



UNIMORE

UNIVERSITÀ DEGLI STUDI DI
MODENA E REGGIO EMILIA

Corso di Laurea Magistrale in Ingegneria Meccatronica

Smart dispenser di acqua potabile

Corso di Sistemi Embedded

Docenti:

Prof. Alessandro Bertacchini

Prof. Fabrizio Pancaldi

Studenti:

Nicholas Boccalini

Simone Fornaciari

Mattia Briselli

Elia Pavarani

Andrea Melegari

Matteo Vaccari

Anno Accademico 2024-2025



Indice

1 Introduzione	4
1.1 Obiettivo del progetto	4
1.2 Contesto e motivazioni	4
1.3 Analisi dei requisiti	5
1.3.1 Requisiti funzionali	5
1.3.2 Requisiti prestazionali	6
2 Progettazione concettuale	7
2.1 Architettura generale del sistema	7
2.2 Funzionamento globale	8
2.2.1 Interfaccia uomo-macchina	8
2.2.2 Circuito di controllo	9
2.2.3 Circuito di potenza	10
2.2.4 Circuito idraulico	11
3 Soluzioni hardware adottate	13
3.1 Progettazione idraulica	13
3.2 Progettazione elettrica	19
3.2.1 Circuito di controllo	19
3.2.2 Circuito di potenza	40
3.2.3 Circuito di alimentazione	43
3.3 Layout finale di prodotto	46
3.3.1 Complessivo CAD	46
4 Configurazione sul Cube MX	49
4.1 Configurazione PIN di comando della scheda STM XNUCLEO IHM04A1	49
4.2 Configurazione PIN Flussimetro	51
4.3 Configurazione PIN pulsanti	52
4.4 Configurazione PIN di comando dei Relè	53
4.5 Configurazione PIN sensore ultrasuoni	55
4.6 Configurazione PIN Display LCD	58
4.7 Configurazione PIN sensore di movimento	59
4.8 Configurazione PIN sensori presenza bicchiere/borraccia	60
4.9 Configurazione PIN modulo Bluetooth HC-05	61
5 Implementazione software su Cube IDE	62
5.1 Controllore PID	62
5.2 Gestione pompa	68
5.3 Gestione macchina a stati	70
5.4 Implementazione software flussimetro	86
5.5 Implementazione software del sensore ultrasuoni	88
5.6 Implementazione software display LCD	89
5.7 Implementazione software sensore di presenza umana	93
5.8 Implementazione software PT1000	94
5.9 Implementazione software modulo Bluetooth HC-05	96

6 Sviluppo dell'applicazione smartphone	102
6.1 Linguaggio di programmazione	102
6.2 Struttura dell'applicazione	103
6.3 Programmazione via app	106
6.3.1 Homepage	106
6.3.2 Timer 1 – Salvataggio della quantità d'acqua giornaliera (Clock1) .	107
6.3.3 Timer 2 – Ricezione dei dati via Bluetooth (TimerRicezioneDati) .	108
6.3.4 Selezione della quantità con lo Slider (inviaComando)	110
6.3.5 Pulsante “Calda” (Calda.Click)	111
6.3.6 Funzioni di test e debug: salvataggio manuale dei dati giornalieri .	112
6.3.7 Pagina ”Storico Dati” – Visualizzazione settimanale	114
6.3.8 Storico dati – selezione giornaliera	115
6.3.9 Manutenzione	116
7 Conclusioni	119

1 Introduzione

Il contenuto di questa relazione tecnica ripercorre le tappe della progettazione e realizzazione dello smart dispenser di acqua potabile. Nella prima parte della relazione vengono esposti gli obiettivi ricercati del progetto, le motivazioni che hanno portato al suo sviluppo e i requisiti funzionali e prestazionali che il dispositivo deve soddisfare.

Nel secondo capitolo si descrive la progettazione concettuale alla base del dispositivo. Viene mostrata l'architettura generale del sistema e spiegato il suo funzionamento globale, approfondendo le funzioni delle singole parti e sottoparti.

Nel terzo capitolo si riportano le soluzioni hardware adottate nella realizzazione del dispositivo. Il capitolo è organizzato per progettazione meccanica, progettazione elettrica e layout finale del prodotto. Nella progettazione meccanica si descrive il circuito idraulico del dispenser e i componenti circuitali. In modo analogo si descrive il circuito elettrico nella progettazione elettrica. Infine si riporta la disposizione finale dei vari elementi hardware che compongono il dispositivo.

Il quarto capitolo vede la descrizione della configurazione dei pin utilizzati del microcontrollore. Avendo utilizzato un microcontrollore della STMicroelectronics, l'ambiente di sviluppo impiegato è stato l'STM32CubeIDE che l'azienda rende open source.

Nel quinto capitolo si descrivono tutte le implementazioni software adottate per gestire la logica di funzionamento del dispenser. Sono state riportate le parti di codice più importanti e descritto il loro funzionamento.

Il sesto capitolo racchiude lo sviluppo software dell'applicazione Android supportata dai dispositivi mobili. Tale applicazione sblocca funzionalità personalizzate per il singolo utente che utilizza il dispenser.

1.1 Obiettivo del progetto

1.2 Contesto e motivazioni

Il tema del rispetto ambientale e della lotta contro i consumi ha accelerato la progettazione di prodotti sempre più sostenibili e a basso impatto ambientale. In questo contesto, il team di sviluppo ha voluto progettare un dispositivo che potesse sposare alcune di queste tematiche. Le problematiche ambientali che hanno spinto alla realizzazione dello smart dispenser sono:

1. la diminuzione dell'uso della plastica
2. la diminuzione dello spreco idrico ed energetico

Lo smart dispenser è pensato per essere collocato in ufficio o in ambienti pubblici. Si vuole realizzare un dispositivo che produca acqua calda, fredda e a temperatura ambiente, intuitivo da usare e che possa collegarsi ai dispositivi mobili presenti oggigiorno. Il dispenser, inoltre, è visto come un incentivo all'uso di risorse rinnovabili, come l'impiego di una borraccia o di un bicchiere riutilizzabile.

1.3 Analisi dei requisiti

I requisiti che il dispositivo deve rispettare sono stati classificati in funzionali e prestazionali. I requisiti funzionali comprendono le modalità e i vincoli di utilizzo del dispositivo, in altre parole tutte le funzioni che il dispenser deve garantire al cliente. I requisiti prestazionali, invece, riguardano le performance numeriche che il dispositivo è in grado di eseguire entro certi limiti e tolleranze.

1.3.1 Requisiti funzionali

I requisiti funzionali sono riportati nella tabella 1.1 sottostante.

ID Req	Descrizione del requisito	Criterio di verifica
RF-01	Il dispenser deve erogare acqua a temperatura calda, fredda e ambiente	Misurazione con termocoppia
RF-02	L'utente deve poter selezionare la temperatura desiderata tramite pulsanti/interfaccia	Test funzionale con input da utente
RF-03	L'erogazione deve interrompersi automaticamente al rilascio del pulsante di selezione	Controllo visivo durante prove di erogazione
RF-04	Il sistema deve rilevare il livello dell'acqua nel serbatoio e segnalare quando è vuoto	Simulazione con serbatoio vuoto
RF-05	Il dispositivo deve segnalare lo stato di funzionamento con LED o display	Controllo visivo durante il funzionamento
RF-06	Il dispositivo deve mostrare la temperatura dell'acqua calda e fredda con LED o display	Controllo visivo durante il funzionamento
RF-07	Il dispositivo deve erogare acqua solo in presenza del bicchiere o borraccia	Controllo visivo durante prove di erogazione
RF-08	Il dispositivo deve andare in modalità stand-by in assenza di utenti dopo alcuni minuti	Controllo visivo durante il funzionamento
RF-09	Il dispositivo deve comunicare con smartphone e registrare il consumo idrico del singolo utente	Controllo durante il funzionamento
RF-10	L'utente può prenotare dall'applicazione la quantità d'acqua desiderata nelle tre temperature	Controllo visivo durante prove di erogazione
RF-11	Il dispositivo deve prevedere uno stato di manutenzione	Test funzionale di manutenzione

Tabella 1.1: Requisiti funzionali

1.3.2 Requisiti prestazionali

I requisiti prestazionali sono riportati nell'elenco sottostante.

- Il dispenser deve garantire una portata minima di 1L/min nelle modalità di acqua calda, fredda e temperatura ambiente.
- L'acqua calda deve avere una temperatura compresa tra 85-95°C.
- L'acqua fredda deve avere una temperatura compresa tra 10-15°C.
- Il dispositivo deve includere un sistema di protezione elettrica (fusibili o interruttore termico) contro picchi di corrente.
- Il dispositivo deve essere operativo (acqua calda e fredda pronta in temperatura) entro massimo 10 minuti dall'accensione.
- Tempo di risposta:
 - acqua calda pronta in meno di 3 minuti dall'accensione
 - acqua fredda pronta in meno di 10 minuti dall'accensione
- Stabilità Bluetooth: connessione stabile fino a massimo 10 m.
- Consumo a regime: ≤ 100 W in stand-by, ≤ 500 W durante riscaldamento/raffreddamento.
- Rumorosità operativa: ≤ 50 dB a 1 m di distanza (compatibile con ambienti domestici/uffici).

2 Progettazione concettuale

La progettazione concettuale definisce le interazioni tra le diverse parti e sottoparti del dispositivo. La rappresentazione mediante diagrammi a blocchi dispone in maniera gerarchica le parti che compongono il sistema.

2.1 Architettura generale del sistema

L'architettura generale del sistema è rappresentata in figura 2.1.

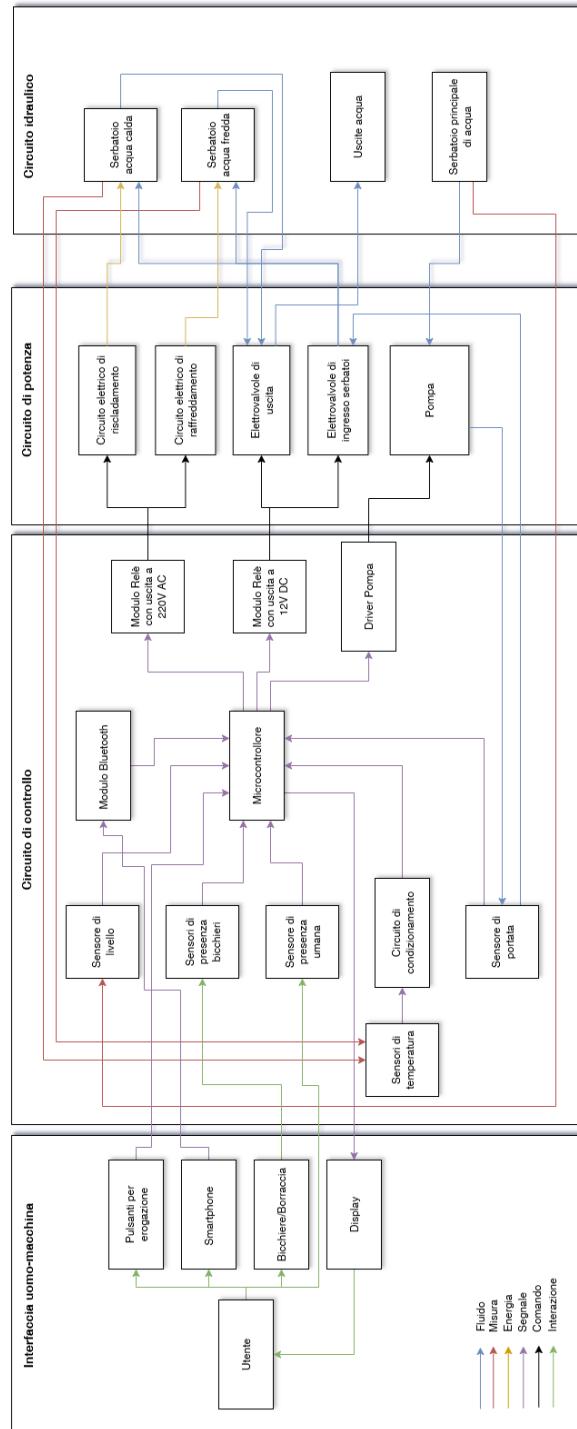


Figura 2.1: Architettura generale

I blocchi rettangolari rappresentano le parti che compongono il sistema uomo-dispenser. Le frecce colorate rappresentano i flussi di fluido, energia, segnale, comando, misura e interazione tra le parti del sistema.

2.2 Funzionamento globale

Il sistema è formato da quattro blocchi principali che interagiscono fra di loro:

- Interfaccia uomo-macchina
- Circuito di controllo
- Circuito di potenza
- Circuito idraulico

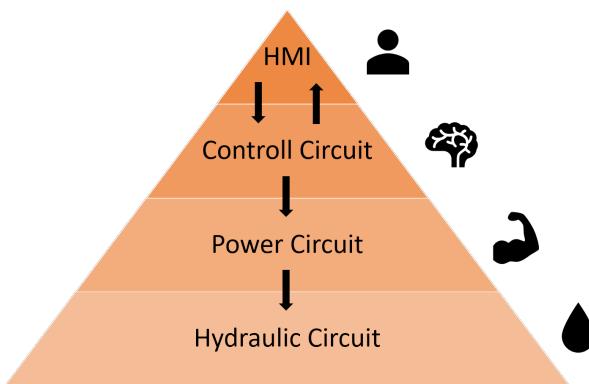


Figura 2.2: Gerarchia delle parti

2.2.1 Interfaccia uomo-macchina

L'utente interagisce con il dispenser attraverso i pulsanti di erogazione, il sensore di presenza umana, il bicchiere/borraccia e il proprio smartphone. Il dispenser, invece, comunica con l'utente attraverso il display integrato.

Il dispositivo è dotato di tre pulsanti distinti per l'acqua calda, fredda e a temperatura ambiente. Tenendo premuto un pulsante, il dispositivo eroga la tipologia d'acqua richiesta fino al suo rilascio.

I sensori di presenza bicchieri rilevano la presenza del bicchiere/borraccia sotto gli erogatori del dispenser e consentono l'erogazione del flusso d'acqua. In mancanza del bicchiere/borraccia l'erogazione non avviene.

Il sensore di presenza umana rileva l'avvicinamento dell'utente al dispositivo, abilitando le funzioni di standby o risveglio automatico.

Utilizzando l'apposita applicazione installata sul proprio smartphone, l'utente può prenotare una quantità d'acqua richiesta e visualizzare il proprio consumo idrico giornaliero.

Il display integrato nel dispenser informa l'utente sullo stato della macchina e sulle temperature dell'acqua presente nei serbatoi.

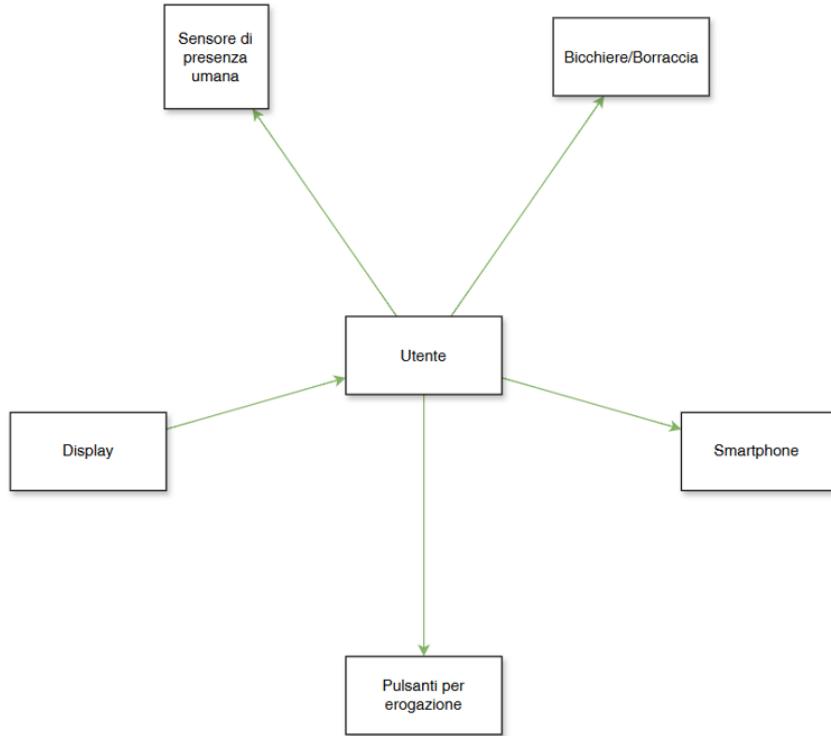


Figura 2.3: Interazioni utente

2.2.2 Circuito di controllo

Il cuore del sistema è rappresentato dal microcontrollore, figura 2.3, che svolge il ruolo di unità di controllo centrale. Esso riceve informazioni dai sensori e dai comandi utente, le elabora secondo la logica software implementata e comanda gli attuatori per garantire il corretto funzionamento del dispenser.

Ingressi (input al microcontrollore):

1. Sensore di portata: misura la portata d'acqua erogata consentendo un controllo preciso della quantità.
2. Sensore di presenza umana: rileva l'avvicinamento dell'utente al dispositivo, abilitando le funzioni di standby o risveglio automatico.
3. Sensore di presenza bicchieri: impediscono l'erogazione in assenza di un bicchiere/borraccia, aumentando la sicurezza del sistema.
4. Sensore di livello: monitora la quantità di acqua presente nel serbatoio principale, prevenendo il funzionamento a secco.
5. Pulsanti per erogazione: permettono all'utente di selezionare la tipologia di erogazione (acqua calda, fredda o a temperatura ambiente).
6. Circuito di condizionamento: adatta e filtra i segnali provenienti dai sensori di temperatura per una corretta acquisizione da parte del microcontrollore.
7. Modulo Bluetooth: consente la comunicazione wireless del microcontrollore con dispositivi esterni (es. smartphone, tablet).

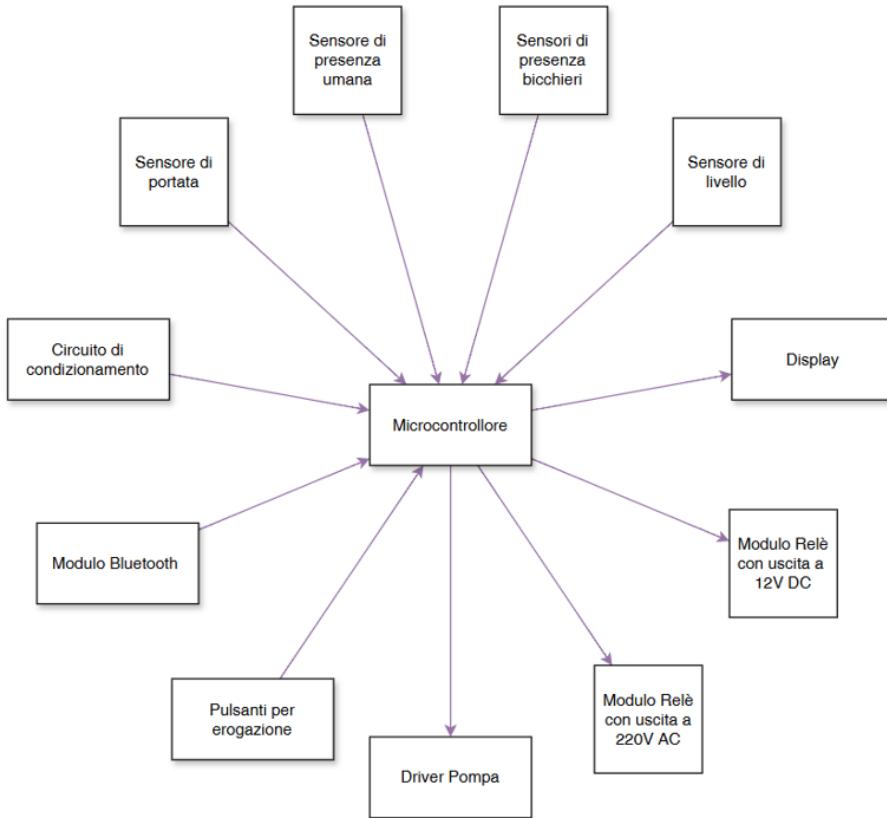


Figura 2.4: Interazioni microcontrollore

Uscite (output dal microcontrollore):

1. Driver pompa: comanda l'attivazione della pompa idraulica per l'erogazione dell'acqua.
2. Modulo relè con uscita a 12V DC: controlla l'apertura e chiusura delle elettrovalvole a bassa tensione continua.
3. Modulo relè con uscita a 220V AC: gestisce componenti alimentati a tensione di rete alternata, come il circuito di riscaldamento o raffreddamento.
4. Display: fornisce feedback all'utente, mostrando informazioni operative, eventuali allarmi e lo stato del sistema.

2.2.3 Circuito di potenza

Il circuito di potenza, figura 2.5 racchiude le parti di attuazione del sistema, cioè i componenti che trasformano un segnale di comando (elettrico in questo caso) in un'azione fisica sul sistema. Gli attuatori implementati nel dispositivo sono:

1. Pompa: preleva l'acqua dal serbatoio principale e la invia nei serbatoi o verso l'uscita (nel caso di acqua a temperatura ambiente).
2. Elettrovalvole di ingresso serbatoi: regolano l'apertura/chiusura dei flussi d'acqua verso i serbatoi.
3. Elettrovalvole di uscita: regolano l'apertura/chiusura dei flussi d'acqua verso gli erogatori.

4. Circuito elettrico di riscaldamento: converte l'energia elettrica della rete in energia termica per risaldare l'acqua nel serbatoio caldo
5. Circuito elettrico di raffreddamento: converte l'energia elettrica della rete in energia termica per raffreddare l'acqua nel serbatoio freddo.

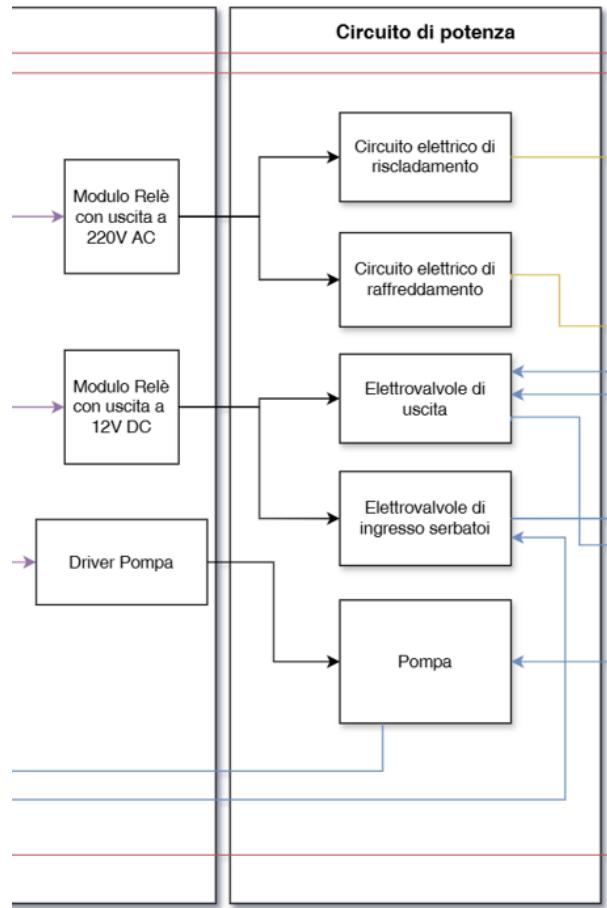


Figura 2.5: Circuito di potenza

2.2.4 Circuito idraulico

Il circuito idraulico, figura 2.6, rappresenta l'ultimo blocco funzionale dell'architettura di sistema. I principali elementi sono:

1. Serbatoio principale: è il serbatoio con maggiore capienza, immagazzina l'acqua e il sensore di livello ne misura la capacità.
2. Serbatoio acqua calda: immagazzina e riscalda l'acqua al suo interno.
3. Serbatoio acqua fredda: immagazzina e raffredda l'acqua al suo interno.
4. Uscite acqua: sono gli ugelli finali dai quali fuoriesce l'acqua verso il bicchiere/borraccia.

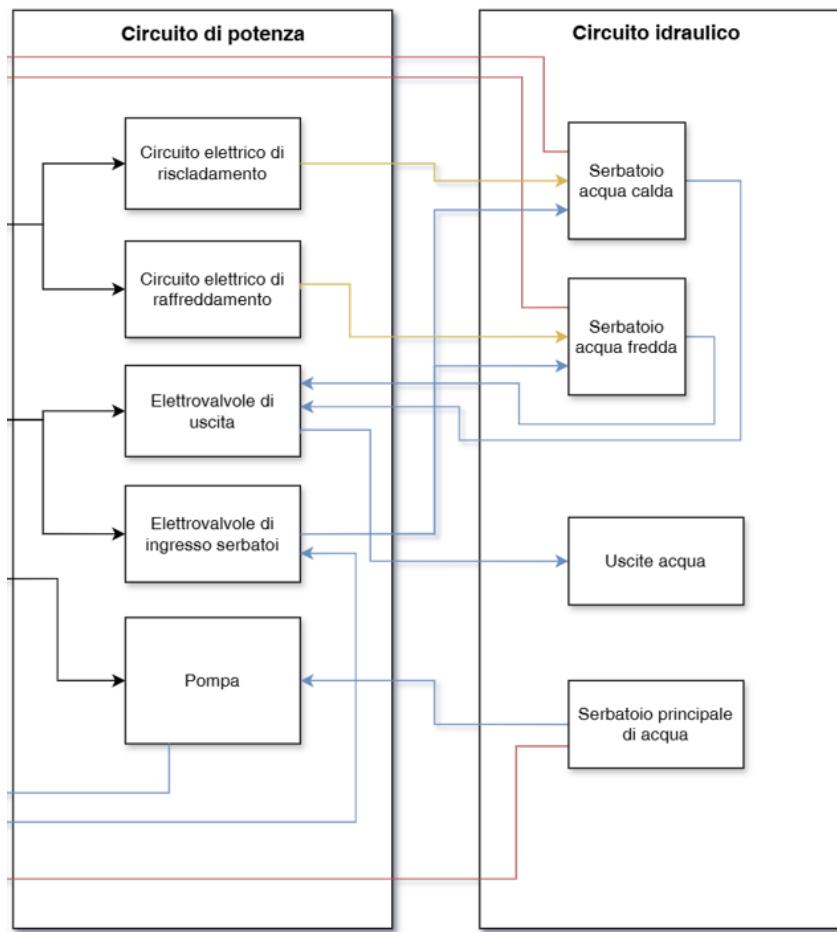


Figura 2.6: Circuito idraulico

3 Soluzioni hardware adottate

In questo capitolo vengono esposte le soluzioni hardware adottate e i relativi calcoli dimensionali. Per consentire una maggiore chiarezza, lo schema idraulico ed elettrico del dispenser sono riportati nella parte finale della relazione. Nel prosieguo della descrizione tecnica si farà riferimento agli schemi come "schema idraulico" e "schema elettrico". Per identificare i componenti si indicheranno il numero di foglio, la lettera di riga e il numero di colonna riportati ai margini dello schema di riferimento.

3.1 Progettazione idraulica

La parte meccanica del dispenser ha visto la progettazione del circuito idraulico e la scelta dei componenti idraulici commerciali che rendono possibile l'intero funzionamento. Il team di sviluppo è partito da un classico dispenser per ufficio con l'obiettivo di automatizzare l'erogazione dell'acqua per mezzo delle modifiche hardware.

Dispenser originale

Il dispenser di partenza, mostrato in figura 3.1, è un classico dispenser da ufficio che eroga l'acqua per effetto della forza di gravità. Può ospitare boccioni d'acqua da 5 a 20 litri e presenta al suo interno due serbatoi separati adibiti a raffreddare e riscaldare l'acqua immagazzinata.

Il serbatoio freddo è costituito da una scatola in acciaio inossidabile coibentata con polistirolo per favorire l'isolamento termico dalle fonti di calore. Il sistema di raffreddamento è realizzato tramite una cella Peltier, alimentata a 12V, che asporta il calore in eccesso all'interno del serbatoio. Una ventola alimentata a 12V e un dissipatore in alluminio sono poi installati in prossimità della cella Peltier per aiutare la dissipazione termica. Il circuito elettrico di raffreddamento è gestito da una scheda PCB già installata nel prodotto.

Il serbatoio caldo è anch'esso realizzato in acciaio inossidabile. Al suo interno è installata una resistenza di riscaldamento da 100Ω che riscalda l'acqua per effetto Joule. Un termostato è installato a contatto con il serbatoio per aprire il circuito quando la temperatura dell'acqua arriva vicina all'ebollizione. Il vapore in eccesso viene trasformato in condensa grazie a un condotto presente in cima al serbatoio che si ricollega al boccione d'acqua.

L'erogazione dell'acqua avviene per gravità azionando le valvole a leva presenti davanti al dispenser, una per ciascun serbatoio. Nella parte frontale del dispenser sono installati tre led luminosi che indicano se il dispenser è in fase di raffreddamento (led giallo), riscaldamento (led rosso) o mantenimento (led verde). Durante il mantenimento, l'acqua presente nel serbatoio caldo ha raggiunto la temperatura prestabilita e il circuito di riscaldamento si interrompe.

Il prodotto richiede l'alimentazione alla tensione alternata monofase a 230V tramite una spina Shuko. Il circuito elettrico del serbatoio caldo funziona interamente a tensione alternata, mentre quello del serbatoio freddo ricava la tensione continua a 12V dalla scheda PCB. Per una maggiore comprensione, nel corso della descrizione, la scheda PCB verrà anche denominata "scheda del freddo".

Il dispenser può ospitare bicchieri di dimensioni comuni, bottiglie e borracce fino a 0,5L. Contenitori di dimensioni superiori richiedono l'inclinazione manuale da parte dell'utente per essere riempiti.



Figura 3.1: Dispenser originale

Raccordi e tubi

Le modifiche al circuito idraulico del dispenser originale prevedono l'installazione di tubi e raccordi idraulici.

La scelta dei tubi idraulici è influenzata dalla varietà di diametri di attacco di componenti (come per esempio le elettrovalvole, la pompa, il flussimetro, ecc), dalle caratteristiche di resistenza alle temperature di esercizio e dalla robustezza e affidabilità nel tempo. Nel corso della fase di assemblaggio si è riscontrato che i classici tubi in Polivinilcloruro (PVC) ad uso alimentare non reggevano le temperature di esercizio (nello specifico acqua a circa 100°C) ed essendo poco spessi si strozzavano, ostruendo il passaggio d'acqua. La scelta è quindi ricaduta sul tubo in Silicone Perossidico, che presenta temperature di esercizio comprese tra -60/+200°C, è inodore e atossico. Nella figura 3.2 si riportano evidenziate le dimensioni commerciali acquistate.

Per i collegamenti idraulici di piccolo diametro (6mm) si è scelto di utilizzare tubi per benzina 5x9mm di colorazione verde, anch'essi resistenti alle alte temperature, atossici e affidabili nel tempo. In figura 3.3a si riporta il tubo per benzina 5x9mm utilizzato.

Le variazioni di diametro sono realizzate impiegando dei raccordi idraulici componibili (figura 3.3b) di diametri 6mm, 8mm e 10mm, mentre le diramazioni del circuito con raccordi a tre vie dello stesso produttore.

SCHEDA PRODOTTO

	Ø esterno (mm)	Ø interno (mm)	Massa (gr/m)
TS2X4	4	2	12
TS3X5	5	3	16
TS3X6	6	3	27
TS4X6	6	4	20
TS4X8	8	4	48
TS5X8	8	5	39
TS5X9	9	5	56
TS6X9	9	6	45
TS6X10	10	6	64
TS7X10	10	7	51
TS8X10	10	8	36
TS8X12	12	8	80
TS9X12	12	9	63
TS10X12	12	10	44
TS10X14	14	10	96
TS12X16	16	12	112
TS14X18	18	14	126
TS18X24	24	18	253

Figura 3.2: Valori commerciali del tubo in Silicone Perossidico



(a) Tubo per benzina 5x9mm



(b) Raccordo idraulico 6mm

Figura 3.3: Tubi e raccordi utilizzati

Tanica 20 litri

Il serbatoio principale nel quale immagazzinare l'acqua da servire è una tanica da 20 L in plastica, mostrata nella figura 3.4. La tanica presenta due aperture: una inferiore tramite un rubinetto manuale e una superiore attraverso il tappo svitabile. Nello schema idraulico, la tanica è rappresentata nella colonna 1 del foglio 1. A pieno carico, la tanica è in grado di soddisfare il fabbisogno idrico giornaliero per un ufficio di 10÷12 persone.



Figura 3.4: Tanica 20L

Elettrovalvole

Le elettrovalvole installate all'interno del circuito idraulico indirizzano i flussi d'acqua che l'utente desidera in fase di erogazione. Le elettrovalvole di ingresso serbatoi sono mostrate in figura 3.5a e figura 3.5b. La tipologia *a* è installata in ingresso al serbatoio caldo, mentre la tipologia *b* presenta due uscite separate (in pratica due elettrovalvole indipendenti), una per il serbatoio freddo e una per l'acqua a temperatura ambiente. Le specifiche tecniche delle due tipologie di elettrovalvole sono:

- Tensione nominale: 12V DC
- Corrente nominale: 300mA
- Tipologia: normalmente chiusa (NC)
- Dimensioni nominali:
 - Tipo *a*: 52mm (L) x 52mm (W) x 60 mm (H)
 - Tipo *b*: 80mm (L) x 65mm (W) x 60 mm (H)
- Dimensioni attacco:
 - Tipo *a*: diametro esterno 6,4mm, diametro interno 4.9mm.
 - Tipo *b*: diametro esterno 8,5mm, diametro interno 6,8mm, diametro ingresso G1 1/4".



(a) del serbatoio caldo



(b) del serbatoio freddo

Figura 3.5: Elettrovalvole di ingresso serbatoi

Le elettrovalvole di uscita sono mostrate in figura 3.6. L'ugello di uscita di queste valvole è inclinato ad angolo retto rispetto all'ingresso, semplificando notevolmente la loro installazione.



Figura 3.6: Elettrovalvola di uscita

Le specifiche tecniche delle elettrovalvole di uscita sono:

- Tensione nominale: 12V DC
- Corrente nominale: 200mA
- Tipologia: normalmente chiusa (NC)
- Dimensioni nominali: 35 mm (L) x 40 mm (W) x 55mm (H)
- Dimensioni attacco: diametro esterno 6,4mm, diametro interno 4,9mm

Implementata nel circuito idraulico è presente una seconda elettrovalvola, come quella in figura 3.5a, collegata tra l'uscita di sfato del serbatoio caldo e il serbatoio principale. La sua funzione è di comandare il circuito di sfato del serbatoio caldo, consentendo al vapore in eccesso di fuoriuscire e condensare nel serbatoio principale.

Valvole unidirezionali

Le valvole unidirezionali permettono il flusso d'acqua verso un'unica direzione. Il circuito idraulico dispone di due valvole unidirezionali:

- La prima, installata a monte della pompa idraulica, evita l'inversione di flusso dell'acqua dalla pompa verso il serbatoio principale. Questa soluzione è importante in quanto un flusso d'acqua contrario porterebbe ad una misura errata da parte del sensore di portata.
- La seconda è situata nel ramo idraulico che svolge la funzione di limitatore di pressione. Per evitare infatti sovrappressioni dannose per i componenti idraulici, è stato inserito un ramo che collega la pompa idraulica e il serbatoio principale. L'estremità finale del ramo presenta un ugello strozzatore che sfocia l'acqua all'interno del serbatoio, fungendo da via di scarico nel caso di sovrappressioni nell'impianto.

