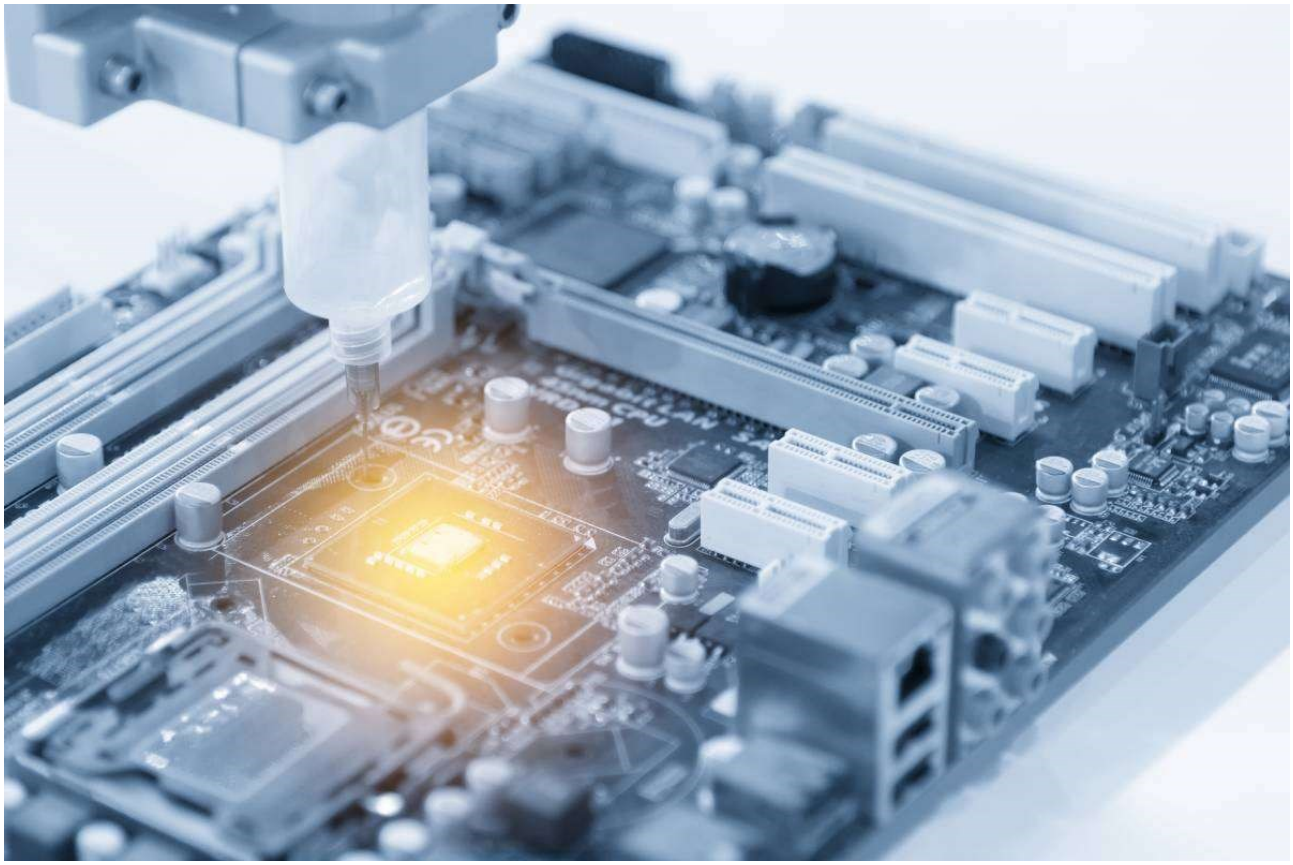
1

PROGETTO SISTEMI OPERATIVI DEDICATI



GIADA GATTI

1108648

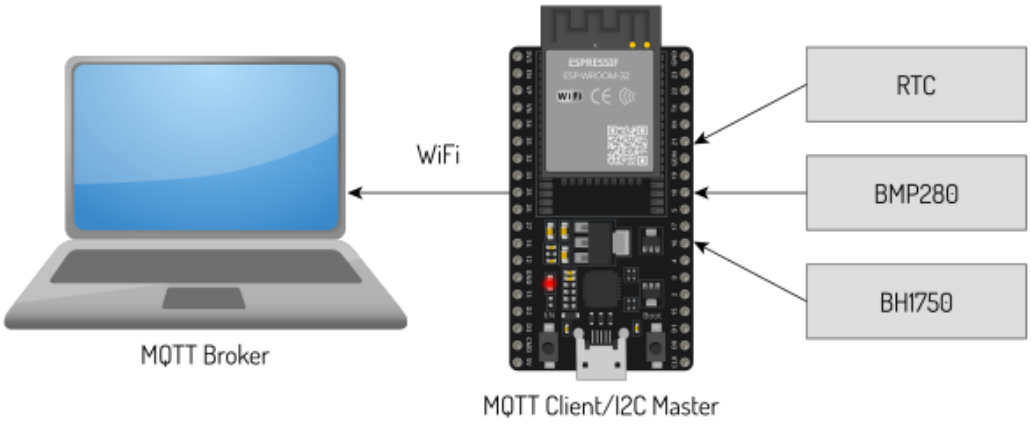
LAUREA MAGISTRALE IN INGEGNERIA INFORMATICA E DELL'AUTOMAZIONE

Link al repository GitHub

Sommario

INTRODUZIONE

Il progetto mira a sfruttare le potenzialità della scheda microcontroller **ESP32**, integrando l'acquisizione dei dati da sensori, come **BMP280** e **BH1750**, la gestione multithreading attraverso *FreeRTOS* e la comunicazione tramite il protocollo *MQTT*.
Il risultato finale è un sistema IoT che consente la visualizzazione dei dati attraverso un'interfaccia Web, fornendo una solida base per applicazioni piu' complesse.



SPECIFICHE DEL SISTEMA

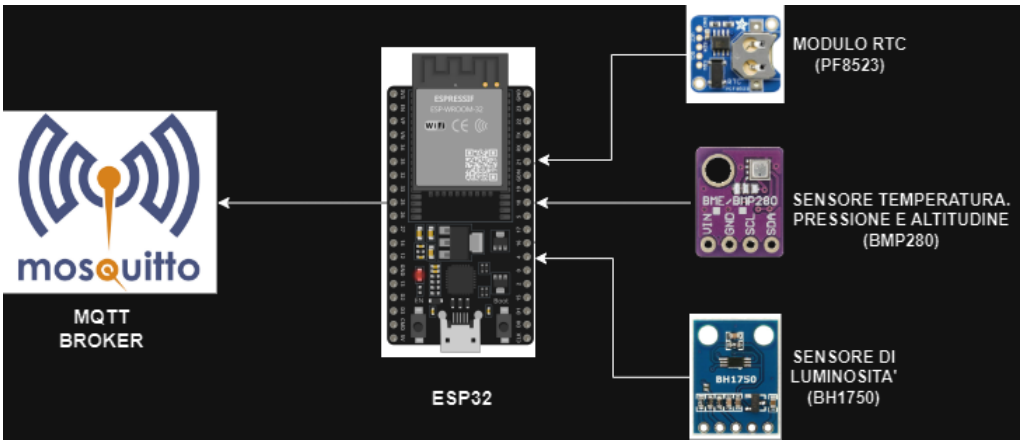
L'architettura del sistema è stata progettata per garantire flessibilità e scalabilità.
La **ESP32** funge da nodo centrale, orchestrando la lettura dei dati dai sensori, la gestione del tempo tramite un modulo **RTC** e la comunicazione *MQTT*.
Un broker *MQTT*, implementato su una macchina virtuale Linux, svolge il ruolo di intermediario nella distribuzione dei dati.
L'interfaccia utente è fornita attraverso un server Web che permette la visualizzazione immediata dei dati acquisiti.

COMPONENTI PRINCIPALI

I componenti principali di questo sistema sono i seguenti:

- **ESP32**: Questa scheda di sviluppo è il cuore del sistema, gestendo la connessione Wi-Fi, le operazioni multithreading tramite FreeRTOS e la comunicazione MQTT.

