

Progetto Sistemi Operativi Dedicati

Antonio Baio - Alexandru Dediu

Matricole 1108645 - 1108807

Laurea Magistrale in Ingegneria Informatica e dell'Automazione

Link al [repository](#)

Descrizione del progetto	2
Specifiche del sistema	3
Funzionamento del sistema	5
Configurazione del Raspberry Pi3	5
Stabilimento connessione WiFi e MQTT tra RPi3 e ESP32	7
Trasferimento dell'immagine da RPi3 a ESP32	9
I. Scelta dell'immagine	9
II. Conversione dell'immagine	9
III. Export del codice dal programma	10
Acquisizione dati dai sensori di movimento e luminosità	11
Controllo della matrice led	14
Conclusioni	16

Descrizione del progetto

Il progetto in questione riguarda lo sviluppo di un sistema composto dai seguenti componenti:

- **Raspberry Pi 3B+**,
- controller **ESP32**,
- una matrice LED con chip **WS2812B**,
- un sensore di luminosità **BH1750**
- un sensore di movimento **HC-SR501**.

L'obiettivo principale del sistema è acquisire dati dai sensori di movimento e luminosità collegati all'ESP32, che funge anche da controller per una matrice LED. La matrice LED verrà invece utilizzata per mostrare delle immagini in stile Pixel Art, regolando la luminosità in base ai dati ricevuti dal sensore preposto.

Il sistema si basa sull'utilizzo di due dispositivi principali: il microcontrollore ESP32 e la Raspberry Pi 3B+. L'ESP32, dotato di FreeRTOS per la gestione dei task, si occupa di acquisire i dati dai sensori di movimento HC-SR501 e di luminosità BH1750. Inoltre, funge da controller per la matrice LED WS2812B, pilotando l'illuminazione degli specifici LED in base sia ai dati acquisiti dai sensori (movimento-luminosità), sia in base all'immagine da visualizzare.

La Raspberry Pi 3B+ funziona invece come un "database" che ospita le immagini che saranno visualizzate sulla matrice LED.

La comunicazione tra l'ESP32 e la Raspberry Pi avviene attraverso una connessione MQTT stabilita tramite una rete Wi-Fi locale, consentendo il trasferimento delle immagini dal database all'ESP32, che provvederà a visualizzarle sulla matrice LED.

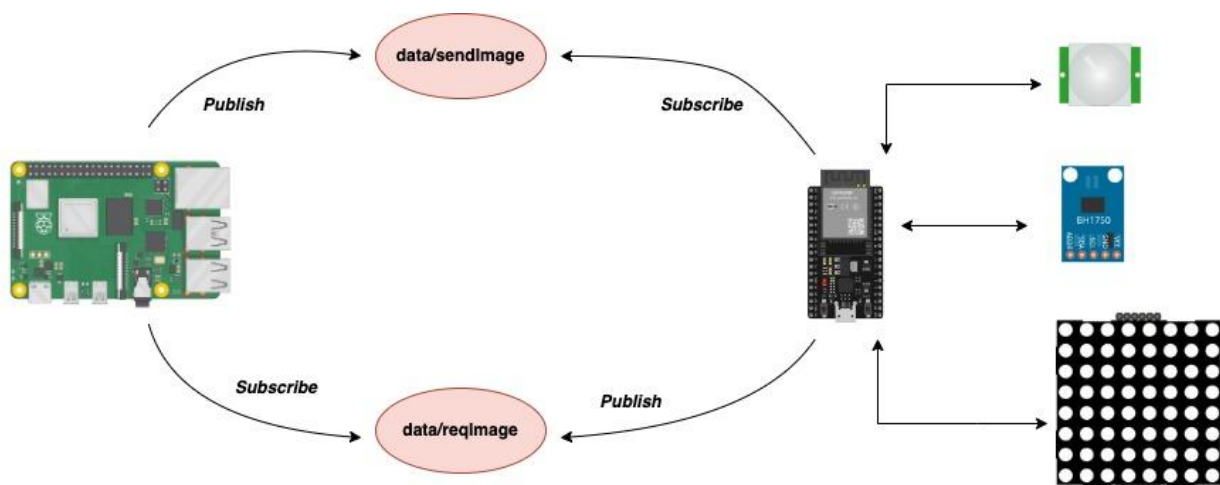
Specifiche del sistema


Il sistema offre le seguenti funzionalità:

- **Acquisizione dati dai sensori di movimento e luminosità:** l'ESP32 è in grado di rilevare i movimenti attraverso il sensore HC-SR501 e la luminosità ambientale tramite il sensore BH1750. Questi dati possono essere utilizzati per attivare specifiche azioni o regolare l'illuminazione della matrice LED in base alle condizioni ambientali.
- **Controllo della matrice LED:** l'ESP32 svolge il ruolo di controller per la matrice LED WS2812B. Utilizzando i dati acquisiti dai sensori di movimento e luminosità, può pilotare l'illuminazione dei singoli LED per visualizzare immagini, pattern o effetti luminosi specifici.
- **Archiviazione delle immagini sulla Raspberry Pi:** La Raspberry Pi 3B+ funge da "database" per le immagini (o meglio, dei file testuali rappresentanti l'immagine, come si vedrà successivamente) che devono essere visualizzate sulla matrice LED. Le immagini possono essere trasferite dalla Raspberry Pi all'ESP32 attraverso la connessione MQTT per una visualizzazione dinamica e personalizzata di quest'ultime sulla matrice LED.

Tutto il sistema (con il codice alla sua base) è gestito con **FreeRTOS**.

FreeRTOS è un kernel in tempo reale open source per microcontrollori e microprocessori embedded. Si tratta di una libreria che fornisce un sistema operativo multitasking per dispositivi a risorse limitate, consentendo l'esecuzione simultanea di più attività indipendenti.





La sua libreria offre funzionalità come la gestione delle attività, la gestione della memoria, la sincronizzazione tra attività, la gestione degli eventi e altro ancora. Questo permette agli sviluppatori di scrivere applicazioni embedded complesse e affidabili, in cui diverse attività possono essere eseguite in modo concorrente, senza interferenze tra di loro.

La flessibilità e l'efficienza di FreeRTOS lo rendono adatto per una vasta gamma di applicazioni, come il caso corrente. Essendo open source, FreeRTOS è ampiamente supportato dalla comunità degli sviluppatori e offre una vasta gamma di port su diverse piattaforme hardware.

Funzionamento del sistema

In questo paragrafo si illustreranno le soluzioni adottate per la realizzazione del sistema.

Configurazione del Raspberry Pi3

La comunicazione tra RPi e ESP32 avviene tramite protocollo MQTT.

MQTT (Message Queuing Telemetry Transport) è un protocollo di messaggistica leggero utilizzato per la comunicazione tra dispositivi con restrizioni di banda e risorse, spesso impiegato in applicazioni IoT (come il presente progetto).

MQTT si basa su un modello di **pubblicazione/sottoscrizione** ed è composto da tre componenti chiave:

1. **Publisher** (Editore):

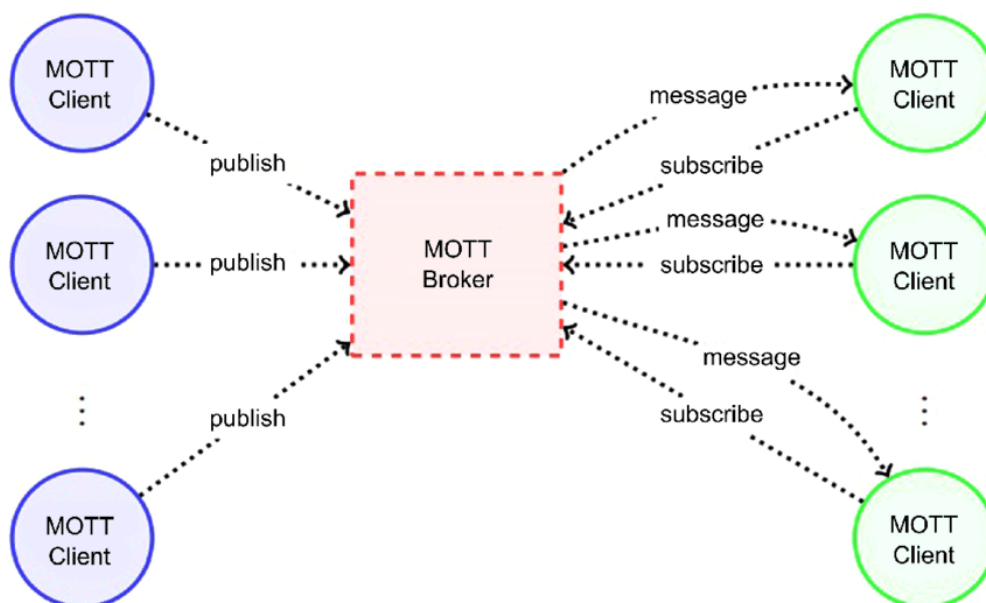
Il publisher è un dispositivo o un'applicazione che **genera messaggi**. Questo componente pubblica i messaggi su specifici "topics" (argomenti), che fungono da canali di comunicazione. Un publisher decide quali messaggi inviare e a quali argomenti pubblicarli.