La valvola implementata nel circuito idraulico è mostrata in figura 3.7. La tipologia è di quelle da rubinetteria domestica, presentando valori di tenuta ritenuti ampiamente sufficienti per il tipo di applicazione.



Figura 3.7: Valvola unidirezionale

3.2 Progettazione elettrica

La progettazione elettrica vede la scelta e l'implementazione dei principali componenti elettrici presenti nel dispenser. All'interno del dispositivo sono presenti tre circuiti differenti:

1. Circuito di controllo
2. Circuito di potenza
3. Circuito di alimentazione

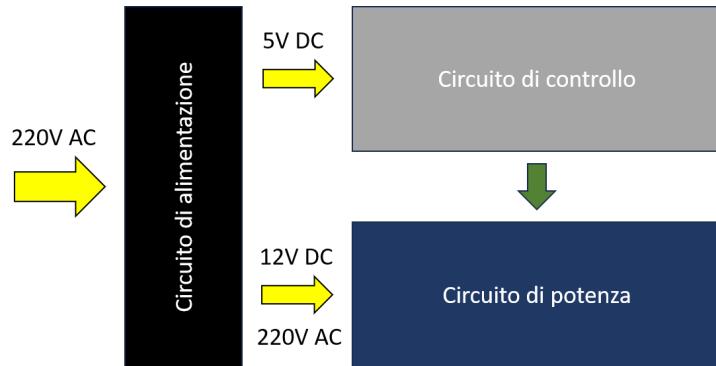


Figura 3.8: Architettura dei circuiti elettrici

3.2.1 Circuito di controllo

Microcontrollore

Il microcontrollore scelto è un STM32L476RG (serie STM32L4 a basso consumo) integrato nella scheda di sviluppo NUCLEO-L476RG, figura 3.9.

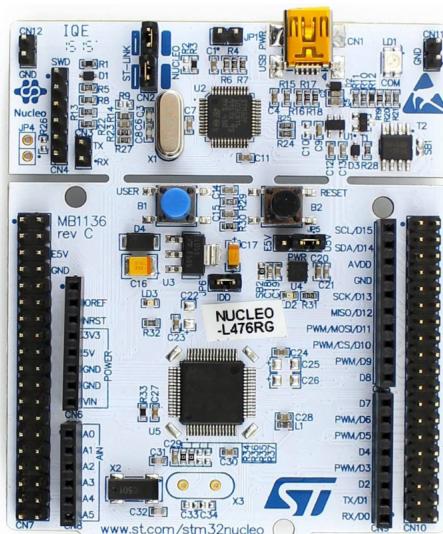


Figura 3.9: Scheda NUCLEO-L476RG

Le principali caratteristiche del microcontrollore sono:

- Core: ARM Cortex-M4 a 80 MHz con FPU (floating point unit)
- Flash: 1 MB
- SRAM: 128 kB
- EEPROM emulata in Flash
- Basso consumo (tecnologia ultra-low power, diverse modalità di sleep/standby)

La scheda di sviluppo presenta:

- Connettori Arduino Uno R3 e ST morpho: compatibilità con shield Arduino e connettori ST per accesso a tutti i pin del micro.
- Programmatore/debugger ST-LINK/V2-1 integrato: non serve un programmatore esterno.
- Alimentazione flessibile: da porta USB (5V, fino a 500mA), da alimentazione esterna tramite il pin VIN (7-12V), dai pin 5V o 3.3V tramite regolatori di tensione onboard.
- Clock: HSI/HSE, RTC con quarzo dedicato (32.768 kHz).
- Comunicazioni integrate: USART, UART, LPUART, I²C, SPI, CAN, SAI,
- Timer: multipli (a 16 e 32 bit), anche PWM.
- Convertitori ADC e DAC: DAC 12 bit (2 canali), ADC 12 bit (16 canali, fino a 5 Msps)

La scheda è indicata per lo sviluppo di sistemi meccatronici con controllo tramite sensori e attuatori e per applicazioni che richiedono basso consumo ma anche potenza di calcolo. Per maggiori dettagli tecnici si rimanda alla documentazione ufficiale del produttore.

La scheda di sviluppo viene alimentata esternamente a 5V DC attraverso il pin VIN e collegando il jumper tra il pin 2 e il pin 3 di JP5, figura 3.10.

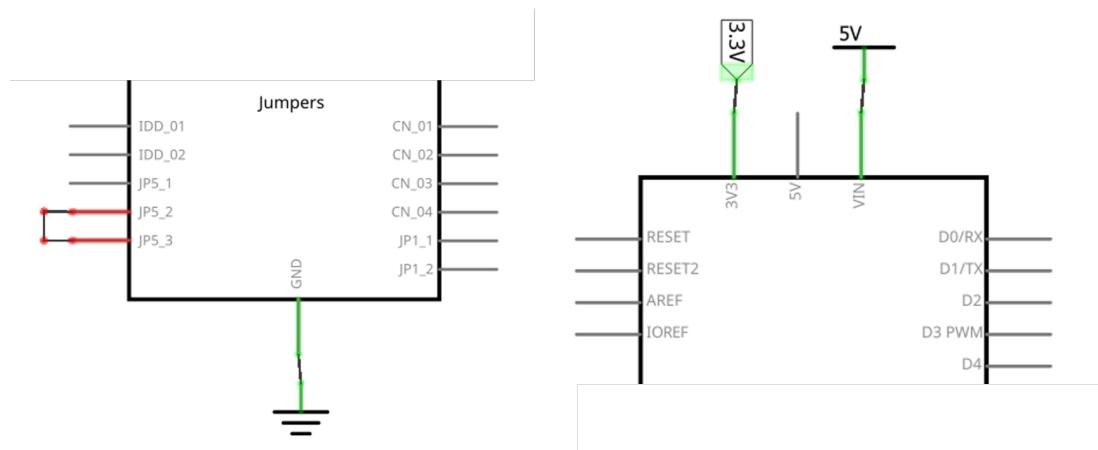


Figura 3.10: Alimentazione microcontrollore

Il pinout completo con i collegamenti del microcontrollore è riportato nella tabella sotto.

Pin Number LQFP64	Pin Name (function after reset)	Pin Type	Alternate Function(s)	Label
1	VBAT	Power		
2	PC13	I/O	GPIO_EXTI13	B1 [Blue PushButton]
3	PC14-OSC32_IN (PC14)	I/O	RCC_OSC32_IN	
4	PC15-OSC32_OUT (PC15)	I/O	RCC_OSC32_OUT	
5	PH0-OSC_IN (PH0) *	I/O	RCC_OSC_IN	
6	PH1-OSC_OUT (PH1) *	I/O	RCC_OSC_OUT	
7	NRST	Reset		
8	PC0	I/O	I2C3_SCL	
9	PC1	I/O	I2C3_SDA	
11	PC3	I/O	ADC1_IN4	
12	VSSA/VREF-	Power		
13	VDDA/VREF+	Power		
14	PA0	I/O	TIM2_CH1	
16	PA2	I/O	USART2_TX	USART_TX
17	PA3	I/O	USART2_RX	USART_RX
18	VSS	Power		
19	VDD	Power		
20	PA4	I/O	ADC1_IN9	
21	PA5 **	I/O	GPIO_Output	LD2 [green Led]
22	PA6 **	I/O	GPIO_Output	
25	PC5	I/O	SYS_WKUP5	
26	PB0	I/O	ADC1_IN15	
28	PB2	I/O	GPIO_EXTI2	PulsanteCalda
29	PB10	I/O	GPIO_EXTI10	PulsanteTAmb
30	PB11	I/O	GPIO_EXTI11	PulsanteFredda
31	VSS	Power		
32	VDD	Power		
33	PB12 **	I/O	GPIO_Output	LedFredda
34	PB13 **	I/O	GPIO_Output	LedTAmb
35	PB14 **	I/O	GPIO_Output	LedCalda
36	PB15 **	I/O	GPIO_Output	EvErogatoreCalda
37	PC6 **	I/O	GPIO_Output	EvErogatoreFredda
38	PC7 **	I/O	GPIO_Output	EvSerbatoioCaldo
39	PC8 **	I/O	GPIO_Output	EvTAmb
40	PC9 **	I/O	GPIO_Output	EvSerbatoioFredda
41	PA8	I/O	TIM1_CH1	

Figura 3.11: Pinout Nucleo-L476RG (1)

Pin Number LQFP64	Pin Name (function after reset)	Pin Type	Alternate Function(s)	Label
42	PA9	I/O	USART1_TX	
43	PA10	I/O	USART1_RX	
44	PA11 **	I/O	GPIO_Output	TRIG
45	PA12 **	I/O	GPIO_Output	Enable_A
46	PA13 (JTMS-SWDIO)	I/O	SYS_JTMS-SWDIO	TMS
47	VSS	Power		
48	VDDUSB	Power		
49	PA14 (JTCK-SWCLK)	I/O	SYS_JTCK-SWCLK	TCK
50	PA15 (JTDI) **	I/O	GPIO_Output	CircuitoCaldo
51	PC10 **	I/O	GPIO_Output	CircuitoFreddo
52	PC11 **	I/O	GPIO_Output	EvSfatoCaldo
53	PC12 **	I/O	GPIO_Output	
55	PB3 (JTDO-TRACESWO) *	I/O	SYS_JTDO-SWO	SWO
56	PB4 (NJTRST)	I/O	TIM3_CH1	
57	PB5	I/O	TIM3_CH2	
58	PB6	I/O	TIM4_CH1	
60	BOOT0	Boot		
63	VSS	Power		
64	VDD	Power		

Figura 3.12: Pinout Nucleo-L476RG (2)

Pulsanti di erogazione

I pulsanti di erogazione sono implementati ciascuno con una rete di pull-down (figura 3.13) che svolge la funzione di partitore di tensione. La pressione dei pulsanti produce i 3.3V logici sugli ingressi assegnati del microcontrollore.

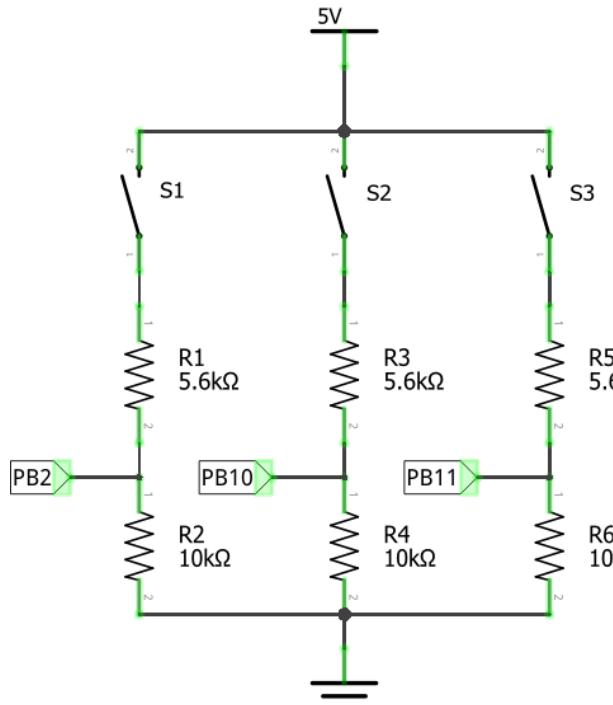


Figura 3.14: Pulsante 5V

Figura 3.13: Rete di pull-down pulsanti

Modulo relè 5V

L’azionamento degli attuatori presenti nel circuito di potenza (tranne la pompa idraulica) avviene con dei moduli relè SRD-05VDC-SL-C, figura 3.15. Nel circuito sono presenti in totale 8 moduli relè:

1. Relè K1: comanda il circuito elettrico di riscladamento
2. Relè K2: comanda il circuito elettrico di raffreddamento
3. Relè K3: comanda l’elettrovalvola dell’acqua a temperatura ambiente
4. Relè K4: comanda l’elettrovalvola di uscita acqua fredda
5. Relè K5: comanda l’elettrovalvola di ingresso serbatoio caldo
6. Relè K6: comanda l’elettrovalvola di uscita acqua calda
7. Relè K7: comanda l’elettrovalvola di sfato del serbatoio caldo
8. Relè K8: comanda l’elettrovalvola di ingresso serbatoio freddo

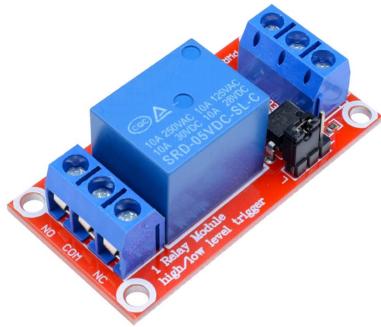


Figura 3.15: Modulo relè SRD-05VDC-SL-C

Le specifiche tecniche del modulo relè sono:

- Tensione nominale: 5V DC
- Corrente nominale: circa 70-75mA in eccitazione
- Resistenza nominale della bobina: 70Ω
- Potenza nominale: 0,36W
- Dimensioni nominali: 50mm (L) x 26.2mm (W) x 18mm (H)

Lo schema di collegamento è riportato sotto in figura 3.16. Gli ingressi logici IN comandano l'eccitazione della bobina. I collegamenti dei contatti NC, NO e COM sono specificati meglio nella sezione del circuito di potenza.

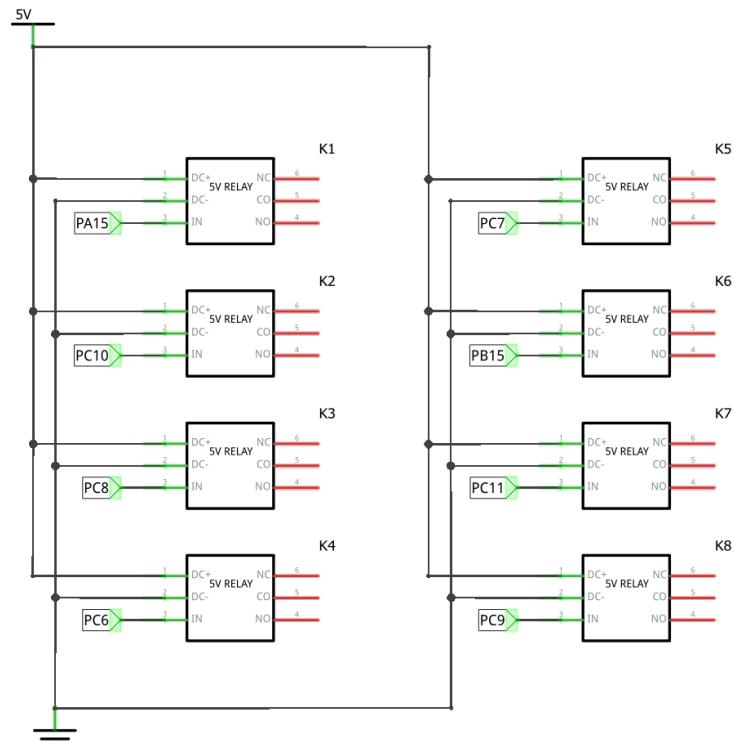


Figura 3.16: Enter Caption

Scheda driver pompa

Il microcontrollore comanda con modulazione d’impulso (PWM) la pompa idraulica attraverso una scheda driver. La scheda scelta è la X-NUCLEO-IHM04A1 (figura 3.17) che consente di pilotare fino a due motori brush DC.

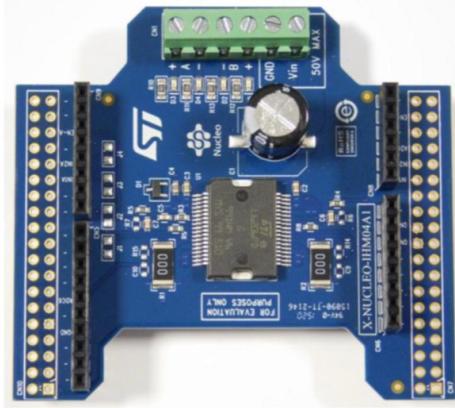


Figura 3.17: Scheda driver pompa

Le specifiche tecniche di questa scheda driver sono:

- Range di tensione da 8-50V max
- Corrente di fase fino a 2.8A rms
- Pilotaggio diretto in uscita
- Possibilità di montaggio per resistenze di shunt
- Protezione da sottotensione (Undervoltage lockout)
- Protezione programmabile da sovraccorrente lato alto (high side overcurrent protection)
- Spegnimento per protezione termica
- Compatibile con schede Nucleo STM32 e connettore Arduino UNO R3

La figura 3.18 mostra il pinout della scheda driver. Durante la progettazione si è scelta di utilizzare l’uscita B collegando gli ingressi di controllo IN1B, IN2B e EN_B ai rispettivi pin elencati della Nucleo. I collegamenti lato potenza verranno descritti nel circuito di potenza.

- In1B → PA12
- In2B → PA11
- En_B → PB4

L’ingresso In2B è stato collegato ma non viene utilizzato durante il pilotaggio della pompa idraulica. Questo perché, come verrà descritto anche nel sottocapitolo successivo, non vi è la necessità di comandare la pompa in entrambi i sensi di marcia. Il collegamento serve solo a non lasciare flottante l’ingresso In2B, mettendolo a false dal codice di gestione della pompa idraulica.

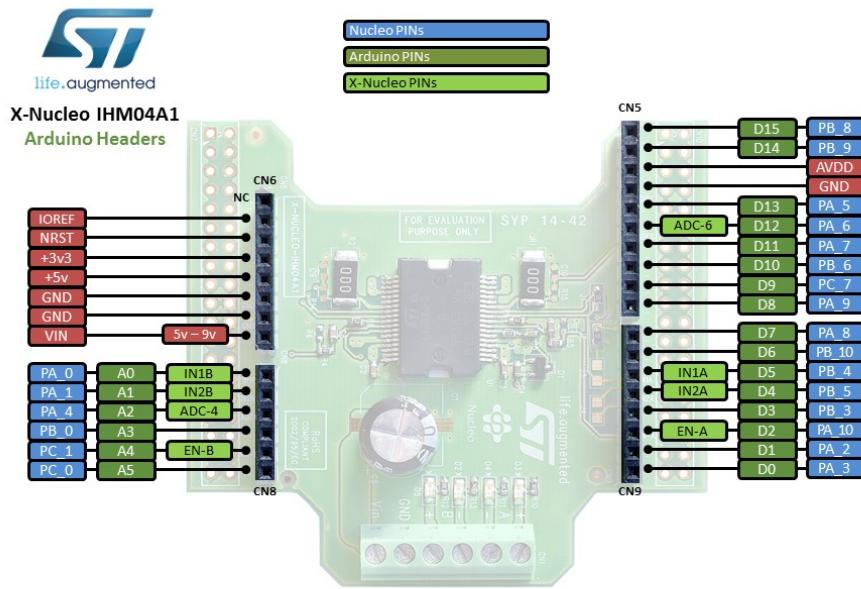


Figura 3.18: Pinout X-Nucleo-IHM04A1

Sensori di presenza bicchiere/borraccia

La presenza del bicchiere/borraccia è rilevata con il sensore a infrarossi IR TCRT5000, figura 3.20. Nel dispenser sono installati due sensori IR sotto gli erogatori. Le specifiche tecniche principali del sensore IR TCRT5000 sono:

- Tensione nominale: 3.3V – 5V DC
- Corrente nominale: circa 23 mA a 3.3V, circa 43 mA a 5.0V
- Intervallo di rilevamento: da 2 cm a 30 cm (regolabile tramite potenziometro)
- Angolo di rilevamento: 35°
- Temperatura di esercizio: -10/+50°C
- Tipo di uscita: digitale (HIGH = ostacolo rilevato, LOW = nessun ostacolo)
- Tempo di risposta: circa 2 ms
- Resistenza alla luce ambientale: buona
- Dimensioni complessive: 45 3m (L) x 14 mm (W) x 7 mm (H)

L'implementazione nel circuito di controllo è mostrata in figura 3.19. Le uscite logiche digitali (DO) dei sensori sono collegate ai pin PB8 e PB9 della scheda di sviluppo.

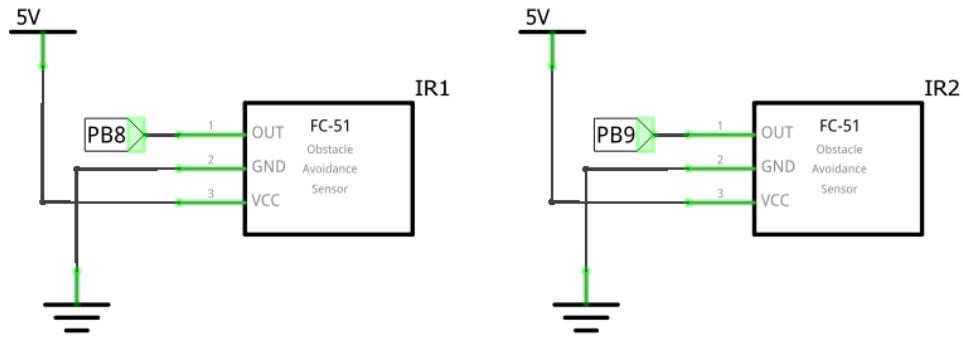


Figura 3.19: Schematico elettrico sensore IR



Figura 3.20: Sensore IR FC-51

Sensore di presenza umana

La presenza umana viene rilevata attraverso un sensore PIR HC-SR501, figura 3.22. Le principali specifiche tecniche sono:

- Tensione di alimentazione: 4.5V – 20V DC
- Consumo di corrente: circa 50 μ A in standby, qualche mA durante il funzionamento
- Portata di rilevamento: regolabile 0,2 – 15mm
- Angolo di rilevamento: circa 120° (copertura a ventaglio)
- Tipo di uscita: digitale (HIGH quando rileva movimento, LOW in assenza)
- Tensione di uscita: 3.3V (logica HIGH)
- Tempo di ritardo (Delay Time): regolabile tramite trimmer da 5 secondi a 200 secondi
- Dimensioni: 32 mm (L) x 24 mm (W) x 18 mm (H)

L'implementazione nel circuito di controllo è mostrata in figura 3.21. L'uscita logica del sensore è collegata al pin PC12 della scheda di sviluppo.

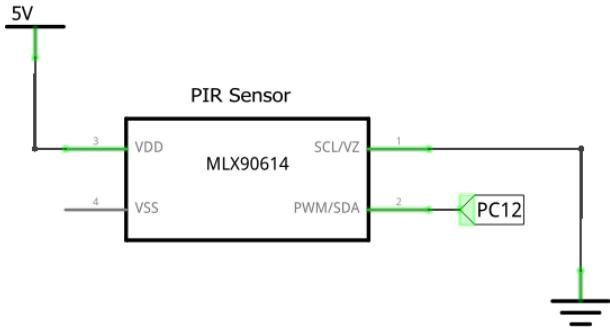


Figura 3.21: Schematico elettrico sensore PIR

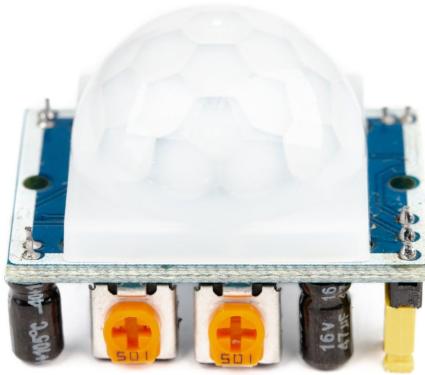


Figura 3.22: Sensore HC-SR501

Sensore di livello

La misura del livello d'acqua nel serbatoio principale è ottenuta impiegando un sensore ad ultrasuoni HC-SR04, figura ???. Le specifiche tecniche del sensore sono:

- Tensione di alimentazione: 5V DC
- Corrente assorbita: 15mA
- Intervallo di misurazione: da 2 cm a circa 400 cm
- Risoluzione: circa 3 mm
- Angolo di rilevamento: 15°
- Frequenza ultrasuoni: 40 kHz
- Tempo di ciclo minimo: 60 ms (circa 16 letture al secondo)
- Dimensioni: 45 mm (L) x 20 mm (W) x 15 mm (H)

L'implementazione nel circuito di controllo è mostrata in figura 3.24. Per adattare i livelli logici delle uscite del sensore (5V) con i livelli logici di ingresso della scheda di sviluppo (3.3V), il circuito implementa un level shifter bidirezionale che si occupa di adattare la tensione. La 3.3V viene fornita dal microcontrollore direttamente all'ingresso logico basso (LV) del level shifter. La figura 3.23 mostra lo schema dei collegamenti.

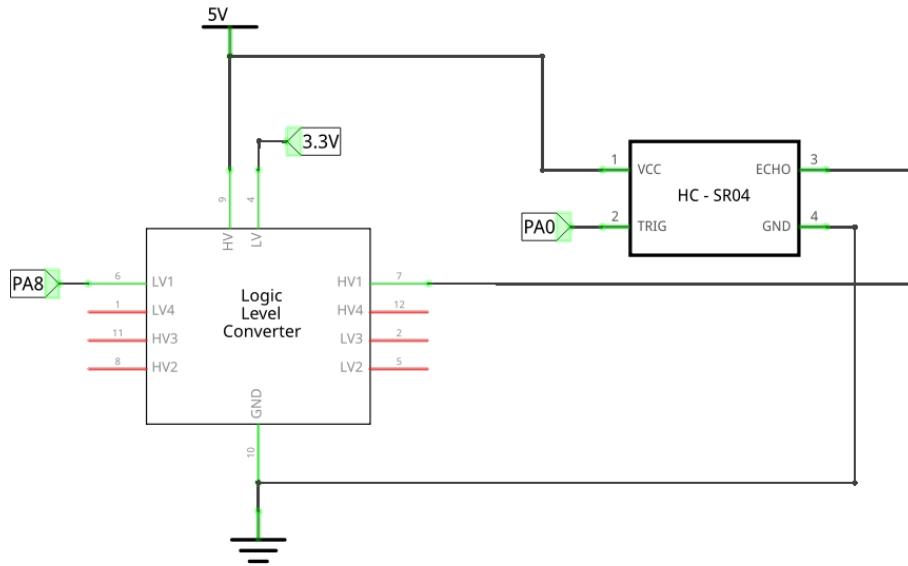


Figura 3.23: Schematico sensore ultrasuoni



Figura 3.24: Sensore ultrasuoni HC-SR04

Display LCD

Per mostrare gli output del dispenser sotto forma di testo e numeri all’utente finale si è deciso di utilizzare un display LCD (Liquid Crystal Display). Il modulo LCD 20x04 (figura 3.25) utilizzato è costituito da 20 colonne e 4 righe, dove ogni cella è formata da 5x8 dots.

Il display basa il proprio funzionamento sul controller Hitachi HD44780, presente nel retro del modulo LCD, che svolge il compito di ricevere comandi e dati e visualizzarli sotto forma di testo. Il microcontrollore non dialoga direttamente con ogni singolo pixel del display. Invece, invia comandi e dati al controller HD44780. Per facilitare la comunicazione I²C tra i due viene utilizzato un modulo PCF8574 che consente di ridurre i pin necessari, tipicamente 6-10, a soli 2 (SDA e SCL).

Le specifiche tecniche del display sono:

- Tipologia: LCD 20 colonne × 4 righe (fino a 80 caratteri visualizzabili).
- Controller: basato su HD44780
- Comunicazione: I²C

- Tensione nominale: 5V
- Corrente nominale: circa 20–30mA con retroilluminazione accesa.
- Contrasto: regolabile con un potenziometro sul modulo
- Dimensioni: 98mm (L) x 60mm (W) x 14mm (H)

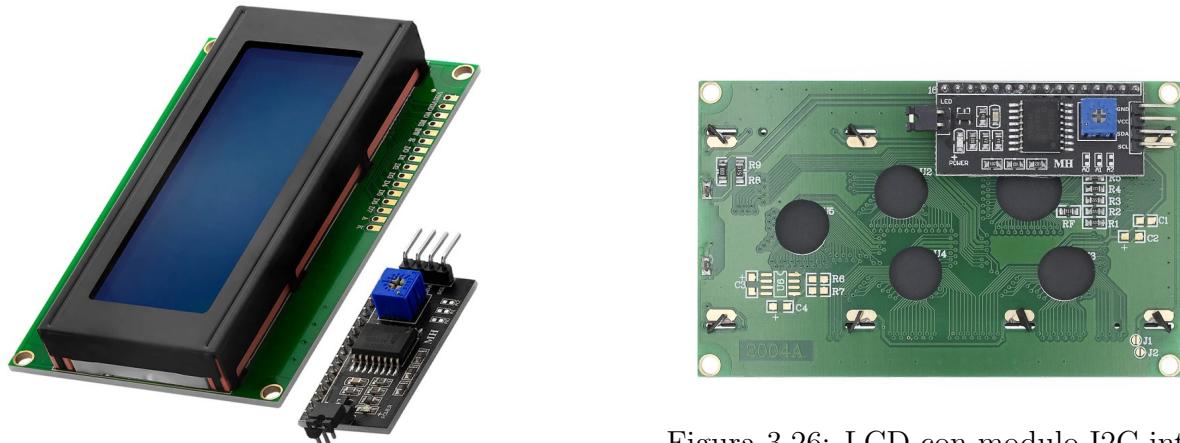


Figura 3.25: LCD 20x4 2004 con controller HD44780

Figura 3.26: LCD con modulo I²C integrato

Dallo studio di un paper online è emerso che per il PCF8574 standard, l'indirizzo 0x4E è quello da utilizzare in HAL per garantire il corretto funzionamento del driver I²C e del protocollo di comunicazione.

L'implementazione nel circuito di controllo è mostrata in figura 3.27. Per adattare i livelli di tensione, come per il sensore ultrasuoni, si è utilizzato il level shifter bidirezionale.

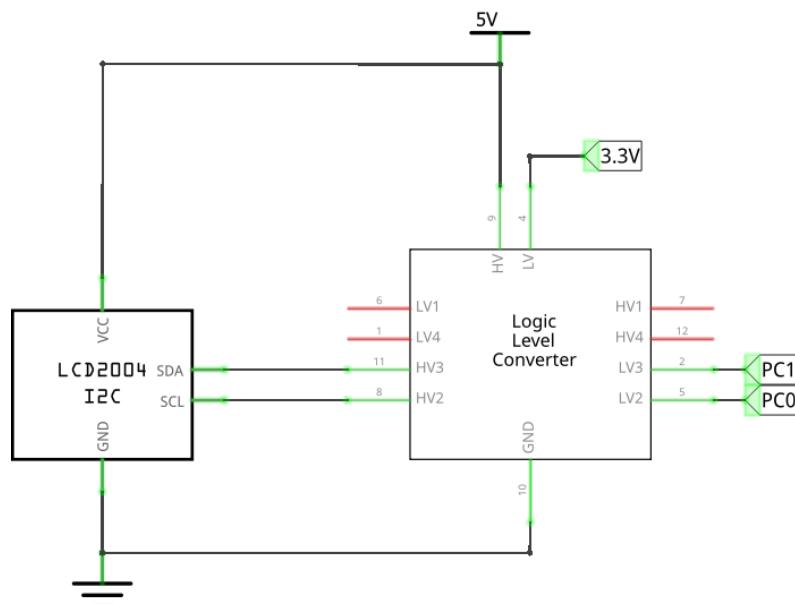


Figura 3.27: Schematico display LCD

Sensore di portata

Il sensore di portata utilizzato è un flussimetro YF-S401, figura ???. Il sensore è composto da un corpo in PVC, un rotore e un sensore ad effetto Hall. Quando l'acqua attraversa il rotore, quest'ultimo ruota e la sua velocità varia in funzione della portata del flusso. Il sensore ad effetto Hall genera un impulso logico alto ad ogni rotazione del rotore. Il risultato è un treno di impulsi a frequenza variabile in funzione della portata d'acqua.



Figura 3.28: Flussimetro YF-S401

Le specifiche tecniche sono riportate in figura 3.29. Il sensore presenta un connettore a 3 pin per il cavo rosso, nero e giallo. Attraverso il cavo rosso (VCC) e nero (GND) si alimenta il sensore, il cavo giallo (OUT) invece è l'uscita logica del sensore che si collega al pin PB6 del microcontrollore.

Specifications

- Mini. Working Voltage: DC 4.5V
- Max. Working Current: 15mA (DC 5V)
- Working Voltage: DC 5V~24V
- Water resistant 0.35MPa
- Flow Rate Range: 1~ 5L/min
- Load Capacity: \leq 10mA (DC 5V)
- Operating Temperature: \leq 80°C
- Liquid Temperature: \leq 120°C
- Operating Humidity: 35%~90%RH
- Water Pressure: \leq 1.75MPa
- Storage Temperature: -25~+ 80°C
- Storage Humidity: 25%~95%RH
- Internal diameter: 1.2mm;
- Error: +/-2L/min;
- Insulation resistance > 100MΩ
- Output pulse duty cycle $50\% \pm 10\%$
- Output pulse high level > DC 4.7V (input voltage DC 5V)
- Flow pulse characteristics $F = (98 * Q) \pm 2\% Q = L / MIN$

Figura 3.29: Specifiche tecniche del flussimetro

Sensori di temperatura

Al fine di poter misurare la temperatura dell'acqua calda e fredda si è scelto di utilizzare due PT1000. Il PT1000 è un sensore di temperatura a resistenza usato per le misurazioni di precisione. È costituito da un elemento sensibile al suo interno, il platino, che varia la sua resistenza elettrica linearmente in funzione della temperatura (1000ohm a 0°C). Esiste anche la versione PT100, tuttavia scartata ai fini del progetto in quanto avente una resistenza nominale più bassa e quindi più sensibile ai disturbi e alle cadute di tensione lungo i cavi.



Figura 3.30: PT1000

Circuito di condizionamento

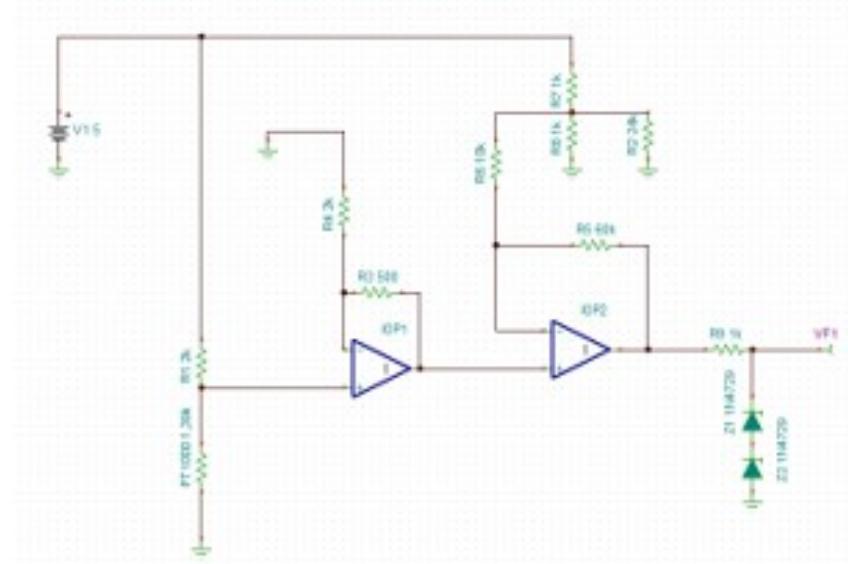


Figura 3.31: Schema del circuito di condizionamento

L'obiettivo del circuito di condizionamento è quello di far sì che al range di valori di resistenza del PT1000, che va da 1000 ohm a 1385 ohm (pari al range che assume tra 0°C e 100°C) corrisponda un range di valori di tensione al punto rappresentato come VF1 nello schema (porta ADC) da 0V a 3.3V. La tensione in ingresso che alimenta i vari componenti del circuito è idealmente di 5 V ed è fornita da un alimentatore (il valore reale fornito alimentando tutti i componenti del dispenser scende a 4.8-4.9V, di ciò se n'è tenuto conto durante l'implementazione software).

