microcomputer

OPERAZIONE NOSTALGIA

#MicroC64: state su che è arrivato!

Annunciazione, annunciazione… no, vabbè, facciamo le persone serie... 🤣

di Andrea de Prisco - AdP

Ne ho già parlato nella mia pagina FB: strada facendo ho anche sfanculato per sempre Gemini AI Pro, che alle prime complicazioni è entrato in un circolo vizioso di tentativi non-risolutivi sparati a casaccio.

La goccia è stata il riuscire a dar voce al SID dell'emulatore tramite l'amplificatore digitale da me utilizzato, ma dopo trentacinque (!!!) tentativi a vuoto e altrettante sue arrampicate sugli specchi, sono andato via senza nemmeno salutare. Rivolgendomi a Claude AI che, nel giro di qualche ora di vero brAInstorming, ha individuato e risolto brillantemente il problema.

Breve Recap

È stato un progettino tenuto in pectore per non meno di un paio d'anni, dai tempi dell' #AmighinoRPi: come suggeriscono le ultime tre lettere, era/è basato sul Raspberry Pi. Il passo successivo, *ça va sans dire,* sarebbe stato un mini emulatore C64, e per questo poteva essere più che sufficiente un ESP32-S3. Con la sua CPU dual core a 240 MHz e, soprattutto, tanta voglia di fare, poteva gestire in scioltezza l'emulazione di tutti i suoi chip.

Naturalmente l'emulatore NON è farina del mio sacco, ce n'è uno famoso disponibile su GitHub... definibile *QUASI pronto all'uso*. Dico questo - e lo sottolineo per bene - perché a seconda dell'hw periferico utilizzato (ampli digitale, display LCD, joystick e le immancabili varie ed eventuali) gli aggiustamenti alla configurazione dei tanti componenti sw presenti diventano, e sono state, la parte più complicata da portare a termine.

Il problema non riguardava l'ESP32-S3 e nemmeno l'emulatore in sé, ma come questi due riescano a parlare efficacemente con il loro piccolo mondo esterno.

Perché insisto tanto su questo? Perché se è vero che rilascio nel pubblico dominio, sotto licenza Creative Commons, tutto il rilasciabile (sorgenti e SDL inclusi), ci tengo a rimarcare che il tutto è calibrato su quelle specifiche periferiche e quei collegamenti, come da schemi prodotti. Non funzionerà con altre, a meno di non effettuare parecchi aggiustamenti. Ed essendoci passato... non vi invidio!

Joystick e tastiera

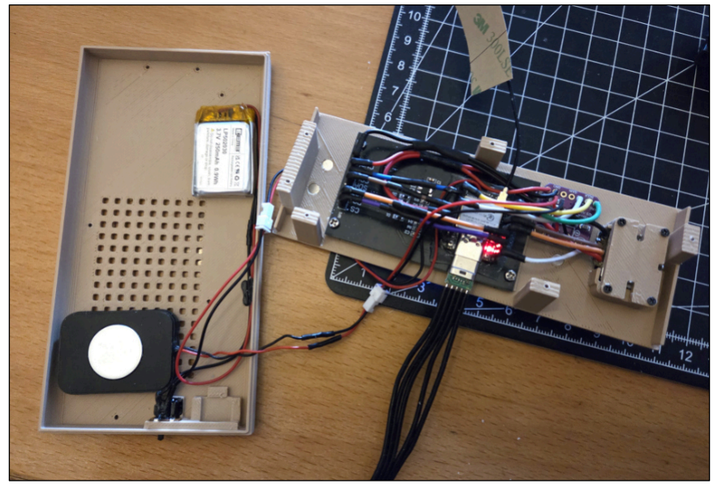
Inizialmente non avevo previsto il joystick integrato, ma solo quello esterno, collegato via USB al PC e in comunicazione *Bluetooth Low Energy* (BLE) con l'ESP32-S3, idem per la tastiera. Chiaramente quella visibile nella foto d'apertura è a fini estetici: non vorrei offenderla, ma è solo un coperchio a protezione del display, tenuto in posizione da un leggero incastro e da alcuni magneti al neodimio. L'STL di questa l'ho recuperato in Rete, ed è l'unica parte 3D non disegnata da me. Con tanto amore! ❤️

Per digitare *la qualunque*, incluso usare i tasti funzione che come ricorderete erano presenti a destra anche sul C64, si utilizza quella del computer, tramite il medesimo canale BLE utilizzato dal joystick esterno. A proposito: quando connesso e correttamente riconosciuto quest'ultimo ha priorità su quello integrato e cablato.

