off()
                print('Brievenbus gesloten')
```

Knopdrukhendelaar: Een callback-functie wordt gedefinieerd om te reageren wanneer de knop wordt ingedrukt. Afhankelijk van de huidige toestand wordt een MQTT-bericht verzonden om een servo te activeren om de brievenbusslot te openen of te sluiten.

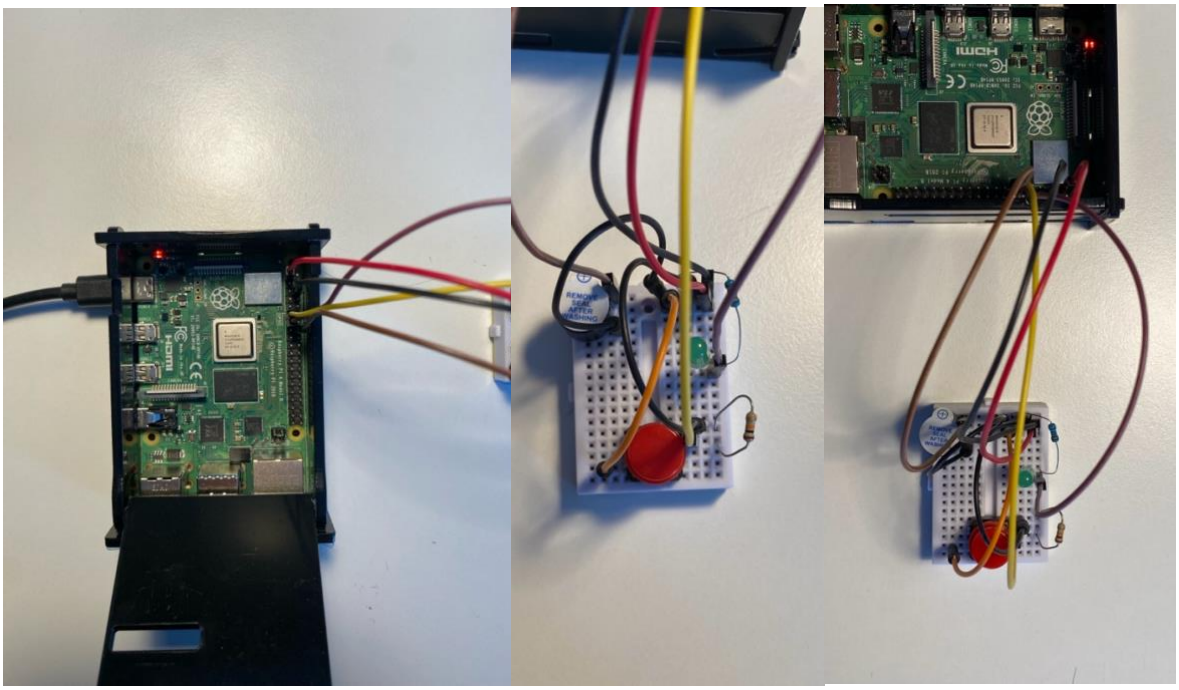
```
def knop1_callback():
        global toggle
        if knop1.is_active == True and toggle == True:
                print("knop is ingedrukt (Slot open)")
                subprocess.run(["mosquitto_pub", "-h", "localhost", "-t", "brievenbus/slot", "-m", "Servo 180", "-u", "EwoutDoms", "-P", "SMOD3344"])
                print("Servo 180")
                toggle = False
                led.on()
                sleep(1)

        if knop1.is_active == True and toggle == False:
                print("knop is ingedrukt (Slot gesloten)")
                subprocess.run(["mosquitto_pub", "-h", "localhost", "-t", "brievenbus/slot", "-m", "Servo 0", "-u", "EwoutDoms", "-P", "SMOD3344"])
                print("Servo 0")
                toggle = True
                led.off()
                sleep(1)
```

MQTT-clientconfiguratie: Een MQTT-client wordt geïnstantieerd met gebruikersnaam en wachtwoord, en de connectie- en berichthandelaars worden ingesteld.

Knopdrukactivering: Wanneer de knop wordt ingedrukt, wordt de knopdrukhendelaar geactiveerd om het juiste MQTT-bericht te verzenden en de acties uit te voeren.

Oneindige lus voor MQTT-berichten: De code wordt in een oneindige lus uitgevoerd om continu te luisteren naar inkomende MQTT-berichten en knopdrukken.