I PT1000 impiegati in totale sono due, per cui sono stati realizzati due circuiti di condizionamento: uno su scheda millefori ed uno realizzato su breadboard, rispettivamente collegati al sensore di temperatura acqua fredda e calda. Il circuito su millefori ha in ingresso un fusibile per salvaguardare il circuito stesso in caso di tensione superiore ai cinque volt. Il circuito su breadboard riceve l'alimentazione da una porta sul circuito della millefori posizionata a valle del fusibile, in modo da poter proteggere anche quest'altro circuito. Per quanto riguarda la massa, il circuito su millefori è collegato alla massa del circuito su breadboard che a sua volta è collegato a massa a un'altra breadboard.

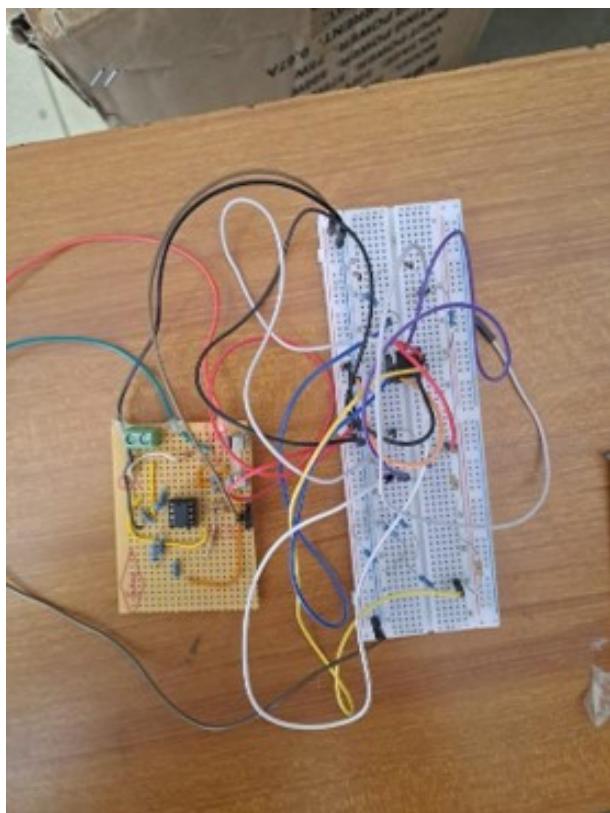


Figura 3.32: Enter Caption

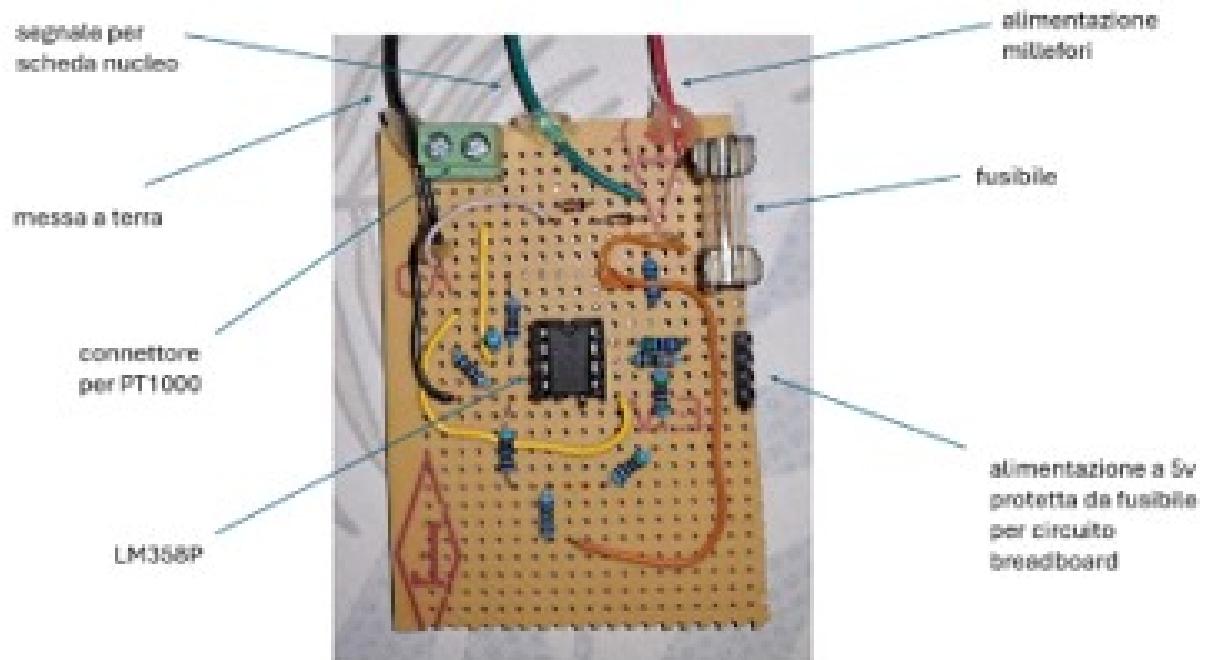


Figura 3.33: Enter Caption

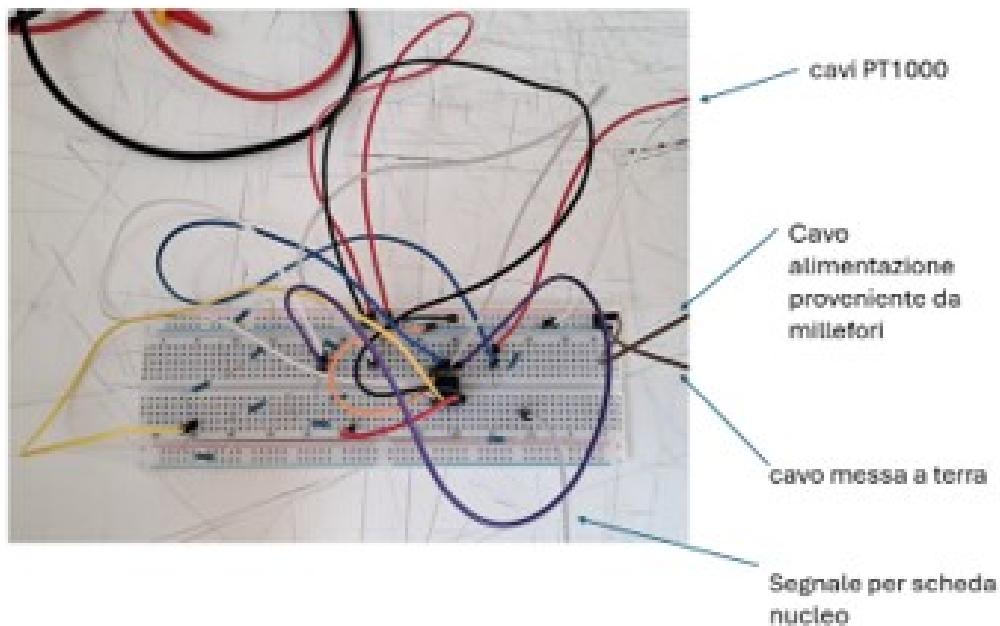


Figura 3.34: Enter Caption

Il motivo per cui i circuiti di condizionamento sono implementati su hardware diversi è dovuto al fatto che inizialmente si sarebbe voluto realizzare entrambi su scheda millefori per una maggiore resistenza meccanica. Successivamente, durante la realizzazione del primo circuito, constatando la necessità di poter apportare modifiche in maniera rapida, si è preferito realizzare il secondo su breadboard per avere una maggiore flessibilità.

I due Op-Amp rappresentati nello schema sono contenuti nei circuiti reali all'interno di chip LM358P. Anche questo componente è alimentato con una tensione di 5V.

LM358P

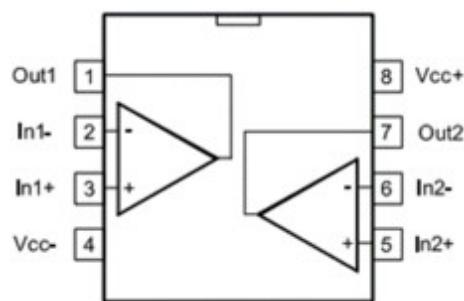


Figura 3.35: Integrato LM358P

I diodi zener da 1.8V in serie sono posizionati in via precauzionale, in quanto la porta per l'adc può ricevere al massimo un valore di 3.3 V. Qualora per qualsiasi motivo dovesse arrivare in ingresso un valore superiore a 3.6 V (valore limite), i diodi “aprirebbero” il circuito che andrebbe a scaricare la tensione sulla resistenza R9. Prima di giungere a questa scelta di diodi sono state provate altre soluzioni tra cui utilizzare diodi singoli da 3.3V e

3.6V. Si è optato per la soluzione a doppio diodo in quanto si è osservato che nonostante il valore nominale di funzionamento i diodi consentivano il passaggio di corrente a valori più bassi del valore nominale. Riducendo la tensione alla porta adc ed utilizzando i due diodi in serie da 1.8V si è notata una minore riduzione di tensione alla porta adc (riduzione di circa 0.1V).

I circuiti sono costituiti da due stadi di amplificazione. Si è scelta questa configurazione piuttosto che una singola amplificazione in quanto questa consente di mantenere una maggiore larghezza di banda complessiva e ottenere un'amplificazione più progressiva piuttosto che un'amplificazione più ampia con un singolo OpAmp.

Primo stadio: il partitore di tensione composto da PT1000, R1=2kohm e tensione in ingresso di 5V genera un range di tensione in uscita di 1.67-2.05V. Il guadagno di tensione dell'OpAmp (pari a 1.25 ottenuto con R3=500ohm e R4=2kohm) incrementa il range di tensione a 2.075-2.56V.

Formula tensione in uscita partitore di tensione: $V_{out} = V_{in} \cdot PT1000 / (R1 * PT1000)$

Formula guadagno di tensione: gain = $1 + R4/R3$

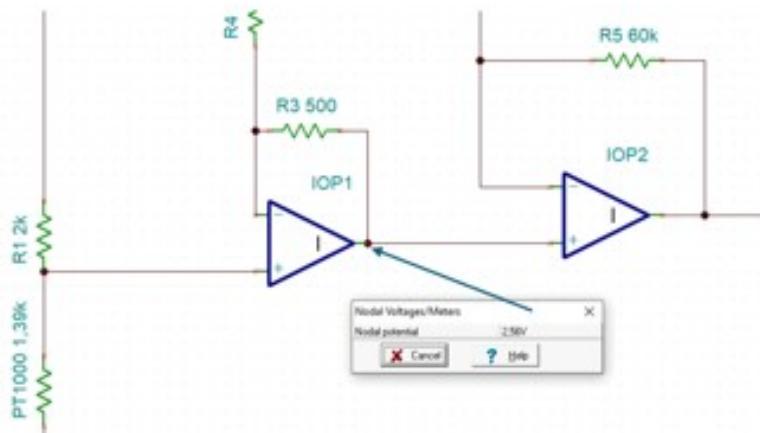


Figura 3.36: Enter Caption

Secondo stadio: questo stadio ha come obiettivo quello di amplificare l'escursione al range: 0-3.3V. Per conseguire l'obiettivo di espansione del range di tensione viene introdotta una tensione di offset. La tensione complessiva di uscita del secondo amplificatore (ottenuta dal contributo del guadagno di tensione dell'amplificatore e dalla tensione di offset) è data dalla seguente formula:

$$V_{out} = -V_{offset} \cdot \frac{R5}{R6} + V_{in} \cdot \left(1 + \frac{R5}{R6}\right)$$

Le resistenze R5 ed R6 sono state scelte in base al guadagno dell'amplificatore che deve corrispondere al rapporto tra il range di tensione in uscita ed in ingresso al secondo OpAmp:

$(3.3-0.0)V / (2.56-2.075)V = 6.95$ approssimato a 7. Da cui il guadagno $\left(1 + R5/R6\right) = 7 \rightarrow R5/R6 = 6$. Di conseguenza sono state scelte R5=60 kohm e R6=10 kohm.

La tensione di offset richiesta è stata individuata dalla formula inversa:

$$V_{offset} = \frac{R6}{R5} \cdot (-V_{out} + V_{in} \cdot (1 + \frac{R5}{R6}))$$

La formula è stata risolta in due modi differenti variando V_{out} e V_{in} con i valori in caso di 0 e 100°C. Nel caso ideale in cui si fosse utilizzato un guadagno dell'amplificatore di 6.95 si sarebbero ottenuti due valori identici, ma avendolo considerato pari a 7 si sono ottenuti valori per 2.43V e 2.45V. È stato scelto un valore di 2.45V in quanto consente una tensione in uscita più vicina allo 0V nel caso di 0°C (valori inferiori a 0.140V) mentre, nel caso di 100°C, si ottiene una tensione di 3.22V (sul modello Tina-TI è a 3.18V, molto vicino a quanto calcolato) che consente di avere un margine rispetto al valore massimo di 3.3V che possa ricevere in ingresso la porta della scheda nucleo (il range leggermente più ristretto rispetto all'ideale è stato tenuto in considerazione durante la fase di programmazione). Per ottenere questo valore di tensione di offset è stato implementato un partitore di tensione con resistenza $R7=1\text{kohm}$ collegata all'alimentazione e una resistenza da 960ohm collegata a massa (ottenuta mettendo in parallelo $R8=1\text{kohm}$ e $R2=24\text{kohm}$).

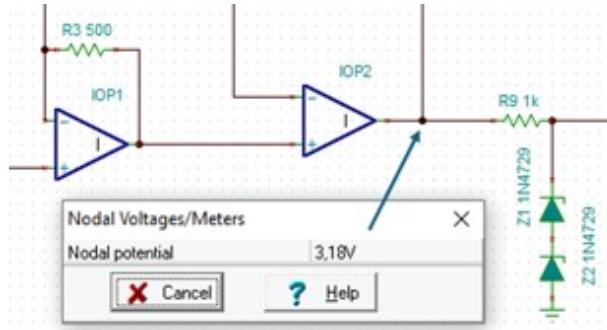


Figura 3.37: Enter Caption

Per avere conferma dei risultati analitici sono state effettuate misurazioni sui circuiti ideali ed è stato anche implementato un modello sul software TINA-TI. Ciò ha portato alla validazione dei dati analitici ed ha anche permesso di comprendere le non idealità dei circuiti come: le tolleranze dei resistori, la mancanza di precisione di erogazione di tensione a partire dalle sorgenti e le non idealità dei diodi zener. Questi ultimi fanno sì che la tensione in uscita dal secondo stadio, pari a 3.22V, arrivi alla porta per l'adc con un valore di tensione pari a circa 3.10V.

Il circuito su millefori è collegato alla porta PC3 (adc1 in4) e misura la temperatura dell'acqua fredda, mentre quello su breadboard è collegato alla porta PA4 (adc1 in9) e misura la temperatura dell'acqua calda. Il sensore dell'acqua calda è affisso tramite filo metallico e pasta termica (per una maggiore condutività) sulla superficie della camera di riscaldamento dell'acqua, mentre quello dell'acqua fredda è posizionato all'interno del circuito subito a valle del serbatoio dell'acqua fredda. Ad entrambe le PT1000 è stata aggiunta colla a caldo nel punto di terminazione della punta metallica al fine di aumentarne l'impermeabilità. Al termine dell'installazione dei sensori, i rispettivi circuiti di condizionamento sono stati allocati nel vano superiore del dispenser.

Modulo Bluetooth HC-05

La comunicazione wireless tra la scheda Nucleo STM32L476RG e lo smartphone è resa possibile integrando il modulo Bluetooth HC-05, figura 3.38. L'obiettivo principale è quello di poter controllare il dispenser d'acqua a distanza, tramite l'invio di semplici comandi da un'app per smartphone.

Il sistema rappresenta un tipico esempio di integrazione tra hardware embedded (STM32) e dispositivi mobili (smartphone), ed è pensato per semplificare l'interazione con il dispositivo finale (il dispenser) senza l'uso di cavi o interfacce complesse.



Figura 3.38: Modulo Bluetooth HC-05

La scelta del modulo bluetooth è ricaduta sul modulo HC-05, le motivazioni e le caratteristiche principali sono riportate di seguito:

- Facilità d'uso:
 - Comunica tramite porta seriale UART (TX/RX)
 - Puoi iniziare a usarlo senza configurazioni avanzate: basta collegarlo, accoppiarlo via Bluetooth e inviare/ricevere dati.
- Prezzo contenuto: è uno dei moduli Bluetooth più economici sul mercato, ideale per progetti didattici.
- Buona compatibilità:
 - Funziona bene con smartphone Android, PC, Raspberry Pi, e altri dispositivi con Bluetooth classico.
 - Supporta il profilo SPP (Serial Port Profile), che simula una porta seriale: perfetto per la trasmissione di dati testuali.
- Portata sufficiente: Portata di circa 10 metri, sufficiente per la maggior parte delle applicazioni domestiche o da laboratorio.
- Funziona subito:
 - Non hai bisogno di entrare nella modalità AT: puoi usarlo in modalità slave base, già pronta all'uso.

- Il modulo si collega automaticamente quando lo accoppi con un dispositivo Bluetooth.

Si è poi collegato il modulo bluetooth alla scheda nucleo effettuando le seguenti connessioni:

- HC-05 TX → STM32 RX (PA10 / USART1_RX)
- HC-05 RX → STM32 TX (PA9 / USART1_TX)
- GND in comune
- Alimentazione 5V su Vcc

3.2.2 Circuito di potenza

Il circuito di potenza alimenta i seguenti attuatori elettrici presenti nel dispenser:

- Circuito elettrico di riscaldamento
- Circuito elettrico di raffreddamento
- Elettrovalvole di ingresso serbatoi
- Elettrovalvole di uscita
- Pompa idraulica

Circuito elettrico di riscaldamento

L'acqua presente nel serbatoio caldo viene riscaldata per effetto joule da una resistenza di riscaldamento integrata all'interno del serbatoio stesso. Dallo schema elettrico, quando il relè K1 si eccita chiudendo il circuito, la corrente alternata scorre attraverso la resistenza R1. Quando l'acqua raggiunge la temperatura di ebollizione, il termostato TM1 (installato a contatto con il serbatoio) si apre e interrompe il passaggio di corrente. Il termostato rimane attivo (cioè aperto) finché l'acqua mantiene circa la temperatura di ebollizione, e si disattiva quando la temperatura cala sotto la soglia minima. I componenti elettrici del circuito caldo sono tutti installati di default nel dispenser originale e le specifiche tecniche non sono note.

Lo schema elettrico del circuito di riscaldamento è riportato nel foglio 1 dello schema elettrico.

Circuito elettrico di raffreddamento

Il circuito di raffreddamento è gestito dalla scheda PCB (Cooling Board) mostrata in figura 3.39. La scheda prende in ingresso la tensione alternata monofase 220V e, tramite un convertitore flyback, produce la tensione continua 12V per alimentare la cella Peltier, la ventola di dissipazione e il sensore NTC.

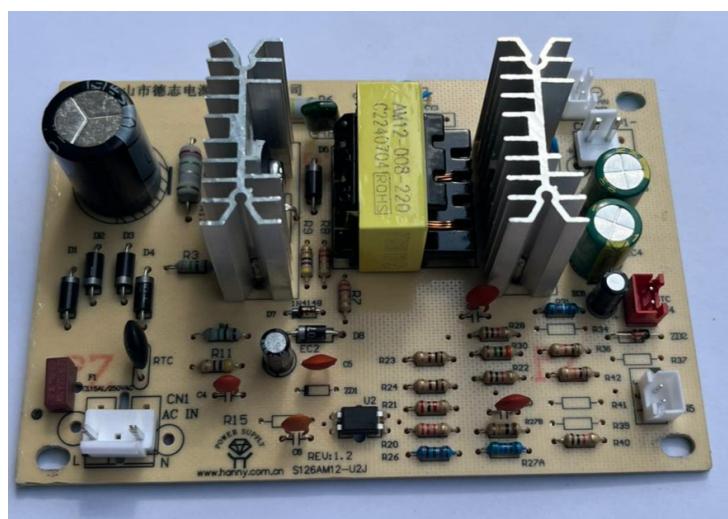


Figura 3.39: Cooling Board del dispenser

Lo schema elettrico del circuito di raffreddamento è riportato nel foglio 1 dello schema elettrico.

Pompa idraulica

La pompa idraulica scelta per l'applicazione è una pompa a membrana a 12V con motore DC. Nel corso della progettazione si sono prese in considerazione anche altre tipologie di pompe idrauliche, con costi e prestazioni differenti, ma la soluzione scelta rappresenta il miglior compromesso tra prestazioni, costo ed facilità d'uso.



Figura 3.40: Pompa idraulica a membrana

Si riportano in elenco le motivazioni di questa scelta tecnica:

- L'alimentazione a 12V della pompa è compatibile con l'alimentazione degli altri attuatori presenti nel circuito di potenza, permettendo l'integrazione diretta.
- La pompa è autoadescante e riesce a pescare l'acqua dal serbatoio principale fino a 2 metri di altezza massima.
- Portata massima dichiarata di 2L/min che consente di riempire bicchieri o borracce senza flussi d'acqua violenti e senza sovraccaricare i serbatoi.
- Pressione massima dichiarata di 0,3 bar sufficiente per aprire le valvole unidirezionali e sostenere le perdite di carico lungo le tubazioni.
- Controllo semplice tramite un relè oppure scheda driver in PWM.
- Tollerante a cicli on/off che la rende adatta al funzionamento intermittente del dispenser.
- Assorbimento moderato di 1-2A in esercizio e 3-4A allo spunto.
- Adatta per applicazioni alimentari (dispenser e macchine del caffè) o per acquari.
- Costo e dimensioni contenute.

Le specifiche tecniche della pompa sono le seguenti:

- Tensione nominale: 12V
- Corrente nominale: a vuoto 0,3-0,5A, in esercizio 1-2A, allo spunto 3-4A.

- Portata: circa 2L/min
- Pressione massima: 0,3 bar
- Prevalenza massima: 2 m

La pompa è pilotata in PWM dalla scheda driver descritta nel sottocapitolo precedente. La scheda consente un pilotaggio full-bridge o half-bridge. Dato che il motore elettrico della pompa deve ruotare solo in un verso, in quanto se ruotasse in senso contrario danneggerebbe le membrane interne della pompa, si è scelto uno schema half-bridge di collegamento. Il polo positivo del motore è collegato all'uscita B+ della scheda, mentre il polo negativo è collegato a massa. La scheda inoltre, per svolgere il suo funzionamento, prende in ingresso la 12V sul morsetto VCC ed è collegata a massa con il morsetto GND. Lo schema elettrico di collegamento della scheda driver con la pompa è mostrato nel foglio 2 dello schema elettrico.

3.2.3 Circuito di alimentazione

Il circuito di alimentazione si allaccia alla rete elettrica monofase 220V AC attraverso la spina shuko. Per preservare l'integrità elettrica dei vari componenti, sono presenti due fusibili F1 e F2 nei loro rispettivi portafusibili. Un interruttore bifase KDC4 è stato installato per comandare l'accensione dell'intero dispenser. I cablaggi elettrici in corrente alternata sono distanziati da quelli in corrente continua della parte di potenza. La figura 3.41 mostra i cablaggi elettrici in corrente alternata, le morsettiera e i giunti elettrici utilizzati.



Figura 3.41: Morsettiera di tensione alternata

I cavi neri rappresentano le fasi che alimentano il circuito elettrico di riscaldamento e raffreddamento, i cavi blu sono i rispettivi neutri collegati a terra (cavi giallo-verdi) come prevedono le norme di sicurezza elettrica. La figura 3.42 mostra invece i collegamenti del circuito di potenza a 12V. I cavi rossi sono a 12V (VCC) mentre quelli blu sono la rispettiva massa (GND).



Figura 3.42: Cablaggio 12V DC

Sono presenti due alimentatori per ricavare rispettivamente le tensioni 5V DC e 12V DC per il circuito di controllo e di potenza. Per il circuito di controllo si è scelto un alimentatore switching modello AP55A (codice 950097) della AlcaPower®. Le specifiche tecniche sono riportate in figura 3.43.

CARATTERISTICHE TECNICHE

Ingresso	100 -240V AC 50-60Hz 0,5A max
Potenza	25,0W
Uscita	5,0V DC 5,0A
Polarità connettore d'uscita	Tutti i connettori hanno il polo positivo centrale 
Classe di isolamento	Isolamento in classe II
Rendimento medio in modo attivo	85,16%
Rendimento a basso carico (10%)	78,79%
Consumo di potenza a vuoto	0,06W
Temperatura d'esercizio	-10°C ~ 40°C
Temperatura di stoccaggio	-25°C ~ 70°C
Lunghezza cavo d'uscita	115cm
Lunghezza cavo d'ingresso	100cm
Connettori d'uscita	5,5x2,1mm; 5,5x2,5mm
Dimensioni	98x45x29mm circa (ingombro contenitore)
Peso	174g circa

Figura 3.43: Specifiche tecniche AP55A

L'amperaggio fornito dall'alimentatore è ampiamente sovradimensionato per l'assorbimento del circuito di controllo. Si stima infatti che, a pieno carico e con un margine extra, il circuito di controllo richieda circa 1,5A di corrente per funzionare correttamente. Non disponendo di un alimentatore adeguato, si è optato per utilizzare l'AP55A. Un altro aspetto riguarda i cali di tensione dovuti ad assorbimenti eccessivi di corrente con alimentatori di amperaggio minore. Utilizzando l'AP55A i cali di tensione sono pressoché inesistenti e questo ha facilitato la fase di testing e convalidazione dei vari circuiti.

Per il circuito di potenza si è scelto un alimentatore switching 12V 150W della Parko®. Le specifiche tecniche dell'alimentatore sono:



Figura 3.44: Alimentatore Parko 12V DC

- Tensione nominale in ingresso: 176-264V AC
- Frequenza nominale in ingresso: 50/60 Hz
- Corrente nominale in ingresso: 2.12-0.92A AC
- Tensione nominale in uscita: 12V +/-5%
- Corrente nominale in uscita: 12.5A
- Potenza nominale: 150W

Anche in questo caso l'amperaggio fornito è ampiamente sufficiente ad alimentare tutti gli attuatori a 12V del circuito di potenza.

3.3 Layout finale di prodotto

In questa sottosezione si discute la disposizione dei principali componenti all'interno del dispenser con le relative immagini CAD.

3.3.1 Complessivo CAD

Prima di realizzare fisicamente il dispenser, il team di sviluppo ha riprodotto su CAD i principali componenti e l'assieme finale. Le figure 3.45 e 3.46 mostrano alcune viste del dispenser. Si tenga a precisare che il complessivo CAD non contiene tutti gli elementi, ma solo i componenti più voluminosi e determinanti per il corretto assemblaggio del dispenser.

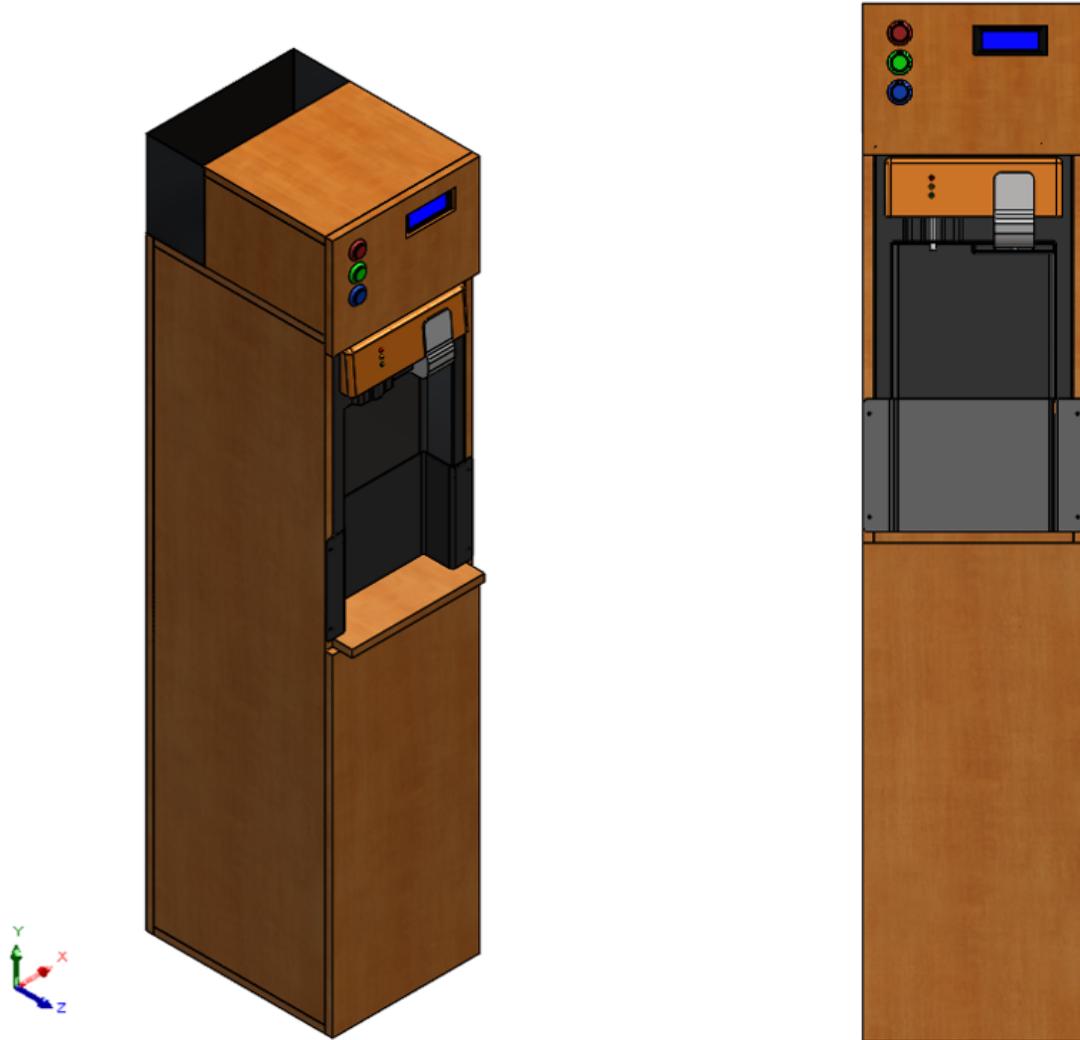


Figura 3.45: Vista trigonometrica dispenser

Figura 3.46: Vista frontale di dispenser

Lo "scheletro" del dispenser è realizzato in legno fenolico satinato spesso 15mm. Il dispenser è alto complessivamente 1400mm, largo 300mm e spesso 400mm. Il dispenser originale è collocato su un cassetto di legno che lo sorregge verticalmente e fissato con delle viti. Un secondo cassetto di legno con i sensori di presenza bicchieri (quello che sporge sotto

in figura 3.45) serve d'appoggio per il bicchiere o la borraccia. La porta apribile di legno, situata inferiormente, nasconde la tanica collocata in basso.

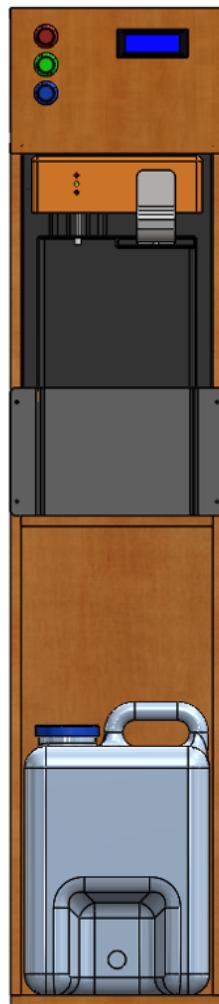


Figura 3.47: Vista con tanica

Per evitare che l'acqua entri in contatto con l'elettronica, il circuito elettrico di controllo è installato superiormente in cima al dispenser. Gli unici componenti che non sono presenti superiormente sono il sensore di livello, installato in cima alla tanica, e il flussimetro, installato dopo la pompa a mezza altezza del dispenser. Guardando frontalmente il dispenser si possono vedere i tre pulsanti di erogazione e il display LCD. Il sensore di presenza umana è installato sulla sommità del dispenser (non visibile nelle figure). Il circuito elettrico di potenza e gli alimentatori sono situati nella parte posteriore del dispenser, coperti con la lamiera nera di figura 3.48.



Figura 3.48: Vista HMI dispenser

4 Configurazione sul Cube MX

4.1 Configurazione PIN di comando della scheda STM XNUCLEO IHM04A1

Per pilotare correttamente la scheda di controllo motore scelta, bisogna collegare tre Pin sulla Nucleo nello specifico:

- PA12 che viene collegato al Pin EnB della XNUCLEO IHM04A1
- PB4 che viene collegato al Pin In1B della XNUCLEO IHM04A1
- PA11 che viene collegato al Pin In2B della XNUCLEO IHM04A1

La scheda XNUCLEO IHM04A1 può pilotare diversi motori a seconda di come si configurano le apposite resistenze sulla scheda e di come si pilotano i pin. Nel caso del dispenser, la configurazione da usare è quella che prevede l'uso di quattro motori in maniera unidirezionale (uno per ogni uscita di potenza). Nel caso considerato, andremo a collegare e pilotare solo 1 motore in accordo con gli schemi elettrici proposti nei capitoli precedenti.

Scelti i Pin da usare sulla Nucleo questi vanno configurati, si parte con i pin PA11 e PA12 che vanno programmati come GPIO Output perché sulla scheda XNUCLEO coincidono rispettivamente con il comando di un transistor del ponte (quello non usato) e l'abilitazione del canale B di potenza. Qua sotto si riporta l'immagine della configurazione di EnB.

Pin Name	Signal on Pin	GPIO output level	GPIO mode	GPIO Pull-up/Pull-down	Maximum output speed	Fast Mode	User Label	Modified
PA12	n/a	Low	Output Push Pull	No pull-up and no pull-down	Low	n/a	EnB	<input checked="" type="checkbox"/>
PA15 (JTDI)	n/a	Low	Output Push Pull	No pull-up and no pull-down	Low	n/a	CircuitoCaldo	<input checked="" type="checkbox"/>
PB2	n/a	n/a	External Interr...	No pull-up and no pull-down	n/a	n/a	PulsanteCalda	<input checked="" type="checkbox"/>
PB6	n/a	n/a	External Interr...	No pull-up and no pull-down	n/a	n/a		<input checked="" type="checkbox"/>
PB8	n/a	n/a	External Interr...	No pull-up and no pull-down	n/a	n/a	IR1	<input checked="" type="checkbox"/>
PB9	n/a	n/a	External Interr...	No pull-up and no pull-down	n/a	n/a	IR2	<input checked="" type="checkbox"/>
PB10	n/a	n/a	External Interr...	No pull-up and no pull-down	n/a	n/a	PulsanteTAmb	<input checked="" type="checkbox"/>
PB11	n/a	n/a	External Interr...	No pull-up and no pull-down	n/a	n/a	PulsanteFredda	<input checked="" type="checkbox"/>

Figura 4.1: Configurazione PA11 e PA12

Configurati i PIN PA11 e 12 è il tempo di configurare PB4 che è collegato all' ingresso In1B della XNUCLEO, ciò significa che PB4 deve generare il segnale PWM necessario al pilotaggio del motore. Per generare una PWM si deve configurare PIN in questione come timer, nello specifico la porta scelta coincide con il timer 3 canale 1.

