



# KHORD: Progettazione e Realizzazione di una Tastiera Stenografica Digitale

Marco Barbaro \* e Valeria Stighezza \*

Dipartimento di Informatica "Giovanni degli Antoni", Università degli Studi di Milano

\*Email: [marco.barbaro@studenti.unimi.it](mailto:marco.barbaro@studenti.unimi.it); [valeria.stighezza@studenti.unimi.it](mailto:valeria.stighezza@studenti.unimi.it)

## Indice

<b>1 Introduzione</b>	<b>3</b>
1.1 La Stenografia . . . . .	3
1.2 Le tastiere meccaniche . . . . .	4
1.3 Motivazioni e Obiettivi del progetto . . . . .	4
<b>2 Progettazione Hardware</b>	<b>5</b>
2.1 Componentistica . . . . .	5
2.1.1 Scelta del Microcontrollore (MCU) . . . . .	5
2.1.2 Through-hole vs SMD . . . . .	6
2.2 Gestione Affidabile dell'Input . . . . .	6
2.2.1 Ghosting e Blocking . . . . .	6
2.2.2 N-Key Rollover . . . . .	7
2.3 Progettazione del Circuito Stampato (PCB) . . . . .	7
2.3.1 Schema Elettrico . . . . .	7
2.3.2 PCB Layout . . . . .	8
2.3.3 Modello 3D . . . . .	11
2.3.4 Prototipo per il Testing del Firmware . . . . .	13
2.4 Assemblaggio . . . . .	13
2.4.1 Produzione del PCB . . . . .	13
2.4.2 Modellazione e Stampa 3D del Case . . . . .	14
2.4.3 Saldatura dei Componenti . . . . .	16
2.4.4 Montaggio Finale della Tastiera . . . . .	17
2.5 Librerie e Plugin Utilizzati . . . . .	17
<b>3 Progettazione Firmware</b>	<b>18</b>
3.1 Pipeline di Sviluppo . . . . .	18
3.1.1 Sviluppo . . . . .	18
3.1.2 Compilazione e Flashing . . . . .	19

3.1.3	Testing e Monitoraggio	19
3.2	Struttura a Matrice	20
3.2.1	Scelte Implementative	20
3.2.2	Rilevamento dell'Input (Matrix Scan)	20
3.2.3	Gestione dell'Input	21
3.2.4	Considerazioni sull'Ottimizzazione	21
3.3	Correzione degli Errori: Debouncing	22
3.3.1	Tecnica di Debouncing Implementata	23
3.3.2	Implementazione Firmware	23
3.3.3	Vantaggi della Scelta	23
3.4	Gestione Asincrona delle Operazioni	23
3.4.1	Interrupt (GPTimer)	24
3.4.2	Deferred Work	25
3.5	Librerie e Plugin Utilizzati	25
3.5.1	TinyUSB	26
3.5.2	ESP-IDF	26
3.6	Protocolli di Codifica Stenografica	27
3.6.1	Scelta del Protocollo (Gemini PR vs TX Bolt)	27
3.6.2	Implementazione del Protocollo di Chording	28
3.7	Plover	29
3.7.1	Configurazione	29
3.7.2	Funzionalità e Personalizzazione	30
<b>4</b>	<b>Analisi e Valutazioni</b>	<b>30</b>
4.1	Potenza Dissipata	30
4.2	Prestazioni (Tempi di Esecuzione)	31
4.3	Analisi dei Costi	32
<b>5</b>	<b>Conclusioni e Sviluppi Futuri</b>	<b>32</b>

# 1. Introduzione

La stenografia moderna rappresenta una tecnologia avanzata per la trascrizione testuale in tempo reale ed è utilizzata principalmente da professionisti come stenografi giudiziari, sottotitolatori e interpreti per catturare il parlato con rapidità ed efficienza. Tuttavia la sua diffusione è limitata perché le tastiere stenografiche in commercio sono poche, costose e spesso basate su hardware e software proprietari che ne restringono personalizzazione, manutenzione e accessibilità.

Nonostante il suo impiego resti prevalentemente confinato a contesti professionali, la stenografia possiede un grande potenziale applicativo anche per studenti che prendono appunti, scrittori che necessitano di input rapido o programmatore che desiderano ottimizzare i tempi di lavoro con l'utilizzo di macro personalizzate.

Il progetto ***KHORD*** mira a superare queste limitazioni rendendo accessibile la stenografia a un pubblico più ampio attraverso la progettazione e la realizzazione di una tastiera stenografica a basso costo, open-source e personalizzabile.

Questa relazione descrive il processo di progettazione, di sviluppo e di testing della tastiera KHORD v1.0.0, illustrando le scelte tecniche, le innovazioni introdotte e i risultati ottenuti.

## 1.1 La Stenografia

A differenza della scrittura tradizionale in cui ogni tasto corrisponde a una singola lettera, la stenografia si basa su un **approccio fonetico** che utilizza **accordi** (o *chords*), ossia la **pressione simultanea di più tasti** per rappresentare sillabe, parole, singole lettere o comandi specifici. Questo metodo riduce significativamente il numero di movimenti necessari, consentendo velocità di scrittura superiori alle 200 parole al minuto, rispetto alle 60-80 parole al minuto delle tastiere convenzionali.

La complessità di questa tecnica richiede un dispositivo hardware capace di registrare con precisione un numero arbitrario di pressioni simultanee, garantendo affidabilità e prevenendo errori come il *ghosting* (registrazione di tasti non premuti) o limitazioni nel *key rollover*, cioè nel numero massimo di tasti che possono essere registrati contemporaneamente. In stenografia, dove un accordo può coinvolgere anche dieci o più tasti premuti nello stesso istante, queste limitazioni renderebbero il sistema inutilizzabile.

La disposizione dei tasti in una tastiera stenografica (fig. 3) è completamente diversa da quella QWERTY. La tastiera è divisa in **tre sezioni principali**:

- a sinistra si trovano le consonanti iniziali della sillaba o parola,
- a destra si trovano le consonanti finali,
- in basso ci sono i tasti delle vocali, usati per rappresentare i suoni vocalici della sillaba o parola.

Inoltre **alcune lettere**, come la *T*, sono ripetute su entrambi i lati per distinguere le consonanti iniziali da quelle finali; e **non tutte le lettere dell'alfabeto sono presenti**: la stenografia si concentra su suoni (fonemi) e quindi alcune lettere (come C, Q) sono rappresentate tramite combinazioni di altri tasti (K, KW). Ad esempio, la parola "cat" si scrive come "KAT" (con un accordo che combina "K", "A" e "T"), mentre "chord" si scrive come "KHORD", da cui deriva il nome della tastiera protagonista di questo progetto. Questa organizzazione consente di "accordare" una parola intera o una sua parte in un unico colpo, in un'unica pressione simultanea dei tasti.

Gli accordi possono rappresentare **interi parole** (come *the, dog, you, hi*), che in stenografia sono scritte con un solo accordo (*the* → T, *dog* → TKOG, *you* → U e *hi* → HEU).

Gli accordi possono anche rappresentare **elementi parziali** come prefissi, radici e suffissi. Ad esempio, la parola *restarting* può essere scritta componendo tre accordi distinti: RE per il prefisso, START per la radice e -G per il suffisso -ing, ovvero RE/START/-G. In molti casi, uno stenotipista esperto riesce a

digitare questi accordi in **rapida successione** o addirittura come un **unico accordo complesso**, che il software di stenografia riconosce e traduce automaticamente nella parola completa.

Questa struttura, pensata per l'efficienza fonetica, permette a uno stenotipista esperto di scrivere alla stessa velocità con cui una persona parla.

## 1.2 Le tastiere meccaniche

Per rispondere alle esigenze della stenografia, il progetto KHORD utilizza la tecnologia delle tastiere meccaniche, le quali si distinguono dalle tastiere a membrana tradizionali per la presenza di **interruttori meccanici individuali (switch)** sotto ogni tasto. A differenza delle tastiere a membrana, le tastiere meccaniche:

- hanno una **maggior durata** (fino a 50 milioni di pressioni per tasto),
- consentono una **pressione simultanea di più tasti (N-key rollover)**, essenziale per la stenografia,
- sono in grado di evitare il **ghosting**, ovvero l'errata registrazione di tasti non premuti (sez. 2.2.1),
- offrono una **varietà di interruttori** con caratteristiche diverse (*lineari, tattici, clicky*), adattabili alle preferenze dell'utente,
- sono più **riparabili e personalizzabili**, sia dal punto di vista hardware che software.

La flessibilità di questa tecnologia è ulteriormente arricchita da diversi firmware open-source, come ad esempio **QMK** (*Quantum Mechanical Keyboard*), un software per tastiere meccaniche personalizzate che permette di configurare layout, gestire la matrice di tasti e implementare funzionalità avanzate come macro e layer multipli.

La diffusione delle tastiere meccaniche personalizzate, unita alla crescente compatibilità con software e protocolli open source come QMK, Gemini PR (sez. 3.6) e Plover (sez. 3.7), ha reso la stenografia accessibile anche a utenti non professionisti.

## 1.3 Motivazioni e Obiettivi del progetto

Nella comunità delle tastiere meccaniche, firmware open-source come QMK sono apprezzati per la loro flessibilità, poiché la personalizzazione del comportamento della tastiera: dalla gestione della matrice alla definizione di macro complesse. Tuttavia, **né QMK né altri firmware open-source comuni supportano nativamente i microcontrollori della serie ESP32-S3**.

L'ESP32-S3 è un microcontrollore ad alte prestazioni, compatto e dotato di connettività Wi-Fi e Bluetooth integrata. Queste caratteristiche lo rendono ideale per lo sviluppo di dispositivi embedded evoluti, come tastiere stenografiche portatili e potenzialmente wireless.

A fronte di queste considerazioni, il progetto **KHORD** è nato con un duplice obiettivo:

- **progettare un firmware open-source dedicato all'ESP32-S3**, capace di gestire efficacemente la scansione della matrice di tasti, la correzione di errori e la trasmissione dei dati in formato compatibile con il software di stenografia *Plover*,
- **sviluppare una piattaforma hardware completa**, compatibile con l'ESP32-S3-DevKitC-1, con una disposizione dei tasti pensata specificamente per la digitazione stenografica a due mani

Il progetto mira a proporre uno strumento di scrittura stenografica più **accessibile, personalizzabile** e adatto alle esigenze di programmatore, studenti e appassionati di scrittura veloce. Il design a **basso costo** e la natura **open-source** di KHORD consentono a chiunque disponga di competenze tecniche di base e risorse minime di **replicare facilmente** la tastiera, promuovendo la diffusione della stenografia.

## 2. Progettazione Hardware

La progettazione hardware è stata realizzata con l'obiettivo di ottenere un dispositivo compatto e affidabile, sfruttando componenti facilmente reperibili e a basso costo. In questa sezione si illustrano le scelte progettuali effettuate in termini di componentistica, strategie anti-erri di input e sviluppo del PCB.

### 2.1 Componentistica

La tastiera è stata costruita impiegando componenti selezionati in base a requisiti di compatibilità con l'architettura progettata, affidabilità operativa e manutenibilità. Di seguito si elencano gli elementi utilizzati e il loro ruolo nel sistema

- **Scheda di sviluppo ESP32-S3-DevKitC-1-N8R8:** utilizzata per il controllo della tastiera e per la comunicazione via USB. Una descrizione più dettagliata della scheda e delle motivazioni che hanno portato alla sua scelta è presentata nella sez. 2.1.1
- **25 switch meccanici Cherry MX brown:** interruttori meccanici con feedback tattile, compatibili con lo standard Cherry MX.
- **25 diodi 1N4148:** inseriti in serie a ciascuno switch della matrice, impediscono fenomeni di ghosting e masking (descritti in sez. 2.2.1) durante la scansione
- **22 copritasti DSA da 1u e 3 copritasti DSA da 2u**
- **3 stabilizzatori da 2u:** compatibili con lo standard Cherry MX, sono montati sotto i tasti da 2u per garantire stabilità e risposta uniforme lungo tutta la superficie del tasto
- **13 viti M2 da 12 mm e 13 dadi:** garantiscono un montaggio semplice e robusto dei layer strutturali della tastiera (top e bottom plate e PCB)

#### 2.1.1 Scelta del Microcontrollore (MCU)

Per questo progetto è stata selezionata la scheda di sviluppo ESP32-S3-DevKitC-1-N8R8 [11], una piattaforma di prototipazione basata sul modulo ESP32-S3-WROOM-1-N8R8 [13].

Il modulo integra il System on a Chip (SoC) ESP32-S3, dotato di un microprocessore dual-core a 32 bit con architettura Xtensa<sup>®</sup> LX7, operante fino a 240 MHz. Include unità di accelerazione per applicazioni di AI e ML, oltre a connettività Wi-Fi e Bluetooth 5 Low Energy, non utilizzata in questo progetto ma utile per sviluppi futuri (sez. 5). La versione N8R8 del modulo offre 8 MB di flash Quad-SPI e 8 MB di PSRAM Octal-SPI.

Il SoC ESP32-S3 mette a disposizione fino a 45 GPIO, molti dei quali sono liberamente utilizzabili dall'utente, mentre altri sono riservati a funzioni di sistema (memorie, controller USB, debug JTAG). Il numero di GPIO è quindi sufficiente per controllare la tastiera a matrice 11×3 protagonista di questo progetto e, al tempo stesso, per lasciare margine per eventuali espansioni future, come ad esempio il collegamento di un display o di altri sensori e periferiche.

La scheda dispone di due interfacce USB

- Una porta USB nativa, collegata al controller USB OTG integrato nel SoC (GPIO 19/20), è utilizzata per la comunicazione USB diretta con il computer host
- Una porta USB-UART realizzata tramite chip bridge CP2102 on-board, dedicata a programmazione e debug via console seriale.

L'alimentazione avviene tramite porta USB o pin a 5 V, con regolazione interna a 3,3 V.

Grazie a queste caratteristiche, la ESP32-S3-DevKitC-1-N8R8 rappresenta una base solida e versatile,

sia per il progetto corrente, sia per l'evoluzione verso versioni più avanzate o funzionalità aggiuntive.

### 2.1.2 Through-hole vs SMD

Per i *diodi* impiegati nel circuito della tastiera KHORD, è stata scelta una versione con montaggio *through-hole*, in linea con l'approccio THT adottato per il resto della componentistica. La THT prevede l'inserimento dei pin dei componenti in fori sul circuito stampato (PCB), con successiva saldatura manuale sul lato opposto. Questo approccio, eseguibile con strumenti di base come un saldatore a punta fine, facilita l'assemblaggio, la sostituzione dei componenti e il debugging, risultando ideale per un progetto accademico con risorse limitate. Inoltre, i componenti THT offrono una robustezza meccanica superiore, adeguata alle sollecitazioni meccaniche tipiche di una tastiera.

