Opdracht MQTT Weerstation

Auteur: Luuk Elkink
Studentnummer: 1642820

Deze code heeft de volgende functionaliteiten:

- Communicatie i2c sensor
- Wifi verbinding
- MQTT verbinding
- Windsnelheid/anemometer

Ten eerste is het belangrijk te noemen dat als extra deel voor deze opdracht is gekozen het ESP-IDF framework te gebruiken in plaats van die van Arduino. Deze geeft meer controlle over specifieke aspecten van de esp, maar is hierdoor iets complexer. Platform-IO kan dit gewoon compileren en debuggen zonder wijziging van instellingen (deze zijn al gedefinieerd in het platformio.ini bestand.

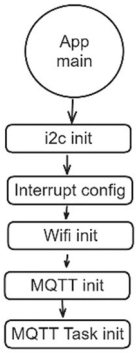
De functionaliteiten worden in het volgende deel individueel besproken. In de code zijn bepaalde delen gedetailleerder uitgelegd. De code is geschreven met de hulp van de ESP-IDF documentatie en de voorbeelden hierin: https://docs.espressif.com/projects/esp-idf/en/stable/esp32/

App_Main()

Dit is de code die runt wanneer het systeem opstart, hier worden alle functionaliteiten geinitialiseerd.

In Figuur 1, Main staat het ruwe proces afgebeeld dat wordt doorlopen bij de initialisatie.

- De i2c bus connectie wordt gemaakt.
- De interrupt instellingen en event queue wordt geinitialiseerd
- De Wifi routine en verbinding wordt gemaakt
- De MQTT routine wordt geinitialiseerd
- De MQTT data verstuur taak wordt geinitialiseerd



Figuur 1, Main

Communicatie i2c sensor

De i2c sensor verbinding wordt gestart met de functie i2c_master_init(). Deze functie maakt gebruik van de i2c_master.h bibliotheek.

Als eerste worden de instellingen van de master bus aangemaakt in de bus_config variabele van het type i2c_master_bus_config_t (struct uit library). Hierin worden de SDA en SCL pins gedefinieerd (in deze code SDA: 21 en SCL: 22), het poortnummer gekozen, de pullup weerstand geactiveerd en meer.

Vervolgens wordt met de instellingen een master bus lijn aangemaakt met i2c_new_master_bus().

Nu moet een connectie worden geregistreerd bij de master bus. Eerst worden de instellingen van de connectie gemaakt en opgeslagen in de variabele dev_cfg van type i2c_device_config_t. Deze bestaat uit de adres lengte, het adres en verbindingssnelheid van de sensor (worden uit datasheet gehaald). Met de instellingen wordt een device op de main bus aangemaakt met i2c_master_bus_add_device(....).

De aangemaakte master bus en device worden opgeslagen in een statisch gealloceerde struct (i2c_zuyd_sensor) zodat deze altijd en overal beschikbaar zijn in het programma. *Wanneer het aantal sensoren onbekend is bij opstarten, of geheugen dynamisch gealloceerd moet worden per sensor zal dit op een andere manier moeten worden geimplementeerd.*

De sensor wordt uitgelezen met de read_sensor_data(*i2c_device device) functie. Deze vereist een pointer naar een i2c device (de eerdergenoemde struct met de bus en device informatie), Hiervoor is gekozen zodat als meerdere sensoren worden aangesloten, dezelfde functie aangeroepen kan worden.

De gemeten data wordt opgeslagen in wat statisch gealloceerd geheugen in een struct, bij iedere meting wordt over de vorige waarde heen geschrven.

De read_sensor_data() functie creert wat geheugen voor het adres waaruit gelezen wordt (uint8_t cmd[1];) hierin wordt het adres geladen voor de temperatuur meting:

```
uint8_t cmd[1];
cmd[0] = I2C_SENSOR_TEMP_REG;
```

Dit adres wordt met een schrijfactie naar de sensor gestuurd om aan te geven dat deze meting gelezen moet worden. Vervolgens wordt kort gewacht zodat de sensor de meting kan doen:

```
ESP_ERROR_CHECK(i2c_master_transmit(device->dev_handle, cmd, sizeof(cmd), -1));
vTaskDelay(pdMS_TO_TICKS(20));
```

twee bytes aan geheugen worden op de stack gealloceerd. Nu wordt geluisterd naar een reactie van de sensor, en wordt de reactie in dit geheugen opgeslagen:

```
uint8_t raw_data[2];
ESP_ERROR_CHECK(i2c_master_receive(device->dev_handle, raw_data, sizeof(raw_data), -1));
```