La configurazione del timer 3 è la seguente: Come si può vedere, rispetto alle impo-

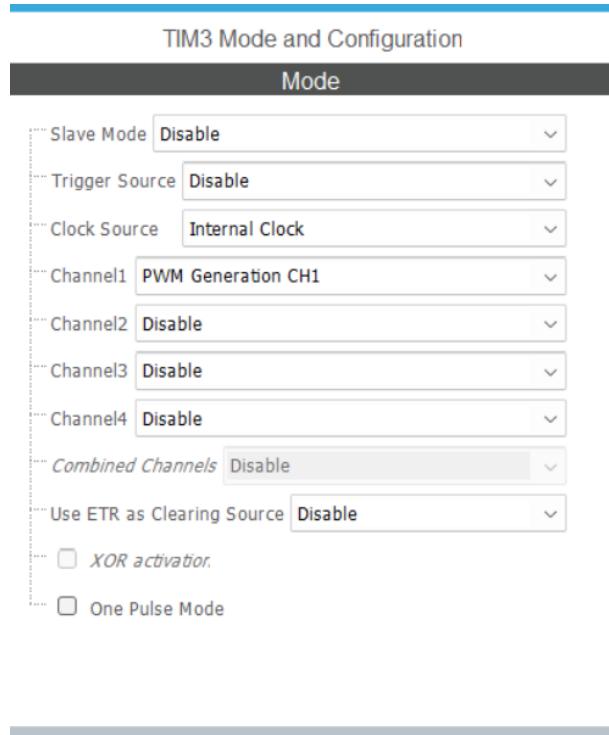


Figura 4.2: Mode timer 3

stazioni di default, è stato impostato il Clock Source come “Internal Clock” ed è stato impostato Channel 1 come “PWM Generation CH1”.

Il passo successivo è modificare i parametri del menù configuration. Come si vede dalla figura 4.3, i parametri modificati sono il “Prescaler” e il “Counter Period” per ottenere dalla PWM una frequenza di 10kHz. Per trovare i valori corretti da inserire è stata usata la seguente formula:

$$f_{PWM} = \frac{f_{timer_clock}}{(Prescaler + 1) \cdot (Counter\ Period + 1)}$$

Tenendo conto che il Clock della scheda è 80MHz i valori di Prescaler e Counter Period sono stati scelti rispettivamente con il valore di 79 e 99.

Trasportando questi valori nella formula si ottiene:

$$f_{PWM} = \frac{80\ 000\ 000}{(79 + 1) \cdot (99 + 1)} = \frac{80\ 000\ 000}{80 \cdot 100} = \frac{80\ 000\ 000}{8000} = 10\ 000\ Hz$$

Quindi si ottiene, come desiderato, una frequenza pari a 10kHz.

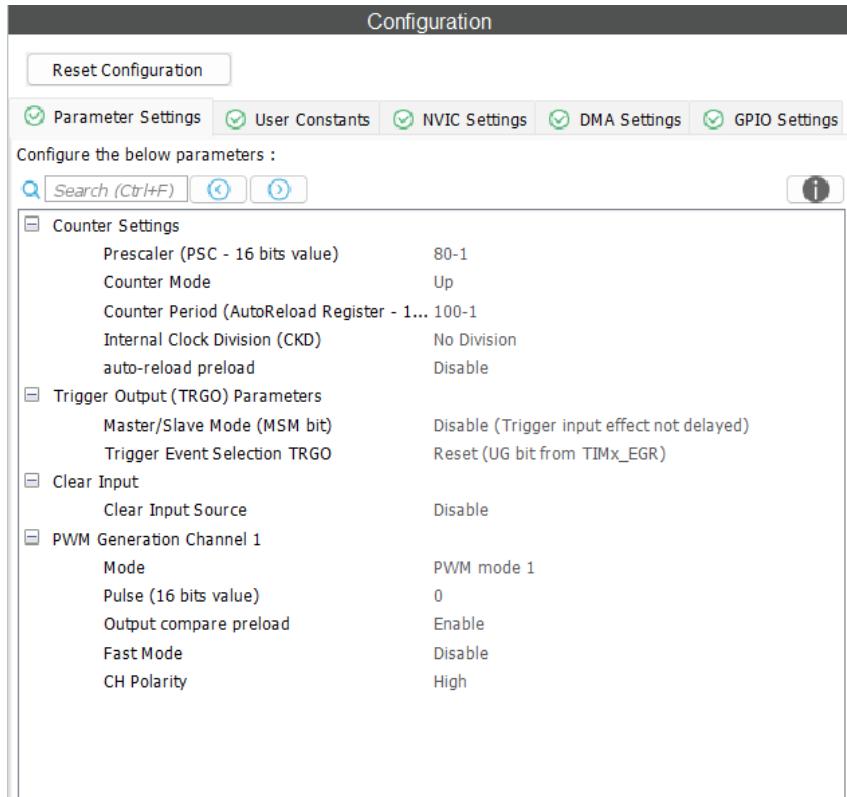


Figura 4.3: Menu Configuration timer 3

4.2 Configurazione PIN Flussimetro

Il collegamento del flussimetro, nel caso in questione si parla di un YF-S401, risulta molto semplice. Tralasciando i cavi di alimentazione del sensore, il flussimetro porta alla Nucleo un segnale ad impulsi tramite un solo filo. La frequenza di questi impulsi, riscalata mediante formule, va a fornire il valore di portata erogata. Per contare la frequenza degli impulsi è necessario contare i fronti di salita che si generano, in maniera analoga a quello che si fa con gli encoder. Il modo più semplice per farlo è settare il PB6 come “GPIO External Interrupt”.

Fatta questa impostazione si deve configurare il PIN scelto, nello specifico bisogna settare il monitoraggio del fronte di salita impostando la GPIO mode come: “External Interrupt Mode with Rising edge trigger detection”.

Nell’ immagine riportata sotto si può vedere la configurazione del PIN PB6.

Pin Name	Signal on Pin	GPIO output level	GPIO mode	GPIO Pull-up/Pul...	Maximum output...	Fast Mode	User Label	Modified
PA12	n/a	Low	Output Push Pull	No pull-up and no...	Low	n/a	EnB	<input checked="" type="checkbox"/>
PA15 (JTDI)	n/a	Low	Output Push Pull	No pull-up and no...	Low	n/a	CircuitoCaldo	<input checked="" type="checkbox"/>
PB2	n/a	n/a	External Interrupt...	No pull-up and no...	n/a	n/a	PulsanteCalda	<input checked="" type="checkbox"/>
PB6	n/a	n/a	External Interrupt...	No pull-up and no...	n/a	n/a	Flussimetro	<input checked="" type="checkbox"/>
PB8	n/a	n/a	External Interrupt...	No pull-up and no...	n/a	n/a	IR1	<input checked="" type="checkbox"/>
PB9	n/a	n/a	External Interrupt...	No pull-up and no...	n/a	n/a	IR2	<input checked="" type="checkbox"/>
PB10	n/a	n/a	External Interrupt...	No pull-up and no...	n/a	n/a	PulsanteTAmb	<input checked="" type="checkbox"/>
PB11	n/a	n/a	External Interrupt...	No pull-up and no...	n/a	n/a	PulsanteFredda	<input checked="" type="checkbox"/>
...	<input checked="" type="checkbox"/>

PB6 Configuration :

GPIO mode : External Interrupt Mode with Rising edge trigger detection

GPIO Pull-up/Pull-down : No pull-up and no pull-down

User Label : Flussimetro

Figura 4.4: Configurazione del Pin PB6 (flussimetro)

4.3 Configurazione PIN pulsanti

Si descrive ora la parte di configurazione del CUBE MX inherente ai tre pulsanti che servono per erogare le rispettive tipologie di acqua dal dispenser, nello specifico: calda, fredda e temperatura ambiente. Per configurare questi pulsanti sono stati individuati i seguenti tre PIN della scheda:

- PB2 per il pulsante dell’acqua calda
- PB10 per il pulsante dell’acqua a temperatura ambiente
- PB11 per il pulsante dell’acqua fredda

Individuati i PIN bisogna capire come gestirli. L’idea è quella di avere una gestione in Interrupt, più efficiente dal punto di vista dei consumi e delle risorse rispetto al polling.

In interrupt si va a monitorare lo stato della periferica solo quando accade qualcosa, all’opposto del polling dove è il codice che, periodicamente o ciclicamente, interroga la periferica in questione.

Nel dispenser i pulsanti sono componenti particolari, infatti, servono sia per attivare o disattivare l’erogazione ma al tempo stesso la loro pressione costante ha una funzione intrinseca di conferma. Per il dispenser, vedere l’ingresso associato al pulsante costantemente a “TRUE”, durante la fase di erogazione è la conferma che lato utilizzatore tutto funziona e si può continuare a lavorare in “sicurezza”.

Alla luce di questo funzionamento dei pulsanti, che si può definire come “Bi-stabile” (Lunghi periodi a “TRUE” alternati da lunghi periodi a “FALSE”) la scelta è stata quella di configurarli in modo analogo al flussimetro, cioè come “GPIO External Interrupt”.

La differenza rispetto al settaggio fatto per il flussimetro sta nell'impostazione del GPIO mode, dove questa volta si è scelto di usare la modalità “External Interrupt Mode with Rising/Falling edge trigger detection” in modo da monitorare sia il fronte di salita che quello di discesa.

Per completezza qua sotto si riporta solo la configurazione del PIN PB2 siccome è uguale a quella degli altri 2 pulsanti. Si fa notare come questi pulsanti sono senza impostazione

Pin Name	Signal on Pin	GPIO output level	GPIO mode	GPIO Pull-up/Pull-down	Maximum output current	Fast Mode	User Label	Modified
PA12	n/a	Low	Output Push Pull	No pull-up and no pull-down	Low	n/a	EnB	<input checked="" type="checkbox"/>
PA15 (JTDI)	n/a	Low	Output Push Pull	No pull-up and no pull-down	Low	n/a	CircuitoCaldo	<input checked="" type="checkbox"/>
PB2	n/a	n/a	External Interrupt	No pull-up and no pull-down	n/a	n/a	PulsanteCalda	<input checked="" type="checkbox"/>
PB6	n/a	n/a	External Interrupt	No pull-up and no pull-down	n/a	n/a	Flussimetro	<input checked="" type="checkbox"/>
PB8	n/a	n/a	External Interrupt	No pull-up and no pull-down	n/a	n/a	IR1	<input checked="" type="checkbox"/>
PB9	n/a	n/a	External Interrupt	No pull-up and no pull-down	n/a	n/a	IR2	<input checked="" type="checkbox"/>
PB10	n/a	n/a	External Interrupt	No pull-up and no pull-down	n/a	n/a	PulsanteTAmb	<input checked="" type="checkbox"/>
PB11	n/a	n/a	External Interrupt	No pull-up and no pull-down	n/a	n/a	PulsanteFredda	<input checked="" type="checkbox"/>
PB15	n/a	Low	Output Push Pull	No pull-up and no pull-down	Low	n/a	CaricoCircolare	<input checked="" type="checkbox"/>

PB2 Configuration :

- GPIO mode: External Interrupt Mode with Rising/Falling edge trigger detection
- GPIO Pull-up/Pull-down: No pull-up and no pull-down
- User Label: PulsanteCalda

Figura 4.5: Configurazione Pulsante PB2

di pull up o down interno, perché la rete è stata implementata esternamente tramite un ponte di resistenze esterno, in accordo con lo schema elettrico mostrato nel capitolo 3.

4.4 Configurazione PIN di comando dei Relè

Si passa ora a descrivere la configurazione dei PIN destinati al comando dei relè, si parla quindi dei PIN di uscita. Nel dispenser, come spiegato nei capitoli precedenti, sono presenti diversi relè che consentono di comandare le valvole e i vari sottosistemi come ad esempio il circuito di riscaldamento dell'acqua.

I relè ricevono il comando dalla scheda NUCLEO mediante cablaggio, diventa quindi essenziale assegnare un PIN ad ogni relè. Alla luce di ciò sono stati impostati un totale di sette PIN come “GPIO Output”. Nello specifico si parla di:

- PB15 elettrovalvola erogazione acqua calda;
- PC06 elettrovalvola erogazione acqua fredda;
- PC07 elettrovalvola caricamento circuito caldo;

- PC08 elettrovalvola caricamento circuito temperatura ambiente;
- PC08 elettrovalvola caricamento circuito freddo;
- PA15 circuito di riscaldamento dell'acqua;
- PC10 circuito di raffreddamento dell'acqua;
- PC11 elettrovalvola di sfiato del serbatoio caldo.

In questo caso non è stato modificato nulla rispetto ai parametri di Default. Siccome questi PIN sono tutti configurati allo stesso modo si riporta solo la configurazione di PC11.

Configuration

Group By Peripherals

GPIO Single Mapped Signals ADC I2C RCC SYS TIM USART NVIC

Search Signals Show only Modified Pins

Pin Name	Signal on Pin	GPIO output level	GPIO mode	GPIO Pull-up/Pull-down	Maximum output speed	Fast Mode	User Label	Modified
PB15	n/a	Low	Output Push...	No pull-up a...	Low	n/a	EvErogator...	<input checked="" type="checkbox"/>
PC6	n/a	Low	Output Push...	No pull-up a...	Low	n/a	EvErogator...	<input checked="" type="checkbox"/>
PC7	n/a	Low	Output Push...	No pull-up a...	Low	n/a	EvSerbatoio...	<input checked="" type="checkbox"/>
PC8	n/a	Low	Output Push...	No pull-up a...	Low	n/a	EVTAmb	<input checked="" type="checkbox"/>
PC9	n/a	Low	Output Push...	No pull-up a...	Low	n/a	EvSerbatoio...	<input checked="" type="checkbox"/>
PC10	n/a	Low	Output Push...	No pull-up a...	Low	n/a	CircuitoFreddo	<input checked="" type="checkbox"/>
PC11	n/a	Low	Output Push...	No pull-up a...	Low	n/a	EvSfatoCaldo	<input checked="" type="checkbox"/>
PC12	n/a	n/a	External Int...	Pull-down	n/a	n/a	PIR	<input checked="" type="checkbox"/>
PC13	n/a	n/a	External Int...	No pull-up a...	n/a	n/a	B1 [Blue Pu...	<input checked="" type="checkbox"/>

PC11 Configuration :

GPIO output level	Low
GPIO mode	Output Push Pull
GPIO Pull-up/Pull-down	No pull-up and no pull-down
Maximum output speed	Low
User Label	EvSfatoCaldo

Figura 4.6: Configurazione del pin PC11

4.5 Configurazione PIN sensore ultrasuoni

Per misurare la distanza mediante il sensore HC-SR04 è necessario generare sul pin TRIG un impulso di almeno 10 μ s e rilevare sul pin ECHO la durata dell'onda di ritorno, espressa anch'essa in microsecondi.

Configurazione del timer TIM1_CH1

È stato scelto TIM1_CH1, mappato su PA8, per l'Input Capture su ECHO del sensore.

- Modalità: Input Capture Direct Mode, permette di catturare automaticamente il valore del counter al fronte di salita e di discesa, senza software polling costante.
- Rising edge Polarity: per quanto riguarda la rilevazione dei fronti si configura la cattura al fronte di salita per iniziare la misura, e si inverte temporaneamente la polarità per catturare poi il fronte di discesa.
- Prescaler: 80 – 1, con un clock di sistema a 80 si ottiene un clock timer a 1 MHz, ovvero un tick ogni 1 μ s.
- Autoreload (ARR): impostato il valore di auto-reload al valore massimo (0xFFFF), si massimizza il range di conteggio prima di overflow. Ciò permette di misurare distanze fino a circa $(65 \text{ ms} * 340 \text{ m/s})/2 = 11 \text{ m}$, più che sufficienti per l'HC-SR04 (range 2 cm – 400 cm).
- NVIC: è necessario abilitare TIM capture compare interrupt per poter eseguire la callback in HAL_TIM_IC_CaptureCallback, dove si calcola la durata dell'impulso ECHO.

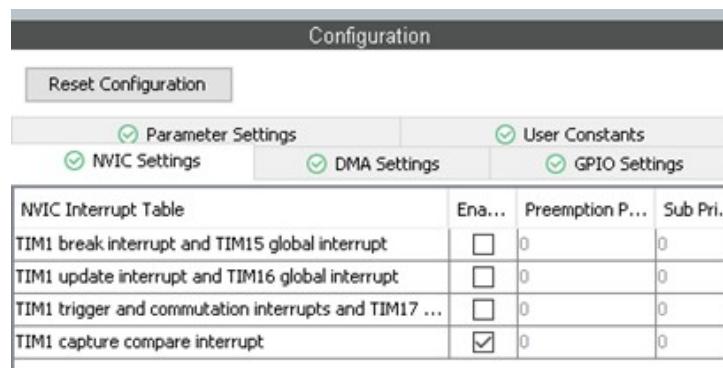


Figura 4.7: Cube MX TIM1_CH1 NVIC setting

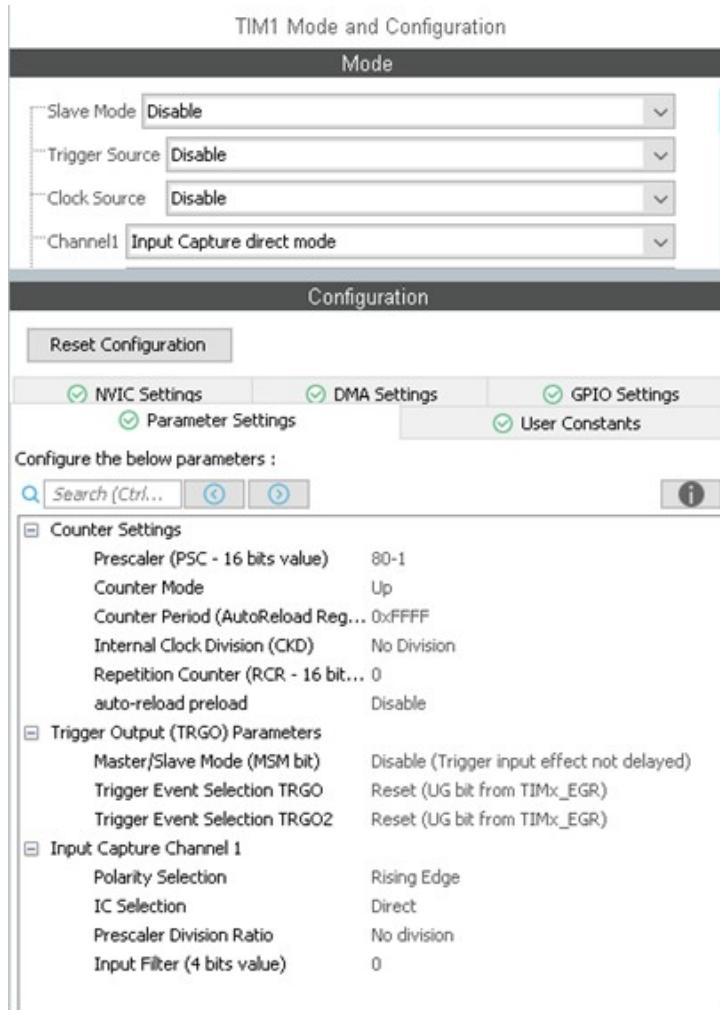


Figura 4.8: Cube MX TIM1_CH1 Mode and Configuration

Configurazione di TIM2_CH1

È stato selezionato TIM2_CH1, mappato su PA0, per la generazione dell’impulso TRIG. A differenza di una chiamata a HAL_Delay() oppure una busy-wait di 10 μ s, configurare il timer in questa modalità permette di non bloccare la CPU e quindi lasciandola libera di processare le altre attività o gestire altri interrupt, inoltre garantisce massima precisione fornendo un impulso esatto di 10 μ s.

- Modalità: PWM Generation Channel 1.
- Prescaler: 80 – 1, con un clock di sistema a 80 si ottiene un clock timer a 1 MHz, ovvero un tick ogni 1 μ s.
- Autoreload (ARR): impostato a 25000 - 1.
- Pulse (CCR1): 10, per avere una durata dell’alto pari a 10 μ s
- NVIC: Anche qui sono stati abilitati gli interrupt.

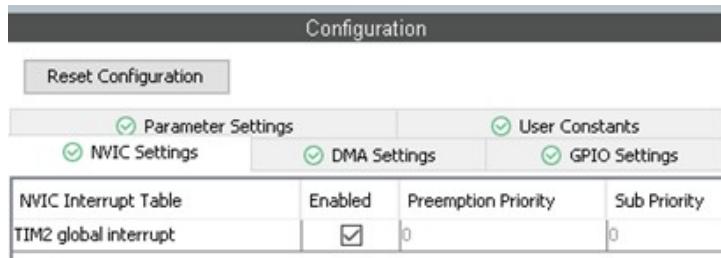


Figura 4.9: Cube MX TIM2_CH1 NVIC setting



Figura 4.10: Cube MX TIM2_CH1 Mode and Configuration

Il pin scelto per l'Input Capture su ECHO del sensore è PA8 e in questo caso è il sensore a comandare la nucleo; infatti, il sensore essendo alimentato a 5V porta questo pin a 5V quando manda l'impulso. Tuttavia, stando a quanto viene riportato nel datasheet del produttore della scheda nucleo è FT, ovvero tollera 5V in ingresso in digitale. Questo, tuttavia, non obbliga a usarlo a 5V e si è deciso di usarlo a 3.3V, usando un level shifter in fase di test, impedendo qualsiasi rischio se per sbaglio venissero impostati male i pin. I contro nell'adottare tale soluzione sono ovviamente i costi aggiuntivi, se anche trascurabili, e una maggiore complessità dovuta alla presenza di un componente aggiuntivo.

In fase di test si è deciso di non usare il level shifter per adattare il livello del TRIG del sensore, siccome è la nucleo che comanda il sensore.

4.6 Configurazione PIN Display LCD

Di seguito sarà possibile vedere le scelte prese per effettuare la configurazione dei pin in STM32CubeMX per l'interfacciamento con il display LCD.

Abilitazione e configurazione del periferico I2C

- Connectivity: I2C.
- I2C Speed Frequency: 10 kHz. Inizialmente, cercando in rete, era emerso che era possibile aggiornare il display con una frequenza di 100 kHz; tuttavia, gli aggiornamenti frequenti e le lunghe stringhe hanno dato come risultato simboli strani, byte corrotti. Per risolvere questo problema si è optato per una frequenza di 10 kHz, riducendo così la velocità di aggiornamento e questo ha portato a meno errori in fase di test.

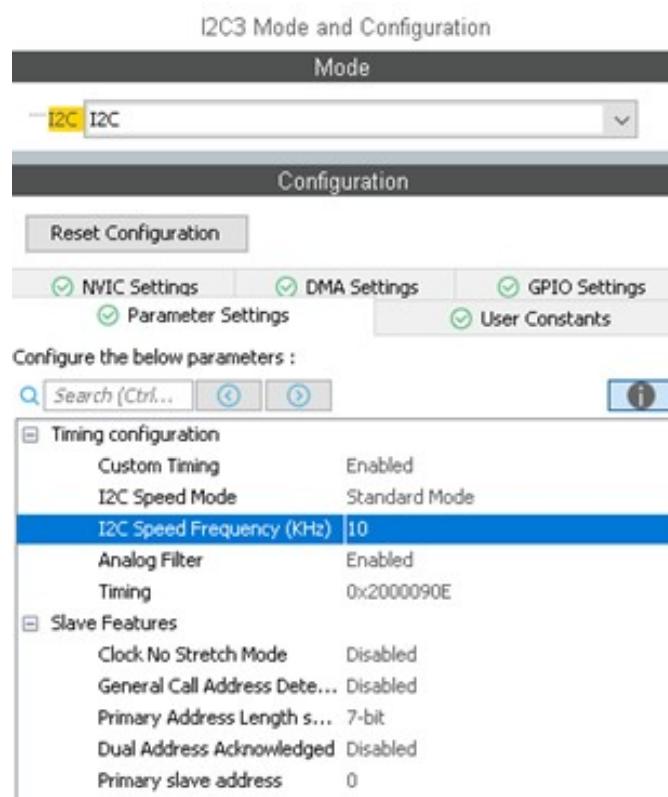


Figura 4.11: Cube MX I2C3 Mode and Configuration

Configurazione dei pin SCL e SDA

- PC0: I2C3_SCL
- PC1: I2C3_SDA

I pin scelti, stando a quanto viene riportato nel datasheet del produttore della scheda nucleo sono FT, ovvero tollerano 5V in ingresso in digitale. Questo, tuttavia, non obbliga a usarli a 5V e si è deciso di usarli a 3.3V, usando un level shifter in fase di test, impedendo qualsiasi rischio se per sbaglio venissero impostati male i pin. I contro nell'adottare tale soluzione sono ovviamente i costi aggiuntivi, se anche trascurabili, e una maggiore complessità dovuta alla presenza di un componente aggiuntivo.

Table 16. STM32L476xx pin defini

Pin Number												Pin name (function after reset)	Pin type	I/O structure
LQFP64	LQFP64_SMPS	WL CSP72	WL CSP72_SMPS	WL CSP81	WL CSP99_SMPS	LQFP100	UF BGA132	UF BGA132_SMPS	LQFP144	LQFP144_SMPS	UF BGA144			
5	5	D9	D9	D9	E11	12	F1	F1	23	23	G2	PH0-OSC_IN (PH0)	I/O	FT
6	6	D8	D8	D8	E10	13	G1	G1	24	24	G1	PH1-OSC_OUT (PH1)	I/O	FT
7	7	E9	E9	E9	E9	14	H2	H2	25	25	H4	NRST	I/O	RST
8	8	F9	F9	F9	F10	15	H1	H1	26	26	H2	PC0	I/O	FT_fla
9	9	F8	F8	F8	F11	16	J2	J2	27	27	H3	PC1	I/O	FT_fla

Figura 4.12: Datasheet nucleo STM32L476RG

4.7 Configurazione PIN sensore di movimento

Per rilevare la presenza di movimento viene utilizzato il sensore PIR HC-SR501 in maniera tale da generare un interrupt sul microcontrollore STM32 sul fronte di salita.

Configurazione del pin di input PIR

- Pin scelto: PC12.
- Mode: External Interrupt Mode with Rising trigger detection, in maniera tale che il sensore riesca a catturare il fronte di salita (LOW - HIGH) quando il sensore rileva un movimento, utile per rilevare l'inizio di un impulso.
- Pull-down: serve a evitare falsi trigger mantenendo il pin a LOW se si trova in stato flottante. Il pin rimane normalmente a 0V e andrà a 1 solo quando il sensore lo porta HIGH.
- NVIC: si associa l'EXTI line [15:10] a PC12 e si abilita l'interrupt nel NVIC affinché ad ogni evento in ingresso venga invocata la callback HAL_GPIO_EXTI_Callback.

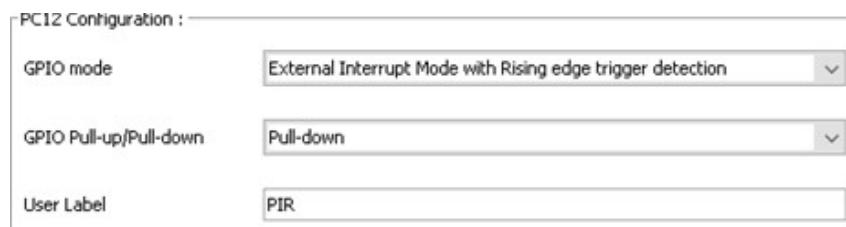


Figura 4.13: Cube MX PC12 Configuration

The screenshot shows the NVIC Interrupt Table configuration in Cube MX. The table has four columns: Enabled, Preemption Priority, and Sub Priority, all set to 0. The first row, 'EXTI line[9:5] interrupts', has its 'Enabled' checkbox unchecked. The second row, 'EXTI line[15:10] interrupts', has its 'Enabled' checkbox checked.

NVIC Interrupt Table	Enabled	Preemption Priority	Sub Priority
EXTI line[9:5] interrupts	<input type="checkbox"/>	0	0
EXTI line[15:10] interrupts	<input checked="" type="checkbox"/>	0	0

Figura 4.14: Cube MX PC12 NVIC setting

Il pin considerato viene mostrato sul datasheet essere FT, ovvero 5V tollerant. Questo significa anche che, se arriva il segnale 3.3 V del pin di uscita del sensore, il pin viene interpretato correttamente come HIGH.

4.8 Configurazione PIN sensori presenza bicchiere/borraccia

Per rilevare la presenza di bottiglia, bicchiere o borraccia viene utilizzato il sensore IR TCRT5000 in maniera tale da generare un interrupt sul microcontrollore STM32 sul fronte di salita e di discesa quando viene rilevata presenza di un oggetto, oppure quando viene rimosso.

Configurazione del pin di input IR1

- Pin scelto: PB8.
- Mode: External Interrupt Mode with Rising/Falling edge trigger detection, in maniera tale che il sensore riesca a catturare il fronte di salita quando il sensore rileva la presenza di un oggetto, e quando viene rimosso.
- Pull-up/Pull-down: Pull-up, perché l'uscita del LM393 funziona meglio con pull-up. In maniera tale che, quando il sensore rileva un oggetto il comparatore chiude verso GND, e viceversa quando non rileva un oggetto, questo va fatto perché il modulo LM393 è un'uscita open-drain, cioè il pin di uscita può solo tirare a massa, ma non può forzare a HIGH.
- NVIC: si associa l'EXTI line [9:5] a PB8 e si abilita l'interrupt nel NVIC affinché ad ogni evento in ingresso venga invocata la callback HAL_GPIO_EXTI_Callback.

The screenshot shows the PB8 Configuration window. It includes fields for 'GPIO mode' (set to 'External Interrupt Mode with Rising/Falling edge trigger detection'), 'GPIO Pull-up/Pull-down' (set to 'No pull-up and no pull-down'), and 'User Label' (set to 'IR1').

Figura 4.15: Cube MX PB8 Configuration

The screenshot shows a table within the Cube MX software interface. The table has a header row with columns: NVIC Interrupt Table, Enabled, Preemption Priority, and Sub Priority. Below the header, there are two rows of data. The first row corresponds to 'EXTI line[9:5] interrupts' and has a checked 'Enabled' checkbox, a 'Preemption Priority' of 0, and a 'Sub Priority' of 0. The second row corresponds to 'EXTI line[15:10] interrupts' and has an unchecked 'Enabled' checkbox, a 'Preemption Priority' of 0, and a 'Sub Priority' of 0.

NVIC Interrupt Table	Enabled	Preemption Priority	Sub Priority
EXTI line[9:5] interrupts	<input checked="" type="checkbox"/>	0	0
EXTI line[15:10] interrupts	<input type="checkbox"/>	0	0

Figura 4.16: Cube MX PB8 NVIC setting

Analogamente si procede con configurazione del pin di input IR2, dove è stato però selezionato il pin PB9.

4.9 Configurazione PIN modulo Bluetooth HC-05

Prima di effettuare il codice, si è andati a configurare il CubeMX, andando subito ad abilitare la USART1 nella sezione "Connectivity" → USART1; per poi andare a selezionare ed impostare le seguenti caratteristiche:

- Modalità: Asynchronous
- TX: PA9
- RX: PA10
- Baud rate: 9600 (default del HC-05)
- Data bits: 8
- Parity: None
- Stop bits: 1
- Mode: TX and RX

5 Implementazione software su Cube IDE

In questa parte di relazione ci si focalizza sulla descrizione del codice inerente alle funzioni implementate nel firmware del dispenser.

Si vuole far notare come questo sia un report, quindi il suo scopo è riassumere le specifiche e focalizzarsi sugli aspetti più importanti del codice. In questo capitolo ci si limita a descrivere solo cosa accade in modo relativamente generico.

Per quello che riguarda le variabili di questa parte di codice si è deciso di usare un file denominato nbVariabili come appoggio, in modo da avere le variabili tutte raccolte in un unico file. Molte variabili come, ad esempio, quelle che gestiscono il PID, la pompa e il dispenser sono di tipo “Struct” in modo tale da rendere il codice complessivamente più intuitivo e ordinato.

Le funzioni verranno descritte lungo il capitolo nell'ordine in cui si presentano nel file “nbFunzioni.c”; di conseguenza le prime sono quelle relative al controllo PID.

5.1 Controllore PID

Il PID si basa essenzialmente su cinque funzioni:

1. nbInitPID
2. nbTunePID
3. nbResetPID
4. nbControllerPID
5. nbPID

Le funzioni sopra descritte hanno tutti scopi ben precisi e servono per controllare il PID e manipolare i dati che fornisce. Partiamo ad analizzare nell'ordine le funzioni sopra citate:

Funzione nbInitPID

Questa funzione, come dice il nome, inizializza il calcolo integrale (somma dei valori di errore moltiplicati per le apposite costanti), azzera il calcolo derivativo e l'offset. Questa funzione viene eseguita all'avvio del codice.

```
void nbInitPID(PIDController_t *PID, float Tc, float u_max, float u_min,
                float offset) {

    PID->Integral = 0; //Azzero il calcolo integrale
    PID->PrevError = 0; //Azzero il calcolo derivativo
    PID->Offset = offset; //Azzero l'offset

}
```

Figura 5.1: Funzione nbInitPID

Funzione nbTunePID

Questa funzione serve per impostare i valori di tutti i parametri del controllore. Nello specifico ci si riferisce alle tre costanti principali: proporzionale, integrativa e derivativa; alla costante anti-windup; alla costante di tempo e ai valori del saturatore di uscita massima e minima. Siccome questi valori non sono soggetti a modifiche durante l'esecuzione del codice la funzione è richiamata solo in fase di avviamento del microcontrollore.

```

void nbTunePID(PIDController_t *PID, float Kp, float Ki, float Kd, float Kb,
                float Tc, float UscitaMassima, float UscitaMinima) {

    PID->Constant.Kp = Kp; //Aggiorno la costante proporzionale
    PID->Constant.Ki = Ki; //Aggiorno la costante integrativa
    PID->Constant.Kd = Kd; //Aggiorno la costante derivativa
    PID->Constant.Kb = Kb; //Aggiorno la costante anti-windup
    PID->Constant.Tc = Tc;
    PID->Constant.UscitaMax = UscitaMassima;
    PID->Constant.UscitaMin = UscitaMinima;

}

```

Figura 5.2: Funzione nbTunePID

Funzione nbResetPID

Questa funzione consente di resettare la variabile che raccoglie i risultati dell'integrale e dell'errore precedente. È molto simile alla funzione “nbInitPID” e viene usata ogni volta che si ultima una erogazione nella quale è stata richiesta una quantità di acqua predefinita oppure quando non si richiede l'uso del controllore.

```

void nbResetPID(PIDController_t *PID) {

    PID->Integral = 0;
    PID->PrevError = 0; //Azzero i termini di sommatoria

}

```

Figura 5.3: Funzione nbResetPID

Funzione nbControllerPID

In questa funzione semplicemente si sceglie se attivare o meno il controllore PID, in caso affermativo si va ad usare la funzione “nbPID”, altrimenti viene fatto un reset del PID e l'uscita è tenuta a zero.

```

float nbControllerPID(PIDController_t *PID, bool FcOnOff, float FcMisura,
                      float FcSetPoint, uint32_t *FcContatore) {

    //Definisco l'uscita del PID
    float Output;

    //Ora richiamo il controllore solo quando richiesto altrimenti uscita nulla
    if (FcOnOff) {

        Output = nbPID(&(*PID), FcMisura, FcSetPoint, &(*FcContatore));

    } else {

        nbResetPID(&(*PID));
        Output = 0;

    }

    return Output;
}

```

Figura 5.4: Funzione nbControllerPID

Questa funzione è usata all'interno della fase di erogazione, di conseguenza, viene richiamata nella macchina a stati presente nella funzione “nbGesioneDIspenser”.

