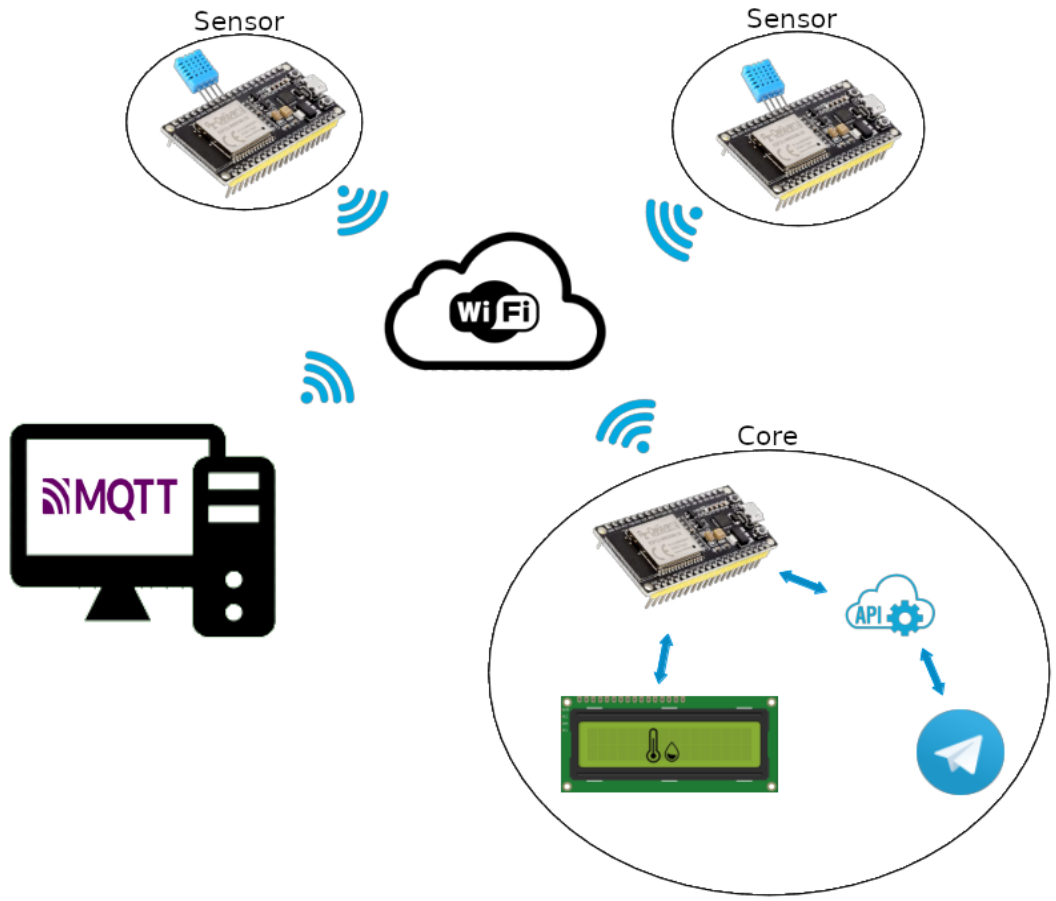
Progetto Arduino di Sistemi di Elettronica Digitale

ESP32 Wifi Thermostat

Realizzato da: Bernocchi Alessandro, Ceresara Gabriele, Cherubini Mattia

Introduzione

L'obiettivo del progetto è quello di realizzare un sistema di controllo distribuito di temperatura, umidità e luminosità.
Il sistema è composto da un Core e da uno o più moduli Sensor. I Sensor sono realizzati mediante un ESP32 (sul quale è eseguito un software chiamato "Client"), un sensore di temperatura e umidità DHT11 e da un fotoresistore GL55 per la misurazione dell'intensità luminosa. Il modulo Core invece è composto da un ESP32 (sul quale è eseguito un software chiamato "Gateway"), da un display LCD non I2C e da un insieme di bottoni per la realizzazione di un'interfaccia utente locale; il Core inoltre mette a disposizione un'altro interfacciamento, realizzato mediante bot Telegram.
Il Core è in grado di rilevare nuovi sensori e aggiungerli alla lista dei sensori connessi.
Per ogni sensore collegato è possibile visualizzare temperatura, umidità e luminosità tramite telegram. Il display LCD mostra le stesse informazioni con l'aggiunta dello stato della connessione e lo stato del relè, entrambi relativi al sensore attualmente selezionato. Sono inoltre presenti 2 led di segnalazione che si attivano quando l'umidità o la temperatura rilevati da un sensore superano una certa soglia.
Dal core è possibile settare per ogni sensore la temperatura di attivazione del relè.
Le diverse entità del sistema comunicano tramite la stessa Wireless LAN oppure attraverso una diversa rete Wireless, appoggiandosi sul protocollo MQTT.