De twee bytes worden gecombineerd door de eerste 8 bits naar links te verschuiven en vast te plakken aan de tweede byte. De resulterende waarde wordt gebruikt in de temperatuur calculatie uit de sensor datasheet en opgeslagen:

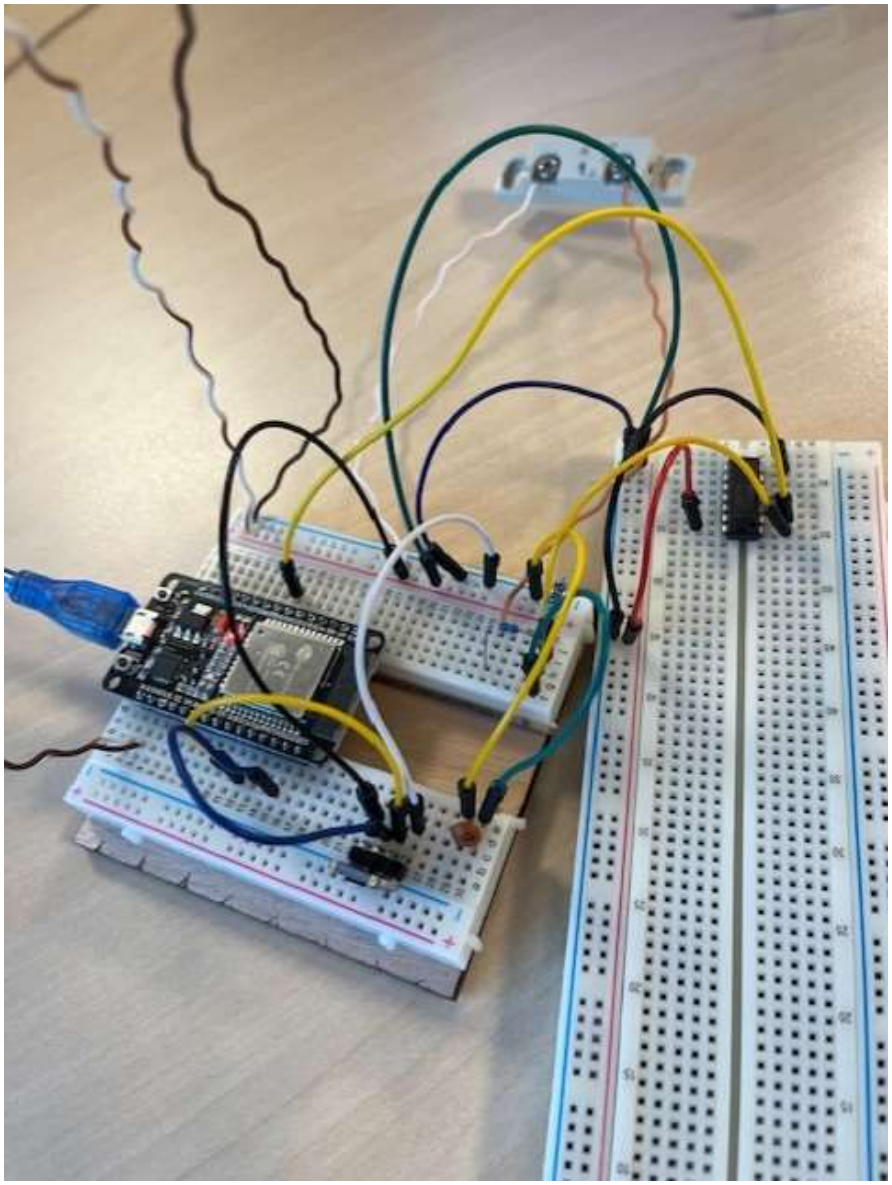
```
        double temp = (double)((raw_data[0] << 8) | raw_data[1]);
        //ESP_LOGI(TAG, "Temperature raw: %.2f", temp);
        temp = (temp * 175.72 / 65536) - 46.85;
        ESP_LOGI(TAG, "Temperature: %.2f", temp);
        sensor_data.temp = temp;
```

De luchtvochtigheid wordt op dezelfde manier verkregen direct hierna en wordt ook opgeslagen in het geheugen.

