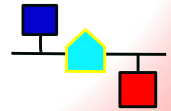
ESPHome

Come creare sensori ed attuatori WiFi da collegare ad Home Assistant

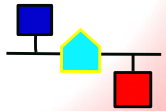


Home Assistant

- Home Assistant è un software open source che ha lo scopo di gestire l'automazione di una casa intelligente dialogando con i vari dispositivi in essa contenuti
- Può essere installato di un pc che farà da server.
- Si possono aggiungere ad home assistant dei moduli per comunicare con una gran quantità di dispositivi di domotica, sia direttamente che tramite i relativi cloud
- Ha una semplice interfaccia Web configurabile
- Può essere programmata per azionare automaticamente i dispositivi collegati tramite delle automazioni programmabili
- Esiste anche un'app per comandarla da cellualre

ESP 8266

- L'ESP8266 è un chip con Wi-Fi integrato a basso costo, con supporto completo al protocollo TCP/IP e funzionalità da microcontrollore prodotto dall'azienda cinese di Shanghai Espressif Systems. (Wikipedia)
- Contiene una CPU a 32 bit
- Viene associato ad una EEPROM seriale che ne consente la programmazione
- Ha una seriale hardware interna ed una serie di piedini di ingresso/uscita che consentono di interrogare sensori e di pilotare dispositivi
- Dato il suo bassissimo costo questo chip vene utilizzato in una enorme quantità di apparati WiFi legati all'internet delle cose (IoT – Internet of things)
- Sono state prodotte anche vari tipi di schede per uso amatoriale, come la ESP-01 che consente di aggiungere capacità WiFi ad esempio a progetti con Arduino



ESP Home

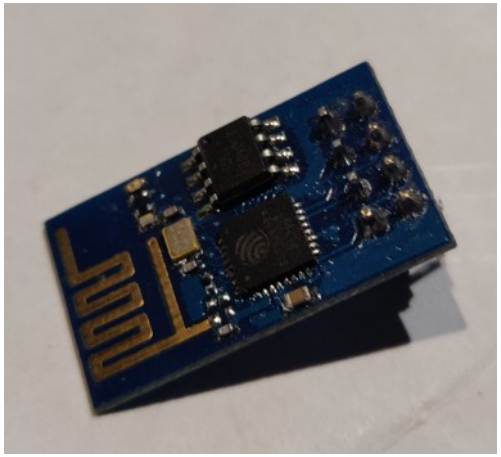
- ESPHome è un software per processore ESP8266 o ESP32 (il suo successore) configurabile, che comunica con Home Assistant
- Può essere configurato per l'hardware connesso all'ESP tramite un file di configurazione
- Ha un sistema di sviluppo che consente di creare il codice per l'ESP in base alla configurazione scelta
- Può caricare il software sul dispositivo tramite interfaccia seriale o USB
- Una volta che il software è installato, se il dispositivo ha una quantità sufficiente di EEPROM, può anche aggiornarsi direttamente tramite il WiFi (ota – On The Air)

Dispositivi commerciali

- Come dicevo, molti dispositivi commerciali sono costruiti attorno ad un ESP82666
- Virtualmente tutti questi possono essere 'trasformati' in dispositivi ESPHome
- Per i didpositivi compatibili spesso esiste già una configurazione adatta negli archivi di ESPHome
- Occorrerà solo caricarla nel sistema di sviluppo e compilarla.
- Occorrerà poi caricarla nel dispositivo
- Se si è fortunati si può sostituire il software intercettando il meccanismo di aggiornamento tramite WiFi e caricando ESPHome al posto del software originale
- Il più delle volte però occorre aprire il dispositivo, connettere la seriale di debug (spesso saldarla ai piedini del processore) e fare il primo caricamento tramite quella.

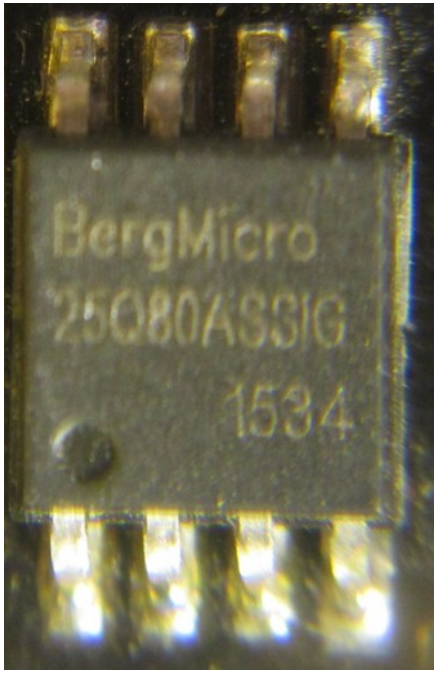
ESP 01

- ESP-01 è una schedina estremamente piccola, prodotta originariamente dalla AI-Tinker, ma ora disponibile a bassissimo prezzo da moltissimi produttori.
- Ha un connettore a lance ad 8 piedini che consente di:
  - Dare alimentazione (3,3V, 200mA max)
  - Colloquiare con la seriale
  - Resettare il dispositivo
  - Impostarne il modo di funzionamento (programmazione o funzionamento normale