2. **Subscriber** (Sottoscrittore):

Il subscriber è un dispositivo o un'applicazione interessato a **ricevere messaggi** su determinati argomenti (topics). Un subscriber si sottoscrive agli argomenti di interesse. Una volta sottoscritto, riceverà automaticamente i messaggi pubblicati.

3. **Broker**:

Il broker è un server che funge da **intermediario** tra publisher e subscriber. I publisher inviano i messaggi al broker specificando l'argomento di destinazione. Il broker mantiene una lista dei subscriber interessati a ciascun argomento. Quando un publisher invia un messaggio su un argomento, il broker inoltra automaticamente il messaggio a tutti i subscriber interessati a quell'argomento. Il broker gestisce anche le operazioni di autenticazione e autorizzazione, garantendo che solo i client autorizzati possano pubblicare o ricevere messaggi su determinati argomenti.



In questo progetto, proprio il RPi3 è stato utilizzato come broker MQTT grazie al software open-source [Mosquitto](#).

La configurazione del RPi3 è stata effettuata attraverso diversi passaggi;

- Installazione del broker Mosquitto
- Verifica della corretta installazione
- Abilitazione dell'avvio automatico del broker dopo il boot dell'OS sul RPi3.

Configurato Mosquitto, questo sistema viene integrato con due script Python eseguiti sul RPi3, ovvero **"pub.py"** e **"sub.py"**.

1. Il primo script si occupa di **pubblicare** messaggi su un broker MQTT utilizzando la libreria "paho.mqtt.client". Legge le righe dal file "colors.txt" e invia ciascuna riga come messaggio MQTT al topic "rpi/images". È inoltre configurato per gestire la conferma di avvenuta pubblicazione dei messaggi.
2. Il secondo script gestisce la connessione al broker MQTT e **riceve** messaggi. Utilizza la stessa libreria "paho.mqtt.client" per stabilire una connessione con il broker e si iscrive al topic "data/reqImage". Quando il client riceve un messaggio su questo topic, viene chiamata la funzione "callback_request_image" che stampa il messaggio ricevuto. Inoltre, questo script controlla ciclicamente lo stato della connessione e cerca di ristabilirla in caso di disconnessione.

Per la spiegazione dettagliata e completa dei comandi utili al setup del broker, si rimanda alla relativa sezione del readme su GitHub, raggiungibile al seguente [link](#).

Stabilimento connessione WiFi e MQTT tra RPi3 e ESP32

Installato il broker

Le due funzioni **"mqttCon"** e **"wifiConn"**, gestiscono rispettivamente la connessione MQTT e la connessione WiFi.