Funzione nbPID

Questa è la funzione dove effettivamente si implementa il controllore PID. Lo schema del controllo che si è voluto implementare è il seguente: Il controllore PID (Proporzionale-

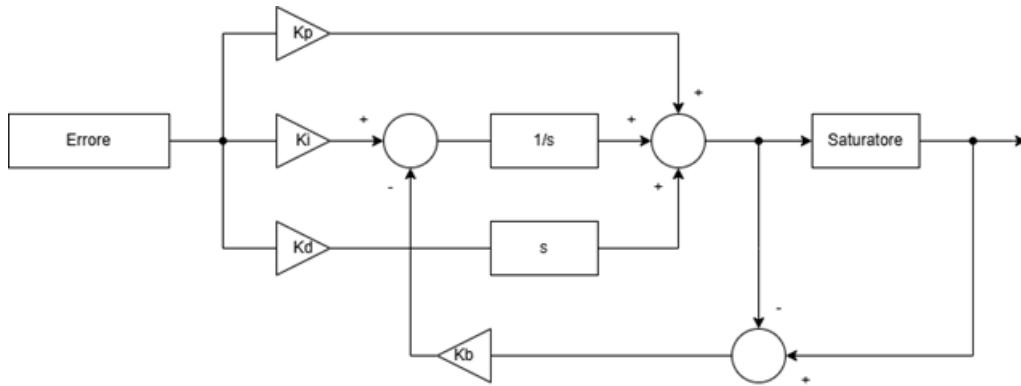


Figura 5.5: Schema controllore PID

Integrativo-Derivativo) è una tipologia di controllore ed è la più diffusa per la regolazione dei sistemi dinamici, grazie alla sua capacità di controllare efficacemente un'ampia gamma di processi. I controllori PID calcolano l'uscita del sistema di controllo utilizzando quattro componenti principali:

- Errore (e): è definito come la differenza tra il segnale di riferimento, ovvero il valore desiderato, e il valore misurato dal sistema
- Proporzionale (P): componente proporzionale all'errore. Un valore proporzionale più alto comporta una risposta più reattiva alle variazioni dell'errore
- Integrativo (I): componente proporzionale all'integrale dell'errore. Tiene conto degli errori passati accumulati nel tempo. La componente integrativa è richiesta per imporre che l'errore si annulli asintoticamente a fronte di segnali di riferimento o disturbi additivi costanti, che sono quelli che si hanno in questo caso
- Derivativo (D): questa componente è proporzionale alla derivata dell'errore. Tiene conto del tasso di cambiamento dell'errore nel tempo e perciò ha il compito di provare ad anticipare l'andamento dell'errore negli istanti futuri, aiutando quindi a prevenire oscillazioni e a migliorare la stabilità del sistema

La combinazione dell'azione proporzionale, integrativa e derivativa consente al controllore PID di adattarsi dinamicamente alle variazioni del sistema e di mantenere la variabile controllata il più vicino possibile al valore desiderato. I parametri del controllore PID ovvero K_p , K_i e K_d , che sono rispettivamente la costante proporzionale, integrativa e derivativa, devono essere ottimizzati per adattarsi al sistema. I valori da scelti per il dispenser sono:

- $K_p = 110$
- $K_i = 5$
- $K_d = 0$

- $K_b = 0.5$

Come si può notare K_d è imposta a zero segno che si vuole escludere la componente derivativa. Questa cosa è possibile siccome non è obbligatorio che tutte e tre le azioni siano contemporaneamente presenti, in particolare è possibile impiegare soltanto una di esse o come in questo caso, la combinazione di due.

Nel caso in questione siamo davanti ad un regolatore PI cioè una rete ritardatrice. Di norma questa rete si usa quando l'azione integrale è indispensabile per le prestazioni statiche (errore nullo a regime), ma è necessaria anche una banda passante più ampia rispetto a quella ottenibile con un semplice regolatore integrale.

Fatte queste premesse riportiamo sotto l'anello di retroazione.

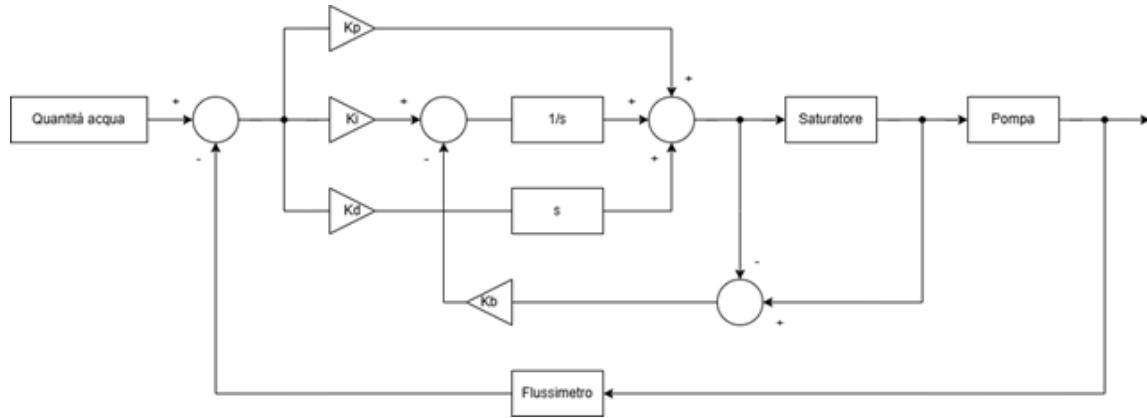


Figura 5.6: Anello di retroazione implementato nel dispenser

Le 3 costanti sono importanti, ma non sono le uniche costanti che servono per far funzionare il controllore. Di seguito si elencano gli altri valori del controllore:

- $T_c = 0.0001$ è la costante di tempo su cui si calcola la derivata e l'integrale
- $U_{Max} = 98$ è il valore massimo oltre il quale interviene il blocco di saturazione
- $U_{Min} = 0$ è il valore minimo al di sotto del quale interviene il blocco di saturazione
- $Offset = 0$ è il valore di offset per il controllore

Il passo successivo, finite le premesse, è quello di descrivere la funzione che implementa questa rete.

```

//Controllore PID
float nbPID(PIDController_t *PID, float FcMisura, float FcSetPoint,
             uint32_t *FcContatore) {

    float Errore = 0;
    float PTerm = 0.0;
    float NewITerm = 0.0;
    float DTerm = 0;
    float Output = 0;
    float SaturatedOutput = 0;
    float Correction = 0;

    //Piccolo check se integrale è ok => no valore infinito o nullo
    if (isinf(PID->Integral) || isnan(PID->Integral))
        nbResetPID(&(*PID));

    //Calcolo l'errore
    Errore = FcSetPoint - FcMisura;

```

Figura 5.7: Funzione nbPID parte 1

Questa funzione restituisce un valore compreso tra i limiti imposti dal saturatore e in accordo con la rete che rappresenta. Nel pezzo di funzione sopra riportato si definiscono le variabili che vengono usate nella funzione, successivamente si esegue un controllo, non inserito nella rete mostrata in figura [5.6]. Questo passaggio verifica che il calcolo integrale e derivativo non siano degenerati rispettivamente verso un valore infinito o nullo. In tal caso c'è stato un errore e bisogna resettare il controllore onde evitare errori di calcolo. Fatti i controlli necessari si può finalmente implementare la rete di figura 5.6, il primo passo da fare per implementare il regolatore PID è calcolare l'errore tra la quantità di acqua richiesta e lo stato attuale di quantità di acqua erogata.

```

//Calcolo il termine proporzionale
PTerm = PID->Constant.Kp * Errore;

//Calcolo il termine integrale
if (PID->Constant.Ki == 0) {
    NewITerm = 0;
} else {
    NewITerm = PID->Integral
        + (PID->Constant.Ki * PID->Constant.Tc * PID->PrevError);
}

//Calcolo il termine derivativo
if (PID->Constant.Kd == 0) {
    DTerm = 0;
} else {

    DTerm = (PID->Constant.Kd / PID->Constant.Tc)
        * (Errore - PID->PrevError);

}

//Calcolo il valore dell'uscita
Output = PTerm + NewITerm + DTerm + PID->Offset;

//Aggiorno l'errore precedente
PID->PrevError = Errore;

```

Figura 5.8: Funzione nbPID parte 2

Nella parte di codice sopra mostrata si calcolano i tre termini: il proporzionale, l'integrativo e il derivativo in accordo con le costanti impostate e la formula del regolatore. I valori così ottenuti sono sommati al fine di calcolare l'uscita che andrebbe direttamente a settare il duty cycle della PWM. Prima però di impostare il valore di uscita è necessario passare per il blocco di saturazione.

Si è già accennato in più parti del capitolo al blocco saturatore, ma precisamente come funziona? Il blocco di saturazione funziona così: si prende l'uscita e, se questa è maggiore del limite superiore del saturatore, la si equipara al valore di duty cycle massimo; se invece il valore di uscita previsto è troppo basso o negativo, lo si impone uguale al valore minimo impostato nel saturatore. Questa descrizione si traduce nel seguente codice.

```

//Saturazione dell'output
SaturatedOutput = Output;

if (SaturatedOutput > PID->Constant.UscitaMax) {

    SaturatedOutput = PID->Constant.UscitaMax;

} else if (SaturatedOutput < PID->Constant.UscitaMin) {

    SaturatedOutput = PID->Constant.UscitaMin;

}

```

Figura 5.9: Funzione nbPID parte 3

Ultimo passaggio ma comunque indispensabile, è il calcolo della correzione da effettuare sul termine integrativo dato dalla rete anti wind-up, dove si va a sottrarre al termine integrale la parte di valore dell'uscita effettivamente utilizzata rispetto a quella calcolata, al fine di contenere l'errore integrale.

Questa parte serve solo quando interviene il saturatore, infatti quando quest'ultimo non è attivo la rete anti wind-up genera un'uscita nulla. Qua sotto si riporta il calcolo implementato a livello software.

```

Correction = (PID->Constant.Kb * (SaturatedOutput - Output))
            * (PID->Constant.Tc);
PID->Integral = NewITerm + Correction;

return SaturatedOutput;

```

Figura 5.10: Funzione nbPID parte 4

5.2 Gestione pompa

Funzione nbGestionePompa

Questa funzione gestisce il funzionamento della pompa, coordinando i vari input e selezionando la velocità corretta della pompa. La funzione in ingresso presenta diversi parametri:

- “FcEnable”: è il booleano per l’abilitazione della pompa
- “FcStartStop”: è il comando per la partenza e l’arresto della pompa
- “FcSelettore velocità fissa”: serve per selezionare se leggere una velocità fissa o se deve usare l’ingresso proveniente dal duty cycle
- “FcDutyCICle”: è come dice il nome il valore di duty cycle da passare alla pompa quando si richiede il funzionamento a velocità variabile
- “FcVelFissa”: è come dice il nome è il valore di duty cycle da passare alla pompa quando si richiede il funzionamento a velocità costante
- “FcStatoPompa”: questa variabile, a differenza delle altre, deve essere non solo letta ma anche scritta e contiene una informazione sullo stato della pompa, lo stato può essere:
 - PompaSPENTA
 - PompaREADY
 - PompaIN_FUNZIONE

Il nome scelto per gli stati spiega già a cosa corrisponda ciascuno di loro; l'unica differenza da sottolineare è tra lo stato di PompaREADY e pompaSPENTA. Nel primo ci si trova quanto FcEnable è a “FALSE”, mentre si è nel secondo stato quando la variabile booleana è a “TRUE”. In entrambi i casi si ha la pompa disattivata e il duty cycle a zero.

Si procede ora ad analizzare nel dettaglio la funzione. La prima cosa da fare nella funzione

è la definizione della variabile che verrà restituita al termine dell'esecuzione della funzione stessa, cioè "FcOutPompa"; questa variabile è di tipo Struct e contiene la velocità e l'abilitazione, più precisamente, il duty cycle della PWM e l'abilitazione booleana da dare al GPIO Output che pilota EnB; Fatto questo si effettuano le verifiche per capire in che

```
typedef struct{
    uint16_t Vel;
    bool Enable;
} OutPompa_t;
```

Figura 5.11: Struttura del tipo di dato restituito dalla funzione nbGestionePompa

funzione si trova la pompa, da notare che qua non è stata definita una struttura di tipo case con tutti gli stati ma dei semplici "if" che hanno l'auto ritenuta nello stato stesso, poichè le condizioni di funzionamento sono abbastanza semplici e non vi sono ambiguità dettate da "stati" diversi, che necessitano delle stesse condizioni per essere attivati. Qua sotto si riporta per completezza la foto dello "stato" di pompaSPENTA. Dopo aver

```
//Pompa SPENTA
if (*FcStatoPompa == PompaSPENTA || FcEnable == false) {

    *FcStatoPompa = PompaSPENTA;
    FcOutPompa.Enable = false;      //Enable uscita =0
    FcOutPompa.Vel = 0;

}
```

Figura 5.12: Funzione nbGestionePompa – stato di pompaSPENTA

velocemente introdotto le logiche dello stato di pompaSPENTA e pompaREADY chiudiamo la trattazione con lo stato di "pompaIN_FUNZIONE", dove semplicemente si va a settare a "TRUE" l'enable della pompa (cioè il PIN PA12 = EnB) e il valore di duty cycle corretto è passato all'uscita.

Le "velocità" passate alla pompa possono essere solo 2: una che deriva dal PID e una fissa che deriva invece che si imposta in tutte quelle situazioni di erogazione manuale o di comando tramite applicazione. In questo secondo caso si imposta un valore di duty cycle e si mantiene quello a prescindere dal tipo di funzionamento

```

//Pompa in Funzione
if (*FcStatoPompa == PompaIN_FUNZIONE || (FcStartStop && FcEnable)) {

    //Aggiorno lo stato
    *FcStatoPompa = PompaIN_FUNZIONE;
    FcOutPompa.Enable = true;

    if (FcSelVelFissa == false) {

        FcOutPompa.Vel = FcVelFissa;

    } else {

        FcOutPompa.Vel = FcDutyCicle;

    }

}

```

Figura 5.13: Funzione nbGestionePompa stato di pompa in funzione

5.3 Gestione macchina a stati

Funzione nbGestioneDispenser

Si passa ora alla funzione di gestione del dispenser ovvero la funzione principale del codice che gestisce la macchina.

Per favorire la comprensione del codice si devono fare alcune precisazioni; siccome il codice che si sta trattando in questa relazione è scritto da più persone è fondamentale che tra le parti vi sia la possibilità di interagire in modo semplice e schematico, così da avere un dialogo tra i blocchi di codice localizzato e ordinato, al fine di validare il sistema il più velocemente possibile e in caso qualcosa non vada risulti più facile isolare il problema.

Fatta questa premessa si passa alla descrizione della funzione, la quale prevede di usare tre macro variabili di tipo strutturato. Perché dati strutturati? La decisione di usare dati strutturati è dovuta alla possibilità che offrono di racchiudere tutti i dati in una unica variabile, per compartimentare meglio il codice ed avere una trattazione più ordinata. Nella funzione nbGestioneDispenser si fa riferimento a tre tipologie di variabili:

- Ingressi: inclusi nella variabile InputDispenser, sono tutte quelle variabili che la funzione nbGestioneDispenser legge e non scrive. La variabile contiene tutti i feedback esterni utili al funzionamento del dispenser stesso
- Parametri: inclusi nella variabile ParametriDispenser, sono tutte quelle variabili proprie del dispenser che solo in questa funzione possono essere modificate
- Uscite: incluse nella variabile OutputDispenser, sono variabili che solo la funzione nbGestioneDispenser scrive e tipicamente non legge. Queste variabili contengono informazioni su come il dispenser deve impostare i suoi attuatori

Fatta questa premessa allora all'interno della funzione, nella variabile “FcInDisp” troveremo tutte le variabili di ingresso, in “FcOutDisp” tutte le variabili di uscita e in “FcParDisp” tutte le variabili interne del dispenser.

Gli stati possibili del dispenser sono:

1. DispenserPRIMOAVVIAMENTO
2. DispenserSPENTO
3. DispenserINIT
4. DispenserSTANDBY
5. DispenserACCESO
6. DispenserPRONTO
7. DispenserPRENOTATO
8. DispenserEROGAZIONE
9. DispenserLAVAGGIO
10. DispenserMANUTENZIONE
11. DispenserALLARME

La funzione si divide in 2 macro parti: una composta dalla macchina a stati vera e propria (struttura switch) che gestisce il dispenser e una parte più piccola all'inizio dove vengono verificate le condizioni per accedere a stati particolari. Si parte proprio con il descrivere quest'ultima parte, che si riporta qua sotto:

```

if (FcParDisp->StatoDispenser != DispenserPRIMOAVVIAMENTO) {
    //Faccio un reset degli alarmi in qualsiasi momento tramite app o pulsante blu sulla nucleo
    if ((FcInDisp.InButtonResetAllarmi == true)
        || FcInDisp.InAppResetAllarmi == true) {
        FcParDisp->ParResetAllarmi = true;
        nbResetApp();
    } else {
        FcParDisp->ParResetAllarmi = false;
    }
    //Faccio il reset degli alarmi
    if (FcParDisp->ParResetAllarmi == true) {
        firstEntry = false; // Reset firstEntry for next cycle
        FcParDisp->StatoDispenser = DispenserINIT;
    }
    //DispenserSPENTO => Condizioni per entrare nello stato SPENTO
    //Da qualsiasi stato
    if (FcInDisp.InOnOff == false && FcParDisp->DispBusy == false) {
        firstEntry = false; // Reset firstEntry for next cycle
        FcParDisp->StatoDispenser = DispenserSPENTO;
    }
    //DispenserMANUTENZIONE => Condizioni per entrare nello stato di MANUTENZIONE
    //Da qualsiasi stato a parte il lavaggio
    if (FcInDisp.InAppManutenzione
        && (FcParDisp->StatoDispenser != DispenserLAVAGGIO)
        && (FcParDisp->StatoDispenser != DispenserMANUTENZIONE)) {
        firstEntry = false; // Reset firstEntry for next cycle
        FcParDisp->StatoDispenser = DispenserMANUTENZIONE;
    }
    //DispenserALLARME => Condizioni per entrare nello stato di allarme da qualsiasi stato a parte quello di manutenzione
    //CheckAllarmi
    if (FcParDisp->StatoDispenser != DispenserMANUTENZIONE) {
        if (FcInDisp.LivelloSerbatorio <= 5) {
            firstEntry = false; // Reset firstEntry for next cycle
            FcParDisp->StatoDispenser = DispenserALLARME;
        }
    }
}

```

Figura 5.14: Funzione nbGestionDispenser prima parte della funzione

Procedendo con ordine vediamo che tutte queste righe di codice sono “governate” da un unico “If” che rende possibile l’esecuzione di queste righe di codice solo se non sì è nello stato di primo avviamento, questo perché il primo avviamento è uno stato particolare in cui molte funzionalità del dispenser non sono disponibili.

Fatta questa precisazione si va a descrivere cosa compone questa parte: in questa prima fase si gestisce il reset degli allarmi, dove per allarmi si intendono tutti i vari malfunzionamenti e problemi che in una qualsiasi macchina si possono verificare.

Visto il basso numero di sensori, l’unico allarme che si può verificare è l’assenza di acqua nel serbatoio. Una volta ripristinato un valore di acqua corretto nella tanica si esegue il reset allarmi tramite l’apposito comando sull’applicazione.

La parte successiva di codice riporta sostanzialmente 3 “if” che considerano tutte le condizioni che possono portare nei seguenti 3 stati:

- DispenserSPENTO
- DispenserALLARME
- DispenserMANUTENZIONE

Per semplicità siccome in questi stati si può entrare in qualsiasi momento, le condizioni per il loro ingresso sono riportate ad inizio codice in modo da evitare inutili ripetizioni. Quello che si vuole fare con questa prima parte di codice è riportato nei seguenti diagrammi di flusso, al fine di rimuovere varie ambiguità di spiegazione. Si passa ora ad una

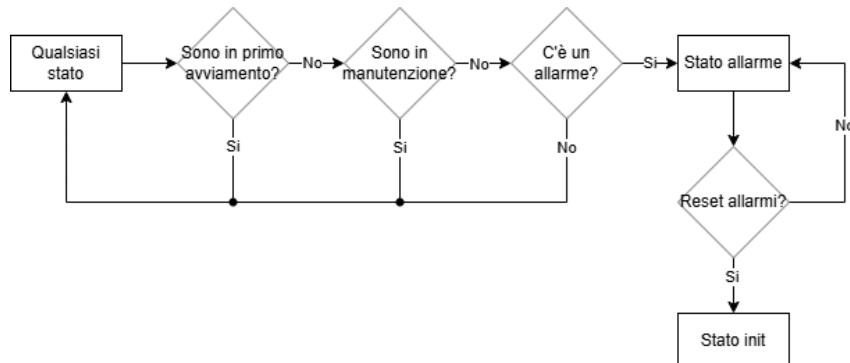


Figura 5.15: Logica semplificata ingresso nello stato di DipenserMANUTENZIONE

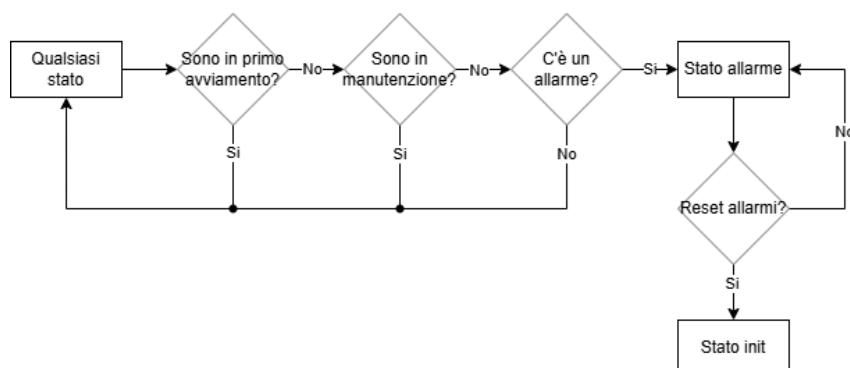


Figura 5.16: Logica semplificata ingresso nello stato di DipenserALLARME

seconda parte del codice, ovvero quella contenuta nella struttura a switch, che gestisce la macchina a stati del Dispenser. Per ogni stato verrà data una breve descrizione, utile a

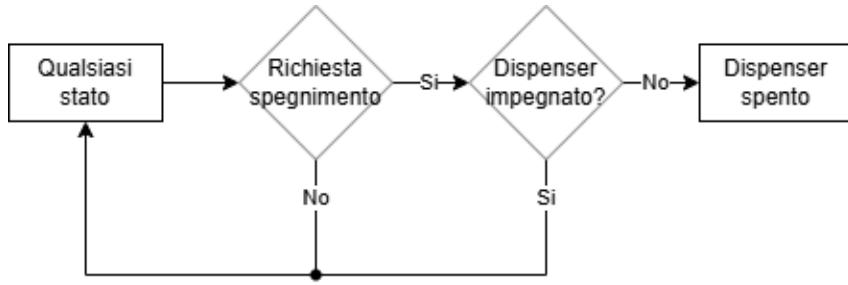


Figura 5.17: Logica semplificata ingresso nello stato di DipenserSPENTO

comprenderne lo scopo. Gli stati sono presentati nell'ordine in cui un generico operatore se li trova davanti durante l'uso del dispenser.

DipsenserPRIMOAVVIAMENTO

Lo stato con cui si ha a che fare accendendo per la prima volta il dispenser è quello denominato DispenserPRIMOAVVIAMENTO, che rappresenta la condizione di prima accensione della macchina. Alla prima accensione del dispenser, bisogna effettuare gli spurghi dell'impianto in modo tale da vuotare le condutture dall'aria e riempire i serbatoi, questa seconda operazione è fondamentale perché permette di evitare la rottura della resistenza di riscaldamento e della cella Peltier, che consente il raffreddamento. La parte di codice che gestisce questa condizione è qua sotto riportata

```

case DispenserPRIMOAVVIAMENTO:

    //Scrivo lo stato in cui sono nella variabile
    FcParDisp->StatoDispenser = DispenserPRIMOAVVIAMENTO;

    //Comando in manutenzione direttamente i tool
    //Elettrovalvole
    FcOutDisp->OutValvolaOutCalda = FcInDisp.InButtonCalda;
    FcOutDisp->OutValvolaCaldo = FcInDisp.InButtonCalda;

    FcOutDisp->OutValvolaFreddo = FcInDisp.InButtonFredda;

    FcOutDisp->OutValvolaTAmbo = FcInDisp.InButtonTAmbo;

    if ((FcInDisp.InButtonFredda || FcInDisp.InButtonTAmbo)) {
        FcOutDisp->OutValvolaOutFreddo = true;
    } else {
        FcOutDisp->OutValvolaOutFreddo = false;
    }

    FcOutDisp->OutCircuitoCaldo = false;
    FcOutDisp->OutCircuitoFreddo = false;

    //Funzione controllo pompa
    FcOutDisp->ParametriPompa = nbGestionePompa(true,
        (FcInDisp.InButtonCalda || FcInDisp.InButtonFredda
        || FcInDisp.InButtonTAmbo), false, 0, 98,
        &FcParDisp->StatoPompa);

    //Dove vado?
    //DispenserSPENTO => Condizioni per entrare nello stato di manutenzione
    if (FcInDisp.InButtonFredda && FcInDisp.InButtonCalda
        && FcInDisp.InButtonTAmbo) {
        firstEntry = false; // Reset firstEntry for next cycle
        FcParDisp->StatoDispenser = DispenserSPENTO;
    }

    break;

```

Figura 5.18: Codice implementato nello stato di DispenserPRIMOAVVIAMENTO

Come si può notare dal codice, l'operatore può solo pilotare il dispenser tramite i 3 pulsanti, i quali rispettivamente azionano la propria parte del circuito idraulico (valvole e pompa). Per effettuare correttamente lo spurgo, bisogna per prima cosa effettuare lo spurgo del circuito del caldo, successivamente quello del circuito del freddo e infine quello del circuito a temperatura ambiente. Una volta che si è sicuri che non ci sia più aria nell'impianto, si può uscire da questo stato premendo contemporaneamente i 3 pulsanti.

DipsenserSPENTO

Lo stato successivo è quello di dispenser spento, concepito per tutte quelle situazioni in cui non si vuole usare il dispenser per lunghi intervalli di tempo. Questo stato, così come molti altri che seguiranno, ha la seguente struttura:

- Si mettono a “FALSE” tutte le uscite
- Si sviluppa il codice mettendo a “TRUE” solo quelle effettivamente usate nello stato

In questo caso il dispenser è spento perciò nessuna uscita è messa “TRUE”. Qua sotto si riporta il codice dello stato. Da questo stato si può uscire solo se il bit della variabile

```
case DispenserSPENTO:
    //Scrivo lo stato in cui sono nella variabile
    FcParDisp->StatoDispenser = DispenserSPENTO;

    //Scrittura uscite
    //Valvole
    FcOutDisp->OutValvolaCaldo = false;
    FcOutDisp->OutValvolaFreddo = false;
    FcOutDisp->OutValvolaTamb = false;
    FcOutDisp->OutValvolaOutCalda = false;
    FcOutDisp->OutValvolaOutFreddo = false;
    FcOutDisp->OutValvolaSfiatoCaldo = false;

    //Impianto caldo e freddo
    FcOutDisp->OutCircuitoCaldo = false;
    FcOutDisp->OutCircuitoFreddo = false;

    //Abilitazione della PWM della pompa
    FcOutDisp->ParametriPompa.Enable = false;
    //DutyCycle nella PWM
    FcOutDisp->ParametriPompa.Vel = 0;

    //Altro
    FcParDisp->DispBusy = false;
    FcParDisp->LavaggioUltimato = false;

    //Dove vado dopo?
    //Condizione per entrare nello stato INIT
    if (FcInDisp.InOnOff && (FcInDisp.InButtonCalda == false) && (FcInDisp.InButtonFredda == false) && (FcInDisp.InButtonTamb == false)){
        firstEntry = false; // Reset firstEntry for next cycle
        FcParDisp->StatoDispenser = DispenserINIT;
    }

    break;
```

Figura 5.19: Codice implementato nello stato di DipenserSPENTO

“InputDispenser.InOnOff” (che nella funzione si chiama “FcInDisp.InOnOff”) viene messo a “TRUE”, se è così lo stato in cui si entra è quello di “DispenserINIT”.

Prima di procedere con la trattazione degli altri stati, si fa notare come ci siano alcuni stati che si assomigliano molto, questo perché la base di partenza era un codice già validato per una macchina industriale più complessa; quindi, il codice è stato riadattato nel modo più efficace possibile cercando un trade-off tra costi (che qui si intendono come ore di lavoro) ed efficacia.

DispenserINIT

Una volta usciti dallo stato di “DispenserSPENTO” o a seguito di un reset degli allarmi si può entrare nello stato di “DispenserINIT”. In questo stato, che assomiglia molto a quello di “DispenserSPENTO”, essenzialmente si esegue una inizializzazione delle periferiche. Questo è solo uno stato di passaggio tra quello di “DispenserSPENTO” e “DispenserSTANDBY”, infatti l’uscita dallo stato è automatica ed avviene esattamente dopo un ciclo di codice. Qua sotto si riporta per completezza il codice dello stato.

```
case DispenserINIT:  
  
    //Scrivo lo stato in cui sono nella variabile  
    FcParDisp->StatoDispenser = DispenserINIT;  
  
    //Scrittura uscite  
    //Valvole  
    FcOutDisp->OutValvolaCaldo = false;  
    FcOutDisp->OutValvolaFreddo = false;  
    FcOutDisp->OutValvolaTAmb = false;  
    FcOutDisp->OutValvolaOutCalda = false;  
    FcOutDisp->OutValvolaOutFreddo = false;  
    FcOutDisp->OutValvolaSfiatoCaldo = false;  
  
    //Impianto caldo e freddo  
    FcOutDisp->OutCircuitoCaldo = false;  
    FcOutDisp->OutCircuitoFreddo = false;  
  
    //Abilitazione PWM pompa  
    FcOutDisp->ParametriPompa.Enable = false;  
    //DutyCycle nella PWM  
    FcOutDisp->ParametriPompa.Vel = 0;  
  
    //Dove vado dopo?  
    //DispenserACCESO => Condizioni per entrare nello stato ACCESO  
    if (FcInDisp.InOnOff) {  
        firstEntry = false; // Reset firstEntry for next cycle  
        FcParDisp->StatoDispenser = DispenserSTANDBY;  
    }  
  
    break;
```

Figura 5.20: Codice implementato nello stato di DipenserINIT

DispenserSTANDBY

Si descrive ora la costruzione a codice di quello che è lo stato di “DispenserSTANDBY”, stato in cui il dispenser si trova tutte le volte che qualcuno non lo sta usando da diverso tempo. In questo stato tutti gli attuatori sono sostanzialmente spenti tranne il circuito del freddo che continua a funzionare indisturbato, quello del caldo che funziona con un controllo ad isteresi tra 2 temperature impostate di default e la pompa che viene impostata con un duty cycle al 25%, al fine di evitare lo svuotamento per evaporazione del serbatoio dell’acqua calda.

Vediamo nel dettaglio l’implementazione del controllo ad isteresi.

```

//Check temperatura del caldo
if (FcInDisp.InTempCalda <= (FcParDisp->SetTempCaldaStandBy - 20)) {
    FcOutDisp->OutCircuitoCaldo = true;
    FcOutDisp->OutValvolaSfiatoCaldo = false;
    FcOutDisp->OutValvolaCaldo = true;
    FcOutDisp->ParametriPompa.Vel = 30;
}

if (FcInDisp.InTempCalda >= (FcParDisp->SetTempCaldaStandBy)) {
    FcOutDisp->OutCircuitoCaldo = false;
    FcOutDisp->OutValvolaSfiatoCaldo = true;

}

```

Figura 5.21: Controllo ad isteresi delle temperature nei serbatoi

Come si può notare, la temperatura scelta è solo una (quella più alta), mentre la soglia minima è dettata dalla temperatura impostata, alla quale si sottraggono 20 gradi. Per uscire da questo stato è necessario che un operatore attivi il sensore di presenza o prenoti una quantità d'acqua tramite applicazione.

Qua sotto si riporta il codice dell'intero stato

```

case DispenserSTANDBY:
    //Scrivo lo stato in cui sono nella variabile
    FcParDisp->StatoDispenser = DispenserSTANDBY;

    //Scrittura uscite
    //Valvole
    FcOutDisp->OutValvolaCaldo = true;
    FcOutDisp->OutValvolaFreddo = false;
    FcOutDisp->OutValvolaTAmb = false;
    FcOutDisp->OutValvolaOutCalda = false;
    FcOutDisp->OutValvolaOutFreddo = false;
    FcOutDisp->OutValvolaSfiatoCaldo = true;

    //Impianto caldo e freddo
    FcOutDisp->OutCircuitoCaldo = false;
    FcOutDisp->OutCircuitoFreddo = true;

    //Abilitazione PWM pompa
    FcOutDisp->ParametriPompa.Enable = true;
    //DutyCicle nella PWM
    FcOutDisp->ParametriPompa.Vel = 25;

    //Check temperatura del caldo
    if (FcInDisp.InTempCalda <= (FcParDisp->SetTempCaldaStandBy - 20)) {
        FcOutDisp->OutCircuitoCaldo = true;
        FcOutDisp->OutValvolaSfiatoCaldo = false;
    }

    if (FcInDisp.InTempCalda >= (FcParDisp->SetTempCaldaStandBy)) {
        FcOutDisp->OutCircuitoCaldo = false;
        FcOutDisp->OutValvolaSfiatoCaldo = true;
    }

    //Dove vado?
    //DispenserACCESO => Condizioni per entrare nello stato ACCESO
    if ((FcInDisp.InStandBy == false) || FcInDisp.InAppTamb || FcInDisp.InAppCalda || FcInDisp.InAppFredda) {
        firstEntry = false; // Reset firstEntry for next cycle
        FcParDisp->StatoDispenser = DispenserACCESO;
    }

    break;

```

Figura 5.22: Codice implementato nello stato di DipenserSTANDBY

Per ricapitolare quanto descritto finora si riporta il seguente diagramma di flusso, che spiega in modo non ambiguo come si interagisce tra i vari stati.