```
client = mqtt.Client(mqtt.CallbackAPIVersion.VERSION1)
client.username_pw_set(mqtt_user, mqtt_password)
client.on_connect = on_connect
client.on_message = on_message
client.connect(mqtt_broker, mqtt_port, 60)

knop1.when_pressed = knop1_callback
client.loop_forever()
```



```
import RPi.GPIO as GPIO
from gpiozero import Buzzer, Button, LED
import paho.mqtt.client as mqtt
import time
from time import sleep
import subprocess  # Voor "servo 180" te versturen

buzzer = Buzzer(23) # (= GPIO 23, ...)
knop1 = Button(17,pull_up=0)
toggle = True
led = LED(27)

mqtt_broker = "192.168.0.242"
mqtt_port = 1883
mqtt_user = "EwoutDoms"
mqtt_password = "SMOD3344"
mqtt_topic_prefix = "brievenbus/"
mqtt_topic = mqtt_topic_prefix + "button_press"

def on_connect(client, userdata, flags, rc):
    print("Geconnecteerd ")
    client.subscribe([(mqtt_topic_prefix + "huis/open", 0),
        (mqtt_topic_prefix + "huis/dicht", 0),])

def on_message(client, userdata, msg):
    topic = msg.topic
    if topic.endswith("huis/open"):
        buzzer.on()
        print('Brievenbus geopend!!!')
    elif topic.endswith("huis/dicht"):
        buzzer.off()
        print('Brievenbus gesloten')
```

```
def knop1_callback():
    global toggle
    if knop1.is_active == True and toggle == True:
        print("knop is ingedrukt (Slot open)")
        subprocess.run(["mosquitto_pub", "-h", "localhost", "-t", "brievenbus/slot", "-m", "Servo 180", "-u", "EwoutDoms", "-P", "SMOD3344"])
        print("Servo 180")
        toggle = False
        led.on()
        sleep(1)

    if knop1.is_active == True and toggle == False:
        print("knop is ingedrukt (Slot gesloten)")
        subprocess.run(["mosquitto_pub", "-h", "localhost", "-t", "brievenbus/slot", "-m", "Servo 0", "-u", "EwoutDoms", "-P", "SMOD3344"])
        print("Servo 0")
        toggle = True
        led.off()
        sleep(1)

client = mqtt.Client(mqtt.CallbackAPIVersion.VERSION1)
client.username_pw_set(mqtt_user, mqtt_password)
client.on_connect = on_connect
client.on_message = on_message
client.connect(mqtt_broker, mqtt_port, 60)

knop1.when_pressed = knop1_callback
client.loop_forever()
```

4.3 2de code van raspberry pi

- Deze code geeft weer hoeveel keer de brievenbus is geopend op de “Grafana-website”.
- Hoe op grafana-website geraken:
    - Ga naar google/safari
    - Schrijf in zoekbalk:
        - “ip van raspberry pi”:3000

Imports: Importeer de vereiste bibliotheken, zoals re voor reguliere expressies, NamedTuple voor het maken van een op maat gemaakte gegevenstypen, paho.mqtt.client voor MQTT-communicatie, en InfluxDBClient voor toegang tot de InfluxDB-database.

```
import re
from typing import NamedTuple
import paho.mqtt.client as mqtt
from influxdb import InfluxDBClient
```

Configuratievariabelen: Stel variabelen in voor de adresgegevens en referenties van de MQTT-broker en InfluxDB-database.

InfluxDB-clientinitialisatie: Initialiseer een client voor de InfluxDB-databaseverbinding met behulp van de verstrekte configuratiegegevens.