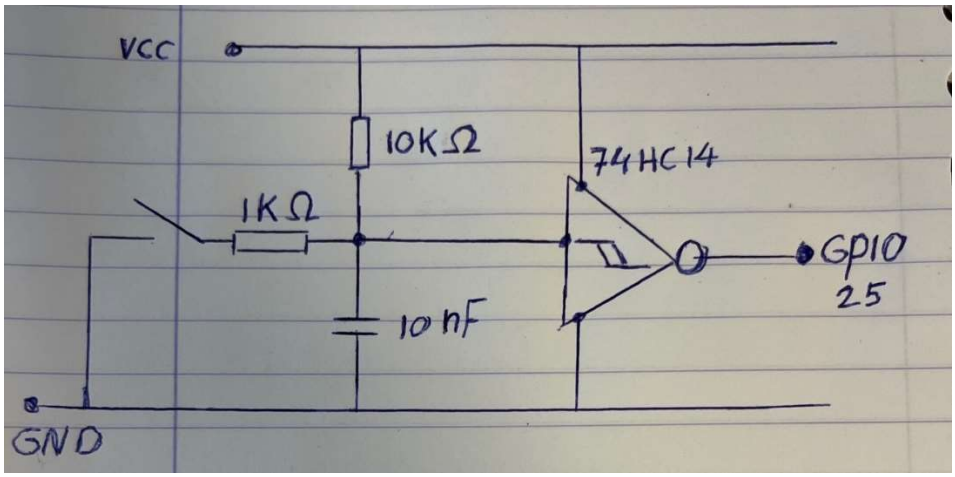
Interrupt wind snelheid

De windsnelheid wordt gemeten door de snelheid van pulsen te gebruiken met een interrupt routine. Eerst zal even de schakeling besproken worden die dit mogelijk maakt.

Schakeling

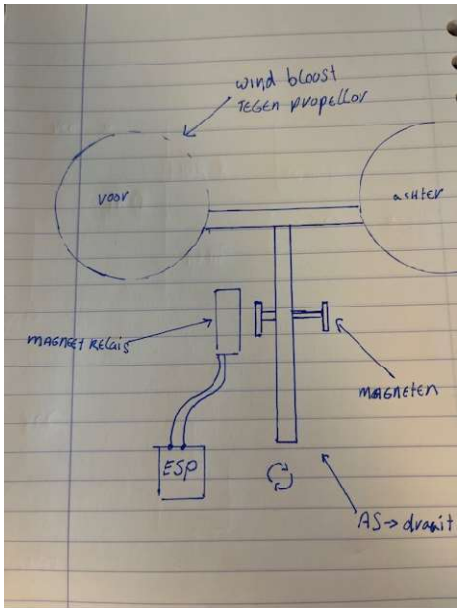


Figuur 2, Foto schakeling



Figuur 3, Schema schakeling

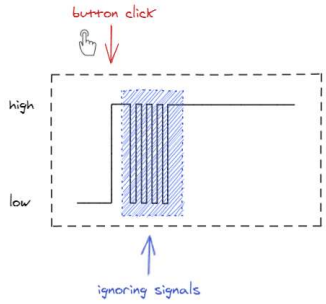
In Figuur 2 en Figuur 3 staat de schakeling afgebeeld die gebruikt wordt voor de windsnelheidsmeting. De windsnelheid zal bepaald worden door het meten hoe vaak de magneet wordt gedetecteerd. Dit geeft een snelheidsmeting van de as, deze kan omgerekend worden naar windsnelheid. Figuur 4 weergeeft een schets van het concept:



Figuur 4, schets concept

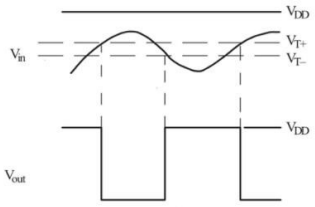
De schakeling bestaat uit een magneet relais (normally open) dat schakelt wanneer een magnetisch veld aanwezig is. Het relais was origineel alleen verbonden met een pullup weerstand, zo werd gemeten wanneer kortgesloten werd naar aarde en dus de pin van hoog naar laag ging. Dit introduceerde een probleem; De meting is zo snel dat bij iedere schakeling meerdere keren een signaal wordt gemeten.

Dit komt doordat de schakelaar niet perfect contact maakt bij de sluiting, en een aantal keer trilt. Zo ontstaat op moment van schakeling een kort hoogfrequent signaal (zie figuur



Figuur 5, schakeling trilling

Om dit te voorkomen is een low-pass filter toegepast, deze filtert het hoog frequente signaal uit de schakeling. Deze trekt de trillingen gelijk, maar de gemiddelde spanning op deze momenten zal dalen. Figuur 6 weergeeft boven nu een gefilterd signaal.