Ecco una breve descrizione di come funzionano:

1. **"wifiConn"**: questa funzione si occupa di stabilire la connessione WiFi. Stampa un messaggio che indica il tentativo di connessione alla rete WiFi specificata (memorizzata nella variabile *"ssid"*). Utilizza il metodo *"WiFi.begin(ssid, wifi_password)"* per iniziare la connessione alla rete WiFi utilizzando il nome della rete (SSID) e la password (memorizzata nella variabile *"wifi_password"*). Nel ciclo *"while"* successivo, la funzione controlla lo stato della connessione WiFi con *"WiFi.status()"*. Finché lo stato non è *"WL_CONNECTED"* (cioè finché la connessione non è stabilita), viene visualizzato un punto ogni mezzo secondo. Una volta stabilita la connessione, vengono stampati i messaggi di successo insieme all'indirizzo IP locale ottenuto tramite *"WiFi.localIP()"*.

```
1 void wifiConn()
2 {
3
4     Serial.print("***** Connessione al WiFi in corso :");
5     Serial.println(ssid);
6
7     WiFi.begin(ssid, wifi_password);
8     while (WiFi.status() != WL_CONNECTED)
9     {
10         delay(500);
11         Serial.print(".");
12     }
13     Serial.println("");
14     Serial.println("->Connessione al WiFi effettuata");
15     Serial.println("-> indirizzo IP: ");
16     Serial.println(WiFi.localIP());
17 }
```


2. ***“mqttConn”***: questa funzione si occupa di stabilire la connessione al server MQTT, che in questo progetto è rappresentato dal Raspberry. La funzione stampa un messaggio che indica il tentativo di connessione al server MQTT specificato (memorizzato nella variabile *“mqtt_server”*). La funzione *“client.setServer(mqtt_server, mqtt_port)”* imposta il client MQTT per connettersi al server specificato all'indirizzo e alla porta corrispondenti. Successivamente, viene chiamata la funzione *“reconnect()”* che è responsabile di gestire la riconnessione in caso di perdita della connessione MQTT.

```
1 void mqttConn()
2 {
3     Serial.print("**** Connesso al server MQTT : ");
4     Serial.println(mqtt_server);
5
6     client.setServer(mqtt_server, mqtt_port);
7     reconnect();
8 }
```

Trasferimento dell'immagine da RPi3 a ESP32

Per poter visualizzare l'immagine sulla matrice, ci siamo avvalsi dell'uso di due strumenti:

- il primo è la libreria **FastLED**: questa libreria per ESP32 semplifica il controllo di una varietà di chipset LED (come WS2812B). Offre funzioni di calcolo RGB ad alte prestazioni e accesso a pin e hardware SPI.
- il programma **LEDMatrixControl**: prende in input un'immagine in pixel art e la divide in più blocchi, uno per led. Nel nostro caso, ci saranno 8x8 blocchi, ovvero 64 led.

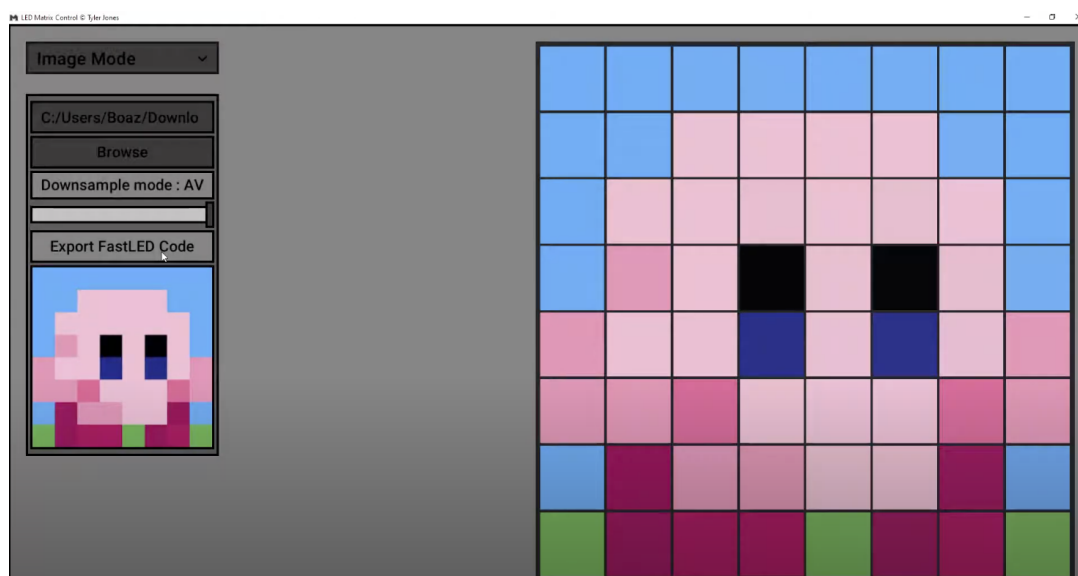
I passi da seguire per poter generare, trasferire e visualizzare l'immagine sulla matrice sono i seguenti:

I. Scelta dell'immagine

Navigando sul web, bisogna cercare un'immagine in pixel art di dimensioni 8x8 (pari alla dimensione della matrice LED in uso. Al seguente [link](#) è possibile trovare un esempio.

II. Conversione dell'immagine

Scelta l'immagine e scaricata sul proprio dispositivo, attraverso l'uso di LEDMatrixControl si è in grado di trasformare l'immagine scelta nel codice utilizzato dalla libreria FastLED per impostare i colori dei singoli LED. Come visibile dal seguente screenshot, una volta caricata l'immagine, il programma si occuperà di suddividerla in 64 diversi blocchi, dove ogni blocco corrisponde ad un LED con un colore specifico:

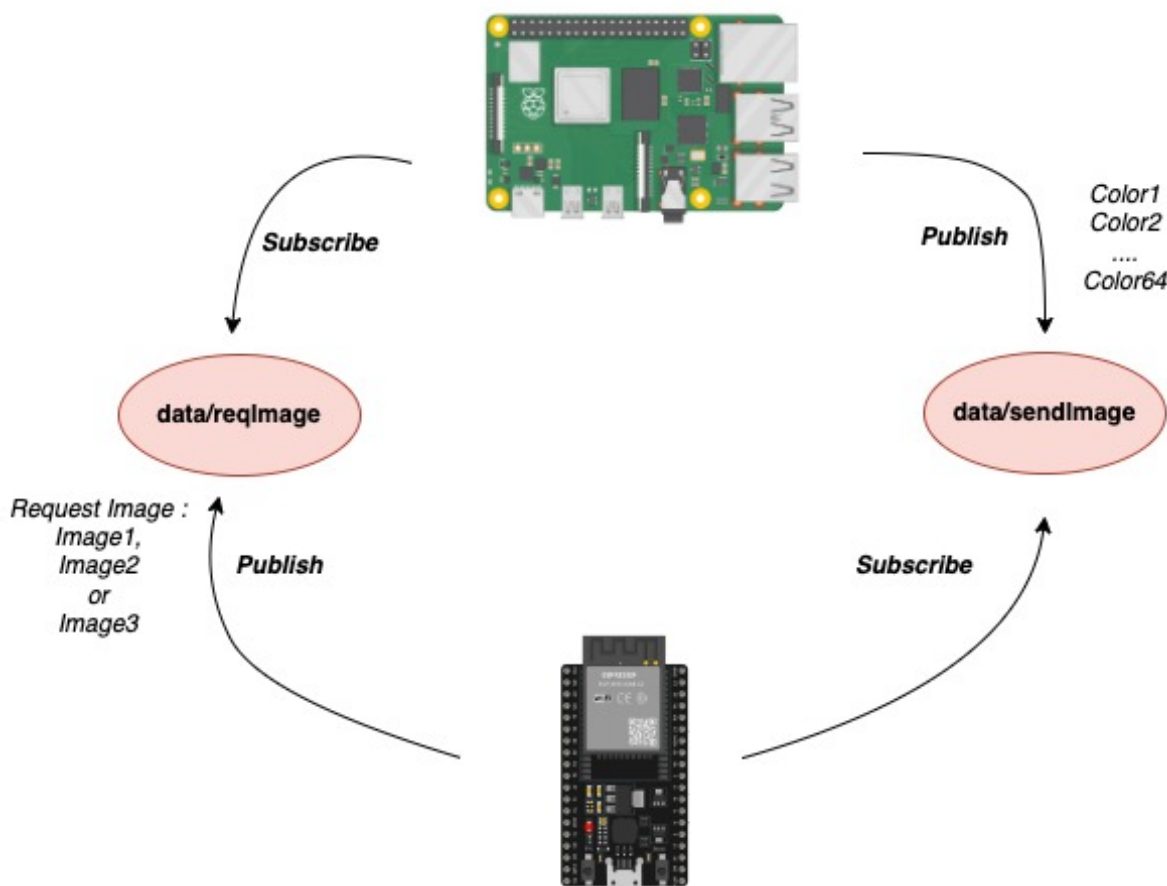


III. Export del codice dal programma

Tramite la funzione dedicata, è possibile generare ed esportare il codice delle funzione della libreria FastLED da utilizzare sul proprio progetto. Questo codice va ad assegnare a ognuno dei 64 led un colore specifico, in modo da ricreare l'immagine originale. Ottenuto il codice, questo viene salvato in un file *txt* e sarà pronto al trasferimento tramite connessione MQTT verso l'ESP32.