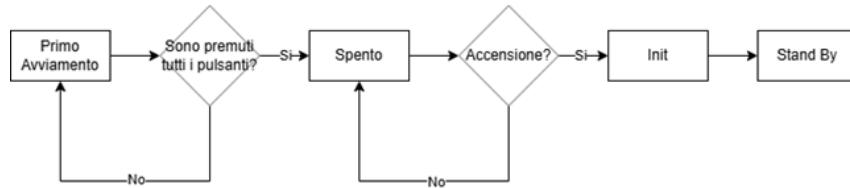


Figura 5.23: Diagramma di flusso per passare dallo stato di primo avviamento allo stato di stand-by

DispenserACCESO

Quando arriva una prenotazione o un operatore passa davanti al dispenser si entra nello stato di DispenserACCESO. In questo stato ci si comporta come nello stato di DispenserSTANDBY con l'unica accortezza che i circuiti del caldo e del freddo sono sempre attivi.

```

case DispenserACCESO:
    //Scrivo lo stato in cui sono nella variabile
    FcParDisp->StatoDispenser = DispenserACCESO;
    //Resetto variabili di sistema
    //FcParDisp->Rit = false;
    FcParDisp->ErogazioneUltimata = false;
    //Scrittura uscite
    //Valvole
    FcOutDisp->OutValvolaCaldo = true;
    FcOutDisp->OutValvolaFreddo = false;
    FcOutDisp->OutValvolaTamb = false;
    FcOutDisp->OutValvolaOutCalda = false;
    FcOutDisp->OutValvolaOutFreddo = false;
    FcOutDisp->OutValvolaSfiatoCaldo = false;
    //Impianto caldo e freddo
    FcOutDisp->OutCircuitoCaldo = true;
    FcOutDisp->OutCircuitoFreddo = true;
    //Abilitazione PWM pompa
    FcOutDisp->ParametriPompa.Enable = true;
    //DutyCycle nella PWM
    FcOutDisp->ParametriPompa.Vel = 25;
    //Check temperature
    if (FcInDisp.InTempCalda >= FcParDisp->SetTempCalda){
        FcOutDisp->OutLedButtonCalda = true; }
    if (FcInDisp.InTempFredda <= FcParDisp->SetTempFredda){
        FcOutDisp->OutLedButtonFredda = true; }
    //Dove vado?
    //DispenserSTANDBY => Condizioni per entrare nello stato PRONTO
    if (FcInDisp.InStandBy) {
        firstEntry = false; // Reset firstEntry for next cycle
        FcParDisp->StatoDispenser = DispenserSTANDBY; }
    //DispenserPRONTO => Condizioni per entrare nello stato PRONTO
    if((FcInDisp.InTempCalda >= FcParDisp->SetTempCalda) && (FcInDisp.InTempFredda <= FcParDisp->SetTempFredda)){
        firstEntry = false; // Reset firstEntry for next cycle
        FcParDisp->StatoDispenser = DispenserPRONTO; }
    //DispenserPRENOTATO => Condizioni per entrare nello stato di PRENOTATO
    if (FcInDisp.InAppTamb || FcInDisp.InAppCalda || FcInDisp.InAppFredda) {
        firstEntry = false; // Reset firstEntry for next cycle
        FcParDisp->StatoDispenser = DispenserPRENOTATO; }
    //DispenserEROGAZIONE => Condizioni per entrare nello stato di EROGAZIONE
    if ((FcInDisp.InPlCupHot && FcInDisp.InButtonCalda) || (FcInDisp.InPlCupCold && (FcInDisp.InButtonFredda || FcInDisp.InButtonTamb))) {
        firstEntry = false; // Reset firstEntry for next cycle
        FcParDisp->StatoDispenser = DispenserEROGAZIONE; }
    break;
  
```

Figura 5.24: Codice implementato nello stato di DipenserACCESO

Dallo stato di DispenserACCESO si esce se:

- Non c'è nessun operatore nelle vicinanze e in quel caso si va nello stato di DispenserSTANDBY
- La temperatura nei serbatoi supera quelle definite come soglia e in quel caso si va nello stato di DispenserPRONTO
- Un operatore chiede l'erogazione dell'acqua e allora in quel caso si va nello stato di DispenserEROGAZIONE

- Un operatore si prenota tramite l'applicazione e in quel caso si va nello stato di DispenserPRENOTATO

DispenserPRONTO

In questo stato sostanzialmente abbiamo la stessa situazione dello stato di acceso, con l'unica differenza che le temperature nei serbatoi sono nel range ottimale.

Dallo stato di DispenserPRONTO si esce se:

- Un operatore chiede l'erogazione dell'acqua e allora in quel caso si va nello stato di DispenserEROGAZIONE
- Un operatore si prenota tramite applicazione e in quel caso si va nello stato di DispenserPRENOTATO
- Non c'è nessun operatore nelle vicinanze e in quel caso si va nello stato di DispenserSTANDBY

Per completezza si riporta qua sotto il codice presente nello stato di DispenserPRONTO.

```

case DispenserPRONTO:
    //Scrivo lo stato in cui sono nella variabile
    FcParDisp->StatoDispenser = DispenserPRONTO;
    //Valvole
    FcOutDisp->OutValvolaCaldo = true;
    FcOutDisp->OutValvolaFreddo = false;
    FcOutDisp->OutValvolaTAmb = false;
    FcOutDisp->OutValvolaOutCalda = false;
    FcOutDisp->OutValvolaOutFreddo = false;
    FcOutDisp->OutValvolaSfiatoCaldo = false;
    //Impianto caldo e freddo
    FcOutDisp->OutCircuitoCaldo = true;
    FcOutDisp->OutCircuitoFreddo = true;
    //Abilitazione PWM pompa
    FcOutDisp->ParametriPompa.Enable = true;
    //DutyCicle nella PWM
    FcOutDisp->ParametriPompa.Vel = 25;
    //Dove vado?
    //DipsenserEROGAZIONE => Condizioni per entrare nello stato di EROGAZIONE
    if ((FcInDisp.InButtonTAmb || FcInDisp.InButtonFredda)
        && FcInDisp.InPlCupCold) {
        firstEntry = false; // Reset firstEntry for next cycle
        FcParDisp->StatoDispenser = DispenserEROGAZIONE;
    }
    if (FcInDisp.InButtonCalda && FcInDisp.InPlCupHot) {
        firstEntry = false; // Reset firstEntry for next cycle
        FcParDisp->StatoDispenser = DispenserEROGAZIONE;
    }
    //DipsenserPRENOTATO => Condizioni per entrare nello stato di PRENOTATO
    if (FcInDisp.InAppTAmb || FcInDisp.InAppCalda || FcInDisp.InAppFredda) {
        firstEntry = false; // Reset firstEntry for next cycle
        FcParDisp->StatoDispenser = DispenserPRENOTATO;
    }
    //DispenserSPENTO => Condizioni per entrare nello stato STANDBY
    if (FcInDisp.InStandBy == true) {
        firstEntry = false; // Reset firstEntry for next cycle
        FcParDisp->StatoDispenser = DispenserSTANDBY;
    }
break;

```

Figura 5.25: Codice implementato nello stato di DipenserPRONTO

DispenserPRENOTATO

Si arriva nello stato di DispenserPRENOTATO solo dopo che un operatore si è prenotato tramite applicazione. In questa condizione il dispenser mantiene tutte le periferiche come nello stato di acceso, però si ha solo un pulsante abilitato, quello che consente di erogare la tipologia di acqua scelta tramite l'applicazione.

Si riporta qua sotto il codice dello stato appena descritto.

```
case DispenserPRENOTATO:
    //Scrivo lo stato in cui sono nella variabile
    FcParDisp->StatoDispenser = DispenserPRENOTATO;
    //Valvole
    FcOutDisp->OutValvolaCaldo = true;
    FcOutDisp->OutValvolaFreddo = false;
    FcOutDisp->OutValvolaTAmb = false;
    FcOutDisp->OutValvolaOutCalda = false;
    FcOutDisp->OutValvolaOutFreddo = false;
    FcOutDisp->OutValvolaSfiatoCaldo = false;
    //Impianto caldo e freddo
    FcOutDisp->OutCircuitoCaldo = true;
    FcOutDisp->OutCircuitoFreddo = true;
    //Abilitazione PWM pompa
    FcOutDisp->ParametriPompa.Enable = true;
    //DutyCycle nella PWM
    FcOutDisp->ParametriPompa.Vel = 25;
    //Gestisco il lampeggio del LED corrispettivo alla prenotazione dall'app
    //Verifico da dove è stata prenotata
    if (FcInDisp.InAppCalda) {
        //Resetto i parametri dell'erogazione
        nbResetPID(&FcParDisp->PIDPompa);
    }
    if (FcInDisp.InAppFredda) {
        //Resetto i parametri dell'erogazione
        nbResetPID(&FcParDisp->PIDPompa);
    }
    if (FcInDisp.InAppTAmb) {
        //Resetto i parametri dell'erogazione
        nbResetPID(&FcParDisp->PIDPompa);
    }
    //Dove vado?
    //DispenserEROGAZIONE => Condizioni per entrare nello stato di EROGAZIONE
    if (FcInDisp.InPlCupCold) {
        if ((FcInDisp.InAppTAmb && FcInDisp.InButtonTAmb) || (FcInDisp.InAppFredda && FcInDisp.InButtonFredda)) {
            firstEntry = false; // Reset firstEntry for next cycle
            FcParDisp->StatoDispenser = DispenserEROGAZIONE;
        }
    }
    if (FcInDisp.InPlCupHot && FcInDisp.InAppCalda && FcInDisp.InButtonCalda) {
        firstEntry = false; // Reset firstEntry for next cycle
        FcParDisp->StatoDispenser = DispenserEROGAZIONE;
    }
    break;
```

Figura 5.26: Codice implementato nello stato di DipenserPRENOTATO

Per uscire dallo stato di “DispenserPRENOTATO” e quindi passare nello stato di “DispenserEROGAZIONE” basta posizionare correttamente la bottiglia e premere il pulsante corrispondente alla tipologia di acqua prenotata.

DispenserEROGAZIONE

In questo stato avviene l'erogazione di acqua da parte del dispenser. L'erogazione può essere di due tipi: erogazione di acqua senza limiti (funzionamento dei dispenser tradizionali) oppure si può erogare una quantità fissata di acqua se si effettua la prenotazione tramite applicazione.

Le 2 tipologie di erogazione differiscono per il fatto che la seconda coinvolge il controllore PID e il flussimetro al fine di avere la quantità di acqua desiderata. Anche per questo stato si riporta il codice nell'immagine sottostante.

```

case DispenserEROGAZIONE:
    //Scrivo lo stato in cui sono nella variabile
    FcParDisp->StatoDispenser = DispenserEROGAZIONE;
    FcParDisp->DispBusy = true;
    FcParDisp->PercorsoOk = false;
    float DutyCicleFloat = 0.0;
    uint32_t DutyCicleInt = 0;
    float SetPoint = 0;
    //Verifico di che tipo di erogazione si tratta
    if (FcInDisp.InAppCalda || FcInDisp.InAppFredda || FcInDisp.InAppTAmbo) {
        //Setto la quantità di acqua che mi serve in ml
        SetPoint = FcInDisp.InQuantitaCalib; }
    FcOutDisp->OutValvolaSfiatoCaldo = false;
    FcOutDisp->OutValvolaOutCalda = false;
    FcOutDisp->OutValvolaCaldo = false;
    FcOutDisp->OutValvolaOutFreddo = false;
    FcOutDisp->OutValvolaFreddo = false;
    FcOutDisp->OutValvolaTAmbo = false;
    //Inizio l'apertura delle Valvole
    //Verifico di che tipo di erogazione si tratta
    if(FcInDisp.InButtonCalda && FcInDisp.InPlCupHot && (!FcInDisp.InAppFredda) && (!FcInDisp.InAppTAmbo)){
        FcOutDisp->OutValvolaSfiatoCaldo = true;
        FcOutDisp->OutValvolaOutCalda = true;
        FcOutDisp->OutValvolaCaldo = true;
        FcOutDisp->OutLedButtonCalda = true;
        FcParDisp->PercorsoOk = true; }
    if (FcInDisp.InButtonFredda && FcInDisp.InPlCupCold && (!FcInDisp.InAppCalda) && (!FcInDisp.InAppTAmbo)){
        FcOutDisp->OutValvolaOutFreddo = true;
        FcOutDisp->OutValvolaFreddo = true;
        FcOutDisp->OutLedButtonFredda = true;
        FcParDisp->PercorsoOk = true; }
    if (FcInDisp.InButtonTAmbo && FcInDisp.InPlCupCold && (!FcInDisp.InAppFredda) && (!FcInDisp.InAppCalda)) {
        FcOutDisp->OutValvolaOutFreddo = true;
        FcOutDisp->OutValvolaTAmbo = true;
        FcOutDisp->OutLedButtonTAmbo = true;
        FcParDisp->PercorsoOk = true; }

```

Figura 5.27: Codice implementato nello stato di DipenserEROGAZIONE parte 1

```

//Incremento di 1
FcParDisp->DutyMan++;
if (FcParDisp->DutyMan >= 98){
    FcParDisp->DutyMan = 98;
}
nbGestioneFlussimetro();
Calib500 = FcParDisp->Misura;
//Gestisco il PID
DutyCycleFloat = nbControllerPID(&FcParDisp->PIDPompa, true, FcParDisp->Misura, SetPoint, &FcParDisp->ContatorePID);
//Converto il duty cicle da float a int
DutyCycleInt = (uint16_t) DutyCycleFloat;
//Funzione controllo pompa
FcOutDisp->ParametriPompa = nbGestionePompa(true, FcParDisp->PercorsoOk,(FcInDisp.InAppCalda || FcInDisp.InAppFredda|| FcInDisp.InAppTamb),
    DutyCycleInt,FcParDisp->DutyMan,&FcParDisp->StatoPompa);
//Dove vado?
//Torno nello stato di prenotato
if (((FcInDisp.InPlCupCold && FcInDisp.InButtonFredda) == false) && FcInDisp.InAppFredda && (FcParDisp->Misura < (FcInDisp.InQuantitaCalib-10)))
    firstEntry = false; // Reset firstEntry for next cycle
    FcParDisp->StatoDispenser = DispenserPRENOTATO;
} else if (((FcInDisp.InPlCupCold && FcInDisp.InButtonTamb) == false)&& FcInDisp.InAppTamb
    && (FcParDisp->Misura < (FcInDisp.InQuantitaCalib-10))) {
    firstEntry = false; // Reset firstEntry for next cycle
    FcParDisp->StatoDispenser = DispenserPRENOTATO;
} else if (((FcInDisp.InPlCupHot && FcInDisp.InButtonCalda) == false)
    && FcInDisp.InAppCalda
    && (FcParDisp->Misura < (FcInDisp.InQuantitaCalib-10))) {
    firstEntry = false; // Reset firstEntry for next cycle
    FcParDisp->StatoDispenser = DispenserPRENOTATO;
} else if (((FcInDisp.InPlCupHot && FcInDisp.InButtonFredda) == false)
    && ((FcInDisp.InPlCupCold && FcInDisp.InButtonFredda) == false)
    && ((FcInDisp.InPlCupCold && FcInDisp.InButtonTamb) == false)) {
    firstEntry = false; // Reset firstEntry for next cycle
    FcParDisp->StatoDispenser = DispenserEROGAZIONEULTIMATA;
}

break;

```

Figura 5.28: Codice implementato nello stato di DipenserEROGAZIONE parte 2

Per uscire da questo stato ci sono 3 condizioni possibili:

- L'operatore non si è prenotato con l'applicazione e rilascia il pulsante o toglie il bicchiere. In quel caso si va nello stato di DipsenserEROGAZIONEULTIMATA
- L'operatore si è prenotato con l'applicazione e rilascia il pulsante o toglie il bicchiere, ma non è ancora stata erogata tutta la quantità di acqua richiesta. In quel caso si torna nello stato di DispenserPRENOTATO
- L'operatore si è prenotato con l'applicazione e il dispenser ha erogato la quantità scelta. In quel caso si va nello stato di DipsenserEROGAZIONEULTIMATA

DispenserEROGAIZONEULTIMATA

Questo stato, di cui sotto si riporta il codice, serve per spegnere tutti gli attuatori del dispenser post erogazione e resettare tutti i dati dell'erogazione stessa, nel caso questa sia ottenuta tramite applicazione.

Da questo stato si torna in automatico nello stato di DispenserACCESO dopo un solo ciclo.

```

case DispenserEROGAZIONEULTIMATA:

    //Scrivo lo stato in cui sono nella variabile
    FcParDisp->StatoDispenser = DispenserEROGAZIONEULTIMATA;

    //Spengo tutto
    FcOutDisp->ParametriPompa.Enable = false;
    FcOutDisp->ParametriPompa.Vel = 0;

    FcOutDisp->OutValvolaSfiatoCaldo = true;
    FcOutDisp->OutValvolaOutCalda = false;
    FcOutDisp->OutValvolaCaldo = false;
    FcOutDisp->OutValvolaOutFreddo = false;
    FcOutDisp->OutValvolaFreddo = false;
    FcOutDisp->OutValvolaTAmbo = false;

    FcParDisp->DispBusy = false;

    FcParDisp->DutyMan = 0;

    if (FcParDisp->Misura >= FcInDisp.InQuantitaCalib - 10) {

        CounterFlussimetro = 0;
        FcParDisp->Misura = 0;
        nbResetApp();

    }

    //Dove vado?
    //Torno nello stato di dispenser acceso
    if ((FcInDisp.InButtonFredda == false)
        && (FcInDisp.InButtonTAmbo == false)
        && (FcInDisp.InButtonCalda == false)) {
        firstEntry = false; // Reset firstEntry for next cycle
        FcParDisp->StatoDispenser = DispenserACCESO;
    }

    break;

```

Figura 5.29: Codice implementato nello stato di DispenserEROGAZIONEULTIMATA

DispenserMANUTENZIONE

Per entrare nello stato di DispenserMANUTENZIONE si deve utilizzare necessariamente l'applicazione. L'accesso a questa parte è riservato solo al personale addetto e in questo stato è possibile comandare tutti gli attuatori in maniera indipendente, al fine di poter controllare il funzionamento di tutto l'impianto. Qui sotto si riporta il codice dello stato.

```

case DispenserMANUTENZIONE:

    //Scrivo lo stato in cui sono nella variabile
    FcParDisp->StatoDispenser = DispenserMANUTENZIONE;
    //Mantenimento stato
    FcParDisp->StatoDispenser = DispenserMANUTENZIONE;
    //Comando in manutenzione direttamente i tool
    //Circuito del caldo
    FcOutDisp->OutCircuitoCaldo = FcInDisp.InAppManuCircuitoCaldo;
    //Circuito del freddo
    FcOutDisp->OutCircuitoFreddo = FcInDisp.InAppManuCircuitoFreddo;
    //Elettrovalvole
    FcOutDisp->OutValvolaOutCalda = FcInDisp.InAppManuValvolaOutCaldo;
    FcOutDisp->OutValvolaOutFreddo = FcInDisp.InAppManuValvolaOutFreddo;
    FcOutDisp->OutValvolaCaldo = FcInDisp.InAppManuValvolaCaldo;
    FcOutDisp->OutValvolaTAmb = FcInDisp.InAppManuValvolaTAmb;
    FcOutDisp->OutValvolaFreddo = FcInDisp.InAppManuValvolaFreddo;
    FcOutDisp->OutValvolaSfiatoCaldo = FcInDisp.InAppManuValvolaSfiatoCaldo;
    //Funzione controllo pompa
    FcOutDisp->ParametriPompa = nbGestionePompa(true,
        FcInDisp.InAppManuPompaStartStop, false, 0,
        FcInDisp.InAppManuPompaVelFissa, &FcParDisp->StatoPompa);
    //Dove vado?
    //Dallo dispenserLAVAGGIO
    //DispenserLAVAGGIO => Condizioni per entrare nello stato di manutenzione
    if (FcParDisp->StatoDispenser == DispenserMANUTENZIONE && FcInDisp.InAppManuLavaggio) {
        firstEntry = false; // Reset firstEntry for next cycle
        HAL_TIM_Base_Start_IT(&htim6);
        FcParDisp->ContatoreLavaggio = 0;
        FcParDisp->StatoDispenser = DispenserLAVAGGIO;
    }
    //Dallo dispenserLAVAGGIO
    //DispenserINIT => Condizioni per entrare nello stato di INIT
    if (FcInDisp.InAppManutenzione == false) {
        firstEntry = false; // Reset firstEntry for next cycle
        FcParDisp->StatoDispenser = DispenserINIT;
    }
}

break:

```

Figura 5.30: Codice implementato nello stato di DipenserMANUTENZIONE

Dallo stato di DispenserMANUTENZIONE si esce solamente usando l'applicazione e le possibilità sono 2:

- Dallo stato di DispenserManutenzione si può entrare nello stato di DispenserINIT, qualora l'intervento di manutenzione sia ultimato
- Dallo stato di DispenserManutenzione si può entrare nello stato di DispenserLAVAGGIO, qualora sia necessaria la pulizia dell'impianto

DispenserLAVAGGIO

Solo il personale addetto può accedere allo stato di manutenzione, di conseguenza solo il personale addetto può eseguire il lavaggio dell'impianto, che consiste nel far circolare dell'acqua attraverso tutto il circuito idraulico del dispenser, per una definita certa quantità di tempo.

In un caso reale il composto che circola in questa fase, non è acqua ma un composto formato da acqua e detergenti specifici.

Il lavaggio dura per una quantità di tempo parametrizzabile, al termine del quale il dispenser esce dalla modalità di lavaggio e torna in modalità di manutenzione. Tramite l'applicazione è possibile interrompere il lavaggio prima che venga raggiunta la soglia di tempo impostata, premendo un apposito pulsante. Si riporta qua sotto il codice dello stato appena discusso.

```

case DispenserLAVAGGIO:

    //Scrivo lo stato in cui sono nella variabile
    FcParDisp->StatoDispenser = DispenserLAVAGGIO;
    //Comando in manutenzione direttamente i tool
    //Circuito del caldo
    FcOutDisp->OutCircuitoCaldo = false;
    //Circuito del freddo
    FcOutDisp->OutCircuitoFreddo = false;
    //Elettrovalvole
    FcOutDisp->OutValvolaOutCalda = true;
    FcOutDisp->OutValvolaOutFreddo = true;
    FcOutDisp->OutValvolaCaldo = true;
    FcOutDisp->OutValvolaTamb = true;
    FcOutDisp->OutValvolaFreddo = true;
    FcOutDisp->OutValvolaSfiatoCaldo = false;
    //Funzione controllo pompa
    FcOutDisp->ParametriPompa = nbGestionePompa(true, true, false, 0, 98, &FcParDisp->StatoPompa);
    FcParDisp->TargetLavaggio = (FcParDisp->DurataLavaggioOre * 3600) + (FcParDisp->DurataLavaggioMinuti * 60) + FcParDisp->DurataLavaggioSecondi;
    FcParDisp->LavaggioUltimato = false;

    if (FcParDisp->ContatoreLavaggio >= FcParDisp->TargetLavaggio) {
        FcParDisp->LavaggioUltimato = true;
        FcParDisp->ContatoreLavaggio = 0;
    }

    //Dove vado?
    if (FcInDisp.InAppManuLavaggio == false) {
        firstEntry = false; // Reset firstEntry for next cycle
        FcParDisp->StatoDispenser = DispenserMANUTENZIONE;
    }

    if (FcInDisp.InAppManuLavaggio == true
        && FcParDisp->LavaggioUltimato == true) {
        firstEntry = false; // Reset firstEntry for next cycle
        FcParDisp->StatoDispenser = DispenserMANUTENZIONE;
        InputDispenser.InAppManuLavaggio = false;
        HAL_TIM_Base_Stop_IT(&htim6);
    }

    break;

```

Figura 5.31: Codice implementato nello stato di DipenserLAVAGGIO

DispenserALLARME

Si chiude la trattazione degli stati con lo stato di DispenserALLARME, stato in cui ci si ritrova ogni volta il dispenser finisce la riserva di acqua situata nella tanica.

In questa condizione si hanno tutte le valvole chiuse e la pompa spenta, e ovviamente i circuiti di riscaldamento e raffreddamento sono disabilitati. Come si è esposto all'inizio, l'unico modo per uscire da questo stato è mediante il ripristino del livello di acqua e il successivo reset allarmi. Si riporta qua sotto il codice appena descritto.

```

case DispenserALLARME:

    //Scrivo lo stato in cui sono nella variabile
    FcParDisp->StatoDispenser = DispenserALLARME;
    //Scrittura uscite
    //Valvole
    FcOutDisp->OutValvolaCaldo = false;
    FcOutDisp->OutValvolaFreddo = false;
    FcOutDisp->OutValvolaTAmb = false;
    FcOutDisp->OutValvolaOutCalda = false;
    FcOutDisp->OutValvolaOutFreddo = false;
    FcOutDisp->OutValvolaSfiatoCaldo = false;
    //Impianto caldo e freddo
    FcOutDisp->OutCircuitoCaldo = false;
    FcOutDisp->OutCircuitoFreddo = false;
    //Abilitazione della PWM della pompa
    FcOutDisp->ParametriPompa.Enable = false;
    //DutyCicle nella PWM
    FcOutDisp->ParametriPompa.Vel = 0;
    //Faccio lampeggiare tutti i led velocemente ed alla stessa frequenza
    if ((HAL_GetTick() - FcParDisp->DelayAllarme) >= FcParDisp->ContatoreAllarme) {
        FcParDisp->ContatoreAllarme = HAL_GetTick();
        if (FcOutDisp->OutLedButtonCalda) {
            FcOutDisp->OutLedButtonCalda = false;
        } else {
            FcOutDisp->OutLedButtonCalda = true;
        }
    }
    //Copio lo stato sugli altri pulsanti
    FcOutDisp->OutLedButtonFredda = FcOutDisp->OutLedButtonCalda;
    FcOutDisp->OutLedButtonTAmb = FcOutDisp->OutLedButtonCalda;

    break;

```

Figura 5.32: Codice implementato nello stato di DispenserALLARME

Default

L'ultimo stato che viene esposto è quello di default, che è stato implementato per una mera esigenza di debug del software. In realtà questo stato non viene e non deve essere usato. Se, qualora il codice per qualsiasi motivo dovesse perdere, finirebbe per entrare nello stato di Default per poi tornare nello stato di DispenserPRIMOAVVIAMENTO.

```

default:
    //FcParDisp->StatoDispenser = DispenserPRIMOAVVIAMENTO;
    defaultState = true;
    break;

```

Figura 5.33: Codice implementato nello stato di default

Descritti tutti gli stati, si può riassumere con un diagramma di flusso il funzionamento del codice esposto a partire dallo stato di “DispenserACCESO”.

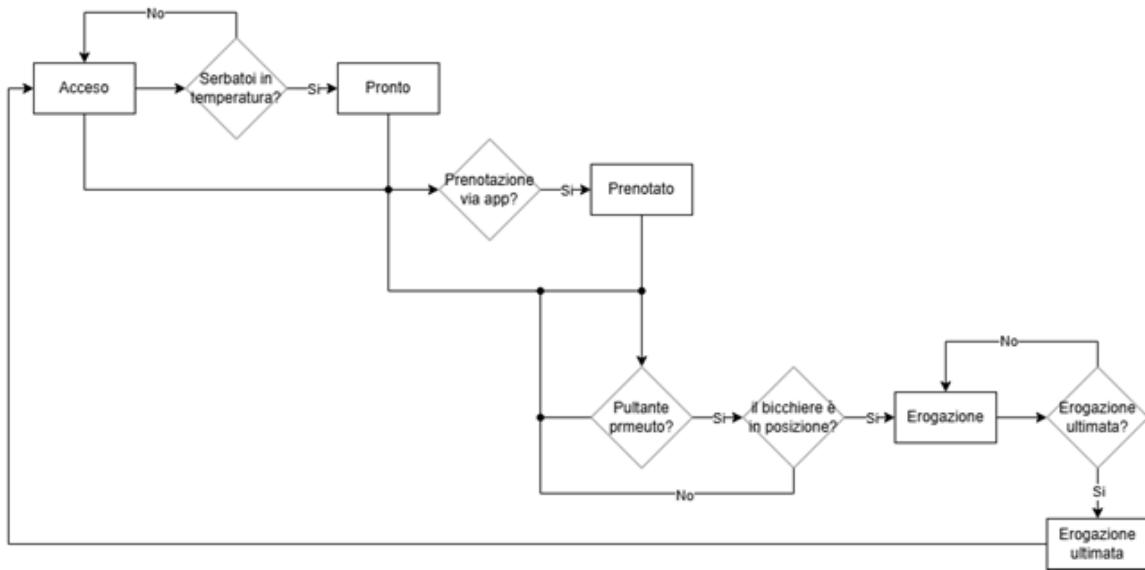


Figura 5.34: Diagramma di flusso di massima del dispenser

5.4 Implementazione software flussimetro

Il flussimetro come già accennato nella fase di implementazione del CUBE MX risulta essere un dispositivo che genera una serie di impulsi. Il numero di impulsi è proporzionale alla portata, di conseguenza il primo passo è contare gli impulsi. Per farlo inseriamo nella Callback denominata: “HAL_GPIO_EXTI_Callback” un contatore in modo da implementare la variabile “CounterFlussimetro” ad ogni fronte di salita. Si riporta qua sotto la parte di codice appena descritta. Come può notare dall’immagine, nella parte di codice

```

if(GPIO_Pin == GPIO_PIN_6){

    if(OutputDispenser.OutStatoDispenser == DispenserEROGAZIONE){

        if((HAL_GetTick() - BounceFlux) >= 13){
            BounceFlux = HAL_GetTick();
            CounterFlussimetro++;

        }
    }
}

```

Figura 5.35: Gestione del flussimetro nella CallBack

inserita nella Callback c’è anche un filtro software che va a scremare quelli che sono gli impulsi da quello che invece è rumore di natura elettrica che si ha sulla porta. L’altra accortezza è il controllo dello stato del dispenser, reso necessario siccome la pompa funziona in quasi tutti gli stati, ma solo in quello di erogazione si deve calcolare la portata. Ricapitolando alla luce del codice inserito nella funzione di CallBack tutte le volte che si è in erogazione e si ha un “vero” fronte di salita si incrementa la variabile “CounterFlussimetro”.

Il passo successivo è il calcolo della portata ma soprattutto il calcolo della quantità erogata. Questi calcoli sono fatti in una seconda funzione denominata “nbGestioneFlussimetro” e richiamata nella funzione “nbGestioneDispenser” più precisamente nello stato di “DispenserEROGAZIONE”. In pratica, dal datasheet del flussimetro YF-S401 si ha che:

$$f = 98 \cdot Q$$

Dove f è la frequenza degli impulsi e Q è la portata in litri al minuto, di conseguenza se si vuole ricavare la portata in millilitri al secondo (più congrua con questa applicazione) si deve ribaltare la formula e moltiplicare per le opportune costanti. La formula risultante è la seguente:

$$Q = \frac{f \cdot 1000}{98 \cdot 60}$$

A questo punto la portata è calcolata ogni 100 ms e, di conseguenza, moltiplicando la portata per 0.1 si ottiene il valore di acqua erogata ogni 100 ms.

Tutto questo ragionamento è stato riportato in “nbGestioneFlussimetro” e il codice che ne risulta è riportato qua sotto.

```
//FLUSSIMETRO (Gestito in Interrupt)
void nbGestioneFlussimetro() {

    float FcPortata;
    float Freq;

    if((HAL_GetTick() - TimerFlussimetro) > 100){

        TimerFlussimetro = HAL_GetTick();

        Freq = (float)CounterFlussimetro/0.1;

        FcPortata = (Freq * 1000)/(98.0*60.0);
        Portata1 = FcPortata;

        if(FcPortata > 33){

            FcPortata = 33;

        }

        Portata = FcPortata;
        ParametriDispenser.Misura = ParametriDispenser.Misura + ((FcPortata*0.1));

        CounterFlussimetro = 0;
    }
}
```

Figura 5.36: Codice della funzione nbGestioneFlussimetro

Alla fine di ogni erogazione il valore del contatore è resettato.

Con questa gestione del flussimetro rimane un piccolo errore di tolleranza sulle quantità erogate. Per gestire questo problema si effettua una calibrazione periodica e si ridefinisce il setpoint della quantità da erogare al fine di ottenere in uscita la quantità di acqua corretta.

5.5 Implementazione software del sensore ultrasuoni

Per il corretto funzionamento del sensore ad ultrasuoni è necessario avviare prima di entrare nel while (1) l'Input Capture in interrupt sul TIM1_CH1 (richiamando la funzione void am_handle_HCSR04_Init (void)), in maniera tale da inizializzare il timer affinché sia pronto a catturare la prima transizione logica sul pin ECHO.

```
void am_handle_HCSR04_Init(void) {
    HAL_TIM_IC_Start_IT(&htim1, TIM_CHANNEL_1);
}
```

Una volta terminata l'erogazione da parte del dispenser, quando il dispenser entra nello stato di Erogazione Ultimata, viene richiamata la funzione all'interno della quale avviene la generazione del segnale TRIG e quindi l'avvio del PWM in interrupt su TIM2_CH1.

```
void am_handle_HCSR04_SendUltrasonicPulse(void) {
    HAL_TIM_PWM_Start_IT(&htim2, TIM_CHANNEL_1);
}
```

Dopo che il timer ha emesso i 10 microsecondi ad alto livello, TRIG viene abbassato automaticamente, perché alla fine del ciclo PWM viene richiamata HAL_TIM_PWM_PulseFinishedCallback () dove il codice ferma il PWM.

```
void HAL_TIM_PWM_PulseFinishedCallback(TIM_HandleTypeDef *htim) {
    if (htim == &htim2) {
        // qui il timer ha già abbassato TRIG automaticamente
        HAL_TIM_PWM_Stop_IT(&htim2, TIM_CHANNEL_1);
    }
}
```

A questo punto è necessario misurare il segnale ECHO attraverso Input Capture. Il primo fronte di salita, TIM1 cattura il valore del counter in IC_Val1. Is_First_Captured, il flag che mi dice che sono nel fronte di salita, passa a 1 E in questo modo si cambia la polarità di cattura sul TIM1, da Rising a Falling.

Sul fronte di discesa del segnale, TIM1 cattura il counter in IC_Val2.

Dopodiché avviene il confronto tra IC_Val2 e IC_Val1 per calcolare Difference. I tick vengono convertiti in una distanza raw, in cm, con la formula che prevede di tenere in considerazione la velocità del suono nel vuoto. La misura raw viene infine aggiustata attraverso una calibrazione in maniera tale da ottenere Distance. Is_First_Captured torna a 0 e si riporta la polarità del timer su Rising in attesa della misura successiva.

```

void HAL_TIM_IC_CaptureCallback(TIM_HandleTypeDef *htim) {
    if (htim->Instance == TIM1) {
        if (htim->Channel == HAL_TIM_ACTIVE_CHANNEL_1) {

            if (Is_First_Captured == 0) {
                IC_Val1 = HAL_TIM_ReadCapturedValue(htim, TIM_CHANNEL_1);
                Is_First_Captured = 1;
                __HAL_TIM_SET_CAPTUREPOLARITY(htim, TIM_CHANNEL_1,
                                              TIM_INPUTCHANNELPOLARITY_FALLING);
            } else {
                IC_Val2 = HAL_TIM_ReadCapturedValue(htim, TIM_CHANNEL_1);

                if (IC_Val2 >= IC_Val1)
                    Difference = IC_Val2 - IC_Val1;
                else
                    Difference = (0xFFFF - IC_Val1) + IC_Val2;

                float raw = Difference * 0.034f / 2.0f;
                Distance = 0.9351f * raw + 1.12f;

                Is_First_Captured = 0;
                __HAL_TIM_SET_CAPTUREPOLARITY(htim, TIM_CHANNEL_1,
                                              TIM_INPUTCHANNELPOLARITY_RISING);
                __HAL_TIM_CLEAR_FLAG(htim, TIM_FLAG_CC1);
            }
        }
        pippo = am_calculateTankLevel(Distance);
    }
}

```

La variabile pippo è una variabile che si occupa di memorizzare il valore del livello di acqua presente all'interno del serbatoio. La variabile assume il valore di output della funzione am_calculateTankLevel(Distance) la quale riceve in ingresso il valore di distanza appena misurato con il sensore nello stato di Erogazione Ultimata del dispenser.

```

float am_calculateTankLevel(float Distance) {
    float livello_percentuale = ((D_MAX_CM - Distance) / (D_MAX_CM - D_MIN_CM)) * 100.0f;
    if (livello_percentuale < 0.0f) {
        livello_percentuale = 0.0f; // Limita il livello minimo al 0%
    } else if (livello_percentuale > 100.0f) {
        livello_percentuale = 100.0f; // Limita il livello massimo al 100%
    }
    return livello_percentuale;
}

```

La precedente funzione permette di ottenere in uscita il livello del serbatoio in percentuale, calcolandolo con una relazione che tiene delle costanti D_MAX_CM e l'offset D_MIN_CM.