```
INFLUXDB_ADDRESS = '192.168.0.242'
INFLUXDB_USER = 'Ewout1'
INFLUXDB_PASSWORD = 'SMOD3344a'
INFLUXDB_DATABASE = 'EINDPRO'
MQTT_ADDRESS = '192.168.0.242'
MQTT_USER = 'EwoutDoms'
MQTT_PASSWORD = 'SMOD3344'
MQTT_TOPIC = 'brievenbus/+/+'
MQTT_REGEX = 'brievenbus/([^/]+)/([^/]+)'
MQTT_CLIENT_ID = 'MQTTInfluxDBBridge'
influxdb_client = InfluxDBClient(INFLUXDB_ADDRESS, 8086, INFLUXDB_USER,
INFLUXDB_PASSWORD, None)
```

SensorData-gegevenstype: Definieer een aangepast gegevenstype SensorData met locatie, meting en waarde als attributen.

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

Verbindingscallback voor MQTT: Definieer een functie on_connect om op te roepen wanneer de MQTT-client verbinding maakt met de broker. Abonneer de client op het MQTT-onderwerp.

```
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)
```

Analyse van MQTT-berichten: Implementeer een functie _parse_mqtt_message om MQTT-berichten te analyseren op basis van een reguliere expressie en de gegevens te extraheren.

```
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, float(payload))
        else:
                return None
```

Verzend gegevens naar InfluxDB: Definieer een functie _send_sensor_data_to_influxdb om de geanalyseerde sensorgegevens naar de InfluxDB-database te schrijven.

```
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)
```

Berichtcallback voor MQTT: Definieer een functie on_message om op te roepen wanneer een PUBLISH-bericht van de server wordt ontvangen. Parse het bericht en verzend de gegevens naar InfluxDB indien geldig.

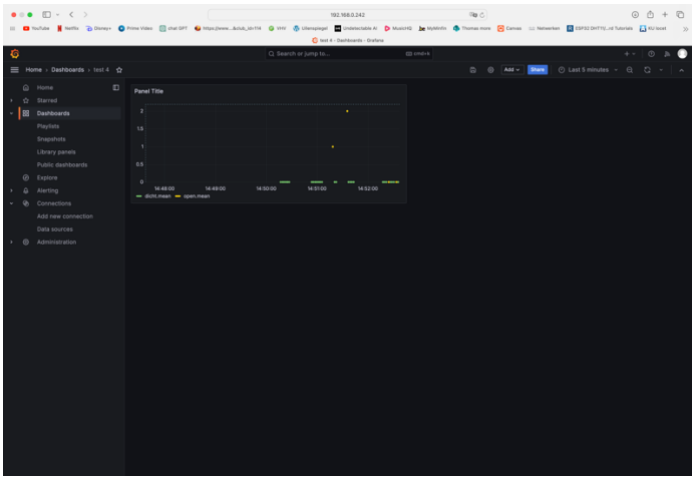
```
def on_message(client, userdata, msg):
        """The callback for when a PUBLISH message is received from the server."""
        print(msg.topic + ' ' + str(msg.payload))
        sensor_data = _parse_mqtt_message(msg.topic, msg.payload.decode('utf-8'))
        if sensor_data is not None:
      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

Initialisatie van InfluxDB-database: Controleer of de database bestaat en maak deze aan indien nodig.

```
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)
```

Hoofdfunctie: Initialiseer de InfluxDB-database, configureer en start de MQTT-client, en start de oneindige lus om MQTT-berichten te verwerken.

```
def main():
        _init_influxdb_database()

        mqtt_client = mqtt.Client(mqtt.CallbackAPIVersion.VERSION1, 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()
```