La tecnologia *Surface Mount Device* (SMD), al contrario, consente il montaggio di componenti su entrambe le superfici del PCB, grazie alle loro dimensioni compatte, ottimizzando la densità circuitale e migliorando l'estetica del prodotto. Tuttavia, il processo di assemblaggio SMD richiede l'applicazione di una pasta saldante tramite uno stencil e il posizionamento preciso dei componenti. La saldatura può essere eseguita in forni a rifusione, che sciolgono la pasta per connessioni stabili, o manualmente con aria calda o saldatore a punta fine, ma tali processi aumentano la difficoltà e i costi rispetto alla THT.

La scelta della THT ha quindi privilegiato semplicità, affidabilità e funzionalità.

## 2.2 Gestione Affidabile dell'Input

Per garantire un'esperienza d'uso affidabile, sono state implementate soluzioni hardware per prevenire errori di input, come *ghosting* e *masking*, e per supportare il *N-Key Rollover* (NKRO).

Queste strategie assicurano la **corretta rilevazione di pressioni simultanee dei tasti**, essenziale per la stenografia, dove più tasti vengono premuti contemporaneamente per formare sillabe o parole.

### 2.2.1 Ghosting e Blocking

Il *ghosting* è un problema comune nelle tastiere a matrice prive di diodi, in cui la pressione simultanea di più tasti può creare percorsi elettrici indesiderati, causando la rilevazione errata di tasti non premuti da parte del microcontrollore.

A titolo di esempio, per spiegare il fenomeno, si consideri una matrice  $3 \times 3$ , con colonne  $(0, 1, 2)$  e righe  $(0, 1, 2)$ , come mostrato in fig. 1a. Si supponga che siano premuti simultaneamente i tasti in posizione  $(1,0)$ ,  $(1,1)$  e  $(2,1)$ , corrispondenti a *A*, *S* e *X*. Durante la scansione, il microcontrollore attiva la colonna 0 e verifica quali righe risultano collegate tramite i tasti premuti. La corrente scorre attraverso il tasto *A*, raggiungendo la riga 1. Da qui, tramite il tasto *S*, arriva alla colonna 1 e, attraverso il tasto *X*, raggiunge la riga 2. Questo genera un percorso elettrico che coinvolge anche il tasto *Z* in posizione  $(2,0)$ , non premuto. A causa di questo flusso, il microcontrollore interpreta erroneamente le coordinate di *Z* come attive, registrando *Z* come premuto: questo fenomeno è noto come *ghosting*.

Per risolvere il problema, si utilizza la tecnica del *blocking*, illustrata in fig. 1b. Aggiungendo diodi in serie a ogni tasto, orientati per consentire il flusso di corrente solo dalla colonna alla riga, si impediscono percorsi inversi. In questo modo, il percorso che causava la falsa attivazione di *Z* viene bloccato, e il microcontrollore rileva solo i tasti effettivamente premuti.

Per eliminare il problema del ghosting, nella progettazione di KHORD sono quindi stati inseriti diodi 1N4148 in serie a ciascun interruttore della matrice  $11 \times 3$ , come mostrato in fig. 2.

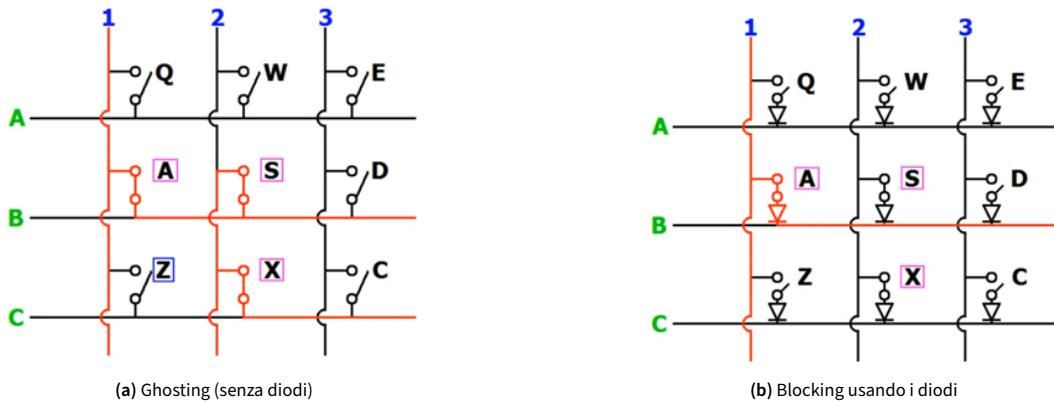


Fig. 1. Problema di ghosting risolto con tecnica di blocking

### 2.2.2 N-Key Rollover

La tecnica di blocking, descritta nella sezione precedente (fig. 1b), è fondamentale anche per implementare il **N-Key Rollover (NKRO)**, una funzionalità hardware che consente la rilevazione simultanea di un numero arbitrario di tasti premuti, senza interferenze o limitazioni.

I diodi 1N4148, ciascuno in serie a ogni tasto, non solo prevengono i percorsi elettrici indesiderati responsabili del ghosting, ma garantiscono anche che ogni pressione di tasto venga registrata correttamente, indipendentemente dal numero di tasti premuti contemporaneamente.

Il NKRO è essenziale in questo progetto, perché la pressione simultanea di più tasti costituisce il nucleo della tecnica stenografica, che richiede un input costantemente preciso e affidabile.

### 2.3 Progettazione del Circuito Stampato (PCB)

Il design del circuito stampato (PCB) per la tastiera stenografica KHORD è stato sviluppato con l'obiettivo di creare un dispositivo compatto, affidabile e facilmente assemblabile, ed è stato progettato per garantire la compatibilità con i componenti selezionati ed elencati nella sez. 2.1.

Il processo di progettazione è stato realizzato utilizzando **KiCad v9.0 [17]**, un software open-source per l'automazione della progettazione elettronica (EDA), il quale ha permesso la creazione dello schema elettrico, del layout del PCB e la visualizzazione tridimensionale del dispositivo.

#### 2.3.1 Schema Elettrico

Lo schema elettrico della tastiera KHORD, mostrato in fig. 2, implementa una matrice  $11 \times 3$  controllata dalla scheda ESP32-S3-DEVKITC-1-N8R8. Questa matrice è composta da 25 interruttori Cherry MX e diodi 1N4148, organizzati in 11 colonne e 3 righe, con gli interruttori collegati verticalmente per formare le colonne e i diodi collegati orizzontalmente per costituire le righe. Usando una matrice di righe e colonne, piuttosto che utilizzare un GPIO dedicato a ogni tasto, si riduce il numero di pin necessari a 14 GPIO, ottimizzando così l'uso dei pin dell'ESP32-S3 rispetto ai 25 richiesti da un collegamento diretto.

Ogni interruttore è connesso in serie con un diodo, orientato per consentire il flusso di corrente esclusivamente dalle colonne alle righe, prevenendo ghosting e masking e supportando il N-Key Rollover (NKRO), come descritto in sez. 2.2.

Lo schema include anche i simboli dei 3 stabilizzatori e dei 26 fori di montaggio, presenti come riferimenti strutturali ma non collegati a nessuna rete elettrica.

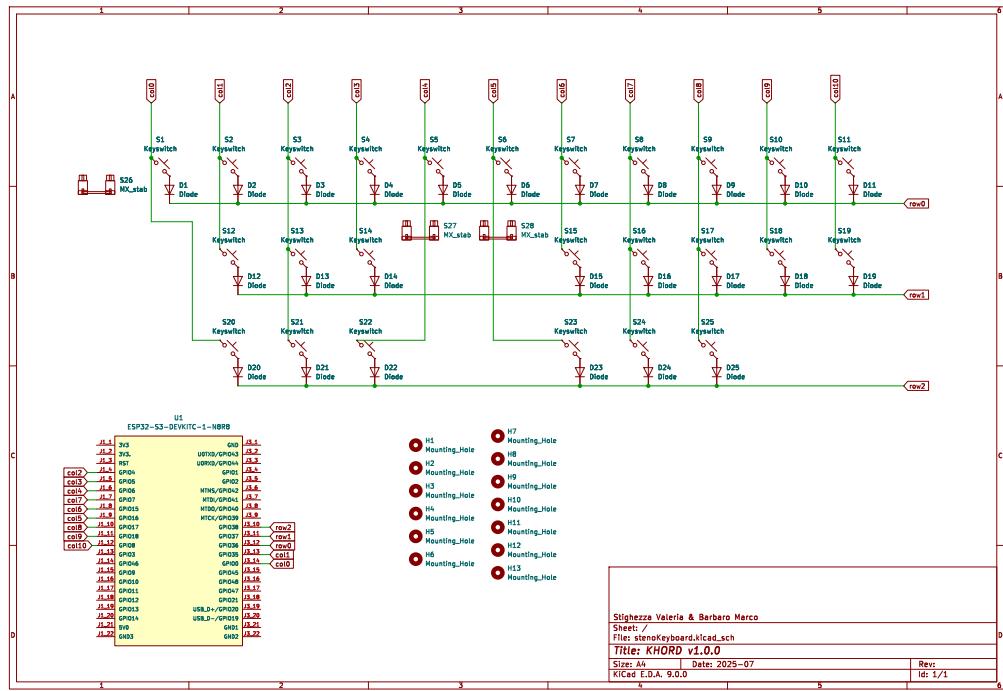


Fig. 2. Schema elettrico della tastiera stenografica KWORD

### 2.3.2 PCB Layout

Il layout del circuito stampato è stato progettato rispettando fedelmente i collegamenti definiti nello schema elettrico.

La disposizione degli interruttori e dei relativi diodi è stata definita a partire da un layout personalizzato (fig. 3), progettato con lo strumento online *Keyboard Layout Editor* [14]. Il file JSON generato da questo editor è stato poi utilizzato dal plugin *KLE Placer* [5] per posizionare automaticamente le impronte degli interruttori sul PCB alle coordinate corrette, garantendo così una corrispondenza precisa tra il progetto digitale e la realizzazione fisica.

Gli interruttori, gli stabilizzatori, la scheda ESP32 e i fori di montaggio sono stati collocati sullo strato frontale (F.Cu, fig. 5), mentre i diodi sono stati posizionati sullo strato posteriore (B.Cu, fig. 6) per evitare sovrapposizioni e interferenze e per ottimizzare lo spazio. In fig. 4 è possibile osservare tutti gli strati contemporaneamente, riflettendo l'approccio adottato durante la fase di progettazione.

Il **routing** è stato strutturato per mantenere il più possibile i collegamenti delle 11 colonne sul lato posteriore (B.Cu) e quelli delle 3 righe sul lato frontale (F.Cu), migliorando l'organizzazione ed evitando intersezioni tra righe e colonne. Tuttavia, poiché lo strato frontale risultava meno congestionato, le colonne 1 e 8 sono state instradate direttamente su F.Cu per ottimizzare l'uso dello spazio disponibile. Nei punti in cui le intersezioni tra tracce sullo stesso strato erano inevitabili (tra le colonne 1 e 4 e all'interno del gruppo di colonne 5, 6 e 7) sono state impiegate *vias*, ovvero fori metallizzati che permettono di trasferire le tracce sul lato opposto della scheda, consentendo di oltrepassare l'incrocio e, se necessario, ritornare al layer originario.

Le tracce, con larghezza di 0.5 mm, mantengono una distanza minima di 1 mm tra loro per garantire

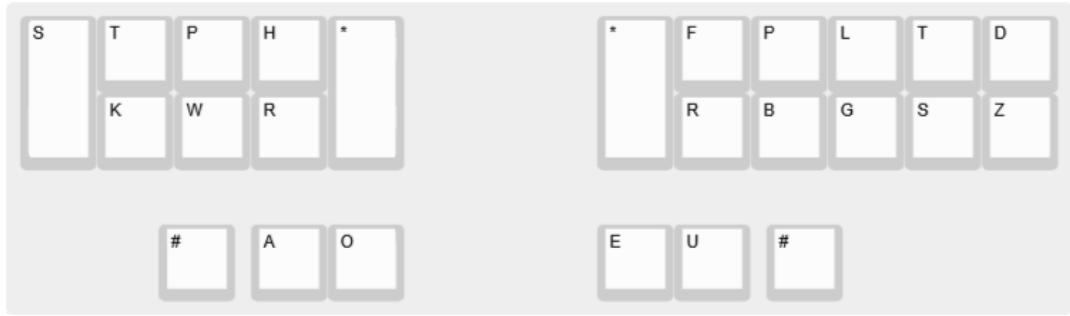


Fig. 3. Layout della tastiera stenografica KHORD

isolamento, e anche le vias, con diametro esterno 0,8 mm e foro da 0,4 mm, sono distanziate di almeno 1 mm per prevenire cortocircuiti.

Durante il routing sono stati evitati angoli di 90°, poiché questi possono provocare accumuli di carica elettrica, ostacolare la fabbricazione (soprattutto durante l'etching, ovvero l'incisione chimica del rame) e creare irregolarità nel flusso del segnale, alterandone la qualità e causando errori di trasmissione o perdita di dati.

Sono infine stati definiti gli aspetti fisici e grafici. Il **perimetro** della scheda è stato tracciato sullo strato Edge.Cuts, che ne determina la sagoma finale. Sullo strato di serigrafia (F.SilkScreen), utilizzato per riportare testi, simboli e riferimenti visivi direttamente sul PCB, è stato aggiunto anche il **logo**, il quale unisce le lettere *S* e *B*, ovvero iniziali dei nomi dei creatori della tastiera KHORD. Per l'inserimento del logo, l'immagine è stata prima convertita in formato bitmap e successivamente importata nel software EDA come un'impronta (footprint) personalizzata.

I fori di montaggio per viti M2, con un diametro di 2.2 mm, sono stati scelti per garantire assemblaggio solido e stabile della tastiera e dei suoi strati strutturali: top plate, PCB e bottom plate.