Figuur 6, Low-Pass filter en ideaal

Dit maakt een nieuw probleem duidelijk, het signaal bevindt zich nu een deel van de tijd in een niet gedefinieerd gebied. De pins van de esp32 meten “hoog” vanaf ~80% van VCC en laag vanaf ~20% VCC, het gebied hiertussen; 20% tm 80% is niet gedefinieerd en resulteerd in onverwachte resultaten, vooral als het signaal bijvoorbeeld vlak onder en vlak boven de 80% trilt.

Hierdoor is de filter niet genoeg om het probleem van extra metingen op te lossen. De volgende stap is het toepassen van een SCHMITT trigger IC. Dit is een Not gate die het signaal versterkt naar een van de uiteindes van het meetgebied.

Dit werkt zoals afgebeeld op Figuur 6. Er is een hoog grens “Vr+” en een laag grens “Vr-“. Als een signaal boven de hoog grens komt wordt de output gelijk aan VCC. Deze wordt pas nul wanneer het ingangssignaal onder de laag grens komt. Zo is ieder gebied tussen nul en VCC gedefinieerd en kan het gefilterde signaal correct gelezen worden door de ESP32.

Code

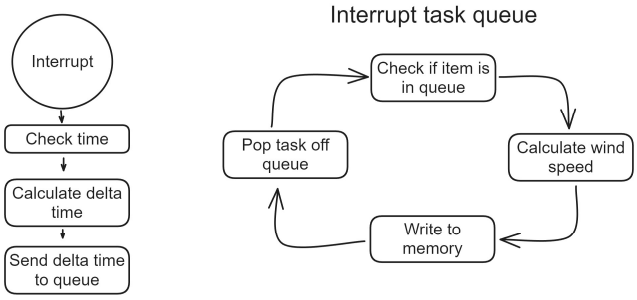
Voor de meting wordt GPIO 25 gebruikt. Eerst worden de relevante instellingen opgeslagen in een struct van de gpio.h bibliotheek. De instellingen worden toegepast met de gpio_config() functie.

```
    gpio_config_t io_conf = {};
    io_conf.intr_type = GPIO_INTR_POSEDGE
    io_conf.pin_bit_mask = GPIO_INTR_PIN_SEL
    io_conf.mode = GPIO_MODE_INPUT;
    io_conf.pull_up_en = 0;
    gpio_config(&io_conf);
```

Het Is zeer belangrijk dat de meting wordt geregistreerd direct wanneer de pin laag wordt, om dit te garanderen worden interrupts gebruikt. Eerst wordt een queue geregistreerd waar interrupt taken kunnen worden opgeslagen; dit garandeert dat een interrupt altijd verwerkt wordt en niet wordt onderbroken door een andere interrupt.

Vervolgens word een taak toegevoegd die de geregistreerde queue afhandeld. Dit is vergelijkbaar met een loop en loopt constant, maar met het verschil dat de activiteit hiervan wordt gemanaged door FREERTOS. (Dit is een soort van OS voor embedded die taken en meer beheert, een groot deel van de uitleg laat ik buiten beschouwing omdat dit verslag al te uitgebreid wordt.)

Hierna wordt de interrupt geregistreerd met gpio_isr_handler_add(). Deze voegt ook de functie toe die wordt aangeroepen wanneer geinterrupt wordt. (in dit geval gpio_isr_handler(). Deze voegt benodigde informatie toe aan de queue.)



Figuur 7, interrupts

Een ruwe representatie van de loop van de code is afgebeeld in Figuur 7. De code is nog iets gedetailleerder, voor verdere info kan naar de comments gekeken worden.

Wifi & MQTT verbinding

De volgende uitleg is beknopt omdat dit grotendeels boilerplate is van het mqtt en wifi voorbeeld van espressif.

Wat flash ruimte wordt vrijgemaakt om wifi gegevens permanent op te slaan. (alleen als niet al beschikbaar)

De wifi_init_sta() functie start de wifi, de instellingen zijn standaard en het gebruikt het gedefineerde SSID en wachtwoord bovenin de code.

Ook wordt een handler aangemaakt die geroepen wordt bij events van de esp_wifi.h bibliotheek. Dit bepaalt wat de code doet wanneer verbinding wordt gemaakt, verbroken etc.