Una volta instaurata la connessione tra i due componenti, il primo passo svolto è la pubblicazione, da parte dell'ESP32, della richiesta di ricevere un'immagine da parte del Raspberry, nel *topic* specifico di **"reqImage"**.

Quest'ultimo, essendo "iscritto" al topic della richiesta, risponde con la pubblicazione in un secondo topic **"sendImage"**, con cui pubblica i messaggi contenenti i colori dei 64 singoli led in formato **(R,G,B)**, e dove ogni led è identificato dalle coordinate **X** e **Y**.



Ricevuti tutti i messaggi, si passa alla fase di parsing del messaggio.

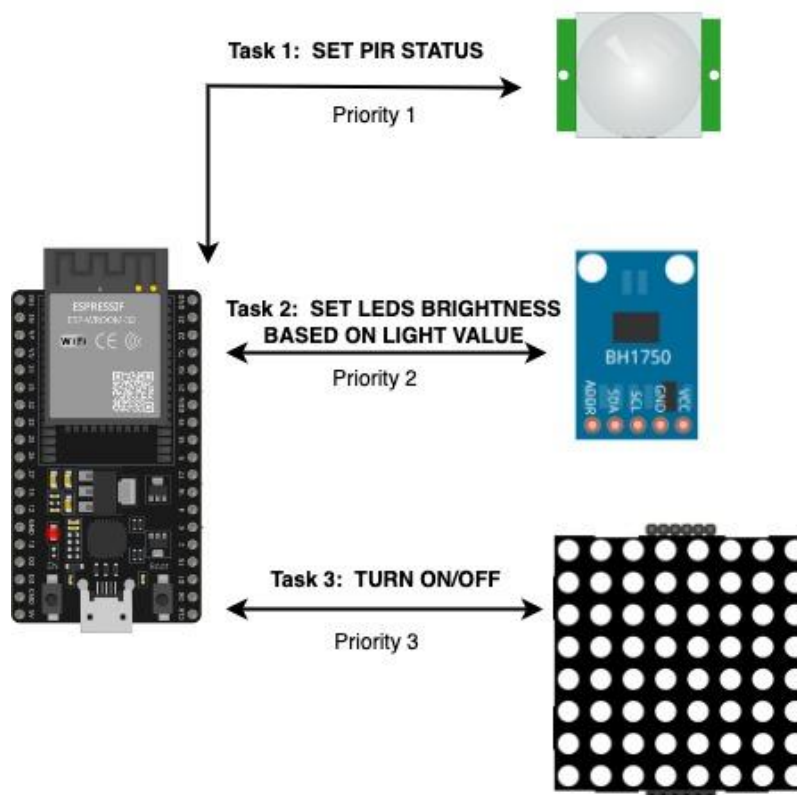
La funzione "*parsePayload*" presente nello script su ESP32 analizza i dati ricevuti attraverso MQTT dal Raspberry. Questi dati, come anticipato, contengono informazioni sui colori dei LED: questo metodo estrae i valori RGB dai dati e li utilizza successivamente, nell'ultimo task programmato, per impostare i colori dei LED della matrice, gestendo la logica per iterare sia attraverso i dati che sui LED.