Il risultato dell'intero processo appena descritto è mostrato in fig. 4

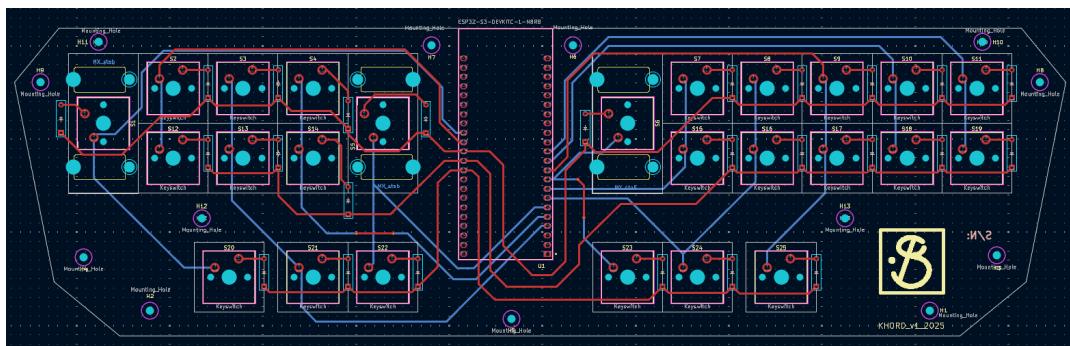
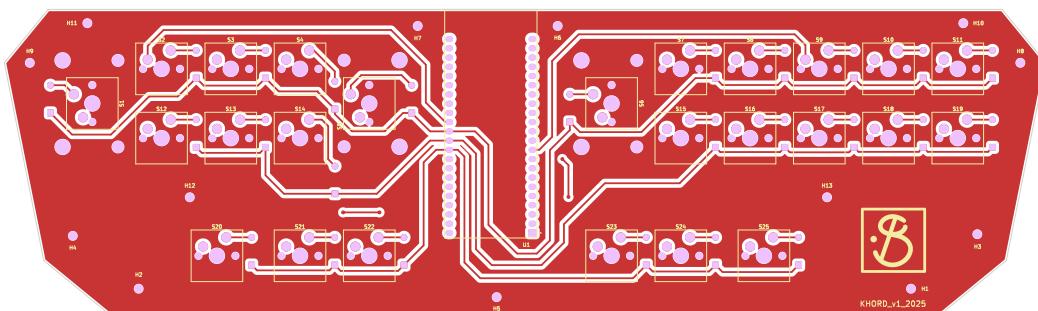


Fig. 4. PCB layout della tastiera stenografica KHORD (tutti gli strati)

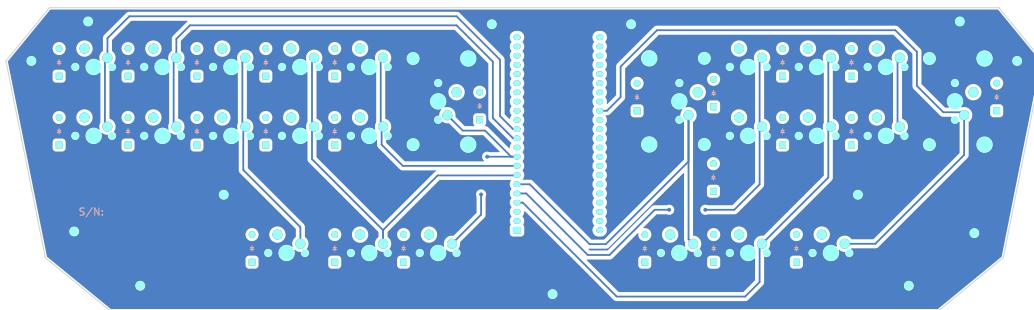
Al termine della progettazione del layout, è stata utilizzata in KiCad la funzione Draw Fill Zones per aggiungere zone di rame sugli strati anteriore (F.Cu) e posteriore (B.Cu) del PCB. Queste zone sono state configurate come <no net>, cioè non sono collegate elettricamente ad alcuna rete del circuito, e sono quindi **elettricamente isolate** dal resto. In questo modo, l'intera superficie del PCB viene quasi completamente ricoperta di rame: le piste, che erano già in rame, costituiscono i collegamenti elettrici tra i componenti, e le aree libere del circuito vengono coperte da zone di rame che non sono collegate ad alcuna rete. Tra le piste e queste zone di rame viene lasciato uno spazio vuoto, chiamato *bordo isolante*, che garantisce che non vi sia contatto elettrico tra le due.

Sebbene non siano collegate al circuito, le zone di riempimento svolgono un ruolo funzionale rilevante: **migliorano la dissipazione del calore** grazie alla maggiore superficie metallica, **riducono il rumore elettrico** dovuto a disturbi esterni e **aumentano la robustezza meccanica del PCB**, migliorando la distribuzione dei materiali e riducendo le deformazioni.

Il risultato di tale procedura è mostrato in fig. 5 e 6.



**Fig. 5.** PCB layout della tastiera stenografica KHORD (strato anteriore, con zone di rame isolate)



**Fig. 6.** PCB layout della tastiera stenografica KHORD (strato posteriore, con zone di rame isolate)

### 2.3.3 Modello 3D

Il modello 3D del PCB è stato generato utilizzando il visualizzatore 3D di KiCad. Questo modello offre una rappresentazione completa della struttura della tastiera.

Una funzionalità chiave del visualizzatore consente di attivare o disattivare la visualizzazione dei modelli 3D dei singoli componenti, facilitando la verifica del posizionamento e della compatibilità durante la fase di progettazione. In fig. 7 e fig. 8 sono illustrate due viste del modello 3D che mostrano esclusivamente il PCB senza i componenti, mentre in fig. 9, fig. 10 e fig. 11 sono presentate tre viste che includono anche i modelli 3D dei componenti derivati dalle librerie specificate nella sez. 2.5, evidenziando l'assemblaggio completo.

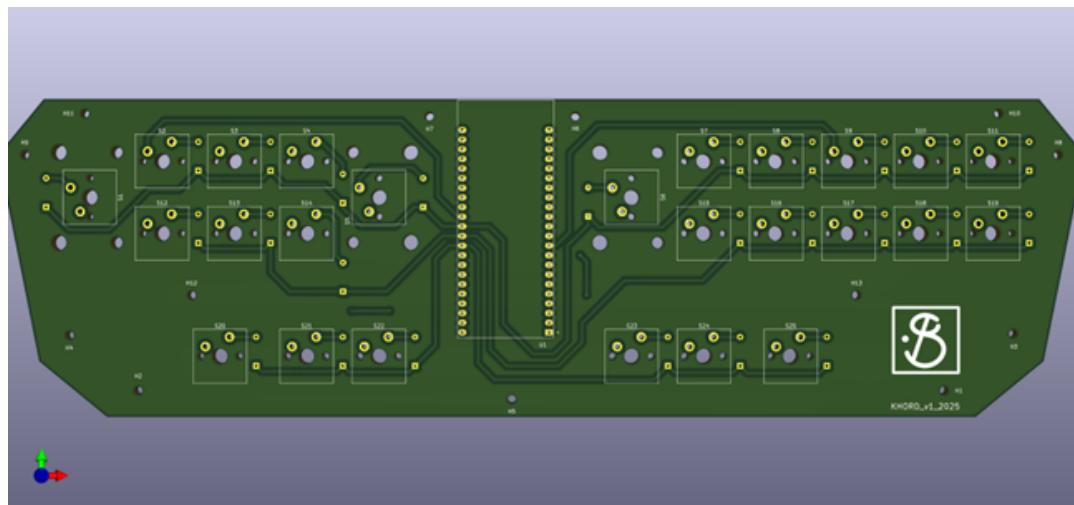


Fig. 7. Modello 3D del PCB della tastiera stenografica KHOORD (strato frontale)

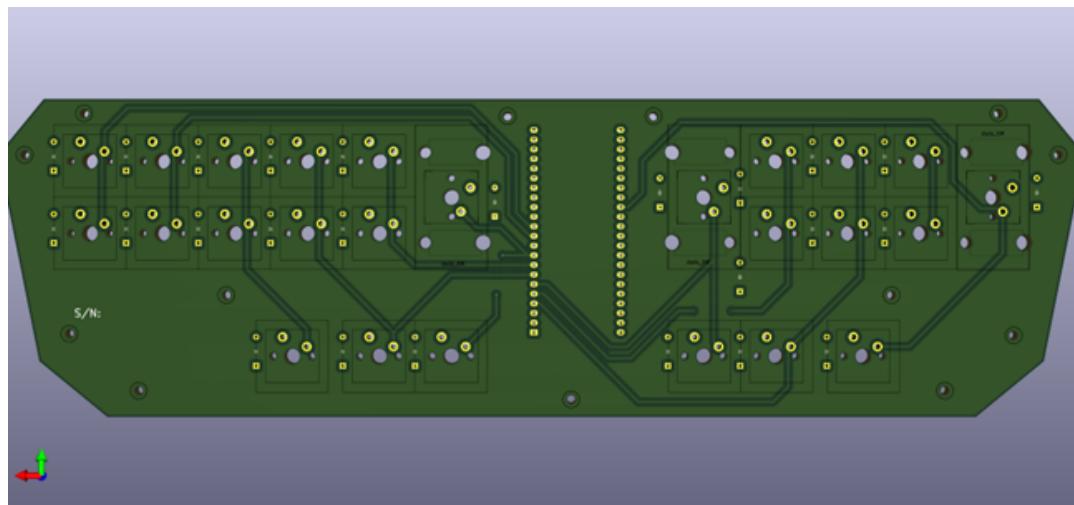


Fig. 8. Modello 3D del PCB della tastiera stenografica KHOORD (strato posteriore)

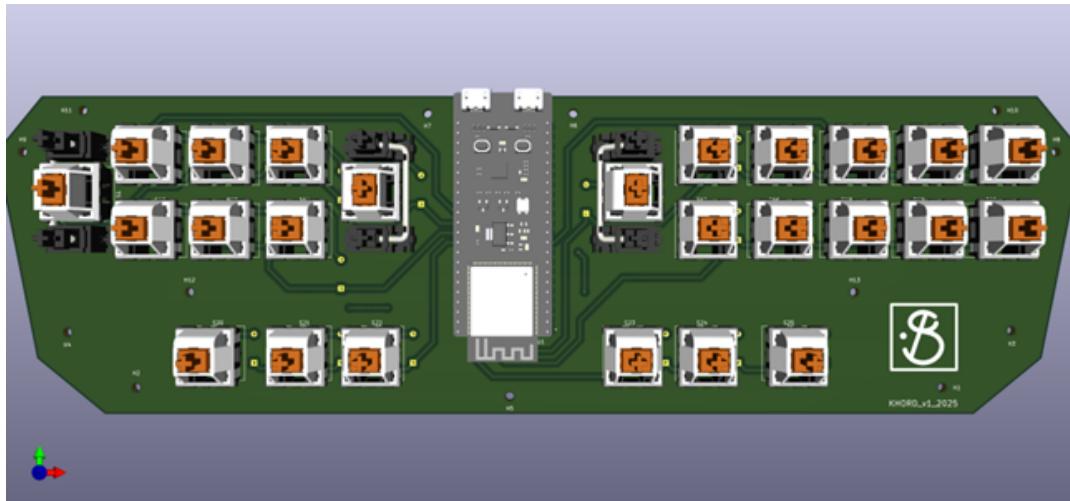


Fig. 9. Modello 3D del PCB della tastiera stenografica KHORD (strato frontale, con modelli 3D dei componenti)

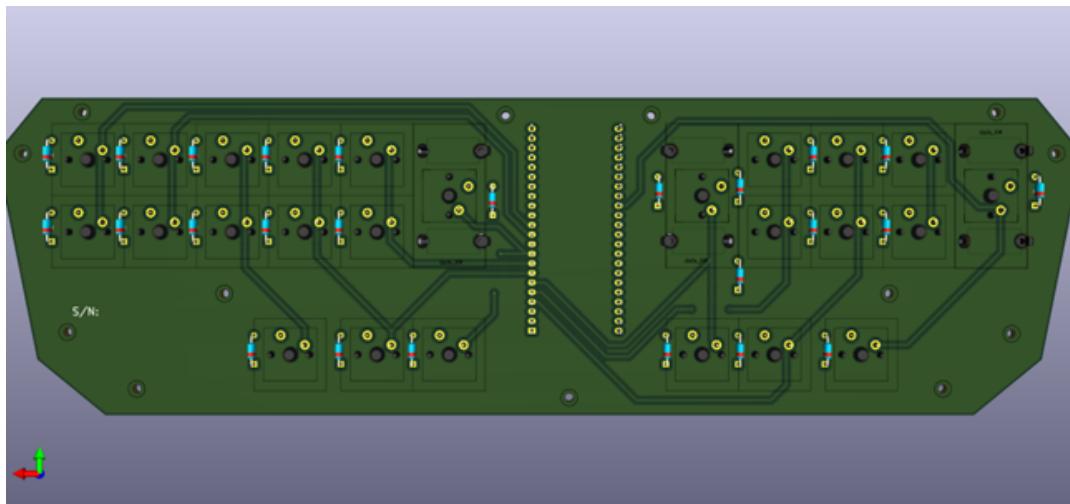


Fig. 10. Modello 3D del PCB della tastiera stenografica KHORD (strato posteriore, con modelli 3D dei componenti)

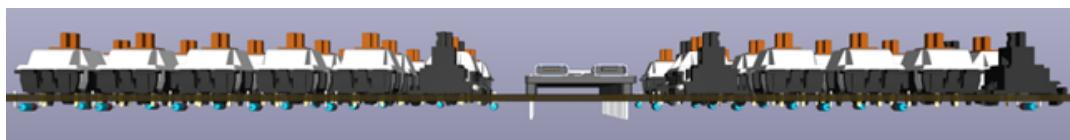


Fig. 11. Modello 3D del PCB della tastiera stenografica KHORD (visione laterale, con modelli 3D dei componenti)

### 2.3.4 Prototipo per il Testing del Firmware

In attesa della fabbricazione e della consegna del PCB è stato realizzato un prototipo (fig. 12) che, composto da una matrice 3x2, ha replicato in scala ridotta l'architettura elettrica fondamentale della tastiera, consentendo lo sviluppo, il testing e il debug del firmware, dettagliati nella sez. 3, verificando la corretta gestione dei segnali GPIO e il supporto al N-Key Rollover (NKRO) prima dell'assemblaggio definitivo.

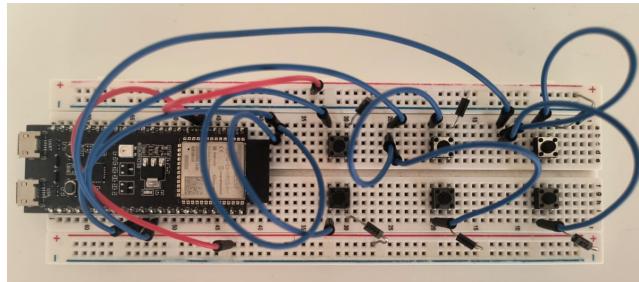


Fig. 12. Prototipo hardware per il testing e il debug del firmware

## 2.4 Assemblaggio

In questa sezione sono presentate le fasi di assemblaggio della tastiera KHORD, che includono la produzione del PCB, la saldatura dei componenti, la modellazione e stampa 3D del case e il montaggio finale.