La differenza tra queste ultime due costanti rappresenta nella pratica la distanza tra il pelo libero dell'acqua e il livello minimo di acqua al di sotto del quale il dispenser va nello stato di allarme.

5.6 Implementazione software display LCD

Di seguito verranno riportate le principali funzioni utilizzate per pilotare il display HD44780 attraverso l'expander I2C basato su PCF8574.

Indirizzo I2C

Nel file “liquidcrystal_i2c.h” deve essere correttamente definito DEVICE_ADDR, che nel caso del display utilizzato è 0x4E, come rappresentato in figura:

```
/* Device I2C Address */
#define DEVICE_ADDR      0x4E
```

Inizializzazione del display

Prima di poter utilizzare correttamente il display è necessario inizializzare correttamente il controller HD44780 in modalità 4-bit attraverso PCF8574.

La funzione utilizzata è void HD44780_Init (uint8_t rows), la quale richiede in ingresso il parametro rows che sarebbe il numero di righe del display, nel nostro caso 4.

```
HD44780_Init(4); // initialize the LCD with 4 rows
```

All'interno di tale funzione vengono inoltre impostati i vari parametri base (numero di linee, modalità a 4 bit, dimensione caratteri, direzione di scrittura, cursor-blinking, ecc.) prima dell'accensione.

Vediamo nel dettaglio cosa fa tale funzione:

1. Salva il numero di righe:

```
dpRows = rows;
```

2. Imposta il valore della variabile che tiene traccia dello stato del led di retroilluminazione (se acceso: LCD_BACKLIGHT, oppure se spento: LCD_NOBACKLIGHT):

```
dpBacklight = LCD_BACKLIGHT;
```

3. Costruisce il byte dpFunction:

```
dpFunction = LCD_4BITMODE | LCD_2LINE | LCD_5x8DOTS;
```

Perché per mettere in piedi la comunicazione con il controllore si invia un byte che usa la seguente struttura:

b7	b6	b5	b4	b3	b2	b1	b0
0	0	1	DL	N	F	-	-

Dove:

- bit7, bit6 e bit5 sono fissi (0x20)
- DL (Data Length): può essere 1 se LCD_8BITMODE, oppure 0 se LCD_4BITMODE
- N (Number of lines): può essere 1 se LCD_2LINE, oppure 0 se LCD_1LINE
- F(Font): può essere 1 se LCD_5x10DOTS, oppure 0 se LCD_5x8DOTS

Costante	Esadecimale	Binario
LCD_4BITMODE	0x00	0000 0000
LCD_8BITMODE	0x10	0001 0000
LCD_1LINE	0x00	0000 0000
LCD_2LINE	0x08	0000 1000
LCD_5x8DOTS	0x00	0000 0000
LCD_5x10DOTS	0x04	0000 0100

Quindi la prima parte del bit è fissa, mentre gli altri bit devono essere impostati. Si parla infatti di maschere di bit, e facendone l'OR si ottengono i byte di comando completi.

Per configurare correttamente il controller, questo bit dovrà più avanti essere inviato al controller stesso tramite la funzione (solo dopo aver inizializzato il controller alla modalità 4-bit, argomento che verrà analizzato successivamente):

```
SendCommand(LCD_FUNCTIONSET | dpFunction);
```

4. Sequenza di delay iniziali per il rispetto del timing HD44780

È necessario rispettare dei delay iniziali per evitare che il display entri in uno stato non definito, mostrando caratteri casuali o strani, oppure potrebbe bloccarsi e ignorare i primi comandi. Una volta fatto questo il controller sarà pronto a ricevere la sequenza di reset a 4-bit.

```
/* Wait for initialization */
DelayInit(); //setup della funzione di delay,
//utile per ottenere un delay preciso in microsecondi
HAL_Delay(50);

ExpanderWrite(dpBacklight); // abilita il backlight
HAL_Delay(1000);
```

5. Transizione graduale da 8-bit mode a 4-bit mode

In ambito embedded è preferibile avere maggiore semplicità hardware (meno pin utilizzati) e un po' più di complessità software: perciò si preferisce usare la modalità a 4-bit, dove però ogni byte con l'informazione viene diviso a metà a formare due nibble, un nibble alto e un nibble basso che vengono inviati uno alla volta. Nonostante questo, il controller HD44780 riconosce i due nibble consecutivi e li assembla in un unico comando da 8 bit. Di seguito la sequenza di comandi per passare da uno stato indeterminato del controller alla modalità 4-bit:

```

/* 4bit Mode */
Write4Bits(0x03 << 4);
DelayUS(4500);

Write4Bits(0x03 << 4);
DelayUS(4500);

Write4Bits(0x03 << 4);
DelayUS(4500);

Write4Bits(0x02 << 4);
DelayUS(100);

```

6. Imposta display on/off, cursor, blink

```

dpControl = LCD_DISPLAYON | LCD_CURSOROFF | LCD_BLINKOFF; // accendo il display,
// spezzo il cursore e il blink
HD44780_Display();
HD44780_Clear(); // pulisco il display

```

```

void HD44780_Clear()
{
    SendCommand(LCD_CLEARDISPLAY);
    DelayUS(2000);
}

```

8. Imposta la modalità di scrittura (entry mode):

```

/* Display Mode */
dpMode = LCD_ENTRYLEFT | LCD_ENTRYSHIFTDECREMENT; // setto la modalità di scrittura
SendCommand(LCD_ENTRYMODESET | dpMode); //
DelayUS(4500);

```

9. Creazione di caratteri speciali se necessario in CGRAM con la funzione void HD44780_CreateSpecialChar (uint8_t, uint8_t []).

10. Riporta il cursore in home con la funzione HD44780_Home ():

```

void HD44780_Home()
{
    SendCommand(LCD_RETURNHOME);
    DelayUS(2000);
}

```

5.7 Implementazione software sensore di presenza umana

Abbiamo visto nel capitolo precedente relativo al sensore che durante i primi 60 secondi dopo l'alimentazione il modulo HC-SR501 entra in una fase di warm-up nella quale il sensore non è ancora stabile; perciò, per il corretto funzionamento del sensore è necessario ignorare i callback finché non sono passati 60 s per evitare falsi positivi. Sono state pensate due diverse strategie per bloccare la lettura del PIR nei primi secondi di avvio: la prima prevedeva di abilitare/disabilitare l'EXTI via NVIC, quindi dopo aver configurato il pin del segnale di output del sensore come EXTI, disabilitare subito la sua IRQ; HAL_NVIC_DisableIRQ (EXTI15_10_IRQHandler). Tuttavia, disabilitando tutti gli interrupt per i primi 60 s portava ad avere dei ritardi con gli altri pin configurati come EXTI sulla stessa linea. Si è perciò deciso di usare un filtro software basato su HAL_GetTick(), che garantisce che eventuali interrupt generati in questa fase vengano silenziati, agendo solamente sul pending bit dell'interrupt.

```
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin) {
    if (GPIO_Pin == PIR_Pin) {
        // Finché non sono passati 60 s, resettiamo solo il pending bit
        if (HAL_GetTick() - init_time_ms < 60000) {
            __HAL_GPIO_EXTI_CLEAR_IT(PIR_Pin);
            return;
        }
        // Solo dopo 60 s
        else {
            GPIO_PinState state = HAL_GPIO_ReadPin(PIR_GPIO_Port, PIR_Pin);

            if (state == true){
                standbymode = false;
                counterStandby = HAL_GetTick();
            }
        }
    }
}
```

Una volta terminata la fase di warm-up, l'implementazione della logica nella callback prevede di gestire correttamente il parametro standbymode, che rappresenta un booleano fondamentale nella transizione dallo stato Standby allo stato Acceso del dispenser. Infatti, il sistema acquisisce lo stato del pin associato al PIR attraverso HAL_GPIO_ReadPin(). Il valore acquisito viene salvato in state.counterStandby è una variabile che funge da contatore in millisecondi in maniera tale da permettere l'ingresso nello stato di Standby del dispenser dopo 5 minuti (300000 millisecondi) di assenza di movimento.

```
void am_HCSR501_handle(){
    if ((HAL_GetTick() - counterStandby) >= 300000){
        standbymode = true;
    }
}
```

5.8 Implementazione software PT1000

Per quanto riguarda il software, viene gestita l’acquisizione delle temperature a livello temporale tramite l’utilizzo di un timer. Sulla scheda nucleo l476rg è stato attivato il timer 7 settandolo con durata pari a 1s (Prescaler a 7999, Period a 9999) in grado di generare un interrupt. Il timer è settato nel file tim.c e viene avviato nel main. Ogni volta che il timer scade, avviene la lettura dei valori di tensione del circuito e viene eseguita una funzione ad hoc per la conversione dei valori dell’adc in gradi Celsius. Per ottenere questa conversione è stata implementata la seguente proporzione: $4095 : 3.3 = x : 3.10 \rightarrow x = 3834$ approssimato a 3800. Questo valore rappresenta il valore massimo che l’adc potrebbe leggere e quindi corrispondente al valore di 100°C. A seguito di campagne di misurazione si è scelto di abbassare $x=3650$, poiché eseguendo misurazioni con tensioni a 3.10V, e quindi aspettandoci un valore di lettura di 100°C, risultavano valori intorno a 90°C. Leggendo documentazione in rete al riguardo sono state individuate tra le possibili cause: un’errata tensione di riferimento dell’adc, un errore di guadagno, cadute di tensione o bassa impedenza del segnale, protezione da segnali in input.

Sono state definite le variabili “temperatura_fredda” e “temperatura_calda” per indicare i valori determinati di temperatura e altre variabili al fine di ottenere questi valori.

```
// Variabili per le PT1000
uint32_t lettura_fredda = 0;
float temperatura_fredda = 0.0;
uint32_t lettura_calda = 0;
float temperatura_calda = 0.0;

float tc = 0.0;
float tf = 0.0;

int i = 0;
```

La funzione di lettura dei valori dell’adc implementa una logica di controllo tale per cui se un valore campionato è superiore di 2°C o inferiore di 2°C rispetto al valore campionato precedentemente, questo non venga considerato in quanto variazioni di temperatura di tale portata richiedono tempi superiori al minuto, e quindi sono da considerare inesatti. La verifica della correttezza del campionamento è stata stabilita arbitrariamente a partire dal decimo campionamento in quanto si presuppone che la lettura da parte dell’adc sia già a regime.

```

void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef *hadc) {
    //HAL_ADC_Start_IT(&hadc1);

    lettura_fredda = HAL_ADC_GetValue(&hadc1);
    lettura_calda = HAL_ADC_GetValue(&hadc1);

    i++;

    tf = ((float) (100 * lettura_fredda) / 3650)/2;
    tc = (float) (100 * lettura_calda) / 3650;

    if (i<10){
        temperatura_fredda = tf;
        temperatura_calda = tc;
    }

    if (i>=10){
        if ((temperatura_fredda - 2 < tf) && (tf < temperatura_fredda + 2)) {
            temperatura_fredda = tf;
        }
        if ((temperatura_calda - 2 < tc) && (tc < temperatura_calda + 2)) {
            temperatura_calda = tc;
        }
    }
}

```

Il posizionamento del sensore di temperatura dell'acqua fredda non è ottimale, in quanto entra a contatto con l'acqua fredda solo durante la fase di chiamata da pulsante. Ciò fa sì che lo scambio termico tra la superficie del PT1000 e l'acqua fredda sia troppo breve affinché il sensore acquisisca correttamente la temperatura. A tale scopo si è optato per implementare una correzione tramite software che consiste nel dimezzare il valore di temperatura rilevato ottenendo un valore coerente con quello reale.

Per poter validare la corretta conversione dei valori della lettura delle porte adc, sono stati testati i circuiti di condizionamento rimuovendo i PT1000 ed inserendo al loro posto resistenze da 1000 ohm e da 1385 ohm per testare i valori estremi del range di nostro interesse.

00= OutputDispenser.OutValvolaTA	_Bool	false	0x2000040e
00= ParametriDispenser.StatoDispense	StatoDispenser_t	DispenserPRONTO	0x200003e0
00= temperatura_fredda	float	23.7142849	0x200002f4
00= temperatura_calda	float	70.1456985	0x200002f8
00= Distance	volatile float	27.1746941	0x20000290
00= InputDispenser.InStandBy	_Bool	false	0x2000031d
00= firstEntry	_Bool	true	0x200003fa

5.9 Implementazione software modulo Bluetooth HC-05

Il codice sviluppato è progettato per gestire la comunicazione tra un modulo Bluetooth HC-05 e una scheda STM32L476RG. L'obiettivo è permettere al microcontrollore di ricevere comandi inviati dal dispositivo Bluetooth, eseguire operazioni in base a tali comandi e rispondere al dispositivo con informazioni pertinenti.

Il programma inizia con la chiamata alla funzione “HAL_Init()”, che inizializza la libreria Hardware Abstraction Layer (HAL). Questo passaggio è essenziale per configurare l’hardware di base del microcontrollore STM32 e abilitare le funzionalità necessarie per la gestione dei dispositivi esterni (come il modulo Bluetooth).

Successivamente, vengono chiamate le funzioni di configurazione del sistema:

- SystemClock_Config(): Imposta la frequenza del clock del microcontrollore. In questo caso, la sorgente del clock è un oscillatore HSI (High-Speed Internal), e la PLL (Phase-Locked Loop) è utilizzata per aumentare la frequenza del sistema.
- MX_GPIO_Init(): Inizializza i pin GPIO, che saranno utilizzati per interagire con i dispositivi hardware, come il LED.
- MX_USART1_UART_Init(): Configura la porta USART1, che è la comunicazione seriale tra il microcontrollore e il modulo Bluetooth HC-05.

```
#include "main.h"
#include "usart.h"
#include "gpio.h"
#include "string.h"

/* Private variables -----*/
uint8_t rx_data; // Dato ricevuto via Bluetooth (USART1)

/* Private function prototypes -----*/
void SystemClock_Config(void);

// Invia una stringa via Bluetooth
void send_bluetooth_string(char *str)
{
    HAL_UART_Transmit(&huart1, (uint8_t*)str, strlen(str), HAL_MAX_DELAY);
}

int main(void)
{
    HAL_Init();
    SystemClock_Config();
    MX_GPIO_Init();
    MX_USART1_UART_Init(); // Solo USART1 = Bluetooth

    // Avvia ricezione interrupt da Bluetooth
    HAL_UART_Receive_IT(&huart1, &rx_data, 1);

    while (1)
    {
        // Tutto gestito da interrupt
    }
}
```

Viene definita poi una variabile globale, `uint8_t rx_data`, che contiene l'ultimo byte ricevuto via UART. Poi viene definita la funzione di invio con il bluetooth, ovvero “void `send_bluetooth_string(char *str)`”:

- `HAL_UART_Transmit()`: Invia una stringa tramite USART1
- La stringa viene convertita in array di byte, “`(uint8_t*)str`”
- `HAL_MAX_DELAY`: attende fino alla fine della trasmissione

Il modulo Bluetooth invia comandi al microcontrollore tramite la comunicazione seriale (USART1). La funzione “`HAL_UART_Receive_IT(&huart1, &rx_data, 1)`” viene utilizzata per avviare la ricezione asincrona dei dati. Questo significa che il microcontrollore non deve continuamente controllare se ci sono dati disponibili sulla UART; invece, verrà interrotto automaticamente quando arriva un dato, e la funzione di callback “`HAL_UART_RxCpltCallback`” verrà chiamata per gestirlo.

In questo modo, il sistema è ”interrupt-driven”, cioè non necessita di un ciclo di polling continuo, migliorando l’efficienza e riducendo il consumo di risorse.

Quindi, la ricezione dei dati avviene in interrupt, il che significa che il microcontrollore non rimane in attesa passiva di nuovi dati, ma viene “svegliato” solo quando un nuovo comando arriva. In questo modo, l’intera comunicazione risulta efficiente e reattiva.

Quando il microcontrollore riceve un dato tramite Bluetooth, la funzione di callback “`HAL_UART_RxCpltCallback` viene eseguita”. Al suo interno, il dato ricevuto viene analizzato per determinare quale comando è stato inviato dal dispositivo Bluetooth.

Quindi, la funzione “`HAL_UART_RxCpltCallback`” è il cuore della logica di controllo. Qui, ogni comando ricevuto viene confrontato con una serie di condizioni if-else, e a ciascun comando è associata un’azione specifica.

Ogni comando:

1. Attiva o spegne un LED per dare feedback visivo
2. Invia un messaggio di conferma via Bluetooth allo smartphone per quanto riguarda la parte di manutenzione
3. Imposta delle variabili booleane nella struttura `InputDispenser`, che verranno usate da altri moduli del software per avviare effettivamente le azioni sul dispenser (come attivare le valvole, far partire la pompa, ecc.)

Il sistema riceve oltre 30 comandi diversi, che vengono gestiti nella callback attraverso una serie di if-else. Ogni comando ricevuto attiva una funzione specifica del dispenser:

- Scelta della quantità e della tipologia di acqua
- Visualizzazione temperatura acqua calda/fredda
- Entrata/uscita dalla modalità manutenzione
- Lavaggio interno del dispenser
- Reset completo del sistema
- Accensione/spegnimento: delle elettrovalvole, dei pulsanti, della pompa, dei circuiti elettrici di riscaldamento e raffreddamento
- Regolazione della velocità della pompa

Di seguito non viene mostrato tutto il programma, ma vengono mostrati alcuni pezzi. In questo piccolo codice viene mostrata la gestione per quanto riguarda la selezione della quantità di acqua calda.

Nello specifico:

1. Comando '0': chiede 100 ml di acqua calda
2. Accende il LED (LD2) per indicare lo stato
3. Imposta i flag per erogare acqua calda (InAppCalda = true)
4. Imposta la quantità desiderata (InQuantita) e quella calibrata per i tempi/volumi reali (InQuantitaCalib)

Stessa cosa poi per il comando '1'(250 ml) e '2'(500 ml).

```
// Callback di ricezione dati UART
void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart) {
    if (huart->Instance == USART1) {
        if (rx_data == '0') {
            HAL_GPIO_WritePin(LD2_GPIO_Port, LD2_Pin, GPIO_PIN_SET); // Accendi LED
            InputDispenser.InAppCalda = true;
            InputDispenser.InQuantita = 100;
            InputDispenser.InQuantitaCalib = 145;

        } else if (rx_data == '1') {
            HAL_GPIO_WritePin(LD2_GPIO_Port, LD2_Pin, GPIO_PIN_SET); // Accendi LED
            InputDispenser.InAppCalda = true;
            InputDispenser.InQuantita = 250;
            InputDispenser.InQuantitaCalib = 317;

        } else if (rx_data == '2') {
            HAL_GPIO_WritePin(LD2_GPIO_Port, LD2_Pin, GPIO_PIN_SET); // Accendi LED
            InputDispenser.InAppCalda = true;
            InputDispenser.InQuantita = 500;
            InputDispenser.InQuantitaCalib = 622;
```

Successivamente viene mostrato un pezzo di codice relativo alla manutenzione del dispenser:

- 'p': entra in manutenzione → attiva flag "InAppManutenzione", invia risposta
- 'q': esce dalla manutenzione
- 'l': inizia il lavaggio
- 's': reset sistema

```

} else if (rx_data == 'p') {
    HAL_GPIO_WritePin(LD2_GPIO_Port, LD2_Pin, GPIO_PIN_SET); // Accendi LED
    send_bluetooth_string("manutenzione attiva\r\n");
    InputDispenser.InAppManutenzione = true;

} else if (rx_data == 'q') {
    HAL_GPIO_WritePin(LD2_GPIO_Port, LD2_Pin, GPIO_PIN_SET); // Accendi LED
    send_bluetooth_string("manutenzione finita\r\n");
    InputDispenser.InAppManutenzione = false;

} else if (rx_data == 'l') {
    HAL_GPIO_WritePin(LD2_GPIO_Port, LD2_Pin, GPIO_PIN_SET); // Accendi LED
    send_bluetooth_string("in fase di lavaggio\r\n");
    InputDispenser.InAppManuLavaggio = true;

} else if (rx_data == 's') {
    HAL_GPIO_WritePin(LD2_GPIO_Port, LD2_Pin, GPIO_PIN_RESET); // Spegni LED
    send_bluetooth_string("macchina spenta\r\n");
    InputDispenser.InAppManuResetAllarme = true;

```

Di seguito vengono mostrate le parti di codice relative a:

- accensione e spegnimento della valvola di uscita dell'acqua calda
- accensione e spegnimento del circuito dell'acqua calda

```

} else if (rx_data == 'a') {
    HAL_GPIO_WritePin(LD2_GPIO_Port, LD2_Pin, GPIO_PIN_SET); // Accendi LED
    send_bluetooth_string("Valvola uscita caldo accesa\r\n");
    InputDispenser.InAppManuValvolaOutCaldo = true;

} else if (rx_data == 'b') {
    HAL_GPIO_WritePin(LD2_GPIO_Port, LD2_Pin, GPIO_PIN_RESET); // Spegni LED
    send_bluetooth_string("Valvola uscita caldo spenta\r\n");
    InputDispenser.InAppManuValvolaOutCaldo = false;

} else if (rx_data == 'w') {
    HAL_GPIO_WritePin(LD2_GPIO_Port, LD2_Pin, GPIO_PIN_SET); // Accendi LED
    send_bluetooth_string("Circuito caldo accesa\r\n");
    InputDispenser.InAppManuCircuitoCaldo = true;

} else if (rx_data == 'x') {
    HAL_GPIO_WritePin(LD2_GPIO_Port, LD2_Pin, GPIO_PIN_RESET); // Spegni LED
    send_bluetooth_string("Circuito caldo spenta\r\n");
    InputDispenser.InAppManuCircuitoCaldo = false;

```

Nel codice successivo viene mostrato il codice per gestire il pulsante dell'acqua calda.

```

} else if (rx_data == 'k') {
    HAL_GPIO_WritePin(LD2_GPIO_Port, LD2_Pin, GPIO_PIN_RESET); // Spegni LED
    send_bluetooth_string("Led calda acceso\r\n");
    InputDispenser.InAppManuLedButtonCalda = true;

} else if (rx_data == ';') {
    HAL_GPIO_WritePin(LD2_GPIO_Port, LD2_Pin, GPIO_PIN_RESET); // Spegni LED
    send_bluetooth_string("Led calda spento\r\n");
    InputDispenser.InAppManuLedButtonCalda = false;
}

```

Qua viene mostrato il codice per accendere e spegnere la pompa. Successivamente invece

```

} else if (rx_data == '(') {
    HAL_GPIO_WritePin(LD2_GPIO_Port, LD2_Pin, GPIO_PIN_RESET); // Spegni LED
    send_bluetooth_string("Pompa accesa\r\n");
    InputDispenser.InAppManuPompaStartStop = false;
    InputDispenser.InAppManuPompaVelFissa = 0;

} else if (rx_data == ')') {
    HAL_GPIO_WritePin(LD2_GPIO_Port, LD2_Pin, GPIO_PIN_RESET); // Spegni LED
    send_bluetooth_string("Pompa spenta\r\n");
    InputDispenser.InAppManuLavaggio = false;
}

```

viene mostrato il codice per andare a selezionare la velocità della pompa. Qua vediamo

```

} else if (rx_data == '!') {
    HAL_GPIO_WritePin(LD2_GPIO_Port, LD2_Pin, GPIO_PIN_RESET); // Spegni LED
    send_bluetooth_string("Velocità pompa 25%\r\n");
    InputDispenser.InAppManuPompaStartStop = true;
    InputDispenser.InAppManuPompaVelFissa = 25;

} else if (rx_data == '?') {
    HAL_GPIO_WritePin(LD2_GPIO_Port, LD2_Pin, GPIO_PIN_RESET); // Spegni LED
    send_bluetooth_string("Velocità pompa 50%\r\n");
    InputDispenser.InAppManuPompaStartStop = true;
    InputDispenser.InAppManuPompaVelFissa = 50;

} else if (rx_data == '%') {
    HAL_GPIO_WritePin(LD2_GPIO_Port, LD2_Pin, GPIO_PIN_RESET); // Spegni LED
    send_bluetooth_string("Velocità pompa 75%\r\n");
    InputDispenser.InAppManuPompaStartStop = true;
    InputDispenser.InAppManuPompaVelFissa = 75;

} else if (rx_data == '/') {
    HAL_GPIO_WritePin(LD2_GPIO_Port, LD2_Pin, GPIO_PIN_RESET); // Spegni LED
    send_bluetooth_string("Velocità pompa 100%\r\n");
    InputDispenser.InAppManuPompaStartStop = true;
    InputDispenser.InAppManuPompaVelFissa = 98;

} else if (rx_data == '=') {
    HAL_GPIO_WritePin(LD2_GPIO_Port, LD2_Pin, GPIO_PIN_RESET); // Spegni LED
    send_bluetooth_string("Pompa spenta\r\n");
    InputDispenser.InAppManuResetAllarme = false;
}

```

le righe di codice fatte per mandare all'app i valori della temperatura dell'acqua fredda e calda:

1. Legge la temperatura (cast da float a int)
2. Costruisce una stringa da inviare all'app
3. La invia via Bluetooth

```

} else if (rx_data == '+') {
    int Tcalda = (int)InputDispenser.InTempCalda;
    int Tfredda = (int)InputDispenser.InTempFredda;
    char msg[50];
    sprintf(msg, "Calda: %d    Fredda: %d\r\n", Tcalda, Tfredda);
    send_bluetooth_string(msg);
}

```

Dopo aver gestito un dato ricevuto, la funzione di callback riavvia la ricezione asincrona con HAL_UART_Receive_IT(&huart1, &rx_data, 1), così il microcontrollore è pronto per ricevere nuovi comandi Bluetooth.

Nel caso si verifichi un errore durante l'esecuzione, come una configurazione errata del clock o dei periphery, il microcontrollore entra in un ciclo infinito grazie alla funzione Error_Handler(). Questo blocca il sistema e permette di gestire errori critici, ad esempio in fase di debug.

```

// Riavvia la ricezione
HAL_UART_Receive_IT(&huart1, &rx_data, 1);
}

void Error_Handler(void)
{
    __disable_irq();
    while (1) {}
}

```

6 Sviluppo dell'applicazione smartphone

Per lo sviluppo dell'applicazione, utile per comandare da remoto (tramite Bluetooth) il dispenser, si è scelto di utilizzare MIT App Inventor, una piattaforma open source online progettata per la creazione di semplici app su dispositivi Android.

Durante la fase di progettazione sono state valutate anche altre soluzioni tra cui:

- Android Studio: Piattaforma ufficiale e potente ma richiede una conoscenza più approfondita del linguaggio di programmazione Java
- Thunkable: Basato su un'interfaccia grafica simile a quella di MIT App Inventor. Tuttavia, MIT App Inventor è stato preferito per la sua maggiore stabilità, l'ampia documentazione disponibile e la semplicità d'uso

MIT App Inventor si è rivelato quindi ideale per un progetto didattico, anche per i tempi rapidi di realizzazione grazie al suo funzionamento generale a blocchi piuttosto che su codice standard.

6.1 Linguaggio di programmazione

Mit App Inventor utilizza un linguaggio a blocchi visuale, basato su Blockly (sviluppato da Google).

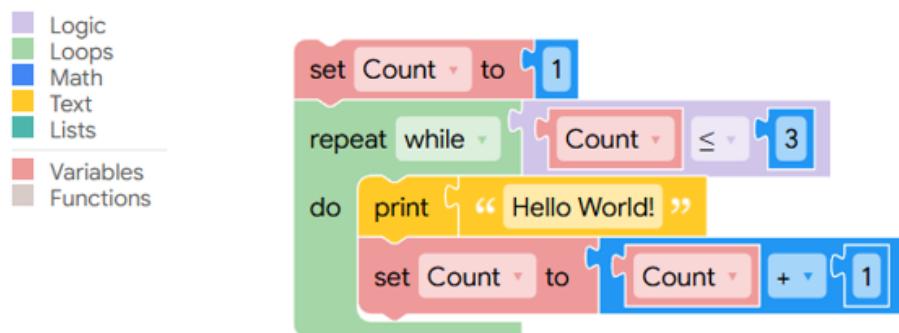


Figura 6.1: Dettaglio Blockly by GOOGLE

Questo tipo di programmazione consente di creare delle strutture logiche/comandi trascinando e collegando i blocchi colorati corrispondenti a una sottocategoria. La programmazione è guidata in quanto non tutti i blocchi possono essere collegati con altri, ne limita però la creatività.

Questo approccio ha vari vantaggi:

- Più accessibile e intuitiva
- Debug più facile
- Adatto per creazione di prototipi

Ulteriore, e fondamentale, punto a favore di Mit APP Inventor è la possibilità di salvare dati nella memoria interna dell'app.

I dati possono essere raccolti tramite componenti quali:

- TinyDB: Database locale che permette di salvare dati persistenti sul dispositivo anche dopo la chiusura dell'app.

- FirebaseDatabase: Per la memorizzazione remota nel caso si volesse espandere il progetto

Nel nostro caso è stato scelto TinyDB poiché consente di salvare in memoria i pochi dati di cui necessitiamo senza utilizzare applicazioni esterne e aggiungere Add-on all'applicazione. Basta infatti “trascinare” il blocco TinyDB all'interno dell'app per far sì che tutti i dati vengano salvati.

6.2 Struttura dell'applicazione

L'app è stata pensata per gestire in modo intelligente l'erogazione di acqua a varie temperature e quantità tramite un dispenser controllato da una scheda Nucleo connessa via Bluetooth.

L'interfaccia utente è articolata in quattro schermate, ciascuna con funzioni specifiche in base al tipo di utente che effettua il login.

Login

La prima schermata dell'app è la pagina di accesso in cui autenticarsi.

Sono stati creati tre account fintizi per simulare l'utilizzo dell'app da parte di figure diverse:

- Professore 1
- Professore 2
- Manutentore

A seconda del profilo che accede, l'applicazione indirizza l'utente verso la schermata di competenza. Ad esempio, solo il profilo di Manutentore può accedere alla pagina di Manutenzione. Professore 1 e Professore 2 sono utenti standard e hanno le stesse funzionalità nell'app.

Homepage

È la schermata principale per gli utenti standard, include varie funzionalità per utilizzare lo smart dispenser:

- Connessione Bluetooth: L'app consente di connettersi ad un modulo bluetooth esterno per comunicare con il dispenser.
- Selezione della quantità di acqua: L'utente può impostare la quantità di acqua desiderata su tre livelli: 100-250-500 [ml]
- Scelta della temperatura: Sono disponibili tre livelli di temperatura selezionabili (calda, fredda, ambiente).
- Visualizzazione dati giornalieri: È possibile analizzare la quantità di acqua erogata nella giornata odierna
- Dati Avanzati: Permette di accedere ad una schermata ulteriore per un'analisi più approfondita dei dati

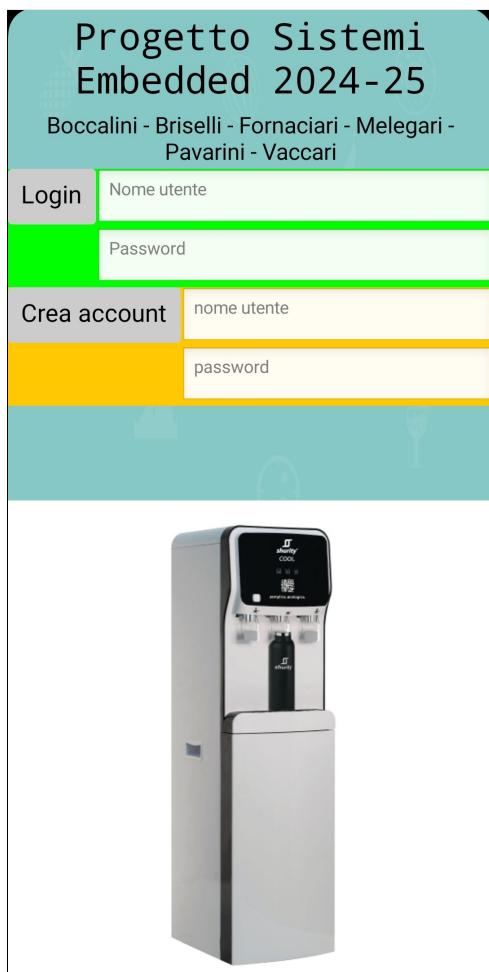


Figura 6.2: Schermata di login

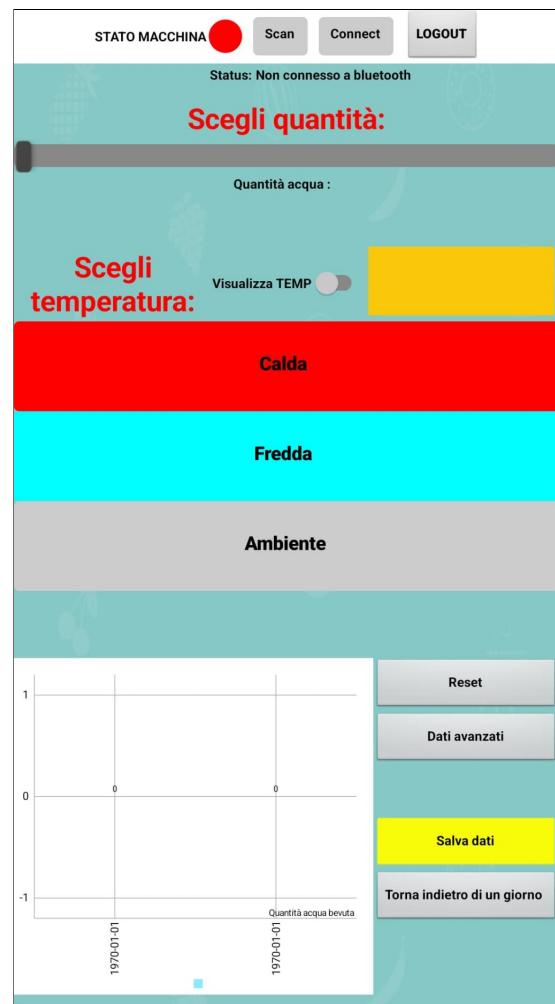


Figura 6.3: Schermata home

Storico dati

La schermata “Storico Dati” è dedicata al monitoraggio a lungo termine delle erogazioni. Include:

- Visualizzazione cronologica: Mostra tutti i giorni in cui è stata erogata dall’acqua con possibilità di selezione singola su tutto l’elenco.
- Analisi settimanale: Possibilità di riassumere la quantità totale d’acqua erogata su base settimanale, facilitando il controllo del consumo.

Questi dati sono memorizzati utilizzando TinyDB.



Figura 6.4: Storico dati utente

Manutenzione

Questa schermata è accessibile solo attraverso login con utente “Manutentore”.