Als de wifi verbonden is wordt de MQTT client gestart met mqtt_start(). Deze definieert de instellingen voor de verbinding met de mqtt://broker.hivemq.com.

Vervolgens wordt een MQTT event handler aangemaakt. De mqtt_client.h bibliotheek regelt dat bij verschillende gebeurtenissen zoal verbinding, data-ontvangst etc. signalen worden uitgestuurd. De handler definieert wat de code moet doen bij ieder signaal.

In ons geval wordt alleen rekening gehouden met wanneer iets fout gaat en iets wordt verstuurd. In beide gevallen wordt in debug geprint wat er is gebeurd.

MQTT data versturen

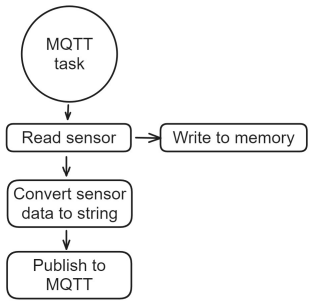
In main wordt de data verstuur taak toegevoegd aan RTOS:

```
xTaskCreate(&publish_sensor_data, "publish_sensor_data", 4096, NULL, 5, NULL);
```

Deze definieert welke functie wordt toegevoegd, hoe deze heet (voor debug), hoeveel geheugen deze mag hebben (ik heb ruim genomen, dit kan veel minder), mogelijke ingangsdata (niet nodig dus NULL), prioriteit (niet heel hoog, dus 5) en een task handle (niet nodig, wordt gebruikt als je later de taak wil bewerken, verwijderen etc.

Publish_sensor_data() bevat een while(1) loop (aka eindeloos). Maar omdat dit in een taak wordt aangeroepen zal RTOS deze beheren, waardoor deze loopt afhankelijk van prioriteit etc.

De publish_sensor_data() functie leest als eerste de sensor data van de i2c sensor. Vervolgens worden alle sensormetingen omgezet naar strings zodat deze via mqtt verstuurd kunnen worden. Figuur 8 beeld de loop van de code grofweg af.



Figuur 8, data versturen

Vervolgens wordt iedere meting individueel verstuurd naar zijn respectievelijke topic.

Overig

Ik heb delen van de uitleg in dit document beknopt gehouden, maar heb uitgebreide comments aan mijn code toegevoegd. (Comments zijn zeer uitgebreid, in werkelijke code zou ik dit beknopter doen, maar wilde extra duidelijkheid omdat dit een school opdracht is.) De comments zijn wel in het Engels, bij coderen vind ik dit prettiger omdat de code en documentatie zelf ook in het Engels is.

Een aantal beslissingen betreffende geheugen en structuur van de code is niet ideaal, ik heb dit nu gedaan door tijdsbeperking en limitatie van scope van de opdracht. Bij een eindproduct van een bedrijf zouden de wifi, mqtt, main, i2c en meetlogica opgesplitst worden in verschillende bestanden, en zou meer tijd gestoken worden in verschillende geheugen optimalisaties.

Functie bewijs:

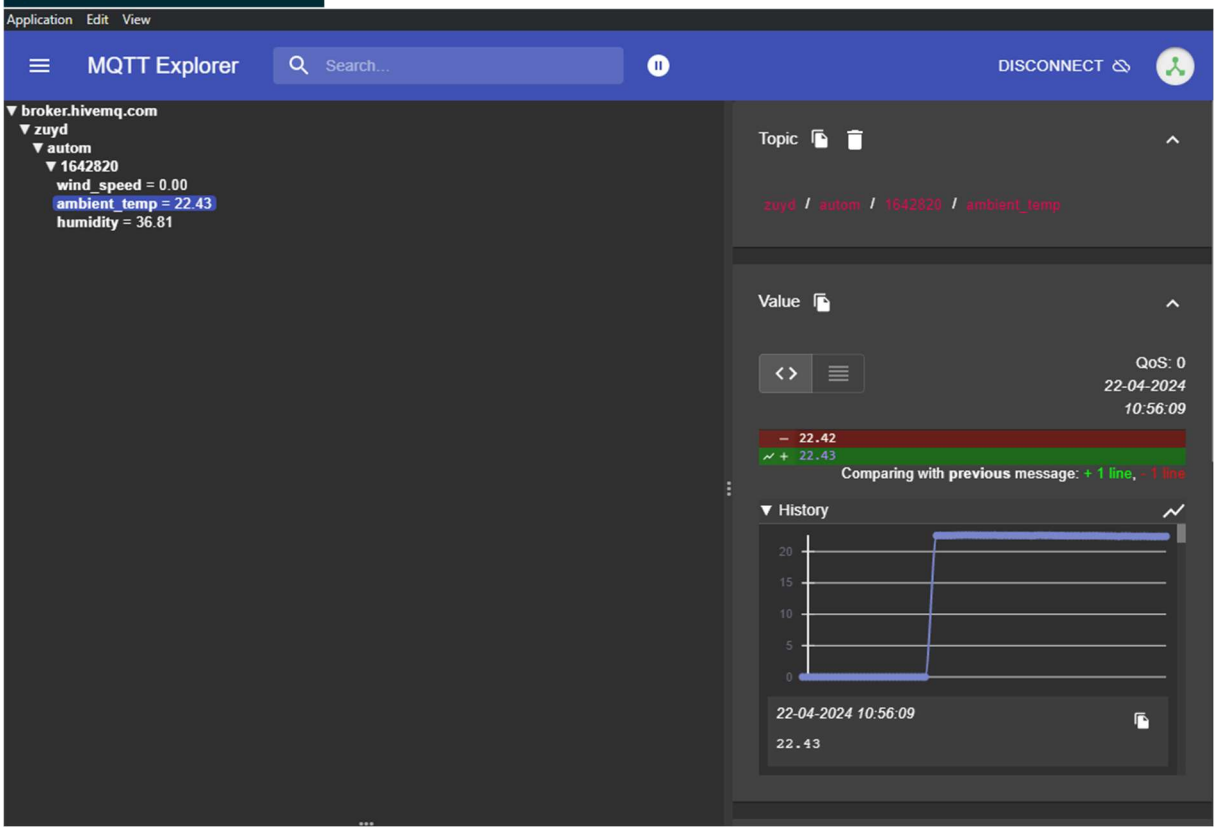
De volgende afbeeldingen laten de debug zien bij versturen van MQTT berichten en interrupts. Die erna weergeven MQTT explorer met de ontvangen data en grafieken. Deze zijn allen functioneel.

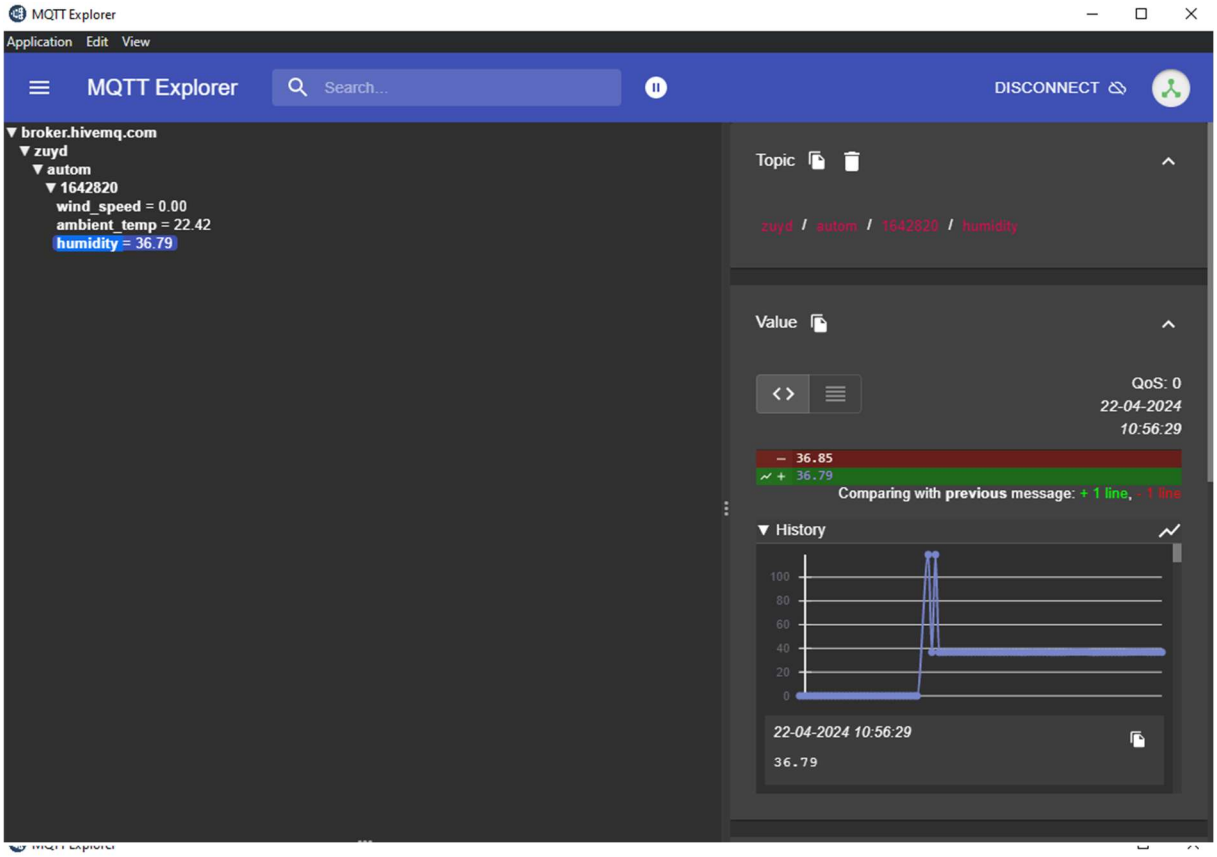
```
80      double temp;
81      double humidity;
82  } sensor_data_t;
83
84  // Structure to store the i2c master bus handle and the i2c master device handle.
85  // This is only used because there is only one i2c device in this project.
86  // In real applications the bus and device handles shlould be handled separately.      You, 1 second ago • Uncommitted changes
    You, 24 seconds ago | 1 author (You)
87  typedef struct {
88      i2c_master_bus_handle_t bus_handle;
89      i2c_master_dev_handle_t dev_handle;
90  } i2c_device;
91
92  static i2c_device i2c_zuyd_sensor;
93
```

