



Scuola Politecnica e delle Scienze di Base  
Corso di Laurea Magistrale in Ingegneria Informatica

Elaborato Computer System Design

## *ESP32 Thermostat*

Anno Accademico 2022/2023

Alessio Carusio  
matr. M63001145

# Indice

<b>1</b>	<b>Introduzione</b>	<b>2</b>
<b>2</b>	<b>Hardware Utilizzato</b>	<b>4</b>
2.1	Microcontrollore . . . . .	5
2.1.1	Programmazione ESP32 . . . . .	6
2.2	Sensore di temperatura ed umidità . . . . .	8
2.3	Modulo Relay . . . . .	9
2.4	Display . . . . .	10
2.5	Collegamenti hardware . . . . .	16
<b>3</b>	<b>Sviluppo Software</b>	<b>17</b>
3.1	Debouncing . . . . .	17
3.2	Timer Interrupts . . . . .	20
3.2.1	Configurazione interruzione . . . . .	21
3.2.2	Definizione ISR . . . . .	23
3.3	Button Interrupts . . . . .	24
3.4	Display . . . . .	26

# Elenco delle figure

1.1	Termostato basato su Esp32 durante lo sviluppo . . . . .	2
1.2	Termostato basato su Esp32 . . . . .	3
2.1	ESP32 Dev Kit . . . . .	5
2.2	CP2102 . . . . .	7
2.3	DHT22 . . . . .	8
2.4	Modulo Relay . . . . .	9
2.5	LCD 1602 . . . . .	10
2.6	Display Pixel . . . . .	11
2.7	Adattatore LCD I2C . . . . .	13
2.8	PCF8574 I2C Registro di indirizzo . . . . .	14
2.9	PCF8574 I2C Pin di selezione dell'indirizzo . . . . .	15
2.10	Collegamenti Hardware . . . . .	16
3.1	Fenomeno di bouncing . . . . .	18
3.2	Timers ESP32 . . . . .	20

# Capitolo 1

## Introduzione

Il progetto realizzato è un termostato basato su ESP32 (Espressif WROOM-32 Developer Kit), che utilizza un sensore di temperatura ed umidità e mostra i valori su un display lcd.



Figura 1.1: Termostato basato su Esp32 durante lo sviluppo



Figura 1.2: Termostato basato su Esp32

Attraverso i pulsanti "+" e "-" è possibile regolare la temperatura target che si vuole ottenere in un determinato ambiente. Sulla base della temperatura ambiente attuale, il sistema deciderà se attivare o disattivare il sistema di riscaldamento.

Per lo sviluppo del progetto applicazioni è stato scelto l'IDE Arduino per la sua semplicità e compatibilità; infatti, la community di utenti Arduino è molto attiva e supporta piattaforme come ESP32.

## Capitolo 2

# Hardware Utilizzato

- Microcontrollore: ESP32 Dev Kit C V2
- Sensore di temperatura ed umidità: DHT22
- Pulsanti
- Relay Board: 4R1B
- Display: HD44780 1602 LCD
- I2C Adapter: I2C Adapter Serial interface for LCD Display 1602
- Breadboard