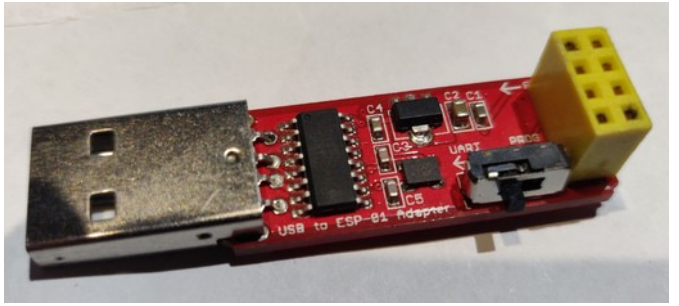
ESP-01 - EPROM

- Come detto, l'ESP-01 monta due chip: l'ESP8266 ed una EEPROM seriale
- La EEPROM può essere di due tipi, esternamente identici:
  - 25Q40 da 512Kib
  - 25Q80 da 1Mib
- Di solito la versione di ESP-01 da 512KiB è su basette di colore verde mentre la versione da 1MiB è su basette di colore nero.
- Entrambe le ROM possono ospitare ESPHome, ma con la prima non è possibile aggiornare il sistema via WiFi (OTA)



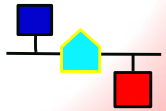
Programmatore per l'ESP-01

- Per programmare l'ESP-01 occorre
  - Alimentarlo a 3,3v
  - Connetterlo ad un'interfaccia seriale
  - Collegare opportunamente alcuni piedini, per falro entrare in modalità programmazione
- A questo scopo si trovano delle piccole interfacce USB che montano
  - Un regolatore di tensione da 5V (USB) a 3,3V (ESP)
  - Un convertitore USB-Seriale
  - (a volte) un interruttore che commuta i piedini per la modalità programmazione



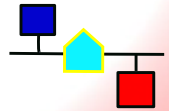
Preparazione per l' installazione

- La programmazione si fa alinea di comando, quindi va bene anche un sistema senza desktop.
- Io ho usato un'installazione minima della Debian 12 Bookworm.
- Occorre che sia installato Python con una versione superiore alla 3.9 – Bookworm monta la 3.11 – OK
- Devo poi installare nel sistema:
  - python3-pip
  - python3.11-venv
  - avahi-daemon (per il riconoscimento del dispositivo installato e l'aggiornamento OTA)
  - Minicom (nel caso serva comunicare via seriale con ESP)
- Occorrerà poi accertarsi che il nostro utente appartenga al gruppo dialout (gpasswd -a <mioutente> dialout) per consentirgli di usare la seriale



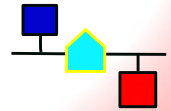
Installazione dell'Ambiente di Sviluppo

- Possiamo ora seguire le semplici istruzioni di installazione: https://esphome.io/guides/installing_esphome.html
- Per linux indicano semplicemente:
  - **`python3 -m venv venv`**
  - **`source venv/bin/activate`**
  - **`pip3 install esphome`**
  - **`esphome version`**
- Se l'ultimo comando non riesce, aggiungete **`$HOME/.local/bin`** alla **`path`**
- Siamo pronti per partire



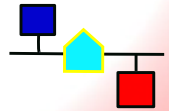
Dispositivo di esempio

- Per questo esempio collegheremo via seriale un Arduino Nano ad un ESP01
- All'Arduino Nano saranno collegati
  - un **pulsante** come input collegato sul piedino 2
  - Il **LED** a bordo collegato sul piedino 13
  - L'**esp-01** su una seriale software: piedini 5 (RXD) ed 4 (TXD)
- Utilizzo una seriale software perché il convertitore USB-Seriale dell'Arduino occupa la seriale hardware
- Attraverso opportuni comandi inviati tramite la seriale, potremo interagire con i sensori ed il pulsante