### 2.4.1 Produzione del PCB

La produzione del PCB è stata affidata all'azienda *NextPCB* [19], a cui sono stati forniti i file Gerber generati tramite il plugin Fabrication Toolkit di KiCad descritto in sez. 2.5.

I file Gerber, un formato vettoriale standard, catturano con precisione le geometrie del circuito (tracce, maschere, serigrafie...) tramite coordinate matematiche. Questi file sono essenziali per l'azienda di produzione, poiché guidano le macchine nelle fasi di incisione, foratura e rivestimento e assicurano che il progetto venga riprodotto fedelmente durante la fabbricazione.

I parametri specifici del PCB ordinato presso NextPCB sono i seguenti:

- **Material Type:** *FR-4*, un materiale standard per PCB, a base di resina epoxidica rinforzata con fibra di vetro, noto per la sua robustezza e resistenza termica
- **2 Layer:** il PCB è composto da due strati di rame (frontale F . Cu e posteriore B . Cu), ottimizzando lo spazio e la complessità del routing
- **Size:** *286.6 × 83.8 mm*. Le dimensioni sono state rilevate automaticamente dal file Gerber
- **Quantity:** *5 pcs* (minimo)
- **PCB Thickness:** *1.6 mm*. È uno spessore standard che garantisce una buona rigidità meccanica e facilità di montaggio
- **TG Rating:** *TG130*. Indica una temperatura di transizione vetrosa di  $130^{\circ}\text{C}$ , ovvero la soglia oltre la quale il materiale FR-4 inizia a deformarsi
- **Outer Copper Weight:** *1 oz*. Indica il peso del rame esterno pari a 1 oncia per piede quadrato (circa  $35\mu\text{m}$ ), sufficiente per la conducibilità richiesta dal circuito

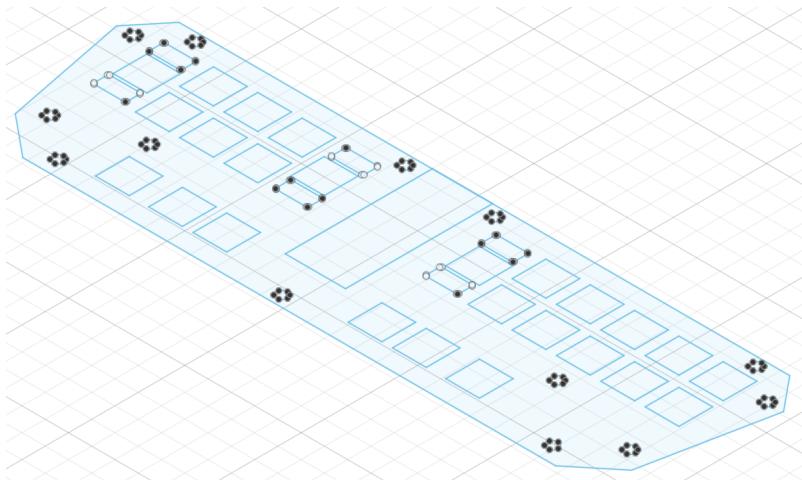
- **Trace Width/Spacing:** *10/10 mil* ↑. Indica la larghezza minima delle tracce e la distanza minima tra di esse pari a 10 mil (0,254 mm)
- **Min. Drill Hole:** *0.3 mm* ↑. Indica il diametro minimo delle vias
- **Via Processing:** *Tented Vias*, ovvero le vias sono coperte da maschera di saldatura su entrambi i lati. Ciò le protegge da cortocircuiti e dall'ossidazione
- **Surface Finish:** *HASL*. Il trattamento a stagno-piombo tramite *Hot Air Solder Leveling (HASL)* protegge il rame esposto dall'ossidazione e ne facilita la saldatura manuale. È una delle opzioni più economiche ed è stata scelta proprio per il suo buon rapporto qualità/prezzo
- **Solder Mask Color:** *Green*
- **Silkscreen:** *White*

Questi parametri hanno assicurato la fabbricazione accurata del PCB rispettando le specifiche del design.

#### 2.4.2 Modellazione e Stampa 3D del Case

Per racchiudere e proteggere i componenti elettronici, fornire stabilità strutturale e conferire al dispositivo un'estetica finita, è stato progettato e realizzato un case su misura tramite modellazione e stampa 3D.

Il processo è iniziato esportando da KiCad il layout finale del PCB nel formato **SVG (Scalable Vector Graphics)**. Questo file è stato successivamente importato in **Inkscape v.1.4.2** [15], dove è stato convertito nel formato interscambio 2D **DXF (Drawing Exchange Format)**. Il file DXF (fig. 13) ha rappresentato la base di partenza fondamentale, poiché conteneva le posizioni esatte di tutti i componenti, inclusi gli switch e la porta USB.



**Fig. 13.** File DXF di partenza per la modellazione 3D del case in Fusion 360

Il file DXF è stato quindi importato in **Autodesk Fusion 360 v.2602.1.25** [3], un software di modellazione CAD 3D. Qui, la progettazione del case è proseguita attraverso i seguenti passaggi. I fori per gli switch sono stati allargati (*offset*) a 14 mm per garantire il montaggio degli interruttori. È stata poi modellata un'apertura in corrispondenza della porta USB della scheda ESP32-S3-DevKitC-1-N8R8. Lo spessore del case durante il processo di estrusione è stato impostato a 1.6 mm.

Il risultato finale del case pronto per la stampa è mostrato in fig. 14.

Una volta completato il modello 3D, è stato esportato nel formato standard per la prototipazione rapida **STL** (*Standard Tessellation Language*).

La stampa è stata eseguita presso **ABprint3D** [1] con una stampante 3D a tecnologia **FDM Delta** (*Fused Deposition Modeling*) con layer height di 0.2 mm, utilizzando come materiale il **PLA** (*Acido Polilattico*) per la sua facilità di stampa, buona rigidità e basso impatto ambientale. Il risultato è un case robusto e funzionale, perfettamente compatibile con l'elettronica progettata.

Con il case prodotto, tutto il materiale necessario per l'assemblaggio della tastiera era pronto, come mostrato in fig. 19.

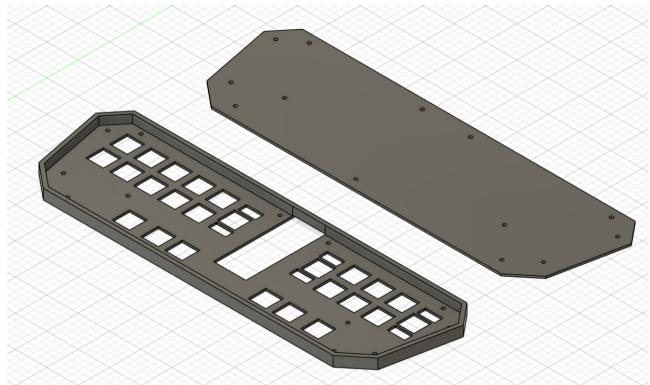


Fig. 14. Modello 3D del case completo, pronto per la stampa 3D

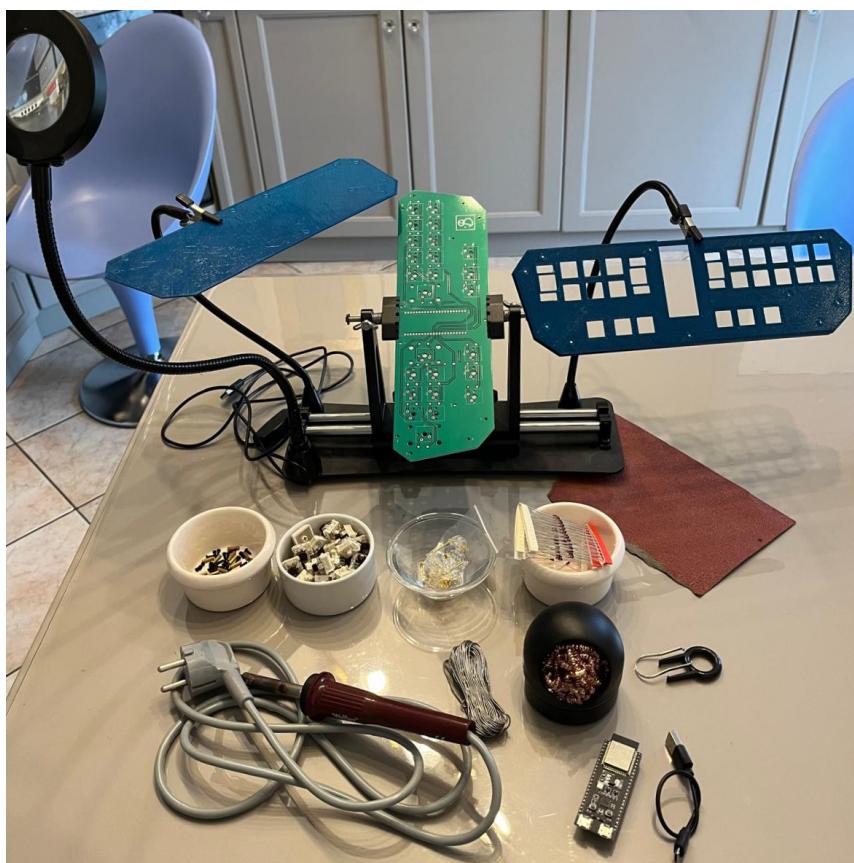


Fig. 15. Elementi necessari al montaggio finale

Il primo passo è stato levigare i bordi interni del case con carta vetrata, per assicurare un adattamento preciso del PCB e dei componenti. Questa operazione è necessaria poiché, nella stampa 3D FDM, le variabili legate all'estruzione (come *temperatura, flusso, ritrazione e velocità*) sono numerose e difficili da controllare con assoluta precisione. Di conseguenza, è improbabile che le dimensioni finali della stampa corrispondano esattamente al modello CAD, soprattutto con tolleranze inferiori al millimetro.

Successivamente, gli interruttori Cherry MX Brown sono stati incaricati nei fori predisposti del top case stampato in 3D (fig. 16). Il top case con gli interruttori è stato poi posizionato sul PCB, inserendo gli switch nei corrispondenti fori della matrice 3x11, e successivamente saldati, come mostrato nella prossima sezione.

#### 2.4.3 Saldatura dei Componenti

La saldatura dei componenti è stata eseguita manualmente utilizzando la tecnologia

**Through-Hole (THT).** Gli interruttori Cherry MX e la scheda ESP32-S3-DEVKITC-1-N8R8 sono stati saldati sullo strato frontale (F.Cu), mentre i 25 diodi 1N4148 sono stati posizionati e saldati sullo strato posteriore (B.Cu). I 3 stabilizzatori sono stati avvitati sullo strato frontale (fig. 17b).

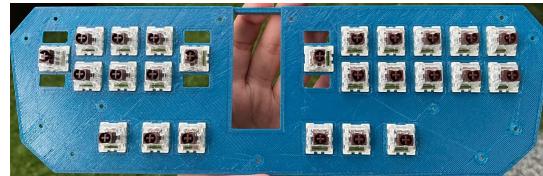
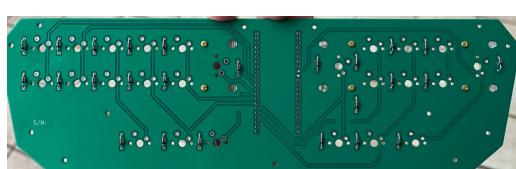
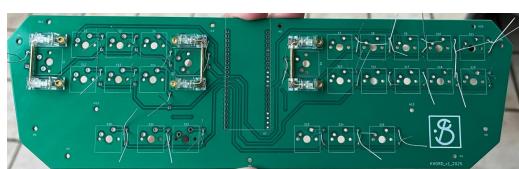


Fig. 16. Interruttori incaricati nel case



(a) Posizionamento corretto dei diodi



(b) Avvitamento stabilizzatori

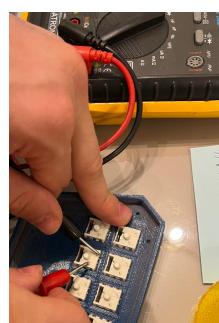
Fig. 17. Posizionamento diodi e montaggio stabilizzatori

La saldatura (fig. 18c e fig. 18d) è stata realizzata con un **saldatore a punta fine**, prestando attenzione alla **corretta polarità** dei diodi (fig. 17a), garantendo la funzionalità della matrice.

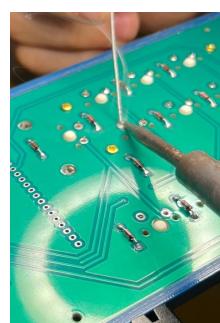
La robustezza delle connessioni, sia tra i vari componenti sia all'interno dei singoli, è stata testata tramite l'utilizzo di un **multimetro** (fig. 18a e fig. 18b).



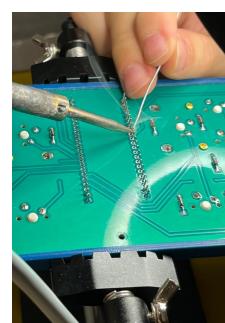
(a) Verifica della conduzione diretta dei diodi



(b) Verifica della continuità degli interruttori



(c) Saldatura degli interruttori e dei diodi



(d) Saldatura della scheda ESP32

Fig. 18. Processo di saldatura avvenuto dopo testing dei componenti tramite multimetro

#### 2.4.4 Montaggio Finale della Tastiera

A questo punto del processo di assemblaggio tutti i componenti principali di KHORD sono pronti: gli interruttori Cherry MX Brown e gli stabilizzatori sono incastri nel top case e saldati sul PCB insieme ai diodi 1N4148 e alla scheda ESP32-S3-DevKitC-1-N8R8.

Il montaggio finale consiste nell'allineare e avvitare il top case (contenente il PCB come descritto sopra) e il bottom case. Le viti (diametro 2 mm) sono quindi inserite nei fori di montaggio (diametro 2.2 mm) e fissate con dadi per garantire un assemblaggio robusto. Il risultato finale è mostrato in fig. ??



Fig. 19. Tastiera fisica assemblata correttamente e terminata

Dopo il montaggio, la tastiera è stata **testata** per verificare la stabilità meccanica e la funzionalità elettrica, confermando il corretto funzionamento dei componenti, dei circuiti, del firmware e della comunicazione USB.