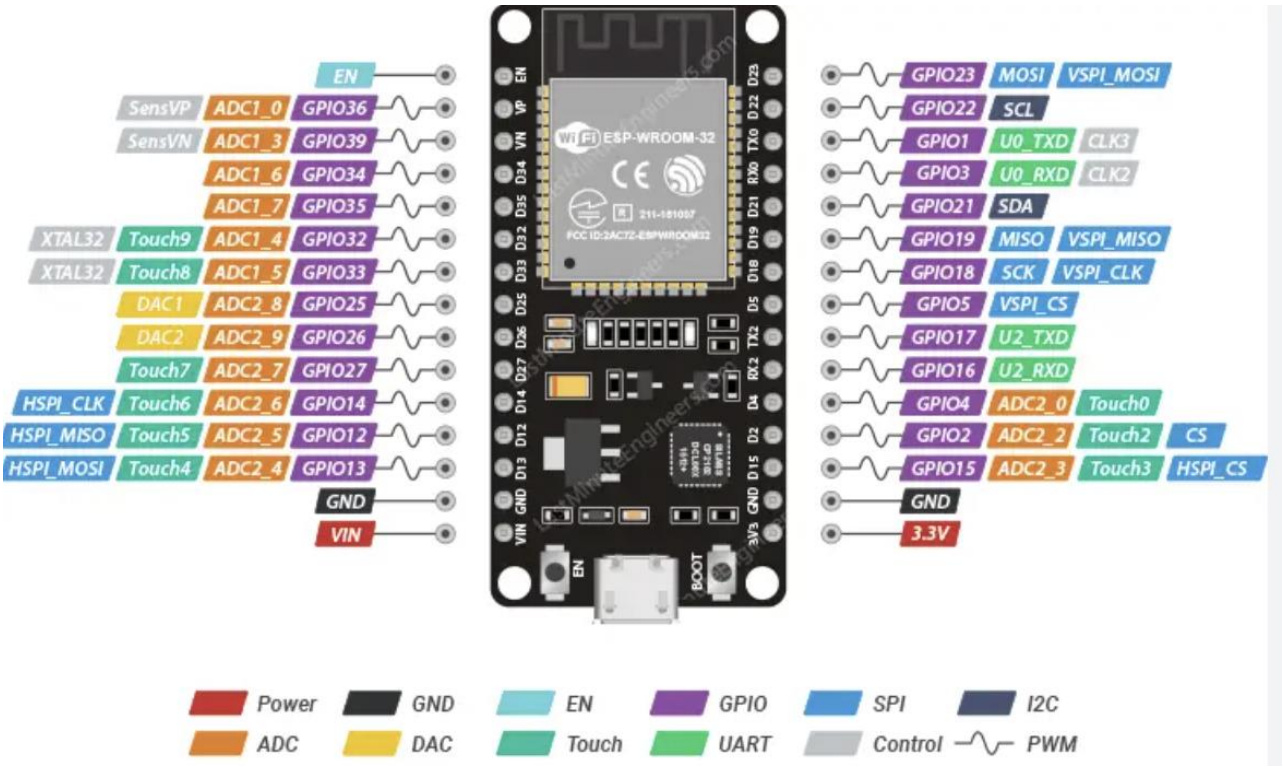


```
import re
from typing import NamedTuple
import paho.mqtt.client as mqtt
from influxdb import InfluxDBClient
INFLUXDB_ADDRESS = '192.168.0.242'
INFLUXDB_USER = 'Ewout1'
INFLUXDB_PASSWORD = 'SMOD3344a'
INFLUXDB_DATABASE = 'EINDPRO'
MQTT_ADDRESS = '192.168.0.242'
MQTT_USER = 'EwoutDoms'
MQTT_PASSWORD = 'SMOD3344'
MQTT_TOPIC = 'brievenbus/+/+'
MQTT_REGEX = 'brievenbus/([^/]+)/([^/]+)'
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, float(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.CallbackAPIVersion.VERSION1, 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()
```