## 2.1 Microcontrollore

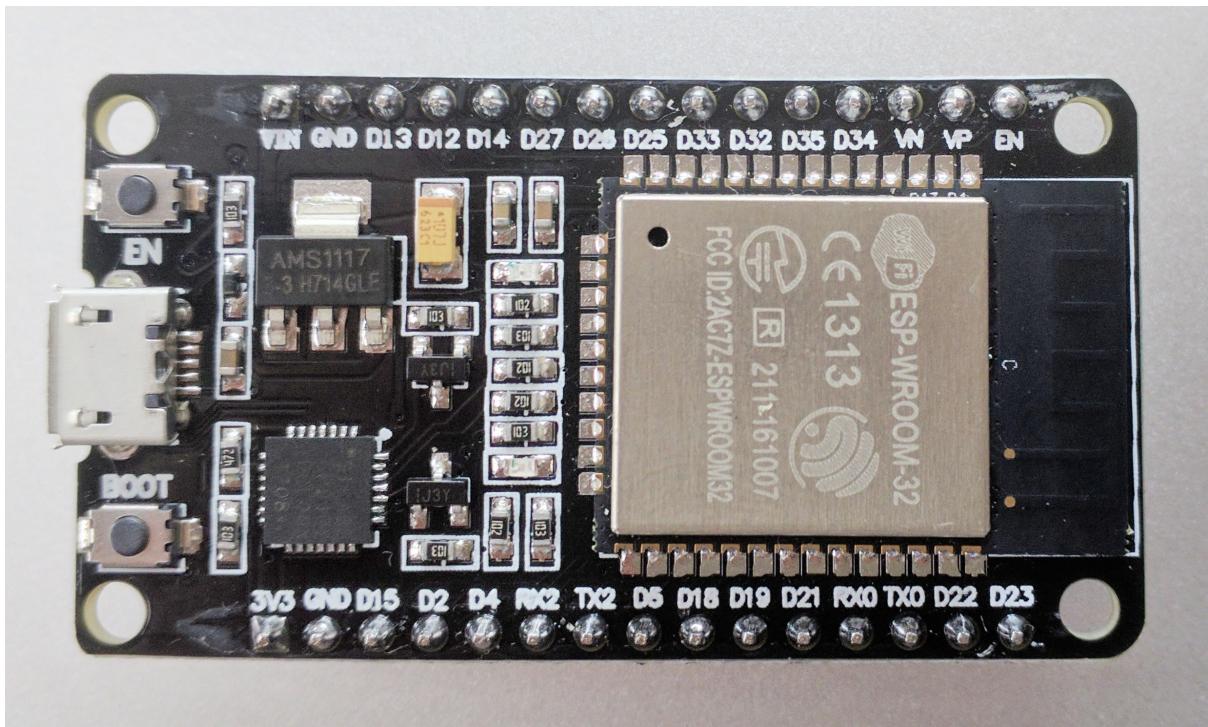


Figura 2.1: ESP32 Dev Kit

L'ESP32 Dev Kit C V2 è una scheda di sviluppo creata attorno al chip ESP32 WROOM 32, contenente un regolatore tensione (AMS1117), un circuito programmatore USB per il chip ESP32 (CP2102) e molte altre funzionalità. La serie ESP32 WROOM 32 di chip Wi-Fi è prodotta da Espressif Systems, azienda cinese con sede a Shanghai. ESP32 WROOM-32 è un modulo Wi-Fi conveniente adatto per progetti fai da te nel campo dell'Internet of Things (IoT). Questo modulo viene fornito con molti GPIO e supporto per una varietà di protocolli come SPI, I2C, I2S, UART e altro. Inoltre, è dotato di una rete wireless inclusa, il che lo rende diverso da altri microcontrollori come Arduino. Ciò significa che può facilmente controllare e monitorare i

dispositivi da remoto tramite Wi-Fi e Bluetooth.

Il core del processore, chiamato LX6 da Espressif, si basa sul controller del processore Xtensa dualcore LX6 a 32 bit e funziona a una gamma di frequenze compresa tra 80 e 240 MHz. Ha una ROM di avvio da 448 kB, 520 kB di SRAM su chip e 4 MB di memoria flash esterna a cui è possibile accedere tramite l’interfaccia SPI.

ESP32 WROOM 32 è dunque un system-on-chip (SoC) che integra un microcontrollore Tensilica a 32 bit, interfacce periferiche digitali standard, interruttori d’antenna, balun RF, amplificatore di potenza, amplificatore di ricezione a basso rumore, filtri e moduli di gestione dell’alimentazione in un piccolo pacchetto.

### 2.1.1 Programmazione ESP32

L’ESP 32 può essere programmato tramite l’interfaccia UART; per farlo, è necessario impostare l’ESP 32 in modalità di programmazione, abbassando il pin GPIO 0. Fatto ciò è necessario riavviare la scheda mantenendo basso il pin GPIO 0. Dopodiché l’ESP 32 è in modalità di programmazione e può essere programmato tramite UART.

Esistono molti circuiti diversi per programmare un ESP 32 ma uno dei più utilizzati è costituito da:

1. un bridge USB-UART CP2102
2. un circuito che imposta l'ESP 32 in modalità di programmazione

ed un'opzione per alimentare il circuito tramite USB o un alimentatore esterno.

Nel dettaglio, il CP2102 è un controller bridge integrato da USB a UART che fornisce una soluzione semplice per l'aggiornamento dei progetti RS232 a USB utilizzando un minimo di componenti e spazio PCB. Il CP2102 include un controller di funzione USB 2.0, un ricetrasmettitore USB, un bus dati seriale asincrono (UART) con segnali di controllo ed altri componenti.