#### 2.5 Librerie e Plugin Utilizzati

Per la progettazione del PCB con KiCad sono state impiegate diverse librerie per reperire *simboli* (per lo schema elettrico), *impronte* (per il layout PCB) e *modelli 3D* (per la visualizzazione 3D) dei componenti utilizzati. Di seguito sono elencate le relative fonti:

- **Diodi 1N4148 e interruttori Cherry MX:** i rispettivi simboli, le impronte e i modelli 3D, salvati come D0-35 e MX\_PCB\_1.00u, sono stati ottenuti dal repository GitHub ScottoKeebs di Joe Scotto, nella cartella Extras/ScottoKicad [16].
- **Stabilizzatori 2u:** sono stati utilizzati il simbolo, l'impronta e il modello 3D STAB\_MX\_2u, reperibili dal repository GitHub Marbastlib [18].
- **Scheda Microcontrollore ESP32-S3-DEVKITC-1-N8R8:** il simbolo, l'impronta e il modello 3D per la scheda sono stati scaricati da SnapEDA [12]. L'impronta scaricata è stato poi modificata in fase di progettazione, nella quale sono stati allargati orizzontalmente i pad per semplificare la saldatura manuale, una scelta coerente con l'approccio THT.
- **Fori di montaggio per viti M2:** sono stati utilizzati il simbolo, l'impronta e il modello 3D MountingHole\_2.2mm\_M2, reperibili dalle librerie ufficiali di KiCad.

In aggiunta alle librerie, sono stati utilizzati due plugin (già citati in precedenza) per semplificare e ottimizzare il processo di progettazione:

- **KLE Placer [5]:** questo plugin è stato utilizzato per posizionare automaticamente le impronte degli interruttori alle coordinate corrette sul PCB, seguendo fedelmente il layout (fig. 3) definito

nel file JSON generato con il tool online *Keyboard Layout Editor* [14], il quale ha consentito di progettare con precisione e in modo personalizzato la disposizione dei tasti.

- **Fabrication Toolkit** [4]: questo plugin integrato in KiCad è stato utilizzato per generare i file Gerber del PCB, i quali sono essenziali per la produzione del circuito stampato poiché contengono le informazioni necessarie per la fabbricazione dei layer di rame, delle maschere di saldatura e delle serigrafie. Il plugin ha semplificato il processo di esportazione e ha assicurato la conformità agli standard di produzione.

### 3. Progettazione Firmware

Il firmware della macchina stenografica *KHORD* è stato interamente sviluppato in linguaggio C, sfruttando l'*Integrated Development Framework* ufficiale del produttore della scheda: ***ESP-IDF*** [7] versione 5.4.2. Tale scelta ha consentito una gestione diretta delle periferiche hardware, della comunicazione USB nativa e della logica asincrona, garantendo così una soluzione efficiente e facilmente estensibile. L'obiettivo è stato di ridurre l'uso della HAL (*Hardware Abstraction Language*) offerta da ESP al minimo in modo che in futuro sia il più semplice possibile passare ad una versione "Bare Metal" del firmware.

Lo sviluppo si è ispirato alle logiche di funzionamento del progetto open-source internazionale di riferimento per la integrazione di tastiere custom: ***QMK*** [27], ma il codice è stato interamente scritto da zero, al fine di ottenere un controllo totale sull'ottimizzazione e sulle specifiche funzionalità richieste dal protocollo *GeminiPR* (sez. 3.6).

La comunicazione seriale via USB è stata realizzata utilizzando la libreria open-source ***tinyUSB*** [29] integrata nel framework *ESP-IDF* tramite il componente gestito ***esp-tinyusb*** [8], consentendo l'esposizione di una porta seriale virtuale conforme alla classe *CDC-ACM*.

In questa sezione vengono descritte le principali componenti firmware, organizzate secondo le aree funzionali di sviluppo.

#### 3.1 Pipeline di Sviluppo

La pipeline di sviluppo adottata prevede l'utilizzo di strumenti open-source e multipiattaforma, con particolare attenzione alla rapidità di iterazione dei prototipi e al monitoraggio real-time.

##### 3.1.1 Sviluppo

L'ambiente di sviluppo integrato utilizzato è ***Visual Studio Code*** [31], scelto per la sua integrazione con il framework *ESP-IDF* e per la disponibilità di estensioni di debugging e flashing.

Per agevolare l'esperienza di sviluppo si è fatto uso di diverse estensioni offerte dal marketplace di Visual Studio:

- **C/C++**: estensione ufficiale Microsoft per il supporto al linguaggio C/C++, utilizzata per l'autocompletamento, la navigazione del codice, il controllo degli errori in tempo reale e il debugging direttamente da Visual Studio Code.
- **CMake Tools**: estensione necessaria per la gestione e la configurazione di progetti basati su *CMake*, utilizzata per facilitare la configurazione dei target di build e la gestione degli ambienti di compilazione del framework *ESP-IDF*.
- **VS Code Counter**: strumento utilizzato per l'analisi quantitativa del codice sorgente, utile per verificare le metriche di progetto come il numero di righe di codice e il bilanciamento tra file sorgente e file di intestazione.

### 3.1.2 Compilazione e Flashing

La compilazione e il flashing del firmware sul processore *ESP32-S3-WROOM-1* avvengono tramite il tool ufficiale `idf.py`, incluso nel framework ESP-IDF. Le operazioni principali eseguite in ambiente CLI sono:

- `idf.py set-target ESP32-S3` per specificare la piattaforma di sviluppo
- `idf.py build` per la compilazione del progetto.
- `idf.py flash -p [COM-port]` per il caricamento del firmware specificando la virtual port a cui è connesso il microcontrollore.
- `idf.py monitor -p [COM-port]` per il monitoraggio dell'output seriale.

### 3.1.3 Testing e Monitoraggio

Il monitoraggio dell'output e il testing della comunicazione USB sono stati condotti utilizzando:

- Terminale *PuTTY* [26] per la ricezione e verifica dei dati trasmessi via porta seriale virtuale tramite protocollo *UART* (*Universal Asynchronous Receiver-Transmitter*) .
- Software *Plover* [24] (sez. 3.7), configurato in modalità *GeminiPR*, per la trascrizione in tempo reale dei dati stenografici.

L'intero processo di sviluppo, compilazione e test si inserisce nella configurazione illustrata in fig. 20 (*placeholder* per eventuale diagramma).

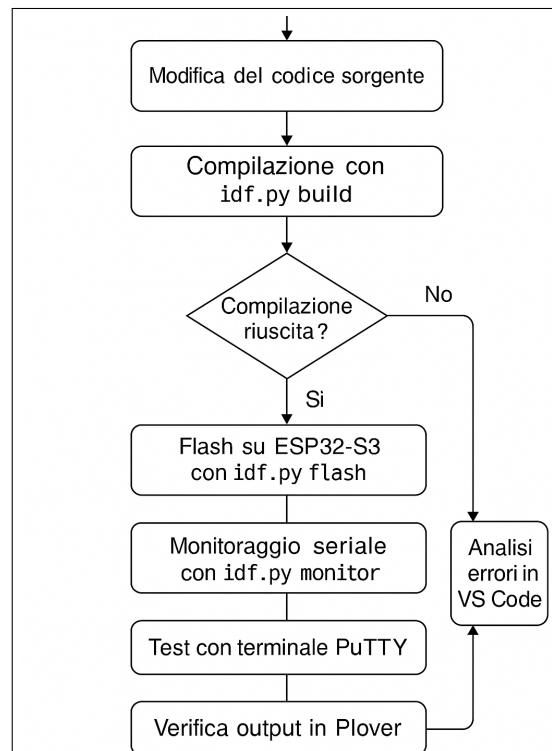


Fig. 20. Pipeline di sviluppo firmware

### 3.2 Struttura a Matrice

Il rilevamento degli input nella macchina stenografica *KHORD* si basa su una matrice di switch organizzata fisicamente su **3 righe x 11 colonne**. Tale configurazione è tipica delle tastiere meccaniche.

Non tutte le celle della matrice risultano attivamente impiegate, poiché nella macchina alcuni tasti di dimensione doppia (come il tasto *S*) occupano fisicamente più spazio ma sono gestiti elettricamente come un singolo switch. Le celle inutilizzate vengono quindi mappate su un'operazione nulla (*KC\_NO*) per prevenire registrazioni di input non voluti.

Il firmware mantiene due versioni logiche della matrice:

- **Matrice RAW:** rappresenta lo stato hardware immediato, non filtrato, degli switch.
- **Matrice FILTRATA:** rappresenta lo stato stabile, sottoposto a debouncing e filtering.

#### 3.2.1 Scelte Implementative

Lo stato delle righe viene memorizzato in un array monodimensionale di tipo `matrix_row_t`, definito come alias di `uint16_t`, dove ogni cella rappresenta una riga. Ogni bit del tipo rappresenta lo stato logico di una colonna (0: premuto, 1: rilasciato).

Tale scelta consente:

- **Compattezza:** solo 6 byte per rappresentare l'intera matrice (3 righe da 16 bit), anziché 33 byte (necessari nel caso di mapping dedicato per ogni tasto).
- **Efficienza:** operazioni di confronto e aggiornamento mediante operatori *bitwise* e funzioni `memcmp` e `memcpy`.

Il file `matrix.h` utilizza la direttiva `#pragma once` per prevenire inclusioni multiple e definisce le interfacce principali di accesso alla matrice, con funzioni dichiarate `inline` per le operazioni più frequenti.

#### 3.2.2 Rilevamento dell'Input (Matrix Scan)

La scansione della matrice avviene periodicamente ogni 3 millisecondi, in seguito ad un interrupt generato dal modulo *GPTimer* (sez. 3.4.1). Tale scansione viene indicata come **Matrix Scan**.

Le macro `MATRIX_ROW_PINS` e `MATRIX_COL_PINS`, definite in `config.h`, consentono la configurazione dinamica dei pin GPIO associati alla matrice.

**Principio di funzionamento:**

- I pin delle righe sono impostati come `OUTPUT` e tenuti in stato logico `HIGH`.
- I pin delle colonne sono impostati come `INPUT` con `PULLUP`.
- Un tasto viene rilevato come premuto quando, selezionando una riga (portandola `LOW`), si rileva un valore `LOW` sulla colonna corrispondente.

**Sequenza operativa:**

1. Seleziona una riga portandola `LOW`.
2. Attendi  $1\mu\text{s}$  per l'assestamento del segnale.
3. Leggi tutte le colonne e memorizza lo stato.
4. Deseleziona la riga riportandola `HIGH`.
5. Attendi  $30\mu\text{s}$  prima di selezionare la riga successiva.

**Listing 1.** Pseudocodice Matrix Scan

```

for (riga = 0; riga < MATRIX_ROWS; riga++) {
    seleziona_riga(riga); // GPIO -> LOW
    delay(1);

    for (colonna = 0; colonna < MATRIX_COLS; colonna++) {
        stato_colonna = leggi_pin(colonna);
        aggiorna_matrice_raw(riga, colonna, stato_colonna);
    }

    deseleziona_riga(riga); // GPIO -> HIGH
    delay(30);
}

```

### 3.2.3 Gestione dell'Input

Per ogni cella della matrice ci interessa registrare esclusivamente due stati:

- **Rilasciato** (bit = 1)
- **Premuto** (bit = 0)

L'utilizzo del tipo `matrix_row_t` consente di sfruttare direttamente le operazioni a livello di bit:

- AND, OR, XOR per la verifica e aggiornamento degli stati.
- `memcpy` e `memcmp` per il confronto tra matrici.

La funzione principale di gestione dell'input è `matrix_scan()`. Essa:

- Esegue una scansione completa della matrice.
- Confronta la matrice RAW appena letta con lo stato precedente.
- Applica la funzione di debouncing (sez. 3.3).
- Restituisce un flag booleano indicante se lo stato della tastiera è cambiato.

**Listing 2.** Pseudocodice Funzione matrix\_scan()

```

matrice_corrente = scan_matrice();

if (memcmp(matrice_corrente, matrice_raw, dimensione)) {
    copia(matrice_raw, matrice_corrente);
    if (debounce_attivo)
        matrice_filtrata = debounce(matrice_raw);
    return true;
}

return false;

```

### 3.2.4 Considerazioni sull'Ottimizzazione

La scelta di una rappresentazione bitwise della matrice permette di:

- Minimizzare il consumo di memoria.

- Eseguire confronti e aggiornamenti in tempo costante.
- Mantenere il firmware leggibile e scalabile.

Operazioni frequenti come `matrix_get_row()` e `matrix_set_row()` sono implementate come funzioni `inline` per ridurre l'overhead di chiamata.

Le celle di tipo "tasto fantasma", non corrispondenti a reali switch, sono mappate su un valore KC\_NO nella tabella di keymap, in modo da evitare l'invio di eventi non validi.

Il prossimo capitolo (3.3) descriverà la logica di **debouncing**, fondamentale per la stabilizzazione degli input.

### 3.3 Correzione degli Errori: Debouncing

La natura meccanica degli switch introduce una problematica inevitabile nella rilevazione dei cambiamenti di stato: il **rimbalzo (debounce)**. Quando un tasto viene premuto o rilasciato, il contatto elettrico oscilla rapidamente tra gli stati alto e basso per una durata dell'ordine di alcuni millisecondi, causando letture errate da parte del microcontrollore. Senza un adeguato filtraggio, queste oscillazioni si tradurrebbero nella registrazione di pressioni multiple non intenzionali. fig. 21

La scelta di implementare la tecnica di debouncing nel firmware e non nell'hardware (tramite l'uso di un circuito RC per ogni switch) permette di ridurre i costi dell'hardware e di rendere più versatile l'algoritmo in quanto sostituibile con altre tecniche in letteratura e configurabile nei parametri.