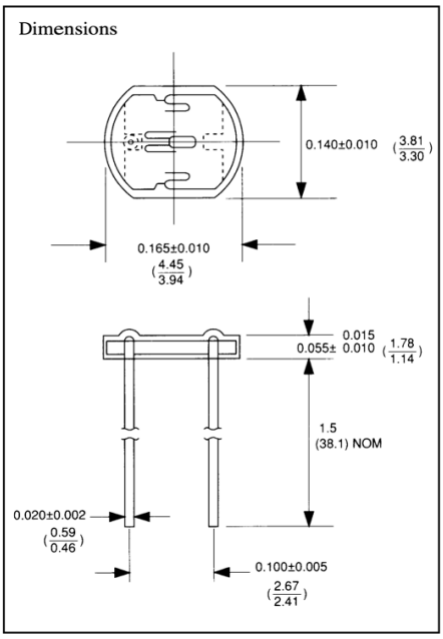
5. Datasheets

5.1 ESP32



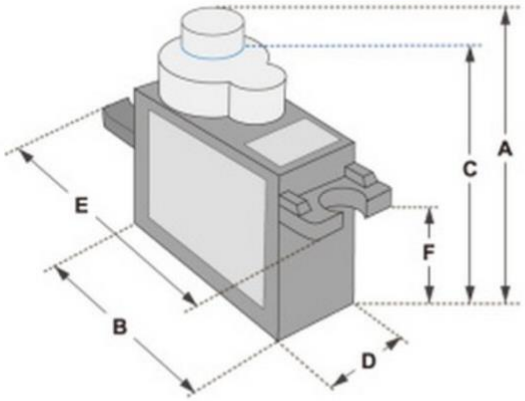
Meer info: https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf

5.2 LDR



Meer info:
https://components101.com/sites/default/files/component_datasheet/LDR%20Datasheet.pdf