Per tutto ciò serviva poi una parte residente sul PC: uno script Python, molto articolato, che oltre a interfacciare joy e tastiera doveva occuparsi anche, o per meglio dire soprattutto, del trasferimento file dal PC all'ESP32-S3.

È possibile infatti memorizzare nella capiente Flash RAM dell' ESP32-S3 un numero considerevole di programmi preinstallati, da selezionare con il joystick integrato tramite un meccanismo aggiunto all'emulatore.

Su e giù per scorrere la lista, click sul Fire per lanciarlo: nel video online è ben mostrato.



Vista dell'interno. La griglia così grande presente sul fondo, necessaria per il mini altoparlante, non è casuale: non sapendo di preciso dove l'avrei posizionato per via dell'ingombro non prevedibile dei collegamenti elettrici. Quello in foto non è ancora il punto finale.

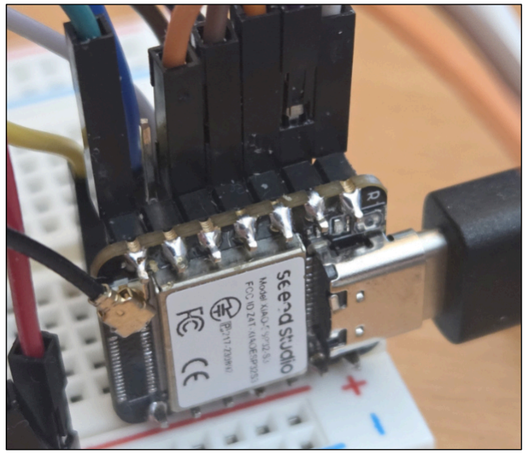
#MicroC64

Prezzo approssimativo materiali (05.2026)

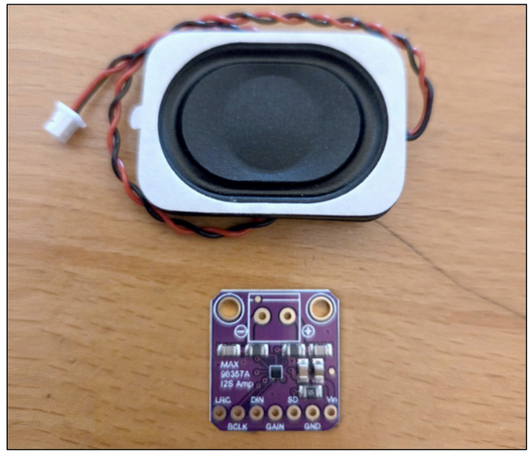
XIAO ESP32-S3	€ 15,00
Display LCD 2”	€ 3,00
Miniamplificatore audio	€ 2,00
Mini cassa audio	€ 3,00
Micro Joystick	€ 4,50
Cavetto USB-USB, interruttore, viti	€ 5,00
Batteria LiPo	€ 7,00
Filamento PLA (stima)	€ 3,00

Si *iniettano* i comandi «Open "nomefile.prg",8» e il successivo «RUN» direttamente nel buffer di tastiera (espediente che conosco e ricordo molto bene, visto che lo utilizzavo anche io sulle pagine di MCmicrocomputer ai tempi dei veri C64 e Vic-20) ed è per questo che li vedremo digitare sulla schermata basic dalla solita *manina fantasma*.

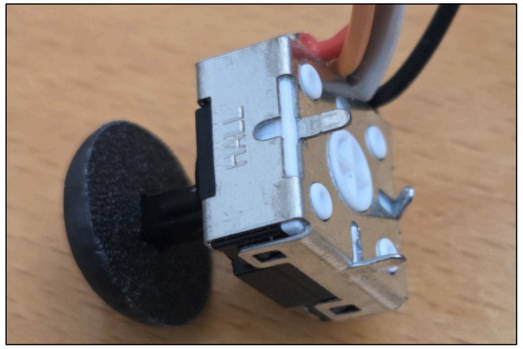
Sempre dall'interfaccia Python, alla quale ho dedicato un dettagliato articolo, è possibile incollare programmi Basic in formato TXT, anche questi magicamente digitati a