- **Sensori(BMP280 e BH1750):** Essi forniscono dati dettagliati sulla temperatura, pressione e luminosità ambientale.
- **Modulo RTC(PCF8523)**: fornisce un timestamp per ogni misura acquisita.
- **FreeRTOS**: Sistema operativo in tempo reale che consente l'esecuzione di concorrente di task indipendenti.
- **Mosquitto(Broker MQTT)**: Broker che facilita la comunicazione asincrona tra la ESP32 e altri dispositivi.
- **Browser Web:** Interfaccia intuitiva per l'utente, permettendo la visualizzazione dei dati in tempo reale.



IMPLEMENTAZIONE DEL CODICE

Qui si andranno a descrivere le varie fasi e soluzioni adottate per la realizzazione del sistema.
Il codice per la **ESP32** è suddiviso in task distinti per leggere i dati dai sensori , gestire la comunicazione *MQTT* e ottenere il timestamp dal modulo **RTC**.
La connessione Wi-Fi è configurata per consentire la comunicazione con il broker *MQTT*.

1.1 Acquisizione dati dai sensori

Come prima cosa si vanno a collegare i sensori all'**ESP32** tramite una breadboard. Dopo avere fatto questi collegamenti, attraverso l'ide di sviluppo *Arduinolde*, si vanno ad installare ed importare sul codice le varie librerie che ci serviranno poi per poter leggere i dati dagli stessi. Le librerie utilizzate sono:

- ***Wire.h:*** permette una corretta connessione con i sensori e i moduli RTC
- ***Adafruit_Sensor.h:*** permette un corretto funzionamento dei sensori dell'azienda Adafruit.
- ***Adafruit_BMP280.h:*** libreria per il sensore BMP280 dell'azienda Adafruit.
- ***RTCLib.h:*** libreria per il modulo RTC dell'azienda Adafruit.

- ***WiFi.h:*** permette di connettersi ad un segnale WiFi
- ***PubSubClient***.h: libreria per utilizzare il protocollo MQTT.
- ***BH1750***.h: libreria per il sensore BH1750 dell'azienda Adafruit

Dopodichè, si a creare un task, gestito con *FreeRTOS*, che ci permette di andare a leggere i dati dai sensori, della temperatura, pressione, luminosità e timestamp e memorizzarli su delle variabili globali, attraverso le quali andremo a inviarli, successivamente tramite protocollo *MQTT*.

```
30  void readSensorData(void *pvParameters){
31    while(1){
32      float currentLuminosity = lightMeter.readLightLevel();
33      float currentTemperature = bmp.readTemperature();
34      float currentPressure = bmp.readPressure() / 100.0F;
35
36      DateTime now = rtc.now();
37      String currentTimestamp = String(now.year()) + "-" +
38                                String(now.month()) + "-" +
39                                String(now.day()) + "" +
40                                String(now.hour()) + ":" +
41                                String(now.minute()) + ":" +
42                                String(now.second());
43
44      xSemaphoreTake(dataMutex, portMAX_DELAY);
45
46      luminosity = currentLuminosity;
47      temperature = currentTemperature;
48      pressure = currentPressure;
49      timestamp = currentTimestamp;
50
51      xSemaphoreGive(dataMutex);
52
53      vTaskDelay(5000 / portTICK_PERIOD_MS);
54    }
55  }
```

I dati vengono poi pubblicati su specifici topic *MQTT*

1.2 Comunicazione MQTT

Per gestire la comunicazione *MQTT*, come prima cosa si deve andare ad installare il broker MQTT*(*nel nostro caso *Mosquitto*), da terminale con il seguente comando:

```
sudo apt install mosquitto mosquitto-clients -y
```

Nel momento in cui, si dovrà avviare la connessione *MQTT* basterà far partire *Mosquitto*.

Per rendere il tutto piu' gestibile, si è creato un task anche per *MQTT*.
Come prima cosa andiamo a controllare la connessione, quindi se è connesso al broker *MQTT* e tenta di riconnettersi se la connessione è persa.