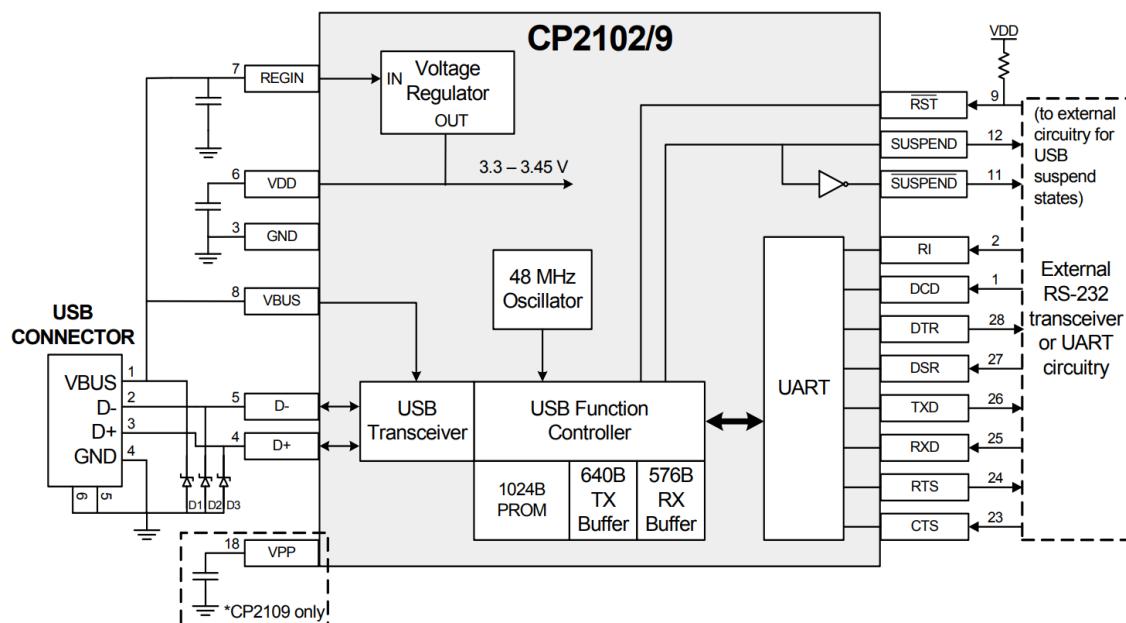


Figura 2.2: CP2102

## 2.2 Sensore di temperatura ed umidità

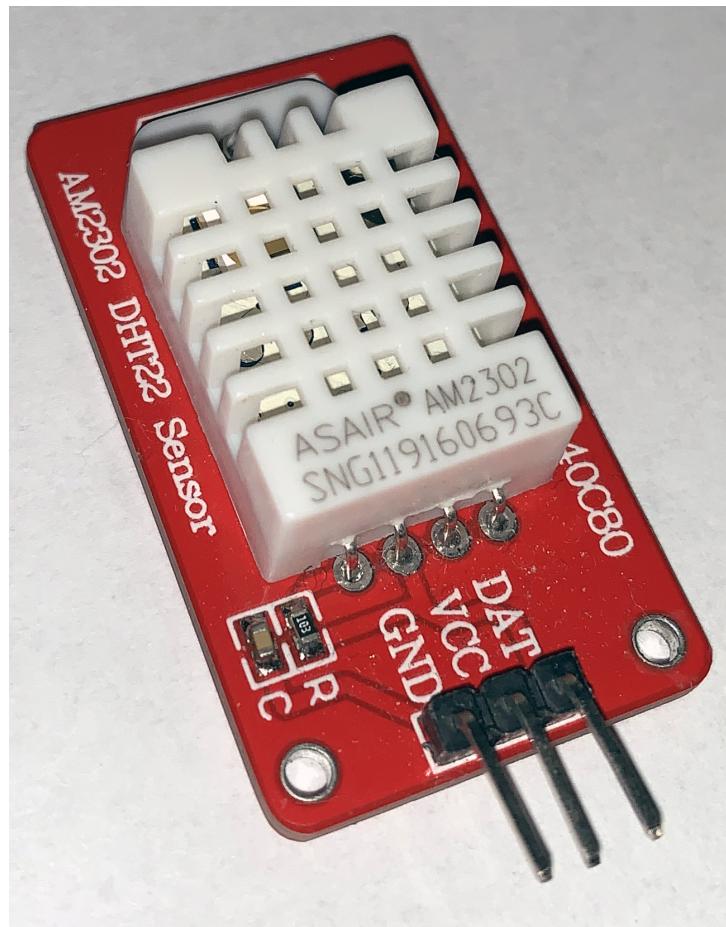


Figura 2.3: DHT22

Il DHT22 è un sensore digitale di temperatura e umidità di base a basso costo. Utilizza un sensore di umidità capacitivo e un termistore (resistore il cui valore di resistenza varia in maniera significativa con la temperatura) per misurare l'aria circostante ed emette un segnale digitale sul pin dati.

Il sensore utilizzato monta un resistore di pull up che garantisce un livello logico valido quando i pin passano dall' input all'output, con un conseguente

funzionamento corretto. Il resistore da  $4.7K - 10K\Omega$  è inserito tra il pin DAT e il pin VCC.

## 2.3 Modulo Relay

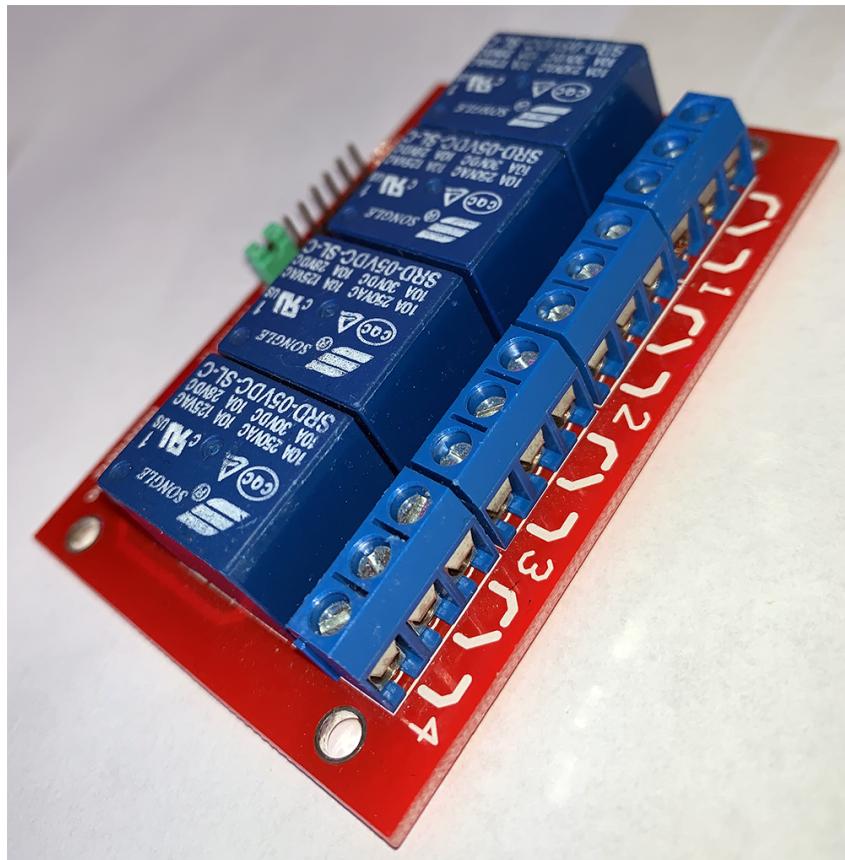


Figura 2.4: Modulo Relay

Il modulo relè a 4 canali può essere utilizzato per controllare vari apparecchi con grande corrente direttamente da un microcontrollore, in quanto possiede un'interfaccia standard. È alimentato a 5v e ogni canale necessita di una corrente di pilotaggio di 15-20 mA.