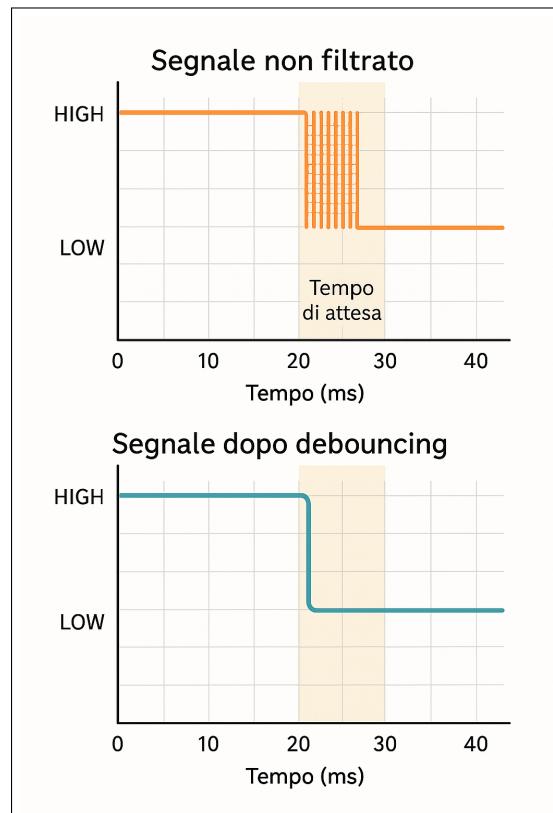


Fig. 21. Esempio applicazione debouncing al segnale

### 3.3.1 Tecnica di Debouncing Implementata

Nel firmware KHORD il **debouncing** viene realizzato tramite un approccio software, utilizzando un *filtro temporale* applicato a livello di matrice. L'algoritmo adottato considera valido un cambiamento di stato solo se permane stabile per una finestra temporale di almeno 10 ms (costante definita tramite la macro `#define DEBOUNCE 10` nel file `config.h`).

Ad ogni evento di cambiamento rilevato nella matrice RAW:

1. Se è il primo cambiamento, viene salvato il timestamp di inizio del rimbalzo.
2. Se ulteriori cambiamenti avvengono entro la finestra temporale di debounce, essi vengono ignorati.
3. Trascorso il tempo di debounce senza ulteriori variazioni, la macchina torna in ascolto di ulteriori cambi di stato.

### 3.3.2 Implementazione Firmware

L'algoritmo è implementato nella funzione `debounce()` del modulo `debounce.c`. La logica è riassumibile nel seguente pseudocodice:

**Listing 3.** Pseudocodice algoritmo di debouncing

```
if (cambiamento_rilevato)
{
    debouncing = true;
    debouncing_time = timestamp_corrente;
}
else if (debouncing
          && (timestamp_corrente - debouncing_time) >= DEBOUNCE)
{
    copia( matrice_filtrata , matrice_raw );
    debouncing = false;
}
```

### 3.3.3 Vantaggi della Scelta

Rispetto ad alternative più complesse (come filtri digitali o contatori per ogni singolo tasto), questa implementazione:

- Minimizza l'occupazione di memoria: non è necessario allocare strutture dati aggiuntive per ogni tasto.
- Garantisce reattività sufficiente per l'uso stenografico, dove la pressione di più tasti simultanei richiede stabilità, ma latenza minima.
- È facilmente parametrizzabile modificando il valore della macro `DEBOUNCE`.

## 3.4 Gestione Asincrona delle Operazioni

Nel firmware di *KHORD*, la sincronizzazione delle operazioni è stata progettata adottando un approccio asincrono. In particolare, si è deciso di evitare la tecnica del *polling*, che avrebbe occupato costantemente il processore in un ciclo di attesa, preferendo invece una gestione tramite interrupt hardware.

La scansione periodica della matrice della tastiera viene richiesta tramite un timer **GPTimer** configurato in modalità periodica. L'interrupt generato dal timer funge da segnale per notificare al firmware la necessità di eseguire un ciclo di *matrix scan* (sez. 3.2).

Per minimizzare l'occupazione della CPU e ridurre il tempo speso all'interno della *callback* di interrupt, è stata adottata la tecnica del **Deferred Work** (sez. 3.4.2). Questo approccio consente di delegare la maggior parte del lavoro computazionale (la scansione vera e propria della matrice) al thread principale, evitando elaborazioni onerose all'interno dell'ISR (Interrupt Service Routine).

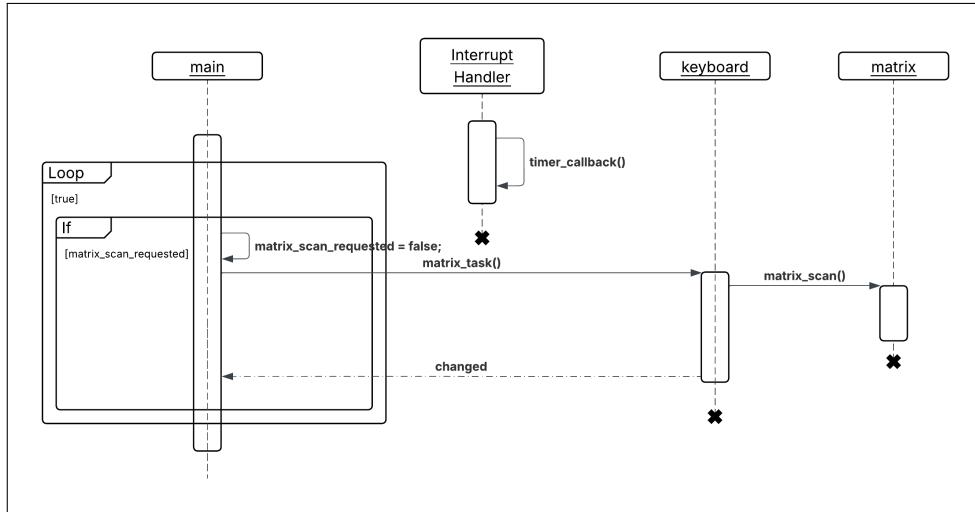


Fig. 22. Sequence Diagram Gestione Asincrona Operazioni

### 3.4.1 Interrupt (GPTimer)

I **GPTimer** [9] sono timer hardware forniti dal framework *ESP-IDF*, utilizzati per generare eventi periodici precisi con risoluzione configurabile. Nel progetto, il timer è stato configurato per generare un interrupt ogni 3 millisecondi (frequenza  $\approx 333$  Hz), rappresentando il segnale per avviare un nuovo ciclo di scansione della matrice.

Durante l'ISR associata all'interrupt del timer non viene eseguita direttamente la `matrix_scan()` ma viene semplicemente settata la flag globale `matrix_scan_requested = true`. Questo permette di mantenere l'interrupt estremamente leggero ed efficiente, minimizzando la latenza e prevenendo la saturazione del sistema.

Listing 4. Callback ISR del GPTimer

```

bool IRAM_ATTR timer_callback (
    gptimer_handle_t timer ,
    const gptimer_alarm_event_data_t *edata ,
    void *user_ctx)
{
    matrix_scan_requested = true;

    //facoltativo: consente di effettuare un context switch immediato
    return true;
}

```

Il timer viene configurato all'avvio tramite la funzione `timer_setup()` all'interno del file `timer.c`, in cui sono impostati:

- Risoluzione del timer (`TIMER_RESOLUTION`) impostata ad 1MHz, ossia un tick ogni microsecondo.
- Frequenza del timer (`TIMER_SCAN_DELAY`).
- Modalità di riarmo automatico (`auto_reload_on_alarm = true`).
- Callback ISR.

### 3.4.2 Deferred Work

La tecnica di **Deferred Work** consiste nello spostare il carico di lavoro dall'interrupt ad un contesto di esecuzione più appropriato (tipicamente il *main loop* o un task dedicato).

Nel firmware di *KHORD*, il *main loop* del programma esegue il controllo periodico della flag `matrix_scan_requested`. Quando essa risulta vera, il firmware:

1. Pulisce la flag (`matrix_scan_requested = false`).
2. Esegue la funzione `matrix_task()` per effettuare la scansione della matrice e gestire gli eventi.

**Listing 5.** Main loop con Deferred Work

```
while (true) {
    if (matrix_scan_requested) {
        matrix_scan_requested = false;
        matrix_task();
    }
    vTaskDelay(1); // consente alle altre task di eseguire
}
```

Vantaggi del Deferred Work:

- Minimizzazione del tempo di permanenza nella ISR.
- Migliore reattività del sistema a nuovi interrupt.
- Semplificazione della logica di gestione degli eventi, confinandola nel contesto del *main thread*.

L'unico grande svantaggio di questa tecnica è che richiede una gestione efficiente della flag di segnalazione per evitare la perdita di eventi. Se il flag non viene resettato prima del prossimo interrupt l'evento scatenante verrà ignorato. Questo problema nel nostro caso non è presente in quanto la scarsità di eventi scatenati e la breve durata di una singola scansione non gravano sulla frequenza di scansione.

L'integrazione di interrupt GPTimer e Deferred Work costituisce il cuore della gestione asincrona del firmware, consentendo una scansione della matrice tempestiva e reattiva senza sovraccaricare il microcontrollore.

### 3.5 Librerie e Plugin Utilizzati

Il progetto **KHORD** si basa su un set ridotto di librerie esterne, tutte open-source, che hanno facilitato lo sviluppo e la gestione del firmware. In questa sezione sono illustrate le principali librerie e framework utilizzati, accompagnate da una descrizione del loro utilizzo concreto nel codice.

### 3.5.1 TinyUSB

**TinyUSB** [29] è una libreria open-source, scritta in C, progettata per implementare stack USB Host e Device nei sistemi embedded. Supporta molteplici architetture hardware e consente la realizzazione di dispositivi conformi a vari standard USB, tra cui *CDC-ACM*, *HID*, *MSC* e altri.

Nel progetto KHORD, TinyUSB viene utilizzata per la gestione della comunicazione seriale USB tramite una **porta virtuale seriale** (Virtual COM Port), grazie alla classe *CDC-ACM*. L'integrazione di TinyUSB nel framework ESP-IDF avviene tramite il componente `esp-tinyusb` [8], che semplifica la configurazione dei driver USB nativi presenti sul microcontrollore ESP32-S3.

La classe **CDC-ACM** (*Communication Device Class – Abstract Control Model*) è uno standard USB [30] che consente di emulare una **serial port** (COM port) sopra il bus USB, mantenendo la retro-compatibilità con i software progettati per comunicare tramite UART classiche. Questa tecnica, nota come *USB Serial-over-CDC-ACM*, è ampiamente utilizzata nei sistemi embedded per implementare interfacce di debug o comunicazione dati.

Durante il normale funzionamento della tastiera, il protocollo **GeminiPR** (sez. 3.6) invia pacchetti di dati testuali al software **Plover** (sez. 3.7) attraverso questa seriale virtuale, simulando l'invio da una tastiera standard.

Il codice sorgente relativo a questa libreria è contenuto all'interno della classe `usb.c` e astratto all'esterno tramite l'header `usb.h`. TinyUSB viene utilizzata per:

- Inizializzare il driver USB device (`tinyusb_driver_install`).
- Inizializzare una interfaccia CDC-ACM (`tusb_cdc_acm_init`).
- Trasmettere dati verso il PC mediante la porta seriale USB (`tinyusb_cdcacm_write_queue`).

**Listing 6.** Pseudocodice inizializzazione TinyUSB

```
usb_init() {
    tinyusb_driver_install(&tusb_cfg);
    tusb_cdc_acm_init(&acm_cfg);
}
```

La funzione `usb_send()` implementa la scrittura asincrona dei dati nel buffer USB:

**Listing 7.** Pseudocodice invio dati USB

```
usb_send(dati, dimensione) {
    tinyusb_cdcacm_write_queue(...);
    tinyusb_cdcacm_write_flush(...);
}
```

### 3.5.2 ESP-IDF

Il **ESP-IDF** (Espressif IoT Development Framework) [7] rappresenta il framework ufficiale per lo sviluppo di firmware su dispositivi ESP32. È un SDK open-source che include:

- Gestione dei GPIO.
- Gestione di timer hardware (`gptimer`).
- Gestione della comunicazione USB.
- API per logging e gestione di eventi.

Nel progetto KHORD, ESP-IDF fornisce il substrato per il controllo diretto dell'hardware e la gestione asincrona delle operazioni. Le principali funzionalità ESP-IDF utilizzate sono:

- `driver/gpio.h` per il controllo dei pin della matrice.
- `driver/gptimer.h` per la gestione dei timer hardware (cap. 3.4.1).
- `esp_log.h` per il logging diagnostico.
- `esp_timer.h` per il conteggio dei millisecondi.

Tali funzionalità sono distribuite nei file:

- `matrix.c` (gestione GPIO)
- `timer.c` (gestione timer)
- `main.c` (gestione ciclo principale e FreeRTOS)

### 3.6 Protocolli di Codifica Stenografica

Il firmware *KHORD* traduce la pressione simultanea dei tasti in una sequenza di bit secondo un protocollo standard di codifica stenografica. Ogni tasto fisico della tastiera è associato a un simbolo stenografico. La loro interpretazione segue un ordine fisso standardizzato, utilizzato universalmente dalla comunità stenografica, secondo la sequenza:

S T K P W H R A O \* E U F R P B L G T S D Z

Questa disposizione, nota come *steno order*, deriva dalla struttura tradizionale delle tastiere per stenotipia meccaniche (vedi [22], [2]) e rappresenta la posizione logica dei tasti dalla sinistra alla destra della tastiera, comprendendo consonanti iniziali, vocali, tasti centrali e consonanti finali.

Ogni chord (*accordo*) è codificato quindi in base a questa sequenza logica, indipendentemente dalla posizione fisica dei tasti, permettendo una corretta interpretazione da parte dei software di trascrizione come *Plover*.

#### 3.6.1 Scelta del Protocollo (Gemini PR vs TX Bolt)

Il progetto *KHORD* supporta la codifica dei chord tramite il protocollo **Gemini PR**. Sono state valutate due principali alternative:

- **TX Bolt**: protocollo storico, sviluppato negli anni '80, con una codifica semplice a 24 tasti. È ormai considerato obsoleto e limitato (supporto di sole 24 chiavi).
- **Gemini PR**: protocollo più recente e versatile, progettato tramite reverse engineering da parte della comunità open-source [28]. Supporta fino a 42 tasti simultanei e offre una rappresentazione binaria più compatta ed efficiente.

La scelta di Gemini PR è stata motivata da:

- maggiore copertura di simboli rispetto a TX Bolt (42 tasti vs 24).
- aderenza agli standard moderni (compatibilità nativa con *Plover*).
- minore complessità di implementazione del protocollo lato firmware.
- possibilità di adattarsi in futuro a layout e teorie alternative (ad esempio teorie fonetiche non anglofone).

### 3.6.2 Implementazione del Protocollo di Chording

All'interno di *KHORD*, il protocollo Gemini PR è stato implementato generando pacchetti binari di 6 byte, in conformità alle specifiche standard [28]. Ogni pacchetto rappresenta un chord completo ed è trasmesso via seriale USB una volta che tutti i tasti sono stati rilasciati.