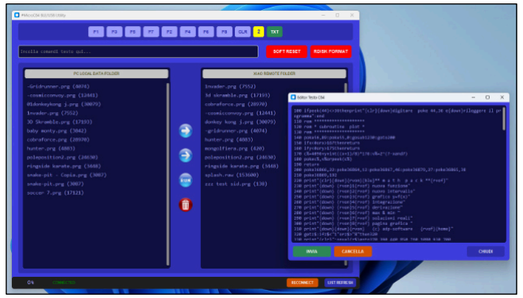


L'ESP32-S3 quando era ancora sulla bread board per i vari, tanti, test.



Il mini altoparlante e il microscopico ampli digitale





Il minuscolo joystick a effetto Hall e, in alto l'interfaccia Python di interfacciamento BLE e USB.

video, riga dopo riga, con la medesima tecnica.

Sarò anche il solito e inguaribile nostalgico, ma mi ha emozionato e non poco!

L'ESP32-S3 di XIAO
e tutto quanto.

La scelta di questa scheda, come del resto per tutti gli altri componenti, è dovuta alle sue dimensioni ultraridotte, a fronte di funzionalità più che complete. Integra finanche la logica di gestione per una batteria ricaricabile ai polimeri di litio che, ovviamente, è integrata anche questa all'interno del piccolo cabinet. Purtroppo non ha un connettore per collegarla "con un click" ma bisogna saldare i fili + e - della batteria direttamente sul fondo della schedina, in due piazzole piccolissime.

Non è stato facilissimo - non sono cintura nera di saldature, e si vede! - ma ci sono riuscito. La cosa più complicata è stata tenere saldatore e fili con le dita incrociate! 😁

Scherzi a parte, nonostante le sue ridotte dimensioni e una capacità di appena 250 mAh assicura circa un'ora e mezza di autonomia, testata tenendo in funzione un gioco in modalità demo, incluso l'audio riprodotto. La ricarica completa avviene in circa due ore e mezza, il che fa dedurre una corrente media di ricarica pari a 100 mA.

Il display è un modulo TFT da 2 pollici con controller ST7789V. È collegato all'ESP32-S3 tramite il bus *Serial Peripheral Interface* (SPI), un protocollo seriale sincrono comunemente usato per periferiche ad alta velocità, a quattro fili: *clock*, *dati*, *chip select* e *data/command*.

Da segnalare una piccola imprecisione nella serigrafia del modulo display: i pin sono etichettati SCL e SDA, denominazioni proprie del bus I2C, mentre come detto il display utilizza in realtà il bus SPI. I nomi corretti sarebbero SCK (*Serial Clock*) e MOSI (*Master Out Slave In*).

Pare sia un errore comune sui moduli di produzione cinese, probabilmente dovuto all'utilizzo della stessa serigrafia per famiglie diverse di display. Riguardo al meccanismo, l'emulatore genera il segnale video replicando il comportamento del chip VIC-II del C64, con la sua palette originale di 16 colori fissi.

L'immagine viene poi trasmessa al display in formato RGB a 16 bit - il formato nativo del controller ST7789V - a una risoluzione di 320×240 pixel. Ad ogni ciclo di refresh - circa 50 volte al secondo, fedele alla frequenza originale del C64 PAL - il framebuffer viene trasferito al display tramite DMA, liberando la CPU per continuare l'emulazione senza interruzioni. La risoluzione nativa del C64 (320×200 pixel) viene mappata direttamente sul display senza alcun ridimensionamento. Le barre laterali presenti nell'output originale del VIC-II - visibili sui monitor dell'epoca - qui semplicemente non sono riprodotte.

In merito all'audio, l'emulatore genera il segnale tramite la replica software del chip SID del C64, producendo campioni digitali a 44.100 Hz.

Questi vengono trasmessi all'amplificatore digitale di classe D MAX98357A tramite il protocollo Inter-IC Sound (I2S), uno standard seriale per il trasferimento di audio digitale tra circuiti integrati, a tre fili: *clock*, *word select* e *dati*. In pratica il MAX98357A riceve i dati digitali, li converte in analogico internamente e li amplifica, pilotando l'altoparlante senza bisogno di ulteriori componenti esterni.