Ogni relè ad alta corrente che funzionano sotto AC250V 10A o DC30V 10A. Questo modulo è otticamente isolato dal lato ad alta tensione per i requi-

siti di sicurezza e impedisce anche il "ground loop" quando si interfaccia al microcontrollore.

Tale dispositivo viene utilizzato, nel progetto, per attivare o disattivare il riscaldamento.

### 2.4 Display

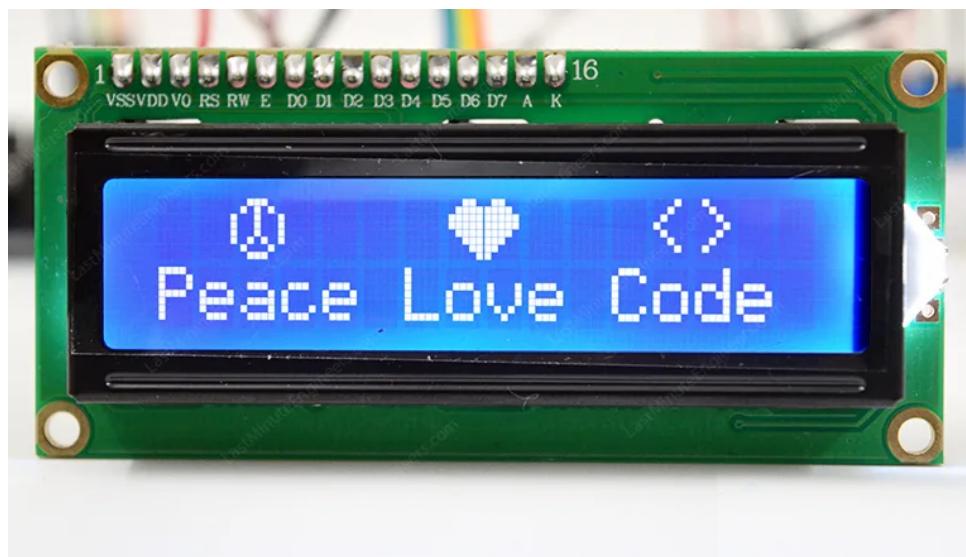


Figura 2.5: LCD 1602

Il 1602 LCD (Liquid Crystal Display) è un tipo di dispositivo elettronico utilizzato per visualizzare messaggi e dati.

Il display è chiamato 1602 perché ha 16 colonne e 2 righe in cui può essere visualizzato un totale di ( $16 \times 2 = 32$ ) 32 caratteri, ognuno costituito da  $5 \times 8$  Pixel, per un totale di  $(32 \times 40)$  1280 pixel.

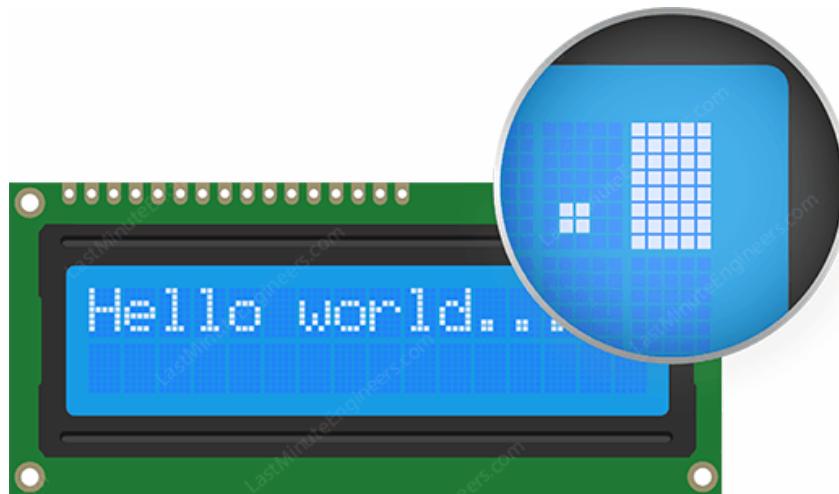


Figura 2.6: Display Pixel

Questi display si basano principalmente su diodi emettitori di luce.

Gli LCD standard possono essere collegati ad un microcontrollore mediante un data bus parallelo, sfruttando 16 diversi segnali:

- **GND** è il pin di collegamento a massa.
- **VCC** è l'alimentazione per l'LCD che colleghiamo al pin 5V sul microcontrollore.
- **Vo (LCD Contrast)** controlla il contrasto e la luminosità del display LCD. Utilizzando un semplice partitore di tensione con un potenziometro, possiamo effettuare regolazioni precise del contrasto.
- Il pin **RS (Register Select)** è impostato su LOW quando si inviano comandi all'LCD (come l'impostazione del cursore su una posizione specifica, la cancellazione del display, ecc.) e HIGH quando si inviano dati all'LCD. Fondamentalmente questo pin viene utilizzato per separare il comando dai dati.