Acquisizione dati dai sensori di movimento e luminosità

Ricevuti i dati dell'immagine, i primi due task che verranno eseguiti sono quelli relativi all'acquisizione dei valori di movimento e luminosità, che andranno ad incidere sulla visualizzazione dell'immagine sulla matrice stessa. Come descritto in precedenza:

- a seconda del dato di movimento rilevato o meno, il sistema mostrerà sulla matrice un'immagine
- a seconda del dato di luminosità rilevato, il sistema regolerà l'intensità con cui la matrice mostrerà l'immagine (secondo diverse fasce di luminosità da noi stabilite).

Dunque, il primo passo da effettuare è il collegamento fisico dei due sensori all'ESP32, che opportunamente programmato, riesce a controllare ed utilizzare i sensori, ricevendo i dati rilevati.



Per la rilevazione del movimento, il sensore utilizzato è il **HC-SR501**: viene integrato all'interno del progetto e gestito tramite il seguente task creato con FreeRTOS:

```
1 void pirStatusTask(void *parameter)
2 {
3     for (;;)
4     {
5         pir_value = digitalRead(19);
6
7         pir_value ? Serial.println("MOVIMENTO RILEVATO") : Serial.println("NESSUN MOVIMENTO RILEVATO...");
8         vTaskDelay(3000);
9     }
10    vTaskDelete(NULL);
11 }
```

Esegue ciclicamente le seguenti azioni:

1. Legge lo stato del sensore di movimento collegato al pin 19 tramite `digitalRead(19)` e assegna il valore a `pir_value`.
2. A seconda dello stato, mostra a schermo un messaggio di movimento rilevato o meno.
3. Attende 3 secondi prima di iniziare il ciclo successivo.

Il sensore di luminosità utilizzato è il **BH1750**. Anche quest'ultimo è perfettamente compatibile con l'ESP32 attraverso l'utilizzo della omonima libreria: creando un oggetto chiamato *lightMeter* di tipo **BH1750**, anche in questo caso è possibile scrivere il task relativo alla gestione del sensore stesso.

```
1 void lightSensorTask(void *parameter)
2 {
3     Serial.println("Valore del pir nel task sensore luminosità: ");
4     Serial.println(pir_value);
5
6     for (;;)
7     {
8         if (pir_value)
9         {
10             float lux = getLux();
11             FastLED.setBrightness(setBrightness(lux));
12
13             Serial.println("Brightness: " + String(setBrightness(lux)));
14         }
15         else
16         {
17             Serial.println("Sensore di luminosità spento (movimento non rilevato)");
18         }
19
20         vTaskDelay(4000);
21     }
22     vTaskDelete(NULL);
23 }
```

La funzione "*lightSensorTask*" controlla la luminosità dei LED in base allo stato di un sensore di movimento. Se il sensore rileva un movimento, la luminosità dei LED viene regolata in base alla luminosità rilevata dalla funzione "*getLux*". In caso contrario, i LED vengono mantenuti spenti. La funzione ciclicamente verifica lo stato del sensore e regola i LED di conseguenza, con intervalli di 4 secondi.

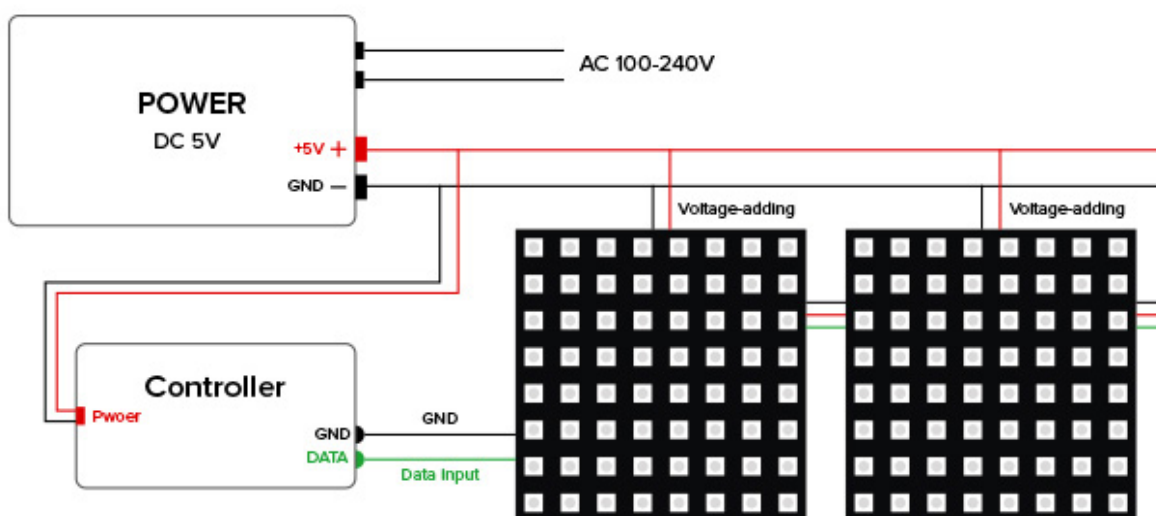
Tramite la funzione "*FastLED.setBrightness()*", si va successivamente a regolare la luminosità dei LED: a seconda della fascia di appartenenza del valore di luminosità rilevato, verrà impostato il valore corrispondente dei LED che compongono la matrice.