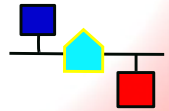
I comandi di esempio

- Saranno implementati i seguenti comandi:
  - **`LON`** accende il LED sul piedino 13
  - **`LOFF`** spegne il LED sul piedino 13
  - **`BTN`** richiede la lettura del pulsante – Arduino risponderà con
    - **`BON`** se il pulsante è premuto
    - **`BOFF`** se il pulsante è spento
- Tutte le richieste e risposte sono terminate da **`<a capo>`** (**`'\n'`**)



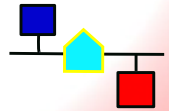
Pianificazione del progetto

- Per iniziare a creare il nostro progetto ci servono alcune informazioni e dobbiamo fare delle scelte:
  - Nome del progetto (name): **esp-test**
  - Processore (ESP32/ESP8266): **ESP8266**
  - Scheda (board): **esp01_1m**
  - Rete Wireless (ssid): <la rete a cui mi collego>
  - Password di rete (PSK): <la password di rete a cui mi collego>
  - Password per entrare nel dispositivo (password): **Ar-t6628-01**



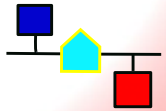
Inizializzazione del progetto

- Creiamo il progetto vuoto
- Se non siamo nell'ambiente virtuale di Python - non compare **(venv)** all'inizio del prompt – diamo il comando
  source venv/bin/activate
- Possiamo ora lanciare la creazione del progetto:
  esphome wizard esp-test.yaml
- Risponderemo alle domande con i dati che abbiamo individuato precedentemente:
  - (name): **esp-test**
  - (ESP32/ESP8266/BK72XX/RTL87XX):**ESP8266**
  - (board): **esp01_1m**
  - (ssid): **OpenWrt-2.4-LD**
  - (PSK): **Gqz9,qpq**
  - (password): **Ar-t6628-01**



Iniziamo la configurazione

- A questo punto nella nostra directory home troveremo il file del nostro progetto: **esp-test.yaml**
- Conterrà
  - le informazioni sulla piattaforma hardware
  - Le informazioni sulla wifi cui collegarsi
  - La password di accesso a EspHome dalla rete
- Non contiene ancora alcuna informazione sulle periferiche da implementare



Il file yaml iniziale

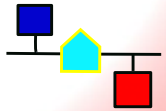
Il file che abbiamo così ottenuto è questo:

```
esphome:
  name: esp-test

esp8266:
  board: esp01_1m
# Enable logging
Logger:
```

```
# Enable Home Assistant API
api:
  password: "Ar-t6628-01"

ota:
  password: "Ar-t6628-01"
wifi:
  ssid: "OpenWrt-2.4-LD"
  password: "Gqz9,qpq"
```

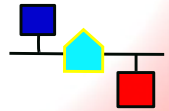
```
# Enable fallback hotspot
# (captive portal) in case
# wifi connection fails
  ap:
    ssid: "Esp-Test Fallback
Hotspot"
    password: "EK2kwHAo8Q7k"

captive_portal:
```

Sarà opportuno aggiungere **`baud_rate: 0`** sotto **`Logger:`** per disabilitare il log sulla seriale, che è utilizzata per comunicare con l'Arduino



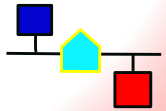
Compiliamo e scarichiamo via seriale il progetto

- Nella nostra directory ora abbiamo il file **`esp-test.yaml`** contenete i dati di base della scheda e dell'ambiente
- Come abbiamo visto non ci sono né sensori né switch
- Ci conviene comunque provare a compilarla e scaricarla tramite il programmatore sull'ESP-01 perché da questo momento potremo programmalo tramite WiFi (*OTA*) senza staccarlo dalla basetta
- Per fare questo primo caricamento:
  - Inseriremo l'ESP-01 sul programmatore
  - Metteremo l'interruttore del programmatore (se presente) nella posizione programmer
  - Inseriremo il programmatore con l'ESP-01 in una presa USB
  - Lanceremo il comando
    `esphome run esp-test.yaml`
  - Alla fine della compilazione (parecchio lunga) sceglieremo **`[1]`** per scaricare tramite il programmatore il programma



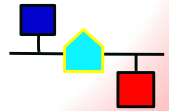
Componenti base

- Per costruire la nostra periferica EspHome offre i componenti
- La pagina dei componenti è https://esphome.io/components/
- I componenti possono essere
  - Oggetti virtuali
    - Interruttori (**`switch`**)
    - Sensori (**`sensor`**)
    - Altro ...
  - Metodi per interagire (**`platform`**)
    - Collegamento Seriale (**`UART`**)
    - Piedini dell'integrato (**`GPIO`**)
    - Pezzi di codice (**`template`**)
    - Altro...