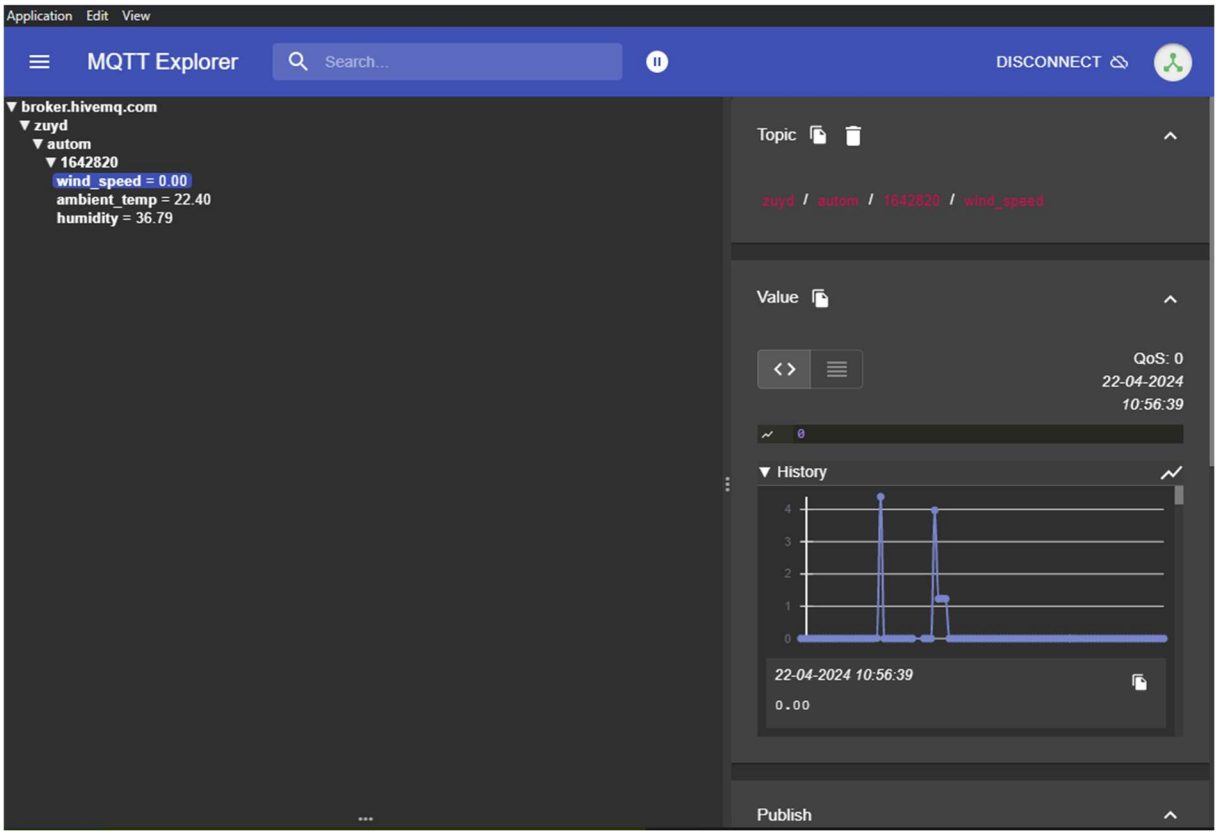
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS GITLENS

```
I (564464) mqtt tester: Temperature raw: 25844.00
I (564464) mqtt tester: Temperature: 22.44
I (564484) mqtt tester: Humidity raw: 22474.00
I (564484) mqtt tester: Humidity: 36.87
I (564524) mqtt tester: MQTT_EVENT_PUBLISHED, msg_id=42815
I (564564) mqtt tester: MQTT_EVENT_PUBLISHED, msg_id=38495
I (564564) mqtt tester: MQTT_EVENT_PUBLISHED, msg_id=42092
I (574504) mqtt tester: Temperature raw: 25848.00
I (574504) mqtt tester: Temperature: 22.46
I (574524) mqtt tester: Humidity raw: 22502.00
I (574524) mqtt tester: Humidity: 36.92
I (574644) mqtt tester: MQTT_EVENT_PUBLISHED, msg_id=61336
I (574694) mqtt tester: MQTT_EVENT_PUBLISHED, msg_id=51922
I (574694) mqtt tester: MQTT_EVENT_PUBLISHED, msg_id=1088
I (584544) mqtt tester: Temperature raw: 25840.00
I (584544) mqtt tester: Temperature: 22.43
I (584564) mqtt tester: Humidity raw: 22530.00
I (584564) mqtt tester: Humidity: 36.97
I (584614) mqtt tester: MQTT_EVENT_PUBLISHED, msg_id=45479
I (584664) mqtt tester: MQTT_EVENT_PUBLISHED, msg_id=4651
I (584664) mqtt tester: MQTT_EVENT_PUBLISHED, msg_id=60981
I (594584) mqtt tester: Temperature raw: 25836.00
I (594584) mqtt tester: Temperature: 22.42
I (594604) mqtt tester: Humidity raw: 22542.00
I (594604) mqtt tester: Humidity: 37.00
I (594644) mqtt tester: MQTT_EVENT_PUBLISHED, msg_id=38677
I (594694) mqtt tester: MQTT_EVENT_PUBLISHED, msg_id=47996
I (594694) mqtt tester: MQTT_EVENT_PUBLISHED, msg_id=29623
```

```
GPIO[25] intr, val: 1
current time: 139908
average time: 127242174
speed: 0.00

GPIO[25] intr, val: 1
current time: 165561
average time: 127250930
speed: 0.00

GPIO[25] intr, val: 1
current time: 120082
average time: 127253404
speed: 0.00

GPIO[25] intr, val: 1
current time: 175427
average time: 127260277
speed: 0.00

GPIO[25] intr, val: 1
current time: 124966
average time: 127264537
speed: 0.00

GPIO[25] intr, val: 1
current time: 176420
average time: 127272875
speed: 0.00

GPIO[25] intr, val: 1
current time: 109987
average time: 127274782
speed: 0.00

GPIO[25] intr, val: 1
current time: 184880
average time: 151286
speed: 2.86

GPIO[25] intr, val: 1
current time: 304989
average time: 168112
speed: 2.57
```




MQTT Explorer
Application Edit View
MQTT Explorer
Search...
DISCONNECT
broker.hivemq.com
zuyd
autom
1642820
wind_speed = 0.00
ambient_temp = 22.42
humidity = 36.79
Topic
zuyd / autom / 1642820 / humidity
Value
QoS: 0
22-04-2024
10:56:29
– 36.85
+ 36.79
Comparing with previous message: + 1 line, - 1 line
History
100
80
60
40
20
0
22-04-2024 10:56:29
36.79


Application Edit View
MQTT Explorer
Search...
DISCONNECT
broker.hivemq.com
zuyd
autom
1642820
wind_speed = 0.00
ambient_temp = 22.40
humidity = 36.79
Topic
zuyd / autom / 1642820 / wind_speed
Value
QoS: 0
22-04-2024
10:56:39
0
History
4
3
2
1
0
22-04-2024 10:56:39
0.00
Publish