Comunicazione

La comunicazione avviene tra il Core e i vari Sensor in modo bidirezionale (entrambi i moduli inviano e ricevono informazioni).

I vari Sensor inviano al Core ogni 7 secondi i valori di temperatura, umidità e luminosità rilevati tramite i loro sensori.
Il Core invia ai vari Sensor i valori di temperatura impostati dall'utente tramite l'interfaccia locale, utili ad accendere o spegnere il relè usato per simulare un sistema di riscaldamento. Tale dato viene mandato ad un Sensor specifico (non viene ricevuto quindi da tutti, ma solo dal Sensor selezionato dall'utente).

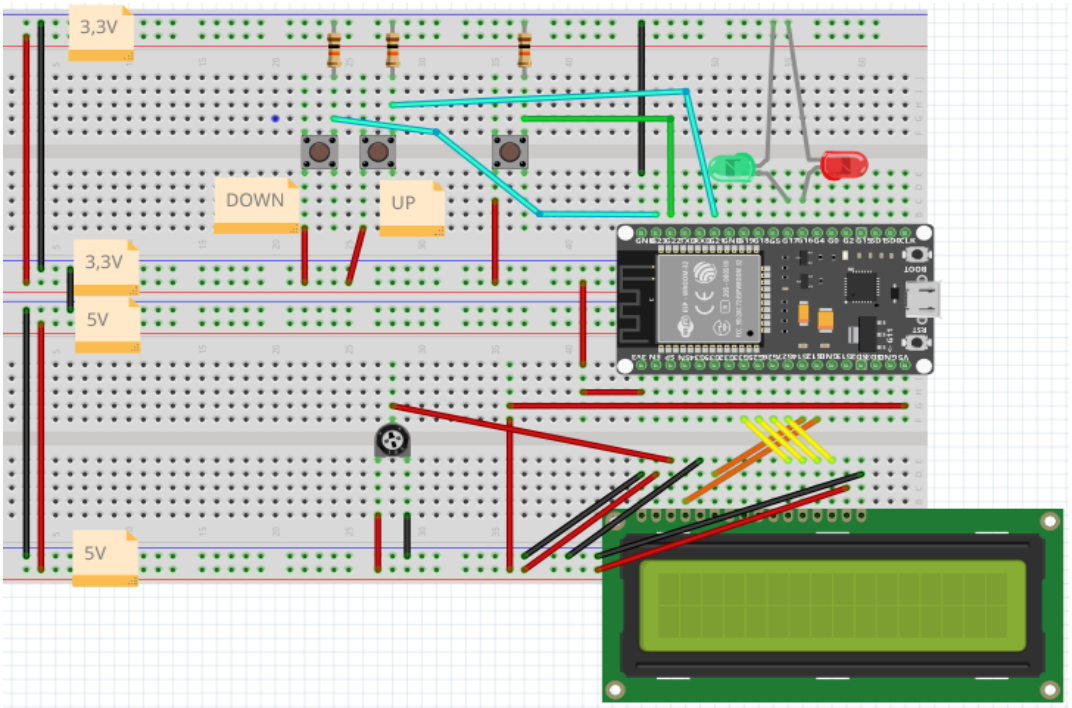
How to build it

Core (sistema di controllo dei sensori)

Lista componenti

- 1x ESP32
- 1x Display 16x02 (non I2C)
- 1x Potenziometro 10kΩ
- 3x Bottone NA
- 3x Resistenza 10kΩ
- 2x Led

Realizzazione circuitale



Sensore

Lista componenti

- 1x ESP32
- 1x Resistenza 10kΩ
- 1x Modulo DHT11
- 1x Fotoresistore GL55
- 1x Led

Fotoresistore GL55

Per la misurazione della luminosità ambientale si utilizza un fotoresistore, componente passivo che fa variare la sua resistenza inversamente alla quantità di luce presente.