Controllo della matrice led

La matrice a disposizione è formata da 64 leds (8x8). Il chip di cui è fornito ogni singolo led è il WS2801B. Grazie alla libreria [FastLED](#), è possibile controllare in modo individuale ogni led della matrice, in modo da creare quindi l'immagine e visualizzarla sulla matrice tramite l'ESP32.


La matrice è collegata ad un alimentatore in grado di erogare una corrente di 4A e un voltaggio di 5V, per un totale di 20W: la matrice, a piena luminosità, assorbe circa 22W.

Viene collegata successivamente all'ESP32 attraverso il seguente schema:



Come osservabile dallo schema, il controller è il microcontrollore ESP32. Eventualmente, avendo più matrici, è possibile collegarle anche in serie: il pin di data input (verde nell'immagine) sarà comune a tutte. Nel nostro caso, ne è presente solo una.

Il task relativo alla gestione della matrice è il terzo, chiamato "ledMatrixTask", dove la funzione principale è "setColors()":

A screenshot of a code editor with a dark background and three colored window control buttons (red, yellow, green) in the top left corner. The code is written in C++ and defines a function named setColors. The function takes an integer parameter pir_status. It uses an if-else statement to check the value of pir_status. If it is non-zero, it prints "Matrice accesa..." and calls FastLED.show(). If it is zero, it prints "Matrice spenta...", sets the brightness to 0, and calls FastLED.show(). The code is numbered from 1 to 15.

```
1 void setColors(int pir_status)
2 {
3
4     if (pir_status)
5     {
6         Serial.println("Matrice accesa...");
7         FastLED.show();
8     }
9     else
10    {
11        Serial.println("Matrice spenta...");
12        FastLED.setBrightness(0);
13        FastLED.show();
14    }
15 }
```

Se lo stato del sensore di movimento è attivo, la matrice viene accesa e i colori dei LED vengono visualizzati tramite "FastLED.show()", seguendo lo schema dei colori ricevuti precedentemente tramite messaggio MQTT.

In caso contrario, la matrice viene spenta: la luminosità dei LED viene azzerata, e "FastLED.show()" viene chiamato per garantire che i LED siano spenti. Durante ogni azione, vengono stampati messaggi sulla porta seriale per indicare lo stato della matrice.

Conclusioni

In conclusione, il progetto in esame si basa sulla comunicazione tra varie componenti e coinvolge diverse funzioni che coordinano l'interazione con vari sensori e la gestione di una matrice LED.

La libreria FastLED viene impiegata per il controllo efficiente e preciso dei LED, con l'obiettivo di massimizzare le risorse disponibili per la generazione dei modelli luminosi.

Le attività definite attraverso il sistema operativo in tempo reale (Free RTOS) permettono una gestione parallela delle operazioni, compresa l'analisi di dati provenienti da un Raspberry tramite MQTT, il controllo della luminosità in base al sensore di movimento e la manipolazione dei colori dei LED.

L'architettura complessiva del progetto mira quindi a creare un sistema di illuminazione reattivo e dinamico, dimostrando l'integrazione di diverse competenze di programmazione e tecnologie hardware.