Le funzioni principali sono:

- Controllo valvole: l’interfaccia permette di monitorare lo stato delle valvole presenti nel dispenser potendole azionare singolarmente e valutarne il funzionamento
- Reset allarmi

La pagina è pensata per fornire un’interfaccia semplice ma funzionale al manutentore senza interferire con le funzionalità riservate agli utenti standard.



Figura 6.5: Schermata di manutenzione

6.3 Programmazione via app

Nel seguente sottocapitolo andremo ad analizzare nel dettaglio la programmazione usata per far funzionare l'applicazione. Per ogni schermata verranno spiegati i vari macro-blocchi.

6.3.1 Homepage

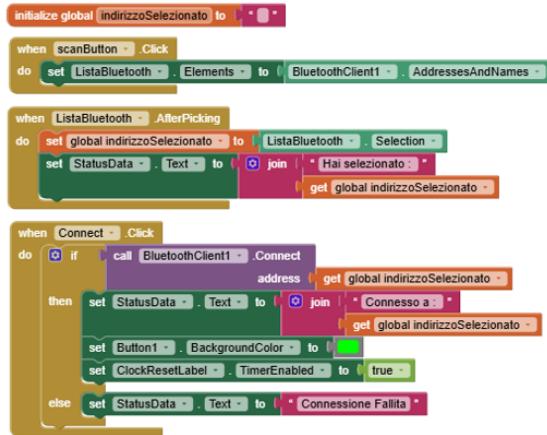


Figura 6.6: Blocchi per connessione bluetooth

Questa sezione dell'app consente di visualizzare la lista di dispositivi Bluetooth già connessi sul cellulare, selezionarne uno e tentare la connessione.

Inizializzazione della variabile globale

Crea una variabile globale chiamata indirizzoSelezionato, inizializzata come una stringa vuota. Serve per memorizzare l'indirizzo del dispositivo Bluetooth selezionato dall'utente.

Blocco scanButton.Click

Quando l'utente clicca sul pulsante di scansione, l'app:

- Popola il menu a tendina ListaBluetooth con tutti i dispositivi Bluetooth trovati.
- Gli indirizzi e i nomi sono recuperati tramite BluetoothClient1.AddressesAndNames.

Blocco ListaBluetooth.AfterPicking

Dopo che l'utente ha scelto un dispositivo dalla lista:

- L'indirizzo selezionato viene memorizzato in indirizzoSelezionato.
- Viene mostrato a schermo un messaggio di conferma.

Blocco Connect.Click

Quando l'utente preme il pulsante Connect:

- Viene effettuato un tentativo di connessione Bluetooth con l'indirizzo selezionato.
- Se la connessione ha successo:
 1. Il testo StatusData viene aggiornato per mostrare il successo.
 2. Il colore di Button1 cambia per confermare la connessione.
 3. Un timer viene attivato (ClockResetLabel), per il reset automatico del testo StatusData
- Se fallisce, viene mostrato un messaggio di errore.

6.3.2 Timer 1 – Salvataggio della quantità d'acqua giornaliera (Clock1)

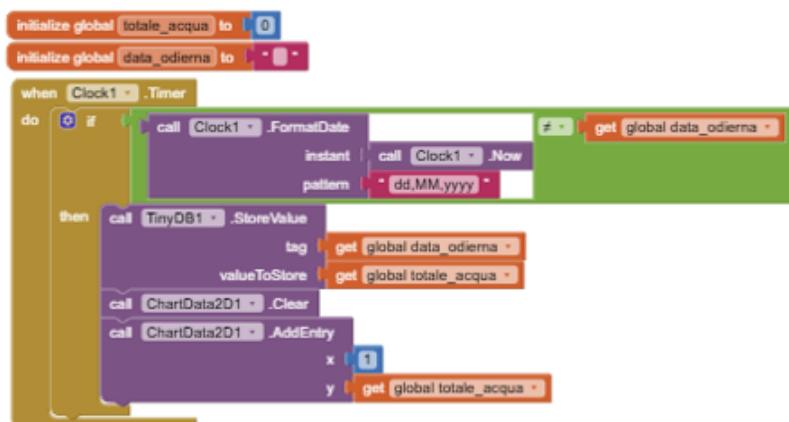


Figura 6.7: Blocchi per salvataggio dati su tinydb

Questo timer serve per registrare la quantità d'acqua erogata ogni giorno e visualizzarla tramite un grafico.

Variabili inizializzate:

- totale_acqua = 0: memorizza la quantità totale d'acqua erogata nel giorno corrente.
- data_odierna = "": verrà popolata con la data formattata.

Azione del timer:

- Viene chiamata la funzione FormatDate per ottenere la data corrente in formato dd,MM,yyyy.
- Se la data ottenuta è diversa da quella memorizzata in data_odierna (quindi è cambiato il giorno):
 1. L'app salva il valore di totale_acqua nella TinyDB, usando la data come chiave.
 2. Viene poi svuotato il grafico ChartData2D1.
 3. Viene aggiunto un nuovo punto con x = 1 e y = totale_acqua.

Utilità:

- Mantiene tracciabilità giornaliera dell'acqua consumata.
- Organizza i dati nel grafico in modo ordinato e aggiornato.
- Automatizza il salvataggio senza bisogno di interazione da parte dell'utente.

6.3.3 Timer 2 – Ricezione dei dati via Bluetooth (TimerRicezioneDati)

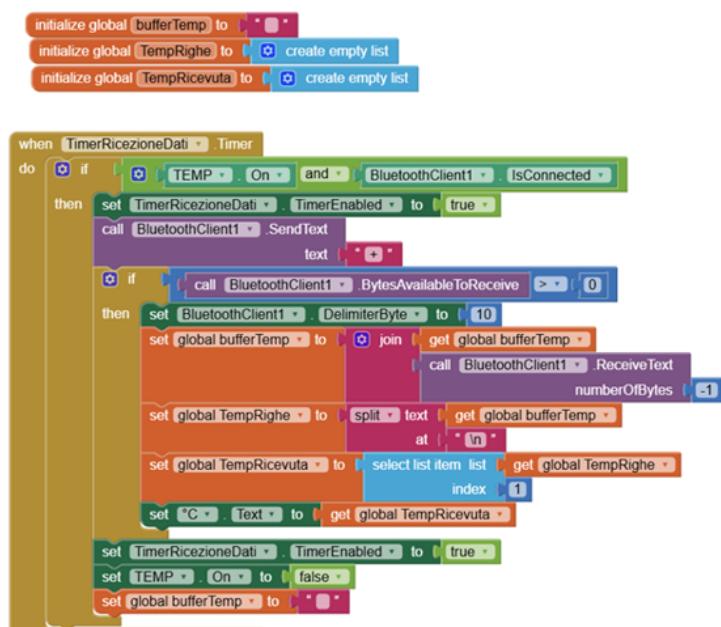


Figura 6.8: Blocchi timer ricezione datB

Questo timer gestisce la comunicazione dati dalla scheda verso l'app, ricevendo messaggi via Bluetooth, è stato utilizzato per feedback della temperatura dei due serbatoi di acqua calda e fredda.

Variabili utilizzate:

- bufferTemp: testo ricevuto.
- TempRicevuta: messaggio interpretato.
- TempRighe: lista che conterrà le righe del messaggio ricevuto.

Azione del timer:

1. Il timer si attiva solo se il Bluetooth è connesso.
2. Verifica se ci sono byte disponibili da ricevere.
3. Imposta il delimitatore (fine messaggio).
4. Riceve il testo e lo inserisce in bufferTemp.
5. Divide il messaggio in righe con split, salvando i dati in TempRighe.
6. Il primo elemento ricevuto viene interpretato come segnale (TempRicevuta).
7. Mostra sul bottone giallo il messaggio ricevuto contenente le temperature dei serbatoi.
8. Disabilita temporaneamente il timer per evitare letture multiple sovrapposte.
9. Resetta il buffer per la prossima ricezione.

6.3.4 Selezione della quantità con lo Slider (inviaComando)

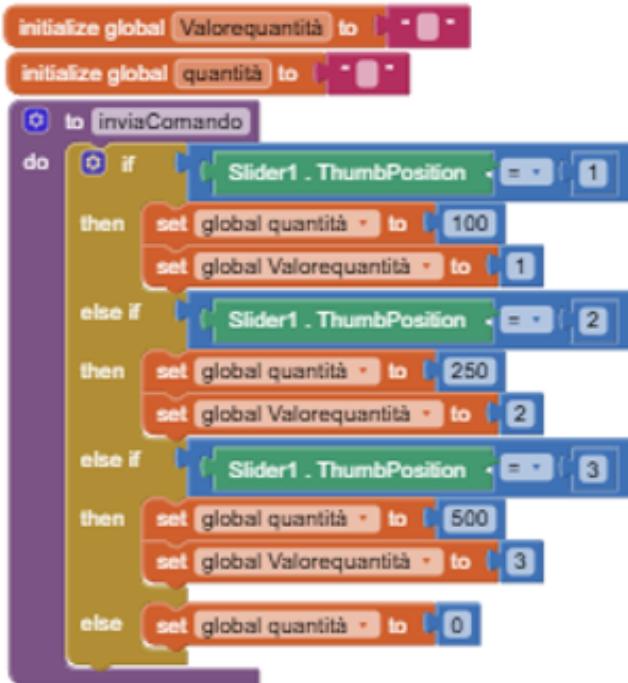


Figura 6.9: Funzione selezione quantità acqua

Imposta il valore della quantità d'acqua da erogare in base alla posizione dello slider. Il valore viene salvato in due variabili globali:

- Quantità, usata per i grafici e statistiche
- Valorequantità, usata per inviare il codice alla scheda via Bluetooth

Panoramica dei blocchi:

Posizione Slider	Valore quantità	Codice Valorequantità
1	100 mL	”1”
2	250 mL	”2”
3	500 ml	”3”
Altro defualt	0	”0”

6.3.5 Pulsante “Calda” (Calda.Click)

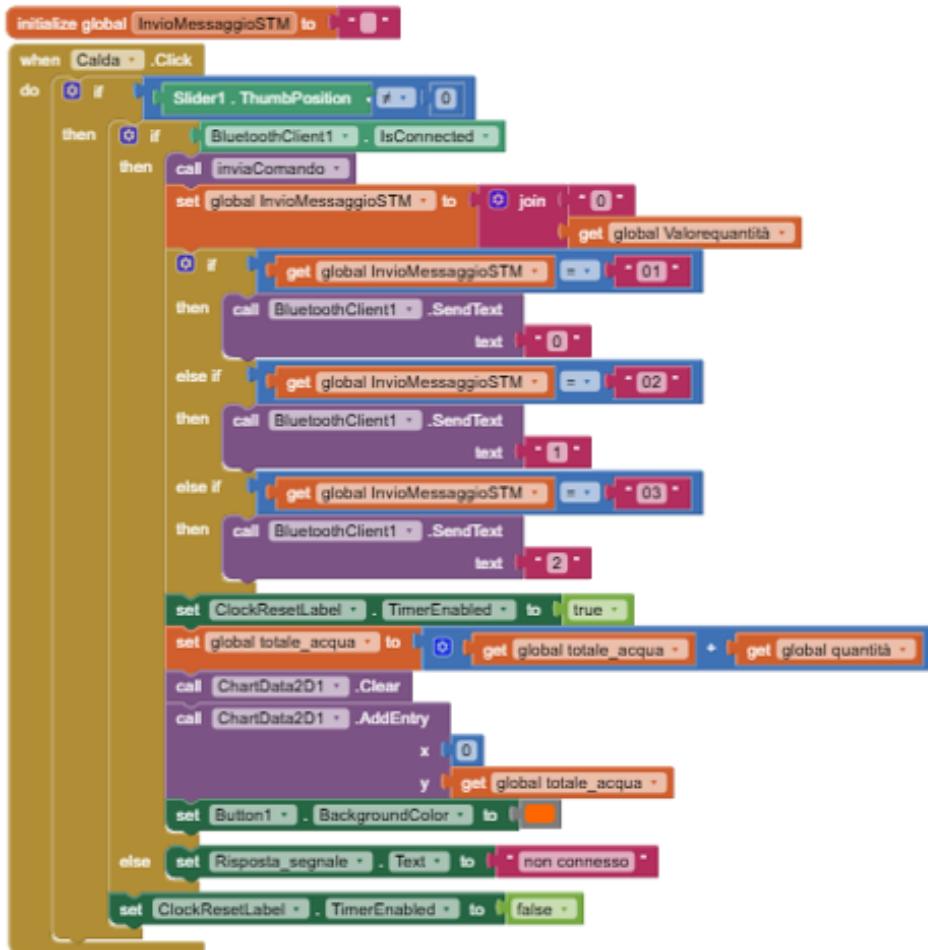


Figura 6.10: Blocchi pulsante Calda

Quando si preme il pulsante, se il Bluetooth è connesso e lo slider non è in posizione 0, l'app invia il comando alla STM32 con le istruzioni per erogare acqua calda, nella quantità scelta tramite lo slider.

Analisi dei passaggi:

1. Controllo connessione Bluetooth. Se non connesso, mostra "non connesso".
2. Chiama `inviaComando`. Prepara le variabili `quantità` e `Valorequantità`.
3. Costruisce il comando per STM32. Usa la variabile `Valorequantità` per comporre una stringa da inviare tramite bluetooth come:
 - "0" → 01 (calda 100 mL)
 - "1" → 02 (calda 250 mL)
 - "2" → 03 (calda 500 mL) Il suffisso "0" identifica il tipo di acqua (calda). La cifra successiva viene prelevata da `Valorequantità`.

4. Invia il messaggio con SendText. Il comando completo viene inviato alla scheda via Bluetooth.
5. Aggiorna il grafico. Somma la quantità attuale a totale_acqua.
6. Timer e interfaccia. Riattiva il ClockResetLabel e cambia il colore di un bottone in arancione per indicare lo stato di erogazione. Il bottone torna al colore originale in base alle impostazioni del timer che lo resetta.

Ovviamente, questo schema viene ripetuto per le ulteriori due temperature presenti nell'app. La differenza consta nel prefisso nella creazione della variabile “InvioMessaggioSTM” (1,2 invece di 0). Le seconde cifre sono le stesse prese dalla posizione dello slider.

6.3.6 Funzioni di test e debug: salvataggio manuale dei dati giornalieri

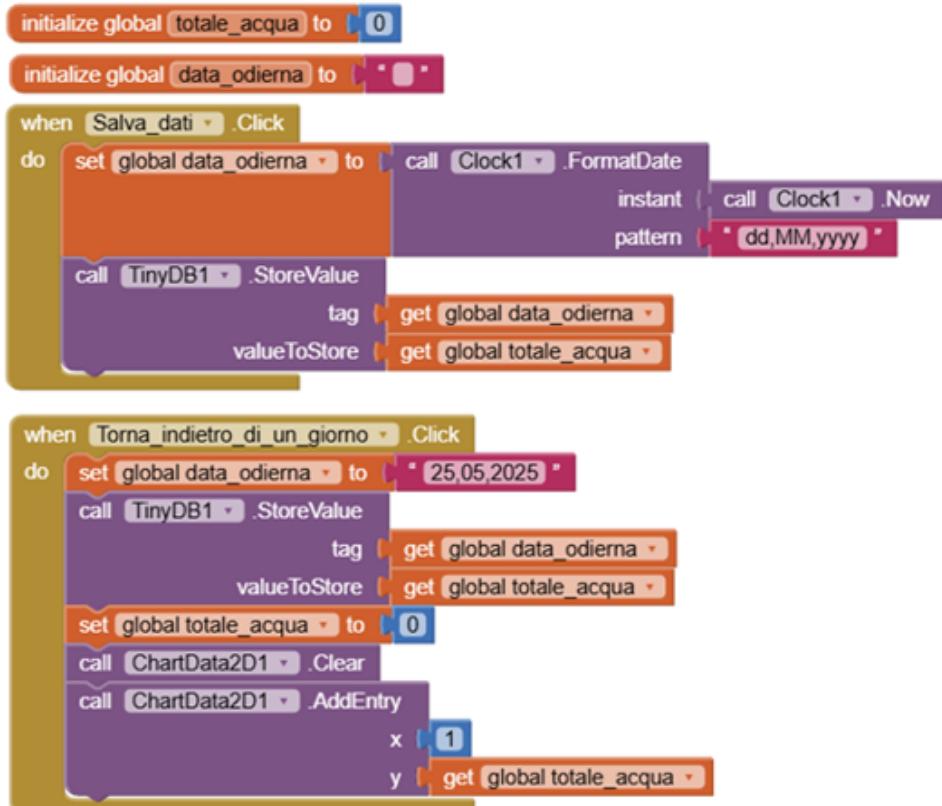


Figura 6.11: Blocchi per test-debug

Per validare il corretto funzionamento del salvataggio dei dati giornalieri nel componente TinyDB, sono stati implementati due pulsanti di controllo nella pagina principale.

Pulsante ”SALVA DATI”

Questo pulsante permette di forzare il salvataggio dei dati della giornata corrente, senza dover attendere l'esecuzione automatica da parte del timer. Funzionamento:

- Acquisisce la data odierna formattata (dd,MM,yyyy) tramite il componente Clock1.

- Salva all'interno della TinyDB la quantità totale di acqua erogata fino a quel momento (totale_acqua).

Pulsante ”TORNA INDIETRO DI UN GIORNO”

Utilizzato per simulare la presenza di dati in giorni differenti rispetto a quello corrente, utile per testare la pagina dello storico dati. Funzionamento:

- Imposta manualmente la variabile data_odierna a una data specifica (es. ”25,05,2025”).
- Salva in TinyDB i dati totale_acqua associati a quella data.
- Azzera il contatore totale_acqua e aggiorna il grafico per simulare il passaggio al giorno successivo.

Procedura di test utilizzata:

1. Simulazione erogazione acqua: L'utente utilizza i pulsanti per erogare acqua in quantità diverse.
2. Salvataggio forzato: Premendo SALVA DATI, i dati vengono memorizzati nel database locale con la data attuale.
3. Cambio giorno manuale: Con TORNA INDIETRO DI UN GIORNO, si simula l'inizio di una nuova giornata.
4. Nuova erogazione: L'utente può nuovamente testare l'erogazione con quantità diverse. Nuova erogazione: L'utente può nuovamente testare l'erogazione con quantità diverse.
5. Salvataggio successivo: I dati vengono salvati anche per la nuova data simulata.
6. Verifica nello storico: Attraverso la pagina dedicata, è possibile verificare che entrambi i giorni abbiano i rispettivi dati registrati, a conferma del corretto funzionamento.

6.3.7 Pagina "Storico Dati" – Visualizzazione settimanale

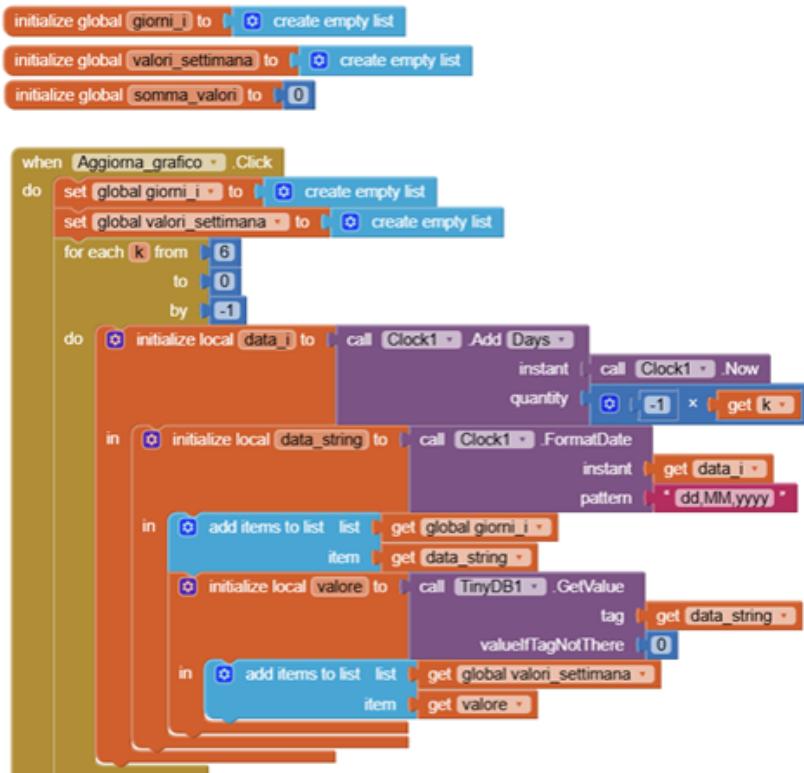


Figura 6.12: Blocchi visualizza storico dati settimanale (1)

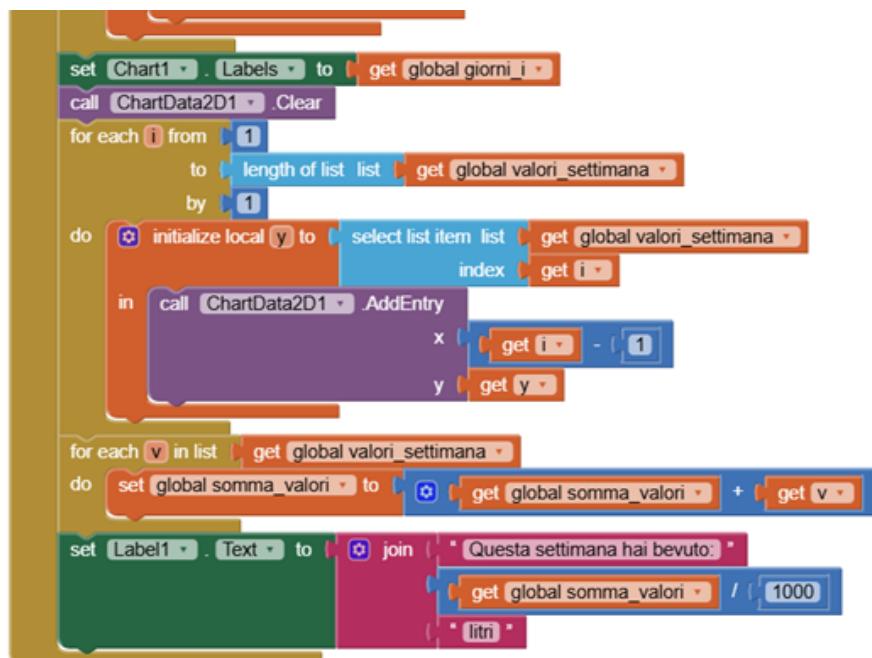


Figura 6.13: Blocchi visualizza storico dati settimanale (2)

Questa schermata consente di visualizzare i consumi giornalieri d'acqua della settimana corrente o della settimana relativa a una data selezionata. I dati vengono letti dalla memoria interna (TinyDB) e rappresentati in un grafico a barre.

Quando l'utente preme il pulsante "Aggiorna grafico", l'app esegue il seguente processo:

1. Calcolo delle date. Viene eseguito un ciclo da 6 a 0 per ottenere i sette giorni precedenti, incluso quello attuale. Per ciascun giorno:

- Viene calcolata la data esatta con Clock1.AddDays.
- La data viene formattata come stringa "dd,MM,yyyy" e salvata nella lista giorni_i.

2. Lettura dei dati da TinyDB. Per ogni data calcolata:

- L'app legge il valore associato (quantità d'acqua bevuta) dalla TinyDB.
- Se non esiste alcun valore, imposta 0 come default.
- I valori letti vengono salvati nella lista valori_settimana.

3. Aggiornamento del grafico. Per ogni giorno viene inserito un punto sul grafico con x = indice e y = valore (ml d'acqua).

4. Calcolo totale settimanale:

- Un secondo ciclo somma tutti i valori di valori_settimana nella variabile somma_valori.
- Il risultato viene diviso per 1000 per ottenere i litri.
- Infine, viene mostrata una frase informativa all'utente tramite Label1, del tipo: "Questa settimana hai bevuto: 3.5 litri".

6.3.8 Storico dati – selezione giornaliera

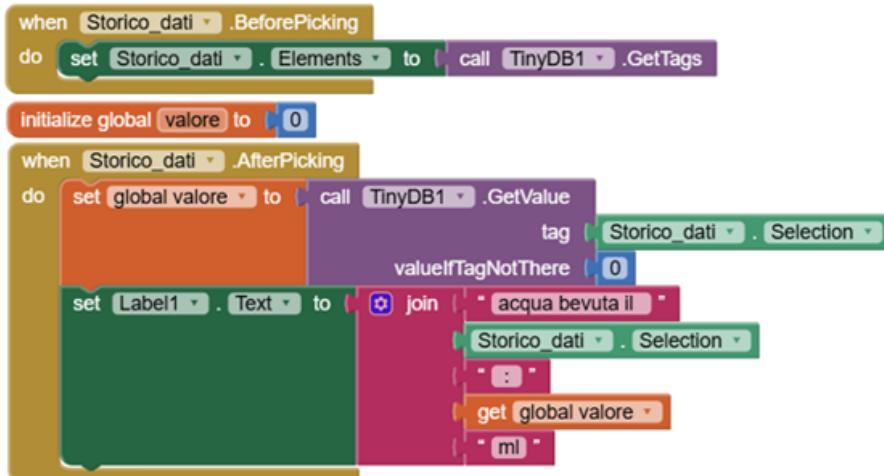


Figura 6.14: Blocchi visualizza quantità acqua giornaliera

Nella schermata "Storico dati" vi è la possibilità di visualizzare l'acqua erogata in un qualsiasi giorno in cui è stata utilizzata l'app. Tramite un pulsante "Storico dati" nella schermata appare una lista di tutte le date, selezionandola appare a schermo un messaggio tipo: "acqua bevuta il – giorno selezionato – global valore – ml". I valori della lista sono presi attraverso la memoria interna dell'app TinyDB1.

6.3.9 Manutenzione

La pagina Manutenzione viene utilizzata dal manutentore per verificare eventuali malfunzionamenti nel dispenser. Questa è formata da 14 tasti che azionano 14 elementi diversi del dispenser. I pulsanti sono i seguenti: I pulsanti sono i seguenti:

1. Relè Caldo
2. Relè Freddo
3. Lavaggio
4. Bottone Calda
5. Bottone Fredda
6. Bottone Ambiente
7. Reset Allarme
8. Uscita Calda
9. Uscita Fredda
10. Sfiato Serbatoio
11. Valvola Calda
12. Valvola Ambiente
13. Valvola Fredda
14. Accensione Pompa

I 14 pulsanti in base ad uno switch I/O possono indicare l'accensione o lo spegnimento del comando. Oltre a questi comandi sono stati inseriti ulteriori due per far sì che il dispenser entri/esca dallo stato di manutenzione (stato software dalla scheda). Nel dettaglio il pulsante di accensione pompa è collegato ad uno switch per regolare la velocità di rotazione a 4 scelte (25,50,75,100 [%]). Successivamente analizzeremo nel dettaglio il linguaggio di programmazione di:

- Un pulsante (sui 14 totali) poiché il formato è il medesimo
- Pulsante accensione pompa

Uscita calda valvole

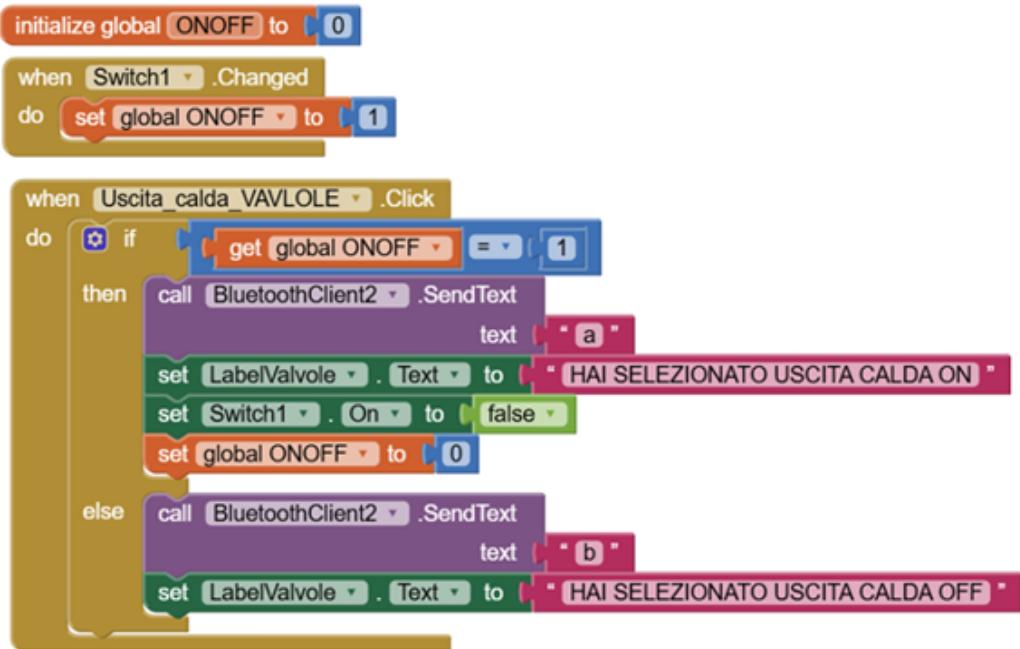


Figura 6.15: Blocchi Uscita calda valvole con switch

Lo schema parte inizializzando una variabile globale denominata ONOFF che è collegata ad uno switch I/O, quando lo switch viene acceso la variabile passa da 0 a 1. Quando il pulsante Uscita.calda_VALVOLE viene premuto, se lo switch è acceso (global ONOFF=1) allora l'app invia una stringa prestabilita (es. "a") alla scheda. Stringa che, quando ricevuta dalla scheda fa partire il comando programmato.

Una volta inviato il segnale, lo switch torna in stato False e la variabile globale ONOFF torna ad 1.

Questo comando funziona anche senza attivare lo switch inviando una stringa diversa (es. "b") alla scheda.

Accensione pompa

Il pulsante accensione pompa è collegato allo slider velocità (stesso funzionamento dello slider temperatura presente nella Homepage) e allo switch I/O (stesso pulsante presente nel precedente schema).

Lo schema dei blocchi è una combinazione di schemi già visti in precedenza. Se lo switch è acceso (variabile globale ONOFF = 1) in base alla posizione dello slider (se diversa da 0), quando il pulsante è premuto, l'app invia una stringa diversa per ogni combinazione possibile.

Se lo switch non è acceso o lo slider in posizione zero, la stringa inviata è differente e corrisponde alla configurazione OFF dell'accensione pompa.

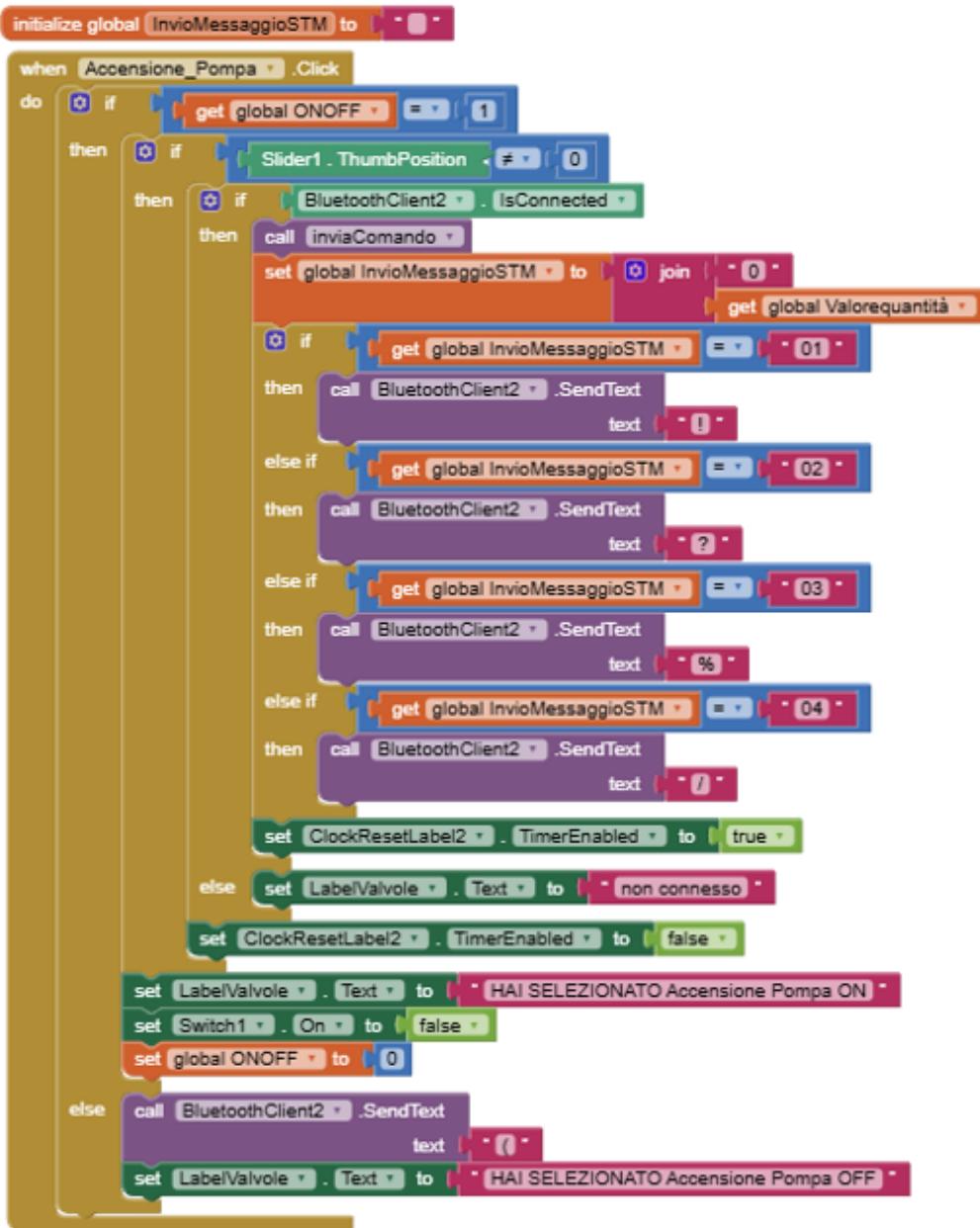


Figura 6.16: Blocchi Accensione pompa con slider velocità

7 Conclusioni

Il progetto dello smart water dispenser si è rivelato una vera sfida per tutto il team di sviluppo, tuttavia siamo soddisfatti del prodotto finale. Questo primo prototipo vuole gettare le basi per future migliorie e implementazioni che possono incrementare le performance attuali del dispositivo. Alcuni spunti e idee erano sorte sin dall'inizio nella fase di sviluppo concettuale, nonostante ciò non sono state applicate per vincoli di tempo e risorse materiali. Si vogliono dunque riportare in elenco alcune aggiunte ed implementazioni future:

- Integrare un kit di analisi chimica della qualità dell'acqua presente nel serbatoio principale e mettere a disposizione l'analisi dei dati rilevati all'utente consumatore.
- Implementare un sistema di raffreddamento basato su compressore e serpentini (come quello del frigorifero). Questo aumenterebbe notevolmente la capacità di raffreddamento del dispenser.
- Aumentare la portata volumetrica del flusso d'acqua riprogettando il circuito idraulico con tubi di sezione maggiore.
- Realizzare una scheda PCB per l'intera gestione del circuito di controllo del dispenser. In particolare realizzare un PCB che integri il microcontrollore STM32L476RG e tutti i circuiti di condizionamento dei sensori, pulsanti e display. Questa operazione ridurrebbe la quantità di cavi aumentando l'affidabilità e la robustezza del dispositivo.