In particolare, in condizioni di buio totale varia nell'ordine dei MOhm, mentre in un ambiente illuminato è nell'ordine dei KOhm .
Il sensore non può essere direttamente collegato all'ingresso del microcontrollore poiché esso è in grado di elaborare solamente dati in tensione. Va quindi inserito in un partitore di tensione con un resistore dal valore adeguato. (Nel nostro caso 10kΩ)
Non sapendo quale sensore della serie GL55 fosse a nostra disposizione è stato necessario stimare il valore di γ in modo da ricavare un coefficiente opportuno da utilizzare poi per la conversione del segnale in ingresso in Lux.
La lettura avviene tramite la funzione Arduino analogRead() la quale legge il valore del pin tramite l'attivazione di un ADC a 12 bit integrato nel microcontrollore in grado di restituirci
un valore nel range [0-4095].
Tramite il partitore di tensione si ricava la relazione che descrive la resistenza del componente che potrà essere convertita in Lux tramite le seguenti formule.

$$V_0 = 3,3V \frac{R_p}{R_p + 10K\Omega}$$

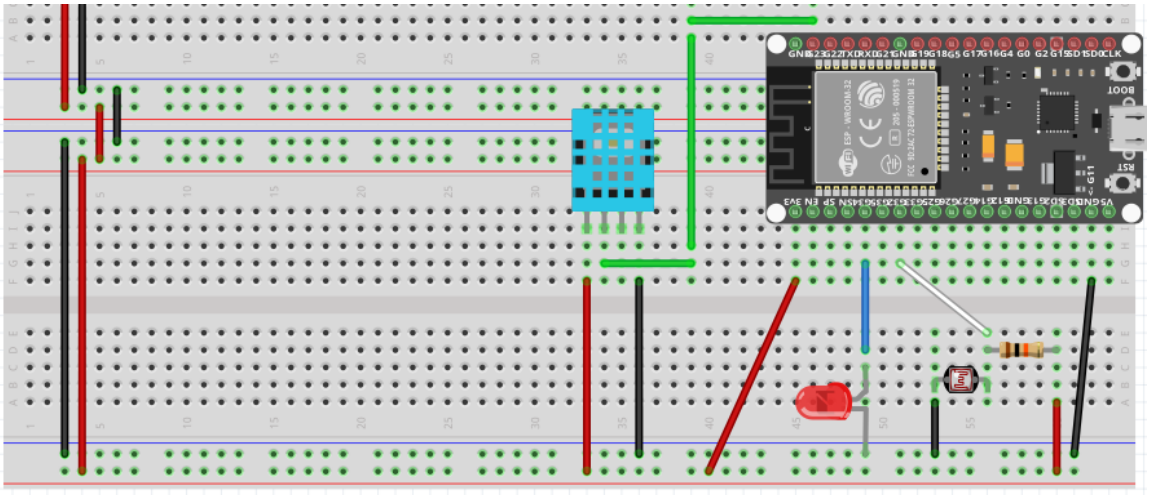
$$V_0(R_p + 10K\Omega) = 3,3R_p$$

$$R_p = V_0 \frac{10K\Omega}{3,3V - V_0}$$

$$\log(R_p)\text{-}\log(100K\Omega) = -\gamma_{10}^{100}[\log(lux) - \log(1)]$$

$$Luce[lux] = 10^{[(\frac{-1}{\gamma})\log(\frac{R_p}{100K})]}$$

Realizzazione circuitale



Communication Protocols

WIFI

La connessione al wifi è gestita tramite la libreria <Wifi.h>.
E' necessario però definire la modalità di utilizzo tramite il metodo WiFi.mode(WIFI_STA) poiché il microcontrollore può agire anche come Access Point.