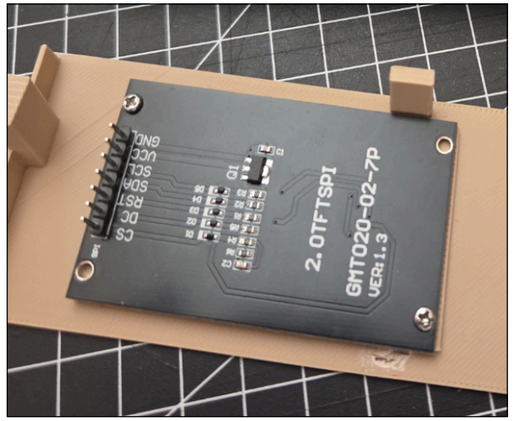
Nello schema riportato sul sito, trovate i collegamenti (sono esclusi per semplicità solo gli ovvi Vcc e GND rispettivamente collegati ai pin 3V3 (3.3 V) e GND dell'ESP32-S3. Per mia comodità ho indicato anche i colori dei fili utilizzati, così da non fare (io) confusione durante l'assemblaggio finale.

Fili? In che ssenZo??? 🤔

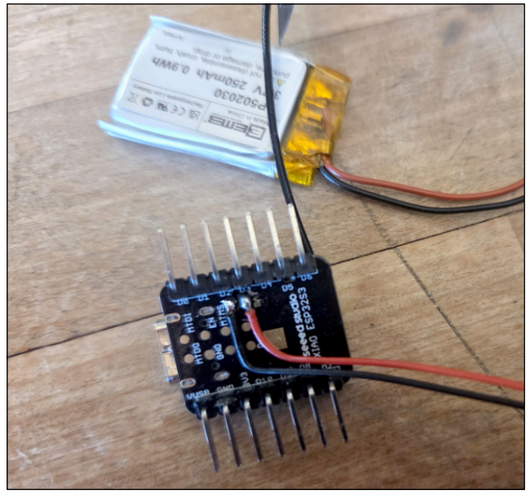
Sì, c'è un dettaglio sul quale non posso sorvolare, che a molti non sarà sfuggito osservando la prima foto. Per rimanere nei tempi, ma anche perché potrebbe esserci ancora qualche modifica ulteriore, non ho previsto da subito un PCB.

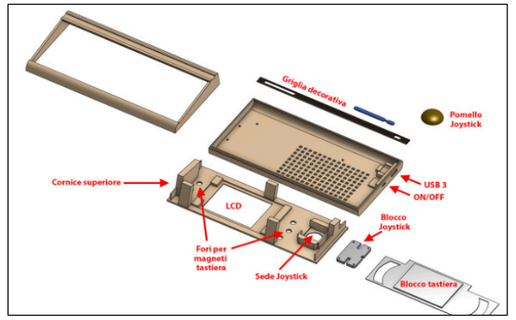
Anche perché inizialmente temevo di dovermi rimboccare troppo le maniche, visto che l'ultimo da me realizzato risale a qualche decennio fa, e all'epoca si procedeva - quando andava bene - con i trasferibili appositi e tanta pazienza.

Però posso anticipare che sto già familiarizzando con un CAD 3D online per i PCB - citofonare EasyEDA.com - e sono a buon punto,



Retro del display TFT 320x240 da due pollici e, a destra, il collegamento della batteria LiPo all'ESP32-S3 di XIAO





Le parti da stampare 3D. Manca qui la tastiera, recuperata sul Web.

pur distante ancora dalla meta. Gli ho dedicato finora un paio d'ore, e posso dire che è stato più semplice di quanto temessi.

Al momento i componenti elettronici sono tutti fissati alla cornice superiore: display e joystick con le viti previste, ESP32-S3 e ampli con semplice biadesivo gommato.

Per ridurre al massimo l'ingombro verticale ho dovuto adattare i classici connettori Dupont dandogli una forma a L, non prima di aver eliminato soltanto la parte in plastica e utilizzato al suo posto della normale guaina restringente.

Alta sartoria! 👀

Stampa 3D

Direi di chiudere in bellezza (?) con qualche dettaglio sugli STL per la stampa 3D del piccolo case.

Come dicevo la tastiera l'ho, per mia fortuna, trovata già bell'e fatta online. Non ricordo dove, ma l'ho scaricata immediatamente e scalata alle dimensioni di cui avevo bisogno.