- Il pin **R/W (lettura/scrittura)** consente di leggere i dati dall'LCD o di scrivere dati sull'LCD. Poiché stiamo utilizzando questo LCD solo come dispositivo di output, imposteremo questo pin LOW. Questo lo forza in modalità WRITE.
- Il pin **E (Enable)** viene utilizzato per abilitare il display. Quando questo pin è impostato su LOW, l'LCD non si preoccupa di ciò che sta accadendo sulle linee R/W, RS e del bus dati. Quando questo pin è impostato su HIGH, il display LCD elabora i dati in arrivo.
- I pin **D0-D7 (Data Bus)** trasportano i dati a 8 bit che inviamo al display. Ad esempio, se vogliamo vedere un carattere "A" maiuscolo sul display, impostiamo questi pin su 0100 0001 (come da tabella ASCII).
- I pin **A-K (anodo e catodo)** vengono utilizzati per controllare la retroilluminazione del display LCD.

Il collegamento in parallelo di un display 16x2 ad un microcontrollore necessita di molti pin. La soluzione a tale problema è quella di utilizzare un collegamento di tipo seriale basato sul protocollo hardware I2C. Un tipico display LCD I2C è costituito da un display LCD a caratteri basato su HD44780 e da un adattatore LCD I2C.

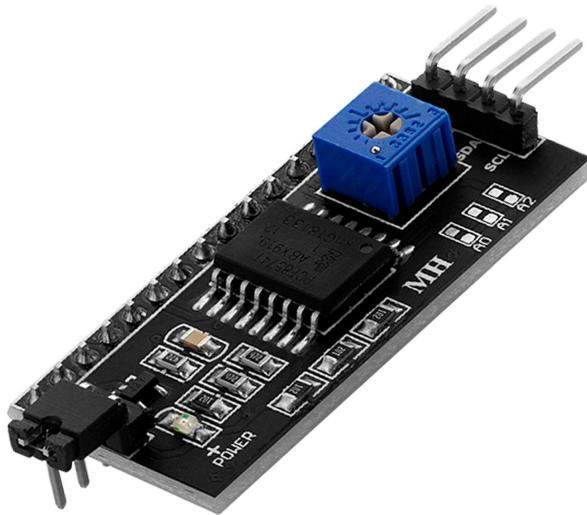


Figura 2.7: Adattatore LCD I2C

Quest’ultimo ha come elemento fondamentale un chip di espansione I/O a 8 bit, denominato PCF8574. Questo chip converte i dati I2C da un microcontrollore nei dati paralleli richiesti per un display LCD. La scheda è inoltre dotata di un piccolo potenziometro per effettuare regolazioni fini al contrasto del display.

Se si utilizzano più dispositivi sullo stesso bus I2C, potrebbe essere necessario impostare un indirizzo I2C diverso per l’adattatore LCD in modo che non entri in conflitto con un altro dispositivo I2C. Per fare ciò, l’adattatore dispone di tre ponticelli a saldare (A0, A1 e A2), ognuno dei quali viene utilizzato per codificare l’indirizzo. Se un jumper viene cortocircuitato con un punto di saldatura, viene impostato fisicamente l’indirizzo.

Poichè diverse aziende producono lo stesso chip PCF8574, ad esempio *Texas Instruments* e *NXP Semiconductors*, l’indirizzo I2C del display LCD dipende

dal produttore del chip. In particolare, avendo utilizzato un chip prodotto da *Texas Instruments*, i tre bit di selezione dell'indirizzo (A<sub>0</sub>, A<sub>1</sub> e A<sub>2</sub>) sono posizionati alla fine del registro degli indirizzi I<sub>2</sub>C a 7 bit.



Figura 2.8: PCF8574 I<sub>2</sub>C Registro di indirizzo

Essendoci 3 bit di selezione dell'indirizzo I<sub>2</sub>C, che possono assumere stato Low o High, è possibile creare 8 diverse combinazioni di indirizzi.

Per impostazione predefinita, tutti e 3 i bit di selezione vengono impostati ad "1", utilizzando delle resistenze di pullup integrate e fornendo al PCF8574 un indirizzo I<sub>2</sub>C predefinito di 0100111Binary o 0x27Hex.

Se si utilizzano più dispositivi sullo stesso bus I<sub>2</sub>C, potrebbe essere necessario impostare un indirizzo I<sub>2</sub>C diverso per l'adattatore LCD in modo che non entri in conflitto con un altro dispositivo I<sub>2</sub>C. In tal caso, è possibile cortocircuitare alcuni o tutti i pin di selezione dell'indirizzo con un intervallo compreso tra 0x20 a 0x27.