Componenti composti o derivati

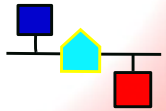
- Sono poi presenti dei componenti composti, che associano al componente un particolare metodo di comunicazione.
- Ad esempio noi potremo usare:
  - Un **`uart switch`** per comandare l'accensione e spegnimento del LED tramite la seriale (UART)
  - Un **`binary sensor`** per il pulsante
- Per i due sensori non abbiamo un componente già costituito nel quale indicare solo, ad esempio, il piedino da utilizzare o la stringa da inviare
- Dovremo usare dei piccoli programmi in C, detti **`lambda`**



Aggiungiamo il primo componente: l'UART

- Visto che la nostra periferica colloquia in seriale, il primo componente da inserire è appunto la UART.
- Nell'ESP-01 la UART0 (che ne è anche un'altra nel chip) è collegata ai piedini:
  - Tx: pin 1
  - Rx: pin 3
- Il nostro Arduino comunica a 9600 Baud
- Dobbiamo aggiungere, in fondo al file yaml, le seguenti righe:

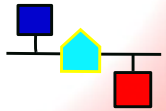
```
uart:
    id: serial
    tx_pin: 1
    rx_pin: 3
    baud_rate: 9600
```

Aggiungiamo una variabile

- Lo switch comandato dall'UART non prevede accensione e spegnimento separati, come il nostro Arduino
- Dovremo quindi prevedere una variabile booleana che memorizzi lo stato
- La variabile si chiamerà **test_led_status** e verrà definita così:

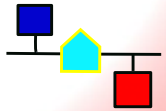
```
globals:
  - id: test_led_status
    restore_value: no
    type: bool
    initial_value: 'false'
```

Ora aggiungiamo lo switch

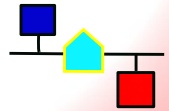
- Ora che abbiamo la variabile per lo stato possiamo definire lo switch come **`template`**, vale a dire che fa delle azioni
- Vediamo come definirlo:

```
switch:
  - platform: template
    name: "test_led"
    id: test_le
    lambda: !lambda |-
      return id(test_led_status);
```

La definizione ed il valore dello switch

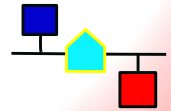
- Abbiamo definito uno **`switch`** con:
- la **`platform`** dello **`switch`** come **`template`**
- Il **`name`** e l'**`ID`** entrambi **`test_led`**
- Per finire il **`lambda`** vale a dire il codice che deve restituire il valore dello **`switch`** che nel nostro caso sarà **codice C incorporato**.
- Per incorporare codics occorre utilizzare la stringa:
  `!lambda |-`
  che precede le righe di codice
- Per finire, il codice:
  `return id(test_led_status);`
  restituisce il valore della variabile.



Il codice per l’accensione

- Quando Home Assistant genera l’evento **`turn_on`** dobbiamo inviare la stringa **`LON`** all’arduino.
- Dobbiamo anche modificare la variabile di stato per ricordarci che lil LED è acceso
- Questo lo facciamo con il seguente codice:

```
turn_on_action:
    - uart.write: 'LON\n'
    - globals.set:
        id: test_led_status
        value: 'true'
```

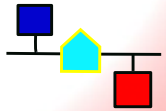
La spiegazione del codice per l’accensione

- **`turn_on_action`** è il comando che arriva da Home Assistant (l’azione).
- Quel che segue è il ***tempalte***. Le azioni elencate vengono eseguite in sequenza
- Per prima cosa, con **`uart.write: 'LON\n'`** mando il comando via seriale all’Arduino
- Poi con **`globals.set:`** imposto la variabile con **`id: test_led_status`** al **`value:'true'`** per annotare che il LED è acceso

Il codice di spegnimento

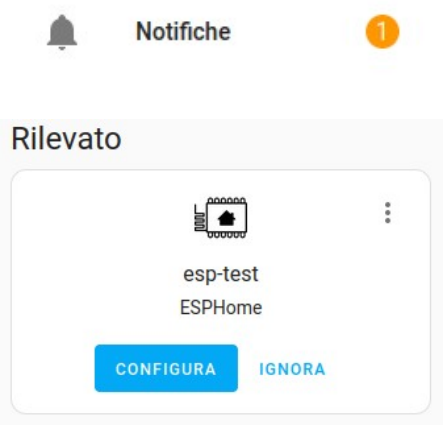
- Nel codice di spegnimento cambiano solo l'evento, la stringa ed il valore.
- Notate le differenze:

```
turn_off_action:
  - uart.write: 'LOFF\n'
  - globals.set:
      id: test_led_status
      value: 'false'
```