Ovvero *q.b.* per coprire l'intero display, che si espande in verticale fino alla barra spaziatrice.

Questo mi ha permesso di rispettare le proporzioni originarie del C64, quello vero, perfino con la misura "64" stessa da qualche parte. Infatti la sua impronta, non a caso, è 125x64 mm 🤔

(*caxxo ridete????* 🤣)

Idiozie mie, lo ammetto, a parte, torniamo a parlare di cose ben più serie. Come mostrato nell'immagine bisogna stampare, oltre alla tastierina, ben otto parti.

Due, facilmente riconoscibili, costituiscono il cabinet vero e proprio, privo della cornice superiore, e pertanto il terzo pezzo importante è quest'ultima, ovvero il supporto per tutta l'elettronica: l'LDC e il joystick, ma come ho anticipato anche per l'ESP32S3 e l'ampli audio. Le altre parti, più piccole, riguardano il blocco per il joystick, quello per la tastiera, il pomello e la griglia decorativa, a sua volta composta da una parte principale (da stampare beige come tutto il resto) e i due piccoli intarsi qui mostrati in blu, da stampare marrone scuro come la stessa tastiera e il pomello del joystick.

La griglia decorativa beige e i due intarsi marroni, vanno incollati nella loro sede con semplice colla epossidica (Attack e simili), idem per il blocco tastiera con quest'ultima.

Le aree curve ai due estremi del blocco tastiera hanno quella forma perché come parte metallica (sensibile al magnetismo del neodimio) ho usato due fette di un anello metallico, peraltro già dotato di biadesivo sottile, di quelli per rendere *magsafe-compatibile* qualsiasi cover di qualsiasi cellulare.

Ho previsto i fori passanti per le viti ma non quelli dove le stesse dovranno avvitarsi.