## CAPITOLO 2. HARDWARE UTILIZZATO

---

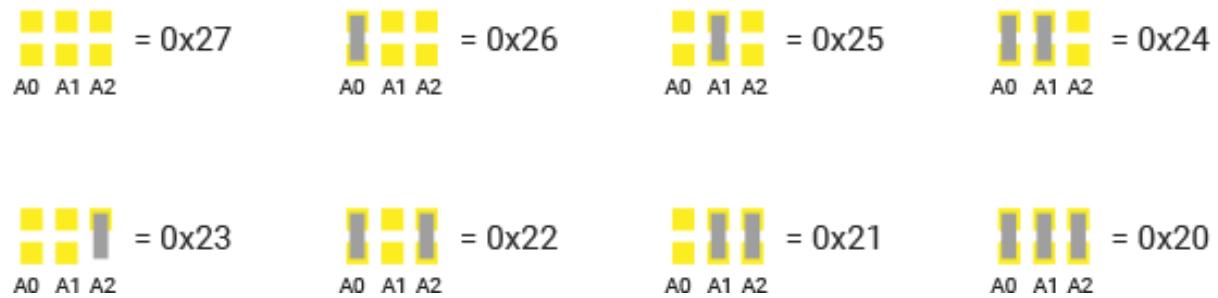


Figura 2.9: PCF8574 I2C Pin di selezione dell'indirizzo

## 2.5 Collegamenti hardware

I collegamenti hardware realizzati sono illustrati nella figura seguente:

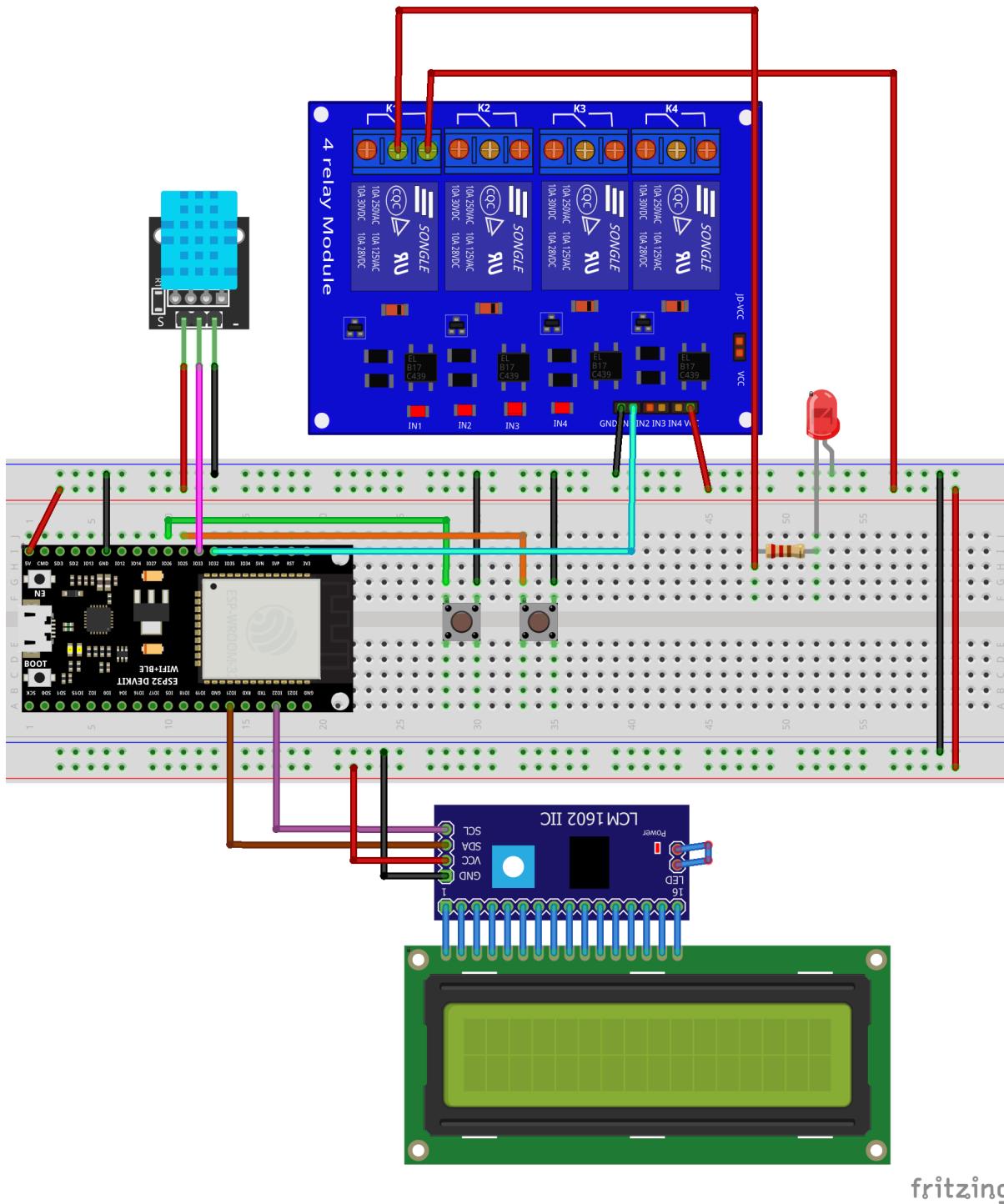


Figura 2.10: Collegamenti Hardware

# Capitolo 3

## Sviluppo Software

Il progetto è stato realizzato utilizzando l'IDE Arduino in linguaggio C/C++.