MQTT

Il protocollo di comunicazione usato dai dispositivi appartenenti al sistema è MQTT, un protocollo di rete basato sul pattern publish-subscribe, dove lo scambio di informazioni avviene tramite

messaggi. Tale protocollo è ampiamente utilizzato in tutti gli ambiti nei quali esistono dei vincoli relativi alle risorse utilizzabili e dove la larghezza di banda è limitata.
MQTT definisce principalmente 2 entità: un message broker e dei client. Il broker è il server che riceve tutti i messaggi dai client e si occupa dell'inoltro delle informazioni alle destinazioni corrette, mentre i client sono un numero arbitrario di dispositivi che si collegano al broker (tramite la rete) per comunicare con il resto del sistema. Lo scambio di messaggi tra i vari client connessi al broker avviene tramite 2 azioni principali: publish e subscribe. Con l'esecuzione di una publish, un dispositivo manda informazioni; in questo caso è necessario specificare il topic sul quale eseguire l'azione e il messaggio da inviare. Sarà poi il broker ad inoltrare il messaggio a tutti i client che preventivamente avevano effettuato una subscribe nei confronti di quello specifico topic.
Nel nostro sistema l'uso del protocollo MQTT avviene tramite la libreria <PubSubClient.h>, la quale fornisce delle strutture dati e funzioni che permettono un'implementazione semplificata di tale protocollo. I metodi usati sono i seguenti:

PubSubClient client(WifiClient wifiClient)	Costruttore del client mqtt, al quale è necessario fornire il client wifi.
setServer(char* host. int port)	Imposta le informazioni relative al broker MQTT al quale ci si vuole collegare.
connect(char* name)	Si collega al broker MQTT specificato (necessario chiamare prima il metodo indicato sopra).
subscribe(char* topic)	Azione di subscribe al topic specificato come parametro
unsubscribe(char* topic)	Annulla l'azione di subscribe al topic specificato come parametro
publish(char* topic, char* msg)	Azione di publish del messaggio sul topic specificato
setCallback(function* callback)	Imposta la funzione di callback del client. Essa è chiamata ogni volta che arriva un messaggio su uno dei topic ai quali il client ha eseguito l'azione di subscribe. La funzione di callback ha come parametri un char* dove viene specificato il topic dal quale proviene il messaggio, e un char*, il quale rappresenta il messaggio vero e proprio

Nel sistema i client effettuano delle publish su topic che presentano un pattern simile: {device_id}/{data}, ovvero nella prima parte indicano il loro id (per distinguersi dagli altri dispositivi) mentre nella seconda il tipo di dato che è presente nel messaggio (nel nostro caso vengono inviate temperatura, umidità e luminosità).

Breve nota sulla sicurezza

Per semplificare la visione del progetto funzionante in università abbiamo utilizzato un MQTT Broker online (www.mqtt-dashboard.com) che non richiede credenziali. Di conseguenza:

- I messaggi inviati dai sensori sono non cifrati e visibili a tutti
- Non c'è autenticazione tra sensori e sistema centrale pertanto chiunque scriva sullo stesso Topic MQTT può introdurre dati errati che poi verranno visualizzati senza nessun controllo.

In un'applicazione reale del progetto i sensori pubblicano i loro messaggi cifrati su un Broker MQTT Privato tramite autenticazione (user e password). In questo modo si risolvono entrambi i problemi citati prima.

Telegram

La comunicazione con Telegram è gestita tramite un BOT. L'utente può inviare i comandi nella chat del BOT. Se l'utente è autorizzato gli verrà inviato un messaggio contenente una visione dei dati inviati al Core da parte dei Client MQTT al momento connessi, in caso contrario riceverà un messaggio di errore.

Lista dei comandi validi:

/start	Utente autorizzato avvia la comunicazione
/sensors	Visualizza la lista di sensori al momento connessi
/RoomN	Visualizza i dati di uno degli N sensori connessi

L'interazione è resa possibile dalle librerie **<UniversalTelegramBot.h>** e **<ArduinoJSON.h>** tramite il seguente procedimento. Quando una richiesta da parte di un utente autorizzato ha successo le informazioni relative al bot, alla chat ed allo stesso utente vengono salvate in un oggetto con il formato JSON. La comunicazione è gestita tramite un'apposita libreria che ci fornisce i seguenti metodi:

UniversalTelegramBot bot (BOTtoken, BOTclient)	Costruttore del bot con il suo token ed il client Wi-Fi
getUpdates (int msg)	Restituisce l'aggiornamento di quanti messaggi sono stati recentemente ricevuti
sendMessage (chat_id, message)	Invia un messaggio "message" sulla chat "chat_id"
bot.messages[int i]	Posizione specifica del messaggio scambiato nell' array che li contiene
chat_id	Identificazione dell'utente
text	Contenuto testuale del comando
from_name	Contiene l'username dell'utente che invia la richiesta

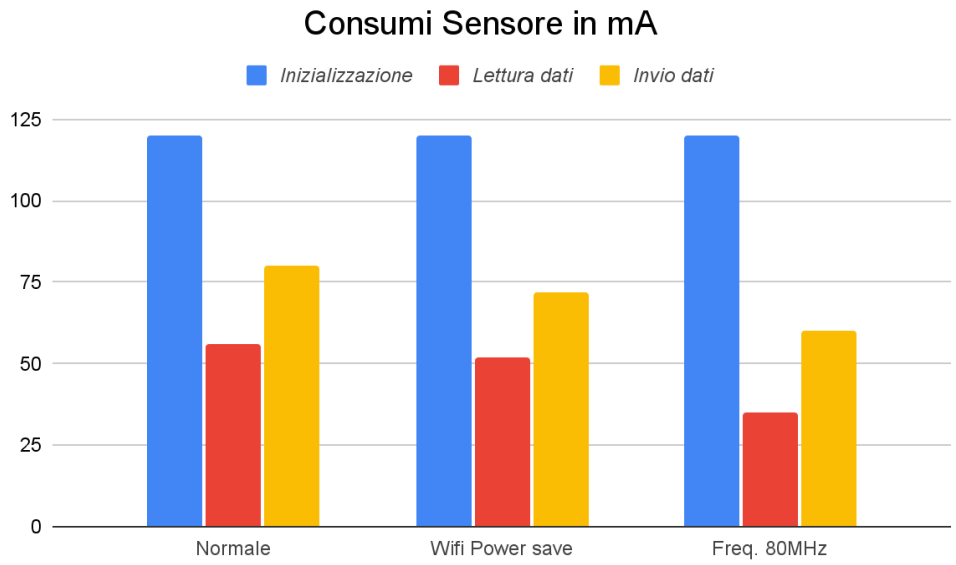
MultiCore

Gli ESP-32 impiegati nel sistema sono tutti dotati di un processore dual-core, grazie al quale possono essere eseguiti più processi contemporaneamente per migliorare le prestazioni di ogni dispositivo. Questa funzionalità è utilizzabile grazie alla presenza di un sistema operativo real time (FreeRTOS) tramite il quale, con opportune API, è possibile creare dei task specificando il core sul quale devono essere eseguiti. La presenza di un RTOS permette anche di assegnare delle priorità ai vari task, che verranno gestite da quest'ultimo. La funzione usata per la generazione e l'esecuzione dei vari task è *xTaskCreatePinnedToCore* (documentazione delle API messe a disposizione da FreeRTOS qui).
Nel nostro caso, il dual core è stato sfruttato dal sistema di controllo per gestire principalmente i task relativi alla scrittura su display LCD e alla gestione del bot Telegram, i 2 computazionalmente più "pesanti". Ad essi è stato assegnato un core separato ed una priorità maggiore, per fare in modo che l'interazione con l'utente sia il più veloce possibile (a discapito di altri task reputati meno importanti, come ad esempio la gestione degli interrupt generati dall'arrivo di un messaggio MQTT).

Power Save

Il sistema di controllo utilizza 2 core e sfrutta al massimo le potenzialità del processore, inoltre ha un display sempre attivo, per questo si presuppone che sia alimentato tramite apposito alimentatore.