La codifica prevede:

- 6 byte (48 bit) per chord.
- Gli ultimi 7 bit di ogni byte rappresentano un gruppo di tasti.
- L'MSB (bit 7) del primo byte è impostato ad 1 per indicare l'inizio del pacchetto.
- Tutti gli altri byte hanno l'MSB settato a 0.

**Listing 8.** Generazione pacchetto Gemini PR

```
uint8_t chord[CHORD_BYTES] = {0};

for each key in pressed_keys:
    group_index = key / 7
    intra_group_index = key % 7
    bit = 1 << (6 - intra_group_index)
    chord[group_index] |= bit

chord[0] |= 0x80 // MSB del primo byte = 1
```

#### Spiegazione dettagliata dei passaggi

1. `uint8_t chord[CHORD_BYTES] = {0};`
  - Inizializza un array di 6 byte a zero, che rappresenterà il pacchetto da inviare via seriale. Ogni byte codifica fino a 7 tasti (42 tasti totali supportati).
2. `for each key in pressed_keys:`
  - Scorre tutti i tasti fisicamente premuti nel chord attuale. Ogni tasto è rappresentato da un indice intero (da 0 a 41) definito dalla mappatura in ordine logico standard.
3. `group_index = key / 7`
  - Calcola in quale byte del pacchetto scrivere il bit corrispondente al tasto. I tasti sono suddivisi in 6 gruppi da 7 tasti ciascuno ( $42 = 6 \times 7$ ).
4. `intra_group_index = key % 7`
  - Determina la posizione del tasto all'interno del byte corrente. Si utilizza un offset inverso (6 - indice) per rispettare la rappresentazione Gemini PR, che memorizza il primo tasto del gruppo nel bit più significativo.
5. `bit = 1 << (6 - intra_group_index);`
  - Genera il bitmask con un singolo bit a 1 nella posizione corrispondente al tasto. Questo permette di impostare il bit corretto nel byte.
6. `chord[group_index] |= bit;`
  - Imposta il bit del tasto nel byte corretto tramite operazione OR bit a bit. Consente di rappresentare più tasti nello stesso byte.
7. `chord[0] |= 0x80;`

- Imposta il bit più significativo (MSB) del primo byte del pacchetto a 1. Questo identifica il byte iniziale del pacchetto secondo le specifiche del protocollo Gemini PR. Gli altri byte del pacchetto hanno l'MSB a 0.

**Esempio di Pacchetto Generato** Consideriamo il chord rappresentante S T K:

```
Byte 1: 10000010
Byte 2: 00100100
Byte 3: 00000000
Byte 4: 00000000
Byte 5: 00000000
Byte 6: 00000000
```

Il file `keyboard.c` gestisce il protocollo nella funzione `send_chord()`, richiamata al rilascio dell'ultimo tasto di un chord.

**Listing 9.** Funzione di invio chord

```
void send_chord(void) {
    usb_send(chord, sizeof(chord));
}
```

Il pacchetto così generato è inviato tramite tinyUSB sulla porta seriale virtuale, conforme allo standard CDC-ACM, descritta nel dettaglio nella sezione [3.5.1](#).

### 3.7 Plover

**Plover** [24] è un software open-source per la trascrizione stenografica in tempo reale, sviluppato come alternativa libera e gratuita ai costosi software proprietari storicamente dominanti nel settore della stenotipia professionale. L'obiettivo primario del progetto è rendere la stenografia accessibile a tutti, in particolare a studenti, hobbyisti e comunità open-source, abbattendo le barriere economiche che per anni hanno limitato la diffusione di questa tecnologia [23].

Plover interpreta i dati provenienti da una tastiera stenografica, rappresentati sotto forma di *chord* e li traduce in output testuale leggibile grazie all'utilizzo di dizionari personalizzabili. Il software supporta diversi protocoli di comunicazione e può operare in tempo reale per scrivere direttamente su qualsiasi campo di testo del sistema operativo.

L'adozione di Plover ha rappresentato un grande vantaggio per il progetto **KHORD**, in quanto ha permesso di concentrarsi esclusivamente sull'hardware e sul firmware per l'acquisizione e trasmissione dei chord, delegando completamente la parte di decodifica, traduzione fonetica e gestione dei dizionari al software esistente [25].

#### 3.7.1 Configurazione

Una volta connessa la tastiera **KHORD** al computer tramite USB, è necessario configurare Plover per stabilire la comunicazione con il dispositivo. La procedura da seguire al primo avvio è la seguente:

1. Aprire Plover.
2. Accedere al menu delle impostazioni tramite **Configure**.
3. Nella sezione **Machine**, selezionare il protocollo **Gemini PR**.
4. Utilizzando il **Device Manager** del sistema operativo, individuare la porta COM associata a KHORD.

5. Inserire la porta COM in Plover nel campo Port.
6. Impostare il Baudrate su 115200.
7. Cliccare Apply e poi OK.
8. Tornare alla schermata principale di Plover e attivare l'output cliccando su Enable.
9. Premere il tasto Reconnect. Quando l'etichetta passa da Disconnected a Connected, il dispositivo è pronto all'uso.

Da questo momento in poi, qualsiasi chord trasmesso dalla tastiera KHORD verrà automaticamente decodificato da Plover e stampato sullo standard output del sistema operativo.

### 3.7.2 Funzionalità e Personalizzazione

Una delle caratteristiche principali di Plover è la possibilità di personalizzare completamente i dizionari di trascrizione. Ogni dizionario è rappresentato come un semplice file JSON in cui ogni voce associa un chord (espresso in notazione stenografica) ad una traduzione testuale.

È possibile creare dizionari personalizzati aggiuntivi oppure modificare direttamente il dizionario dell'utente (`user.json`), che viene caricato automaticamente all'avvio di Plover. Di seguito un esempio minimale:

**Listing 10.** Esempio di dizionario custom Plover

```
{
  "KHORD": "corda",
  "T-K": "tac",
  "STKPWHRAO*EU": "ciao"
}
```

Ulteriori funzionalità offerte dal software includono:

- **Paper Tape:** una finestra di diagnostica che visualizza in tempo reale i chord ricevuti, utile per verificare il corretto funzionamento del firmware e dei keymap.
- **Lookup Tool:** uno strumento di ricerca che consente di trovare tutti i chord corrispondenti a una determinata parola all'interno di tutti i dizionari attivi, supportando lo studio della stenografia e la personalizzazione rapida.
- **Dizionari multipli:** è possibile caricare più dizionari contemporaneamente (es. dizionario inglese base, abbreviazioni, comandi) e definirne la priorità.

## 4. Analisi e Valutazioni

Questa sezione analizza il consumo energetico, le prestazioni e i costi della tastiera KHORD, con l'obiettivo di valutarne l'efficienza, la replicabilità e il basso costo.

### 4.1 Potenza Dissipata

Il consumo energetico della tastiera KHORD è determinato principalmente dal microcontrollore ESP32-S3-DevKitC-1-N8R8, alimentato tramite USB a 5 V e dotato di un regolatore interno che fornisce 3.3 V al SoC. Il modulo opera in modalità attiva con Wi-Fi e Bluetooth non utilizzati e assorbe una corrente media di 23.88 mA [10]. La sua potenza dissipata è quindi

$$P = V \times I = 3.3V \times 23.88mA = 78.80mW \quad (1)$$

Questo valore include il consumo della CPU, delle periferiche interne e della comunicazione USB CDC-ACM.

I 25 diodi 1N4148, utilizzati per prevenire il ghosting, conducono corrente solo durante la pressione dei tasti, contribuendo con una potenza (volendo trascurabile) di circa  $10 \mu\text{W}$ , quando solo un tasto è attivo, [21] e fino a  $250 \mu\text{W}$  nel caso teorico di pressione simultanea di tutti i 25 tasti

$$P_{diodo} = V \times I \approx 1 \text{ V} \times 10 \mu\text{A} = 10 \mu\text{W}$$

Gli interruttori Cherry MX Brown e i 3 stabilizzatori, essendo componenti meccanici, non consumano energia.

La **potenza totale dissipata massima** (con tutti i tasti premuti) è quindi **inferiore a 0.1 W**.

Poiché KHORD è costantemente alimentata tramite USB, compatibile con lo standard 5 V / 500 mA, l'ottimizzazione del consumo energetico non è stata prioritaria. In una futura versione wireless alimentata a batteria, modalità a basso consumo come *modem-sleep*, *light-sleep* o *deep-sleep* saranno essenziali per minimizzare il consumo e garantire maggiore autonomia alla batteria.

## 4.2 Prestazioni (Tempi di Esecuzione)

La valutazione delle prestazioni della tastiera KHORD si è concentrata sul tempo che intercorre tra la pressione dei tasti e la trasmissione del pacchetto USB al software Plover.

Il ciclo di **scansione della matrice** viene eseguito ogni 3 ms tramite un interrupt hardware generato da un GPTimer. Tuttavia, il tempo effettivo richiesto per completare la scansione delle 3 righe è decisamente inferiore. Ogni riga richiede circa  $33 \mu\text{s}$ : 1  $\mu\text{s}$  di delay dopo la selezione della riga, circa 2  $\mu\text{s}$  per la lettura delle colonne e 30  $\mu\text{s}$  dopo la deselectazione della riga. Quindi per scannerizzare tutte le 3 righe è richiesto un totale di:

$$T_{\text{scan}} = 3 \times 33 \mu\text{s} = 99 \mu\text{s}$$

Questo lascia circa 2.9 ms liberi prima dell'interrupt successivo, durante i quali il sistema può gestire altre operazioni asincrone come il debouncing, la gestione della comunicazione USB e l'elaborazione degli eventi.

Il **debouncing** è gestito in modo asincrono con una finestra di 10 ms e garantisce stabilità dell'input senza introdurre latenza percepibile.

I **pacchetti trasmessi** a Plover hanno dimensione di 6 byte. A una velocità di trasmissione di 115200 bps, ogni byte richiede circa  $87 \mu\text{s}$  (considerando 10 bit per byte: 8 bit di dati, 1 start bit e 1 stop bit). Il tempo complessivo di trasmissione è quindi pari a:

$$T_{\text{trasmissione}} = 6 \times \frac{10}{115200} = 6 \times 87 \mu\text{s} \approx 520 \mu\text{s}$$

La frequenza di invio dei pacchetti è asincrona e dipende dalla velocità di digitazione dell'utente, con ogni pacchetto che rappresenta un chord completo che viene trasmesso una volta rilasciati tutti i tasti, come descritto nella sez. 3.6.2.

L'**elaborazione dell'accordo** e la **codifica GeminiPR**, basate rispettivamente su mapping diretto e operazioni bitwise rapide, hanno un impatto trascurabile.

Questi tempi garantiscono un'elevata reattività del sistema anche in condizioni di input intenso.

### 4.3 Analisi dei Costi

L'analisi dei costi del progetto KHORD valuta le spese sostenute per l'acquisto dei componenti necessari alla costruzione della tastiera, considerando sia i costi totali per gli ordini in blocco (inclusa spedizione, dove applicabile) sia i costi proporzionali per i componenti effettivamente utilizzati.

La tastiera utilizza 25 interrutori Cherry MX Brown, 25 diodi 1N4148, una PCB prodotta da NextPCB, 3 stabilizzatori per tasti da 2u, un case stampato in 3D in PLA e viti/dadi M2 per l'assemblaggio.

Il costo totale del prototipo è calcolato come somma dei costi dei componenti utilizzati, garantendo un'analisi precisa per valutare l'economicità del progetto.

**Tab. 1.** Costi economici relativi al progetto KHORD (espressi in euro)

Componente	Costo Totale Acquisto in Blocco (€)	Costo Pezzi Utilizzati (€)
Scheda originale Espressif ESP32-S3-DevKitC-1-N8R8	1 pezzo: 17.09 €	1 pezzo: 17.09 €
Switch meccanici Cherry MX brown	40 pezzi (0.23 €/pz): 9.19 €	25 pezzi (0.23 €/pz): 5.74 €
Copritasti DSA	61 pezzi (0.13 €/pz): 7.75 €	15 pezzi (0.13 €/pz): 3.18 €
Diodi 1N4148	100 pezzi (0.015 €/pz): 1.47 €	25 pezzi (0.015 €/pz): 0.37 €
Stabilizzatori 2u	5 pezzi (0.18 €/pz): 0.90 €	3 pezzi (0.18 €/pz): 0.54 €
Viti M2 12 mm e dadi	410 pezzi (0.045 €/pz): 8.99 €	26 pezzi (0.045 €/pz): 0.11 €
PCB	5 pezzi (5.21 €/pz): 20.85 €	1 pezzo: 5.21 €
<b>Prezzo finale (senza case)</b>	66.24 €	<b>32.24 €</b>
Case	1 pezzo: 40.00 €	1 pezzo: 40.00 €
<b>Prezzo finale (con case)</b>	106.24 €	<b>72.24 €</b>

Il costo del case è stato analizzato separatamente perché la sua produzione presso un negozio specializzato in stampa 3D ha avuto un impatto significativo sul costo totale. Tuttavia, se si dispone di una stampante 3D, il costo del case può essere drasticamente ridotto, rendendo il progetto ancora più economico.

Complessivamente, senza considerare il case e valutando i singoli componenti utilizzati, la tastiera ha un costo di 32.24 €.

## 5. Conclusioni e Sviluppi Futuri

Il progetto KHORD ha raggiunto un traguardo importante: sviluppare una tastiera stenografica open-source, economica e personalizzabile, superando i limiti delle soluzioni proprietarie. Grazie al microcontroller ESP32-S3 e a un firmware personalizzato scritto in C con il framework ESP-IDF, KHORD si è dimostrata una piattaforma versatile, ideale per studenti, scrittori, programmatore e hobbisti. Il design hardware, con il supporto NKRO (N-Key Rollover), offre affidabilità e precisione, fondamentali per la stenografia. Il prototipo è stato testato con successo.

Nonostante la tastiera KHORD abbia raggiunto pienamente gli obiettivi iniziali, il progetto resta aperto a ulteriori evoluzioni. Infatti, durante il processo di progettazione e test, sono emerse numerose possibilità di miglioramento, sia dal punto di vista tecnico che estetico, che potrebbero ampliarne le funzionalità e renderla ancora più portatile, efficiente e personalizzabile.

Di seguito, organizzati nella tabella 2, sono descritti i possibili sviluppi futuri.