Questo perché con le tolleranze in gioco potevano risultare più larghi del previsto (parliamo di meno di un millimetro di diametro, sono micro-viti) e pertanto bisogna praticarli a mano, sovrapponendo le parti prima, con un piccolo trapanino manuale, di quelli che si usano a mo' di giravite. Non certo con il Parkside a percussione di turno.

```
"""


--------------------------------------------------------------------------------

MicroC64 BLE/USB Control Center
Utility di controllo per il #MicroC64, l'emulatore C64 tascabile basato su XIAO ESP32-S3.

Autore: Andrea de Prisco - AdP
Info su: digiTANTO.it
SW distribuito con licenza Creative Commons
Attribuzione - Non commerciale 4.0 Internazionale (CC BY-NC 4.0)

Testo completo della licenza:
https://creativecommons.org/licenses/by-nc/4.0/deed.it

--------------------------------------------------------------------------------



Funzionalità principali:
  - Connessione BLE al dispositivo (nome: THMIC64)
  - Tastiera virtuale via BLE (tasti funzione, caratteri, shortcut)
  - Joystick USB via BLE (polling pygame, invio stato assi e pulsanti)
  - Trasferimento file .prg/.raw via BLE (singolo file) o USB (flash completo)
  - Esecuzione remota di programmi presenti nella flash XIAO
  - Editor testo con conversione automatica codici mnemonici VICE → PETSCII
  - Menu contestuale su entrambe le liste file

Protocollo BLE (comandi inviati alla caratteristica CHAR_UUID):
  [byte1, byte2, byte3]  → pressione tasto (matrice CIA1 del C64)
  0x77, val              → valore joystick (bit 0-4: su/giù/sx/dx/fire, attivi bassi)
  0x78                   → toggle porta joystick (1 ↔ 2)
  0x79, 0x01/0x00        → joystick USB connesso / scollegato
  0x7A, nome, 0x00       → avvia programma per nome (RUN remoto)
  0x81, bytes...         → iniezione diretta PETSCII nel buffer tastiera C64
  0xAA, nome, 0x00       → apri file per upload
  0xBB, data             → chunk dati file
  0xCC                   → chiudi file (fine upload)
  0xDD                   → richiedi lista file (DIR)
  0xEE, nome, 0x00       → elimina file
  0xFF                   → formatta filesystem

"""

import os
# Evita la creazione di finestre SDL superflue: serve solo il polling del joystick
os.environ["SDL_VIDEODRIVER"] = "dummy"

import customtkinter as ctk
import asyncio
import threading
import subprocess
import time
import queue
import serial
import shutil
import tkinter as tk
import serial.tools.list_ports
from bleak import BleakClient, BleakScanner
from tkinter import messagebox
from PIL import Image

# --- SUPPORTO JOYSTICK USB (pygame opzionale) ---
# Tenta prima pygame standard, poi pygame-ce (Community Edition)
JOY_SUPPORT = False
try:
    import pygame
    JOY_SUPPORT = True
except ImportError:
```



```
    try:
        import pygame_ce as pygame
        JOY_SUPPORT = True
    except ImportError:
        JOY_SUPPORT = False

# ==============================================================================
# CONFIGURAZIONE — adattare alle proprie impostazioni di sistema
# ==============================================================================
DEVICE_NAME    = "THMIC64"                      # Nome BLE del dispositivo
SERVICE_UUID   = "695ba701-a48c-43f6-9028-3c885771f19f"
CHAR_UUID      = "3b05e9bf-086f-4b56-9c37-7b7eeb30b28b"
LITTLEFS_OFFSET = "0x670000"                    # Offset flash per LittleFS
LITTLEFS_SIZE   = "1507328"                     # Dimensione partizione LittleFS (byte)

# <<<<<<<<<<<<< SPECIFICARE I TRE PERCORSI CORRETTI >>>>>>>>>>>>>>>>>>

DATA_DIR = r"data" # <<<<<<<<<<<<<<<<<<<<<<<< SPECIFICARE QUI IL PERCORSO COMPLETO DELLA PROPRIA
CARTELLA LOCALE CON I FILE PRG

MKLITTLEFS = r"***percorso_completo***" # SPECIFICARE QUI LA POSIZIONE DI mklittlefs.exe
ESPTOOL    = r"***percorso_completo***" # SPECIFICARE QUI LA POSIZIONE DI esptool.exe