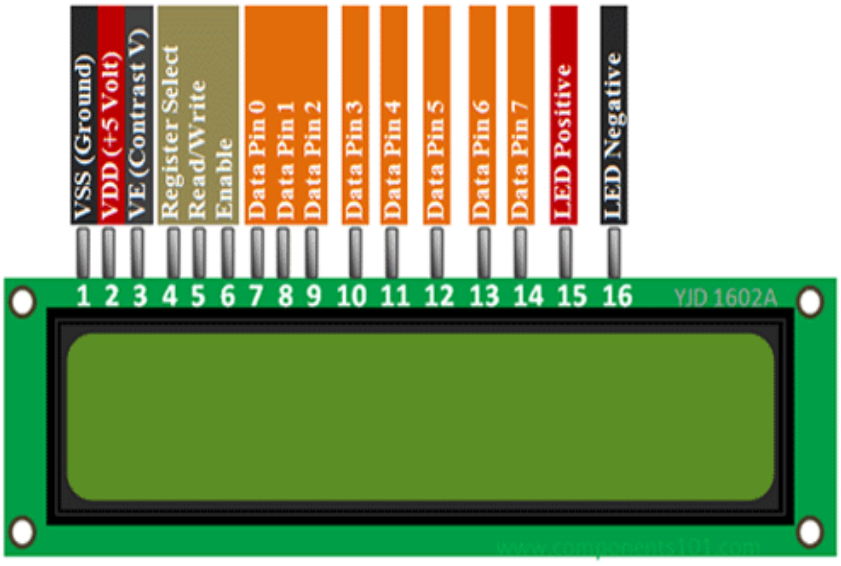
Per quanto riguarda i sensori, si cerca invece di ridurre i consumi per un possibile utilizzo a batteria.
Si utilizza quindi la funzione *esp_wifi_set_ps(WIFI_PS_MIN_MODEM)* la quale rallenta la frequenza di arrivo dei messaggi Beacon con la stessa frequenza del periodo DTIM.
E' necessario inoltre diminuire la frequenza del clock del microcontrollore con la funzione *setCpuFrequencyMhz(int frequency)*. Nel nostro caso la frequenza del clock viene abbassata da 240MHz a 80MHz.
Risultati ottenuti:



(documentazione API di ESPRESSIF qui)

Display LCD

Per la visualizzazione "in locale" dei dati è stato usato un display 16x2 non I2C che dispone di 16 colonne e 2 righe. Indicata qui sotto una tabella con i diversi pin ed uno schema:



La programmazione avviene tramite la libreria **<LiquidCrystal.h>** della quale abbiamo utilizzato i seguenti metodi:

LiquidCrystal lcd(rs, enable, d4, d5, d6, d7)	Permette di creare l'oggetto inizializzando i pin connessi
lcd.begin(col, row)	Inizializzo la righe e le colonne scrivibili
lcd.setCursor(col, row)	Setto il cursore in un punto del display

lcd.print(data)	Scrive la stringa “data” dalla posizione del cursore
lcd.createChar(num, Byte data[8])	Crea un char personalizzato assegnando un numero [0-7]
lcd.write(data)	Scrive il carattere “data” dalla posizione del cursore

Il sistema può trovarsi in uno di due stati:

- **WAITING:** In attesa che qualcuno si colleghi mostra l’interfaccia base, con gli slot dei dati vuoti.
- **DISPLAYING:** Con uno o più Sensori connessi mostra i dati ricevuti da uno di essi.

Poiché il sistema è scalabile fino ad un massimo di 10 Sensori ed è possibile visualizzare sullo schermo le informazioni relative ad un solo client per volta il cambio di visualizzazione della “stanza” è gestito tramite un bottone.

Sistema di Allarmi

Il sistema di allarmi è composto da due LED sulla stazione centrale:

- LED Rosso: si accende se la Temperatura supera la soglia definita
- LED Verde: si accende se l’Umidità supera la soglia definita

Il led viene settato con la funzione **digitalWrite(pin, HIGH/LOW)** la quale dà un uscita digitale sul pin settato come uscita con la funzione **pinMode(pin, OUTPUT)**, attivando il buffer Three-State posto sul GPIO selezionato.

Regolazione temperatura

Ogni sensore ha un led il quale si attiva quando la temperatura letta è inferiore ad una soglia. Tale soglia è definibile soltanto dal Core tramite appositi bottoni. Una volta settata, verrà inviata tramite MQTT al sensore.
Per semplicità il led rappresenta un relè. l’attivazione e la disattivazione corrispondono al funzionamento di un termostato.