**Tab. 2.** Tabella degli sviluppi futuri

Aspetto	Ottimizzazione
Versione wireless	<p>Realizzazione di una versione completamente wireless di KHORD sfruttando Wi-Fi e Bluetooth 5 Low Energy dell'ESP32-S3. Le ottimizzazioni si dividono in</p> <ul style="list-style-type: none"> <li><i>Ottimizzazione Hardware:</i> utilizzo di schede come Olimex ESP32-S3-DevKit-Lipo [20] o DFRobot Beetle ESP32-C3 [6], dotate di circuiti per batterie LiPo (ricaricabili, leggere, ad alta densità energetica). In alternativa, sulla scheda attuale (ESP32-S3-DevKitC-1-N8R8), si potrebbe aggiungere un circuito con chip TP4056, che gestisce la ricarica sicura della batteria LiPo.</li> <li><i>Ottimizzazione Software:</i> ottimizzare il firmware basato su ESP-IDF per trasmissioni a bassa latenza (&lt;10 ms), stabilità della connessione, autonomia della batteria e buona portata (10-30 m). Implementare una modalità ibrida (USB o wireless) con passaggio via tasto dedicato e meccanismi di riconnessione automatica.</li> </ul>
Utilizzo e prestazioni antenna (in caso di versione wireless)	<p>Migliorare la connettività wireless isolando l'area dell'antenna sul PCB, rimuovendo il riempimento di rame per ridurre le interferenze.</p> <p>Si potrebbero valutare varianti ESP32-S3 con connettori per antenne esterne (es. ESP32-S3-DEVKITC-1U-N8R8 [11]), utili per migliorare la ricezione, soprattutto se la tastiera è racchiusa in un case che potrebbe attenuare il segnale.</p>
Funzionalità aggiuntive	<p>Ulteriori funzionalità per arricchire l'esperienza d'uso da parte dell'utente:</p> <ul style="list-style-type: none"> <li>Integrare un sistema di visualizzazione delle informazioni come lo stato della connessione, la velocità di digitazione (WPM) o il livello della batteria. A tale scopo, si possono utilizzare display a 7 segmenti come TM1637 o TM1638 per i dati numerici essenziali, oppure un display IPS (LCD) a colori per una rappresentazione grafica più dettagliata e personalizzabile.</li> <li>Implementare un sistema di retroilluminazione a LED per i tasti per migliorare la visibilità e l'estetica della tastiera KHORD, rendendola più funzionale in ambienti scarsamente illuminati e più attraente per gli utenti. La gestione della retroilluminazione può avvenire tramite driver dedicati con controllo PWM, come il TLC5940, PCA9685 o IS31FL3731, con i quali è possibile regolare l'intensità luminosa e creare effetti luminosi personalizzati.</li> </ul>
Aggiunta di layer alternativi	<p>Estendere il firmware per supportare la nozione di <i>layer</i>. Ogni layer corrisponde a una keymap differente, attivabile temporaneamente o in modo persistente tramite tasti dedicati (es. M0(x) o TG(x) in QMK).</p> <p>Questo permetterebbe, ad esempio, la creazione di un layer di simboli, uno per la punteggiatura o uno per comandi di navigazione, aumentando l'espressività della tastiera senza aggiungere nuovi tasti fisici. L'implementazione richiederebbe una struttura dati per il layer corrente e un sistema di fallback tra layer attivi.</p>
Gestione della pressione di lunga durata	<p>Introdurre una logica di <i>long press</i> per tasti o chord, che consenta la ripetizione automatica dell'output finché il tasto resta premuto, come avviene nelle tastiere standard.</p> <p>L'algoritmo può prevedere un tempo di attivazione iniziale (es. 300ms) seguito da ripetizioni ogni 100ms finché il chord non viene rilasciato. Ciò permetterebbe funzioni come cancellazione continua (*) o ripetizione rapida di simboli senza sollevare le mani.</p>
Introduzione di macro	<p>Integrare un sistema di macro a livello firmware per l'esecuzione di sequenze complesse come tasti funzione, scorciatoie da tastiera o comandi di sistema (es. copia/incolla, volume, schermata). Ogni chord potrebbe essere associato a una funzione speciale anziché a un testo statico, eseguendo più azioni in sequenza. L'implementazione richiede un motore di interpretazione delle macro, con un dizionario interno di mapping tra chord e funzione complessa.</p>

## Bibliografia e Sitografia

- [1] ABPrint3D *ABPrint3D – Stampa 3D Professionale e Assistenza Milano*. <http://www.abprint3d.it/>.
- [2] Art of Chording *How steno works*. <https://www.artofchording.com/introduction/how-steno-works.html>.
- [3] Autodesk *Autodesk Fusion – 3D CAD, CAM, CAE & PCB Cloud-Based Software*. <https://www.autodesk.com/products/fusion-360/overview>.
- [4] Benny Megidish *Fabrication Toolkit – Scripts and Utilities for PCB Fabrication*. <https://github.com/bennymeg/Fabrication-Toolkit>. 2022.
- [5] Danny Tan *kicad-KLE-placer*. <https://github.com/zykrah/kicad-kle-placer>. 2023.
- [6] DFRobot *Beetle ESP32-C3 – A mini size IoT development board based on ESP32-C3*. <https://www.dfrobot.com/product-2566.html>.
- [7] ESP-IDF *Espressif ESP-IDF Programming Guide (v5.4.2)*. 2025. Available at <https://docs.espressif.com/projects/esp-idf/en/stable/esp32s3/index.html>.
- [8] ESP-IDF *Espressif's additions to TinyUSB (v1.7.4)*. 2025. Available at [https://components.espressif.com/components/espressif/esp\\_tinyusb](https://components.espressif.com/components/espressif/esp_tinyusb).
- [9] ESP-IDF *General Purpose Timer*. <https://docs.espressif.com/projects/esp-idf/en/stable/esp32/api-reference/peripherals/gptimer.html>.
- [10] Espressif Systems *Current Consumption Measurement for ESP Modules*. <https://docs.espressif.com/projects/esp-idf/en/stable/esp32s3/api-guides/current-consumption-measurement-modules.html>. 2023.
- [11] Espressif Systems *ESP32-S3-DevKitC-1 User Guide (v1.1)*. [https://docs.espressif.com/projects/esp-dev-kits/en/latest/esp32s3/esp32-s3-devkitc-1/user\\_guide\\_v1.1.html](https://docs.espressif.com/projects/esp-dev-kits/en/latest/esp32s3/esp32-s3-devkitc-1/user_guide_v1.1.html). 2023.
- [12] Espressif Systems *ESP32-S3-DEVKITC-1-N8R8 – Development Board*. <https://www.snapeda.com/parts/ESP32-S3-DEVKITC-1-N8R8/Espressif20Systems/view-part/>. 2024.
- [13] Espressif Systems *ESP32-S3-WROOM-1 / WROOM-1U Datasheet*. [https://www.espressif.com/sites/default/files/documentation/esp32-s3-wroom-1\\_wroom-1u\\_datasheet\\_en.pdf](https://www.espressif.com/sites/default/files/documentation/esp32-s3-wroom-1_wroom-1u_datasheet_en.pdf). 2023.
- [14] Ian Prest *Keyboard Layout Editor*. <https://www.keyboard-layout-editor.com/>. 2024.
- [15] Inkscape Project *Inkscape — Professional Vector Graphics Editor*. <https://inkscape.org/>. 2025.
- [16] Joe Scotto *ScottoKeebs - Handwired Keyboard Designs*. <https://github.com/joe-scotto/scottokeebs>. 2021.
- [17] KiCad: *Open Source Electronics Design Automation Suite*. <https://www.kicad.org/>.
- [18] Moritz MarbastLib - *KiCad Library*. <https://github.com/ebastler/marbastlib>. 2022.
- [19] NextPCB *NextPCB Online PCB manufacturing and quoting service*. 2024. Available at <https://www.nextpcb.com/pcb-quote#/pcb-quote>.
- [20] Olimex Ltd. *ESP32-S3-DevKit-Lipo open source hardware*. <https://www.olimex.com/Products/IoT/ESP32-S3/ESP32-S3-DevKit-Lipo/open-source-hardware>.
- [21] ON Semiconductor *1N4148WS Small Signal Switching Diode Data Sheet*. <https://www.onsemi.com/download/data-sheet/pdf/1n4148ws-d.pdf>. 2025.
- [22] Open Steno Project *Learn Plover, fingers and keys*. <https://www.openstenoproject.org/learn-plover/lesson-1-fingers-and-keys.html>.
- [23] Plover *Plover Beginner's Guide*. 2024. Available at [https://plover.wiki/index.php/Beginner%27s\\_Guide](https://plover.wiki/index.php/Beginner%27s_Guide).
- [24] Plover *The stenography desktop application*. Available at <https://github.com/openstenoproject/plover>.
- [25] Project OS Plover *GitHub Repository*. 2024. Available at <https://github.com/openstenoproject/plover>.
- [26] PuTTY *A free SSH and telnet client for Windows*. Available at <https://www.putty.org/>.
- [27] QMK *Open-source keyboard firmware for Atmel AVR and Arm USB families*. Available at [https://github.com/qmk/qmk\\_firmware](https://github.com/qmk/qmk_firmware).
- [28] QMK *Stenography*. <https://docs.qmk.fm/features/stenography>.
- [29] tinyUSB *Open-source cross-platform USB Host/Device stack for embedded system (v0.18.0)*. 2025. Available at <https://docs.tinyusb.org/en/latest/>.
- [30] USB-IF *Class definitions for Communication Devices (v1.2)*. 2025. Available at <https://www.usb.org/document-library/class-definitions-communication-devices-12>.
- [31] VS Code *The open source code editor*. Available at <https://code.visualstudio.com/>.

## Appendice 1 - Organizzazione dei File - Lato Hardware

`src/Hardware/`: è la cartella principale del progetto lato hardware in KiCad, e contiene:

- `KHORD.kicad_pro`: file principale del progetto. Facendo doppio clic su questo file si apre l'intero progetto in KiCad.
- `KHORD.kicad_sch`: file dello schema elettrico.
- `KHORD.kicad_pcb`: file del layout del circuito stampato (*PCB*).
- `sym-lib-table`: file di testo che elenca le librerie di simboli attive per il progetto.
- `fp-lib-table`: file di testo che elenca le librerie di footprint attive per il progetto.
- `libs/`: cartella con le librerie specifiche del progetto.
  - `_3d_models.3dshapes/`: cartella per i modelli 3D dei componenti.
  - `_footprints.pretty/` e `_logo.pretty/`: cartelle che contengono le impronte (*footprint*) dei componenti, utilizzate per il layout del PCB.
  - `_symbols/`: cartella per i simboli, utilizzati nello schema elettrico.

## Appendice 2 - Organizzazione dei File - Lato Software

`src/Software/`: è la cartella principale del progetto lato software, e contiene:

- `CMakeLists.txt`  
File di configurazione per il build system CMake e l'integrazione con ESP-IDF.
  - `idf_component_register`: registra i file sorgente e le dipendenze ESP-IDF.
  - `EXTRA_COMPONENT_DIRS`: include componenti comuni esterni.
- `idf_component.yml`  
Manifest per il gestore dei componenti ESP-IDF.
  - `espressif/esp_tinyusb`: dipendenza al modulo USB.
- `config.h`  
File centrale di configurazione hardware e costanti globali.
  - `MATRIX_ROWS`, `MATRIX_COLS`: dimensione della matrice.
  - `DEBOUNCE`: tempo in ms per il filtro di debouncing.
  - `MATRIX_ROW_PINS`, `MATRIX_COL_PINS`: macro per i pin GPIO assegnati.
  - `TIMER_RESOLUTION`, `TIMER_SCAN_DELAY`, `TIMEOUT_TICKS`, `TIMER_COUNT_START`: configurazioni del timer GPTimer.
- `matrix.h`  
Interfaccia della logica della matrice.
- `matrix.c`  
Implementazione della gestione della matrice hardware.
  - `matrix_read_cols_on_row()`: legge tutte le colonne per una riga.
  - `select_row()`, `unselect_row()`: funzioni GPIO per selezionare righe.
  - `matrix_output_select_delay()`: delay per stabilizzazione segnali.
  - `matrix_print()`: stampa una matrice di 0 e 1 raffigurante lo stato corrente della matrice,

- usato per debugging.
- `matrix_init()`: inizializza lo stato iniziale della matrice e dei pin GPIO.
  - `matrix_scan()`: legge righe e colonne per verificare che almeno una cella sia cambiata di stato.
  - `debounce.h`  
Header del modulo di debounce.
  - `debounce.c`  
Implementazione dell'algoritmo di debounce con filtro temporale.
    - `debouncing_time`: timer di riferimento.
    - `debounce()`: aggiorna matrice filtrata se superato tempo di assestamento.
  - `timer.h`  
Definizioni e prototipi per il timer GPTimer.
    - `TIMER_DIFF_32(a, b)`: macro per differenza tra 2 timestamp in formato `unit32_t`.
    - `matrix_scan_requested`: flag impostato dall'interrupt.
  - `timer.c`  
Implementazione del timer asincrono per la scansione periodica.
    - `timer_callback()`: ISR che segnala richiesta di matrix scan.
    - `timer_setup()`: configura il GPTimer, associa la callback ISR e fa partire il timer.
  - `usb.h`  
Interfaccia della comunicazione USB.
  - `usb.c`  
Implementazione della comunicazione CDC-ACM tramite TinyUSB.
    - `usb_init()`: inizializzazione dello stack USB.
    - `usb_send()`: invia pacchetti su porta seriale virtuale.
  - `keyboard.h`  
Interfaccia per la gestione della logica stenografica.
    - `keypos_t`: astrazione di una cella della matrice.
    - `keyevent_t`: struttura per la gestione di un evento scatenato dalla macchina.
  - `keyboard.c`  
Implementazione della logica di chording e invio dei pacchetti tramite protocollo GeminiPR.
    - `chord[]`: array di 6 byte che memorizza lo stato corrente dei tasti.
    - `pressed_keys`: numero di tasti attualmente premuti sulla tastiera.
    - `add_key_to_chord()`: codifica un tasto premuto nella posizione corretta.
    - `send_chord()`: prepara e invia pacchetto tramite USB.
    - `matrix_task()`: esegue lo scan, aggiorna lo stato delle matrici e processa gli eventi.
  - `keymap.h`  
Definizione della mappa dei tasti.
    - `keymaps[][]`: matrice [row][col] contenente i keycode logici.
    - `steno_keycodes`: enumerativo per astrarre codifica dei keycodes.

- **keymap.c**  
Mappatura logica dei tasti in codici stenografici.
  - **keymaps**: inizializzazione della matrice con i codici steno (es. STN\_S1).
- **main.c**  
Entry point del firmware.
  - **app\_main()**: funzione principale. Inizializza USB, matrice, timer e gestisce il loop principale.