### 3.1 Debouncing

Il problema del "bounce" si presenta quando all'interno di un circuito si hanno dei pulsanti o degli interruttori. Esso consiste in più commutazioni non volute che si verificano quando in realtà si vorrebbe una singola commutazione di un segnale. Ad esempio, ipotizzando di voler una singola commutazione logica bassa-alta alla pressione di un pulsante, a causa della meccanica dello stesso si otterebbe un segnale di questo tipo:

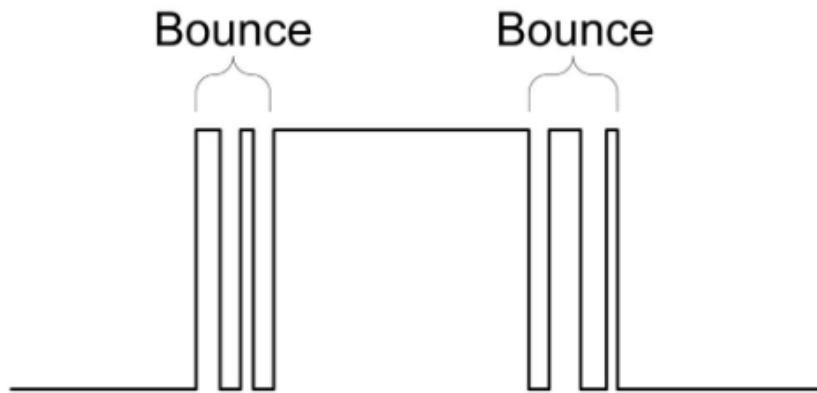


Figura 3.1: Fenomeno di bouncing

L'origine di tali disturbi è dovuta al fatto che la lamina di contatto non garantisce, una volta commutata, un impulso immediatamente stabile. Tale problema ha una durata che può variare in base al pulsante utilizzato. È possibile risolverlo sia tramite soluzioni hardware che software.

- **Tramite l'hardware:** aggiungendo un filtro RC appropriato per agevolare la transizione.
- **Tramite software:** ignorando temporaneamente ulteriori interrupt per un breve periodo di tempo dopo l'attivazione del primo interrupt.

La soluzione scelta è quella software, sia perchè non sono necessari componenti hardware, sia perchè perché non tutti gli interruttori rimbalzano allo stesso modo. Anche due interruttori di tipo identico e dello stesso lotto di produzione possono comportarsi in modo molto diverso, quindi un'unica soluzione valida per tutti potrebbe non funzionare per alcuni interruttori particolarmente "rimbalzanti".

L’obiettivo base di un debouncer è silenziare l’input dal pulsante dopo che il codice ha rilevato un singolo cambiamento di stato. Per farlo, si impone di voler accettare solo un campione prelevato almeno **250 millisecondi** (parametro definito nella documentazione ufficiale dell’ ESP32) dopo l’ultimo campione.

Quando il pulsante viene inizialmente premuto, si registra il tempo del contatto utilizzando il clock interno del microcontrollore con la funzione *millis()* e si memorizza tale informazione nella variabile *button\_time*.

Nell’ISR relativa alla pressione del bottone, come prima cosa l’ora corrente della pressione viene sottratta dall’ultima volta che è stato ricevuto un input legittimo, controllando quindi se l’intervallo di tempo è maggiore della soglia preimpostata di 250 millisecondi, denominata *debounceDelay*.

```
if (button_time - last_button_time > debounceDelay) {  
    ...  
}
```

Tale condizione rappresenta il filtro, che blocca il rumore di un pulsante che "rimbalza". Una volta trascorso un periodo di tempo ragionevole, si accetta l’input e si inizia ad elaborarlo.

## 3.2 Timer Interrupts

Gli interrupt del timer sono delle interruzioni software generate da un timer e rappresentano un modo efficace per garantire che gli eventi temporizzati si verifichino al millisecondo, consentendo operazioni ottimizzate. Le timer-interrupts consentono di eseguire un'attività a intervalli di tempo molto specifici, indipendentemente da cos'altro sta accadendo nel sistema. Tali tipologie di interruzione sono simili agli interrupt esterni, ma invece di essere attivati da un evento esterno, vengono attivati da un timer.

L'attivazione di una timer interrupt interrompe il thread di esecuzione dopo il completamento dell'istruzione corrente, esegue l'ISR associata e ritorna all'istruzione successiva da dove era stata interrotta, proprio come un'interrupt hardware o esterno.

Il chip ESP32 contiene due gruppi di timer hardware, ciascuno dei quali dispone di due timer hardware identici. Tutti i timer sono generici, a 64 bit e basati su prescaler a 16 bit e possono contare ad incremento o a decremento.



Figura 3.2: Timers ESP32

Poiché tali timer sono hardware, tutti i tempi sono correlati al clock della

scheda (80MHz). La velocità del timer può essere determinata dalla seguente formula:

$$VelocitaTimer(Hz) = \frac{ClockESP32(Mhz)}{Prescaler}$$

Utilizzando un prescaler di 80 e considerando 80MHz la frequenza del microcontrollore, si ottiene una frequenza del timer pari ad 1MHz. Per poter poi contare effettivamente ogni secondo, si specifica di voler avere un'interruzione ogni  $1 * 10^6$  microsecondi.