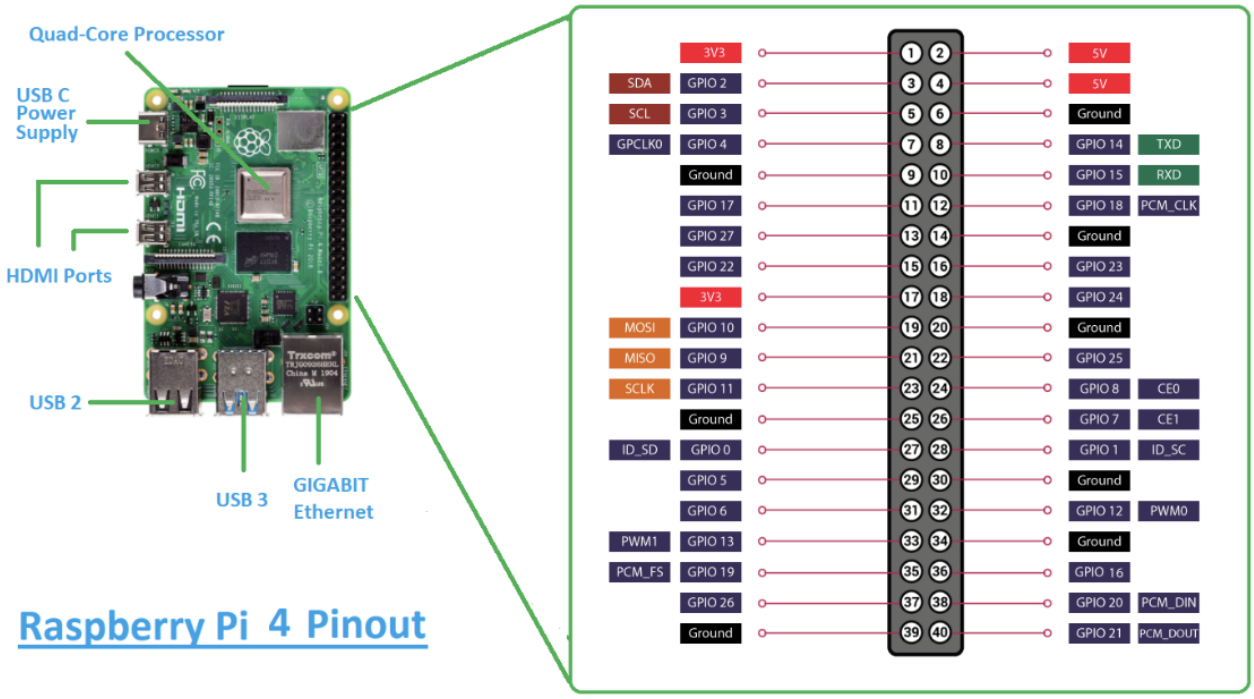
5.3 Servo motor



Dimensions & Specifications
A (mm) : 32
B (mm) : 23
C (mm) : 28.5
D (mm) : 12
E (mm) : 32
F (mm) : 19.5
Speed (sec) : 0.1
Torque (kg-cm) : 2.5
Weight (g) : 14.7
Voltage : 4.8 - 6

Meer info: http://www.ee.ic.ac.uk/pcheung/teaching/DE1_EE/stores/sg90_datasheet.pdf

5.4 Raspberry pi

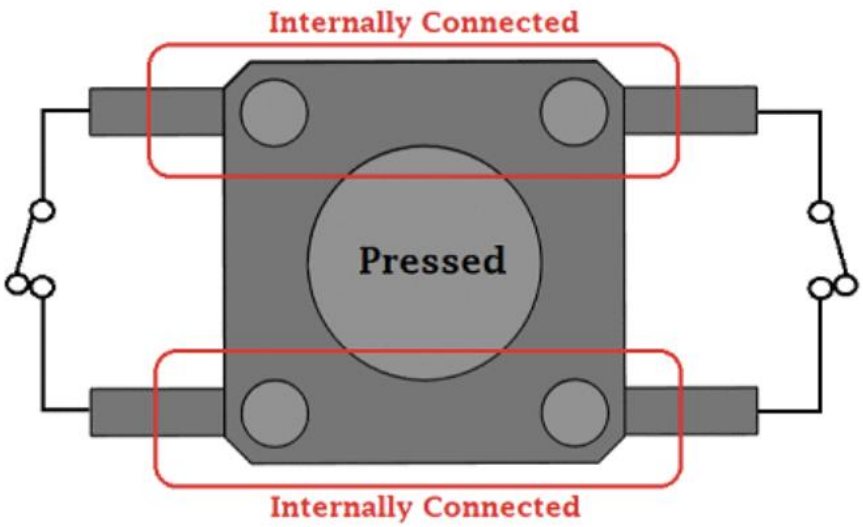


Meer info: https://www.theengineeringprojects.com/2021/03/what-is-raspberry-pi-4-pinout-specs-projects-datasheet.html

5.5 Buzzer

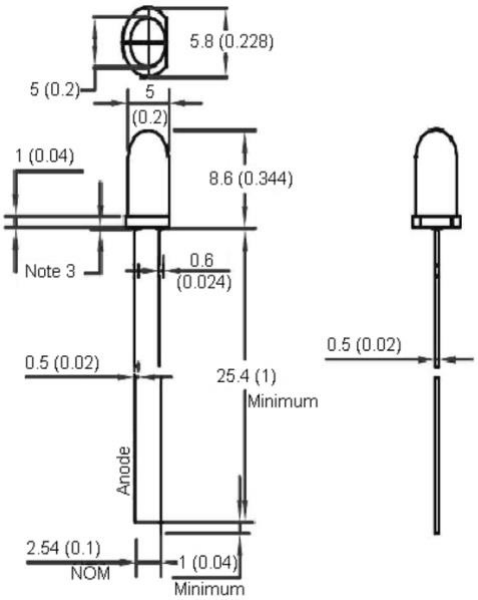
Meer info: https://components101.com/misc/buzzer-pinout-working-datasheet

5.6 Knop



Meer info: https://components101.com/switches/push-button

5.7 LED



Meer info: https://www.farnell.com/datasheets/1498852.pdf