# ---------------------------------------------------
# ------ CONFIGURAZIONE AUTOMATICA PORTA SERIALE ------
# ---------------------------------------------------

def find_esp32_port():
    import tkinter.simpledialog as sd
    import tkinter as tk

    candidates = [p for p in serial.tools.list_ports.comports() if p.vid == 0x303A]

    if len(candidates) == 0:
        root = tk.Tk(); root.withdraw()
        tk.messagebox.showerror("MicroC64",
            "Nessun dispositivo ESP32 rilevato.\n"
            "Collega il MicroC64 via USB e riavvia.")
        root.destroy()
        return None

    if len(candidates) == 1:
        return candidates[0].device  # Rilevamento automatico

    # Più dispositivi: chiedi all'utente
    root = tk.Tk(); root.withdraw()
    port_list = "\n".join([f"{p.device} — {p.description}" for p in candidates])
    result = sd.askstring("MicroC64 — Selezione porta",
        f"Rilevati più dispositivi ESP32.\n"
        f"Inserisci la porta del MicroC64:\n\n{port_list}",
        initialvalue=candidates[0].device)
    root.destroy()
    return result


COM_PORT = find_esp32_port()
if COM_PORT is None:
    exit()

# --- PALETTE COLORI UI (ispirata al C64) ---
C64_BLUE_LIGHT = "#A0A0FF"
C64_BLUE_DARK  = "#3030B0"
C64_BLUE_BG    = "#4040E0"
JOY_COLOR_OFF  = "#FFF9C4"   # Giallo tenue: joystick USB non rilevato
JOY_COLOR_ON   = "#FFFF00"   # Giallo acceso: joystick USB attivo

# ==============================================================================
# MAPPA TASTIERA C64 → MATRICE CIA1
# Ogni entry è una lista di 3 byte: [DC00, DC01, SHIFT]
# DC00 e DC01 rappresentano le righe/colonne della matrice tastiera del C64.
```

```
# Il terzo byte indica se il tasto SHIFT è premuto (0x01) o no (0x00).
# Le lettere minuscole (a-z) inviano il carattere senza SHIFT.
# Le lettere maiuscole (A-Z) inviano il carattere con SHIFT attivo,
# producendo i caratteri grafici del set 2 (es. per le abbreviazioni BASIC).
# ============================================================================
KEY_MAP = {
    'a': [0xFD, 0xFB, 0x00], 'b': [0xF7, 0xEF, 0x00], 'c': [0xFB, 0xEF, 0x00], 'd': [0xFB, 0xFB, 0x00],
    'e': [0xFD, 0xBF, 0x00], 'f': [0xFB, 0xDF, 0x00], 'g': [0xF7, 0xFB, 0x00], 'h': [0xF7, 0xDF, 0x00],
    'i': [0xEF, 0xFD, 0x00], 'j': [0xEF, 0xFB, 0x00], 'k': [0xEF, 0xDF, 0x00], 'l': [0xDF, 0xFB, 0x00],
    'm': [0xEF, 0xEF, 0x00], 'n': [0xEF, 0x7F, 0x00], 'o': [0xEF, 0xBF, 0x00],
    'p': [0xDF, 0xFD, 0x00], 'q': [0x7F, 0xBF, 0x00], 'r': [0xFB, 0xFD, 0x00], 's': [0xFD, 0xDF, 0x00],
    't': [0xFB, 0xBF, 0x00], 'u': [0xF7, 0xBF, 0x00], 'v': [0xF7, 0x7F, 0x00], 'w': [0xFD, 0xFD, 0x00],
    'x': [0xFB, 0x7F, 0x00], 'y': [0xF7, 0xFD, 0x00], 'z': [0xFD, 0xEF, 0x00],
    # Lettere maiuscole = SHIFT+lettera (caratteri grafici C64, usati per abbreviazioni BASIC)
    'A': [0xFD, 0xFB, 0x01], 'B': [0xF7, 0xEF, 0x01], 'C': [0xFB, 0xEF, 0x01], 'D': [0xFB, 0xFB, 0x01],
    'E': [0xFD, 0xBF, 0x01], 'F': [0xFB, 0xDF, 0x01], 'G': [0xF7, 0xFB, 0x01], 'H': [0xF7, 0xDF, 0x01],
    'I': [0xEF, 0xFD, 0x01], 'J': [0xEF, 0xFB, 0x01], 'K': [0xEF, 0xDF, 0x01], 'L': [0xDF, 0xFB, 0x01],
    'M': [0xEF, 0xEF, 0x01], 'N': [0xEF, 0x7F, 0x01], 'O': [0xEF, 0xBF, 0x01],
    'P': [0xDF, 0xFD, 0x01], 'Q': [0x7F, 0xBF, 0x01], 'R': [0xFB, 0xFD, 0x01], 'S': [0xFD, 0xDF, 0x01],
    'T': [0xFB, 0xBF, 0x01], 'U': [0xF7, 0xBF, 0x01], 'V': [0xF7, 0x7F, 0x01], 'W': [0xFD, 0xFD, 0x01],
    'X': [0xFB, 0x7F, 0x01], 'Y': [0xF7, 0xFD, 0x01], 'Z': [0xFD, 0xEF, 0x01],
    # Cifre
    '1': [0x7F, 0xFE, 0x00], '2': [0x7F, 0xF7, 0x00], '3': [0xFD, 0xFE, 0x00], '4': [0xFD, 0xF7, 0x00],
    '5': [0xFB, 0xFE, 0x00], '6': [0xFB, 0xF7, 0x00], '7': [0xF7, 0xFE, 0x00], '8': [0xF7, 0xF7, 0x00],
    '9': [0xEF, 0xFE, 0x00], '0': [0xEF, 0xF7, 0x00],
    # Punteggiatura e operatori
    '.': [0xDF, 0xEF, 0x00], ',': [0xDF, 0x7F, 0x00], ':': [0xDF, 0xDF, 0x00], ';': [0xBF, 0xFB, 0x00],