### 3.2.1 Configurazione interruzione

Per configurare il timer, è stato utilizzato un puntatore a una variabile di tipo **hw\_timer\_t**.

```
My_timer = timerBegin(0, 80, true);
timerAttachInterrupt(My_timer, &onTimer, true);
timerAlarmWrite(My_timer, 1000000, true);
timerAlarmEnable(My_timer);
```

Per inizializzare il timer, è stata usata la funzione timerBegin che prende in input il numero del timer da utilizzare (da 0 a 3), il valore del prescaler ed un flag che indica se il contatore deve contare ad incremento (true) o a

decremento (false).

In tale progetto si è scelto di utilizzare il timer 0 con un prescaler di 80 ed un contatore ad incremento.

In seguito, con la funzione `timerAttachInterrupt` viene collegato il timer all'ISR *onTimer*, che verrà eseguita quando viene generato l'interrupt.

La funzione *timerAlarmWrite* viene utilizzata per specificare il valore del contatore a cui deve essere generato l'interrupt del timer. Si è pensato, per scopi didattici, di voler generare un interrupt ogni secondo. Per fare ciò, viene passato al timer il valore di 1000000 microsecondi, che corrispondono, appunto, ad 1 secondo. Il terzo argomento, impostato a true è un flag che specifica che l'interrupt dev'essere eseguito periodicamente e che il contatore deve ripartire ogni volta che raggiunge il conteggio massimo.

Infine, si procede ad abilitare l'interruzione del timer utilizzando la funzione `timerAlarmEnable`.

### 3.2.2 Definizione ISR

```
void IRAM_ATTR onTimer() {  
    expiredTimer = true;  
  
    if (thermostat.target_temp > dht.readTemperature())  
        ↪ ) {  
  
        if (digitalRead(RELAY_PIN) != 1) {  
            digitalWrite(RELAY_PIN, HIGH);  
            digitalWrite(LED_PIN, HIGH);  
            onActivationThermostat = true;  
        }  
  
    } else if (digitalRead(RELAY_PIN) != 0) {  
        digitalWrite(RELAY_PIN, LOW);  
        digitalWrite(LED_PIN, LOW);  
        onDeactivationThermostat = true;  
    }  
}
```

La logica alla base del termostato realizzato è la seguente: se la temperatura ambiente è più bassa di quella target, il riscaldamento viene acceso, altrimenti viene spento.

Ogni operazione di attivazione e disattivazione del termostato porta all'aggiornamento del display che visualizzerà un opportuno messaggio di "Riscal-

damento acceso" o "Riscaldamento spento".

È possibile notare, nell'ISR, la presenza di una macro **IRAM\_ATTR** che consente di forzare il codice in IRAM invece che in memoria flash. Tipicamente, lo script del linker inserirà tutto il codice in flash memory e tutte le variabili nella RAM condivisa.

Nell' ESP32 l'accesso alla memoria flash è molto più lento dell'accesso alla RAM, motivo per il quale esiste una cache di memoria che può essere utilizzata per risolverne una parte, senza però garanzie di un cache hit. Dunque, un codice eseguito da memoria flash, potrebbe richiedere un accesso alla flash memory e dunque tempi lenti di caricamento.

### 3.3 Button Interrupts

Per aumentare o diminuire la temperatura target del termostato sono stati utilizzati due bottoni a cui sono state associate delle interruzioni hardware:

```
void IRAM_ATTR increaseISR() {
    button_time = millis();

    if (button_time - last_button_time > debounceDelay
        ) {
        if (thermostat.target_temp < 30) {
            thermostat.target_temp++;
        }
    }
}
```

```
increaseButton.pressed = true;
last_button_time = button_time;

} else {
    increaseButton.pressed = true;
    last_button_time = button_time;
    Serial.println("Temperatura_m massima_target_
        ↪ raggiunta");
}
}
```

```
void IRAM_ATTR decreaseISR() {
    button_time = millis();

    if (button_time - last_button_time > debounceDelay
        ↪ ) {
        if (thermostat.target_temp > 10) {
            decreaseButton.pressed = true;
            last_button_time = button_time;
            thermostat.target_temp--;
        } else {
            decreaseButton.pressed = true;
```

```
    last_button_time = button_time;
    Serial.println("Temperatura_minima_target_"
                  "→ raggiunta");
}
}
```

## 3.4 Display

Dal punto di vista software, la configurazione del display è molto semplice ed utilizza la libreria *LiquidCrystal\_I2C*.

Come prima cosa è necessario impostare l'indirizzo del dispositivo I2C, il numero di colonne e il numero di righe.

La fase di configurazione inizia chiamando la funzione `init()` sull'istanza del display creata.

La fase di aggiornamento del display non avviene direttamente nell'ISR, perchè la libreria che gestisce l'aggiornamento stesso utilizza le interruzioni, che vengono disabilitate quando si è all'interno di una ISR.

La soluzione adottata, prevede che nell'ISR, invece di fare direttamente l'aggiornamento del display, si impostino delle una variabili flag (`onActivationThermostat` e `onDeactivationThermostat`) che indicano che è necessario aggiornare il display. Questa variabile è semplicemente un valore booleano che viene settato a true.

## CAPITOLO 3. SVILUPPO SOFTWARE

---

Nel ciclo principale (`loop()`), si procede poi a verificare periodicamente se questa variabile "flag" è stata impostata. Se sì, si procede con l'aggiornamento del display.

In questo modo, si gestisce l'aggiornamento del display in modo sincronizzato con l'esecuzione del programma, evitando problemi legati alle interruzioni e garantendo che i valori vengano visualizzati correttamente.