Se è connesso, si va a proteggere l'accesso alle variabili locali e quindi invia i dati al broker *MQTT* utilizzando la funzione *client.publish()*.
Dopo l'invio dei dati, il task rilascia il *Mutex* per consentire ad altri task di accedere alle variabili globali.
Questo task gestisce, quindi, in modo efficiente l'invio dei dati *MQTT*, gestendo la connessione, la pubblicazione dei dati e garantendo la protezione dell'accesso concorrente alle variabili globali.

```
58  void sendMQTTData(void *pvParameters){
59    while(1){
60      if(!client.connected()){
61        if(client.connect("ESP32Client")){
62        Serial.println("Connesso al broker MQTT!");
63      }else{
64        Serial.println("Connessione al broker MQTT fallita");
65        vTaskDelay(5000 / portTICK_PERIOD_MS);
66        continue;
67      }
68    }
69
70    xSemaphoreTake(dataMutex, portMAX_DELAY);
71
72    client.publish("luminosity", String(luminosity).c_str());
73    client.publish("temperature", String(temperature).c_str());
74    client.publish("pressure", String(pressure).c_str());
75    client.publish("timestamp", timestamp.c_str());
76
77    xSemaphoreGive(dataMutex);
78
79    vTaskDelay(1000 / portTICK_PERIOD_MS);
80    }
81  }
```

1.3 Funzione Setup

In questa funzione viene inizializzata la comunicazione seriale e la libreria *Wire* per la comunicazione **I2C**, che può essere utilizzata per comunicare con sensori e altri dispositivi.
Si va poi a verificare se i sensori sono inizializzati correttamente.

Per quanto riguarda la connessione del dispositivo alla rete WiFi, si vanno a specificare le credenziali e si attende la connessione.

Essa, quindi, è una funzione che va ad inizializzare il funzionamento dell'**ESP32**.

```
83  void setup() {
84   Serial.begin(115200);
85   Wire.begin();
86
87   if(!bmp.begin()){
88     Serial.println("Errore inizializzazione BMP280");
89     while(1);
90   }
91
92   if(!lightMeter.begin()){
93     Serial.println("Errore inizializzazione BH1750");
94     while(1);
95   }
96
97   if(!rtc.begin()){
98     Serial.println("Impossibile trovare il modulo RTC");
99     while(1);
100  }
101
102  dataMutex = xSemaphoreCreateMutex();
103
104  //Connessione alla WiFi
105  WiFi.begin(ssid, password );
106  while(WiFi.status() != WL_CONNECTED) {
107    delay(250);
108    Serial.println("Connessione WiFi in corso...");
```

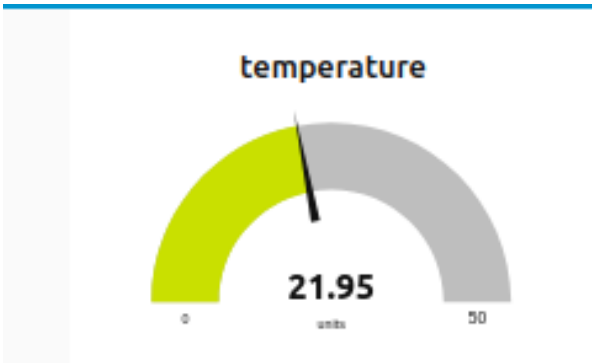
```
113  //Configurazione del client MQTT
114    client.setServer(mqtt_server, mqtt_port);
115
116    xTaskCreatePinnedToCore(
117      readSensorData,
118      "TaskSensorData",
119      10000,
120      NULL,
121      1,
122      &TaskSensorData,
123      0); //core 0
124
125
126
127      xTaskCreatePinnedToCore(
128      sendMQTTData,
129      "TaskMQTT",
130      10000,
131      NULL,
132      1,
133      &TaskMQTT,
134      1); }
```

INTERFACCIA WEB

Questi dati, dopo essere stati inviati, tramite protocollo *MQTT*, al broker *MQTT*, saranno accessibili e visualizzabili attraverso un'interfaccia web creata grazie all'ausilio di ***Node-Red***, installato tramite terminale e raggiungibile attraverso l'indirizzo http://localhost:1880/.

Il risultato che si è deciso di dare a questa interfaccia è del tipo:

TEMPERATURA



PRESSIONE

LUMINOSITA'

CONCLUSIONI

Il progetto offre varie applicazioni nell'ambito dell'IoT, integrando diversi componenti in una sistema complesso.

La collaborazione tra i sensori, FreeRTOS e MQTT fornisce una base robusta per applicazioni future.