Collegamento con HomeAssistant

- Ora compileremo il codice con il comando **`esphome run esp-test.yaml`** che scaricherà ‘On the air’ il codice sl nostro ESP-01
- Passiamo su Home Assistant
- Scopriamo che compare una notifica
- E, passando al pannello delle notifiche troviamo il nostro EspHpme
- Possiamo ora andare a configurarlo inserendom la password che abbiamo configurato e troviamo nel pannello **`Panoramica`** il nostro interruttore che ci consente di accendere e spegnere il LED

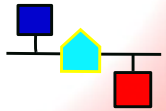


Aggiungiamo il tasto

- Come abbiamo detto, il tasto sarà un **`binary sensor`**, ma per gestirlo utilizzeremo un **`template`**.
- Prima di entrare nel codice, vediamo la definizione del nostro sensore:

```
binary_sensor:
  - platform: template
    name: "test_button"
    id: "button"
```

- Come si vede è un **`binary_sensor`** basato su un **`template`**, quindi script di nome **`test_button`** ed id **`test`**.

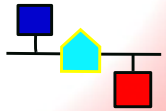


Cominciamo ad entrare nel codice

- Cominciamo a scrivere la funzione (**`lambda`**) che interroga il tasto e ne ritorna lo stato
- Per prima cosa la richiesta dello stato alla periferica

```
lambda: !lambda |-
    char msg[20];
    int i=0;
    id(serial).write_str("BTN\n");
```

- La prima riga indica che comincia lo script (**`lambda:`**) e che non è una sequenza di comandi ***yaml*** ma un sorgente C (**`!lambda |-`**)
- Dobbiamo leggere la stringa che ci ritorna la periferica quindi definiamo un buffer (**`char msg[20]`**) ed un indice (**`int i`**)
- Recuperiamo ora un riferimento all'istanza della seriale (**`id(serial)`**) e su questa invochiamo il metodo **`write_str`** per inviare la nostra stringa



Leggiamo la stringa

- Dobbiamo ora leggere dalla seriale la stringa, terminata con \n, gestendo anche un timeout per evitare errori.
- La scriveremo come se fosse codice per Arduino

```
for (i = 0; i < 20;) {
  for (int j = 0;; j++){
    if (j == 80) {
      return 0;
      ESP_LOGD("BTN","TIMEOUT %d", i);
    }
    if (id(serial).available()) {
      break;
    }
    delay (1);
  }
```

```
    id(serial).read_byte((unsigned char*)msg+i);
    if (msg[i++] == '\n') {
      break;
    }
  }
  msg[i] = 0;
```

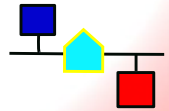
.

Faccio un loop per 20 caratteri (la lunghezza del buffer). All'interno faccio un altro loop attendendo un carattere disponibile, facendo una pausa e attendendo un massimo di 80 mS. Se il limite è raggiunto stampo un messaggio di log tramite la macro **ESP_LOGD**. Quando il carattere è disponibile lo leggo e se è '\n', interrompo il for. Alla fine termino la stringa.

Elaborazione del risultato

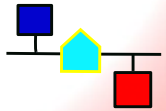
- Una volta recuperata la stringa in msg, la confronto con i messaggi validi e ritorno vero o falso a seconda del messaggio:

```
if (!strncmp(msg,"BON",3)) {
  return true;
} else if (!strncmp(msg,"BOFF",4)) {
  return false;
}
return 0;
```

Sitografia - 1

- ESP8266 Wikipedia: https://it.wikipedia.org/wiki/ESP8266
- Pagina di Espressif: https://www.espressif.com/en/products/socs/esp8266
- Home Assistant: https://www.home-assistant.io/
- Il sensore 18b20 con Arduino: https://randomnerdtutorials.com/guide-for-ds18b20-temperature-sensor-with-arduino/
- Installazione ambiente di sviluppo ESPHome: https://esphome.io/guides/installing_esphome.html
- Componenti EspHome https://esphome.io/components/



Sitografia - 2

- UART bus
  https://esphome.io/components/uart.html
- Template switch
  https://esphome.io/components/switch/template.html
- Template Binary Sensor
  https://esphome.io/components/binary_sensor/template.html
- Lambda
  https://esphome.io/cookbook/lambda_magic.html
- Programmazione UART (uart.h)
  https://github.com/esphome/esphome/blob/dev/esphome/components/uart/uart.h