    '=': [0xBF, 0xDF, 0x00], '+': [0xDF, 0xFE, 0x00], '-': [0xDF, 0xF7, 0x00], '*': [0xBF, 0xFD, 0x00],
    '/': [0xBF, 0x7F, 0x00], ' ': [0x7F, 0xEF, 0x00],
    # Caratteri speciali (con SHIFT)
    '!': [0x7F, 0xFE, 0x01], '"': [0x7F, 0xF7, 0x01], '£': [0xBF, 0xFE, 0x00],
    '$': [0xFD, 0xF7, 0x01], '%': [0xFB, 0xFE, 0x01], '&': [0xFB, 0xF7, 0x01],
    '(': [0xF7, 0xF7, 0x01], ')': [0xEF, 0xFE, 0x01], '?': [0xBF, 0x7F, 0x01],
    '^': [0xBF, 0xBF, 0x00], '"': [0xF7, 0xFE, 0x01],
    '<': [0xDF, 0x7F, 0x01], '>': [0xDF, 0xEF, 0x01],
    # Tasti speciali
    'Return': [0xFE, 0xFD, 0x00], 'BackSpace': [0xFE, 0xFE, 0x00],
    'Right':  [0xFE, 0xFB, 0x00], 'Left':  [0xFE, 0xFB, 0x01],
    'Down':   [0xFE, 0x7F, 0x00], 'Up':    [0xFE, 0x7F, 0x01],
    'Escape': [0x7F, 0x7F, 0x00], 'Home':  [0xBF, 0xF7, 0x00], 'CLR': [0xBF, 0xF7, 0x01],
    # Tasti funzione (F1-F8: pari = SHIFT+dispari sul C64)
    'F1': [0xFE, 0xEF, 0x00], 'F2': [0xFE, 0xEF, 0x01],
    'F3': [0xFE, 0xDF, 0x00], 'F4': [0xFE, 0xDF, 0x01],
    'F5': [0xFE, 0xBF, 0x00], 'F6': [0xFE, 0xBF, 0x01],
    'F7': [0xFE, 0xF7, 0x00], 'F8': [0xFE, 0xF7, 0x01],
}


class C64ControlApp(ctk.CTk):
    """Finestra principale dell'utility di controllo MicroC64."""

    def __init__(self):
        super().__init__()
        self.title("#MicroC64 BLE/USB Utility")
        self.geometry("1100x850")
        self.configure(fg_color=C64_BLUE_BG)

        # --- Stato connessione BLE ---
        self.client = None                      # Client BLE attivo (BleakClient)
        self.loop   = asyncio.new_event_loop()  # Event loop asyncio dedicato al BLE
        self.ble_queue = queue.Queue()          # Coda thread-safe per i pacchetti BLE

        # --- Stato UI ---
        self.xiao_files_raw   = ""              # Lista grezza ricevuta dalla XIAO via seriale
        self.refreshing_usb   = False           # Flag per evitare refresh USB concorrenti
        self.joy_port         = "2"             # Porta joystick attiva (1 o 2)
        self.last_joy_val     = 0xFF            # Ultimo valore joystick inviato (evita invii ridondanti)
        self.pc_buttons       = {}              # Riferimenti ai pulsanti lista PC
        self.xiao_buttons     = {}              # Riferimenti ai pulsanti lista XIAO
        self.selected_pc_file = None
```



```
        self.selected_xiao_full_text = None
        self.selected_xiao_name      = None

        self.setup_ui()
        self.update_pc_list()

        # --- Thread di background ---
        threading.Thread(target=self.run_async_loop,      daemon=True).start()  # Event loop BLE
        threading.Thread(target=self.ble_worker,          daemon=True).start()  # Invio pacchetti BLE
        if JOY_SUPPORT:
            threading.Thread(target=self.joystick_poll_worker, daemon=True).start()  # Polling joystick USB

        # Avvio differito: lascia il tempo alla UI di renderizzarsi prima del primo refresh
        self.after(500, self.initial_startup)

    # ========================================================================
    # COSTRUZIONE INTERFACCIA
    # ========================================================================

    def setup_ui(self):
        """Costruisce tutti i widget dell'interfaccia grafica."""
        img_path = os.path.join(os.path.dirname(os.path.realpath(__file__)), "PNG")

        # Caricamento icone (normale + hover per effetto rollover)
        try:
            self.img_usb_norm  = ctk.CTkImage(Image.open(os.path.join(img_path, "FrecciaALL_norm.png")),  size=(50, 50))
            self.img_usb_hover = ctk.CTkImage(Image.open(os.path.join(img_path, "FrecciaALL_hover.png")), size=(50, 50))
            self.img_ble_norm  = ctk.CTkImage(Image.open(os.path.join(img_path, "Freccia1_norm.png")),    size=(50, 50))
            self.img_ble_hover = ctk.CTkImage(Image.open(os.path.join(img_path, "Freccia1_hover.png")),   size=(50, 50))
            self.img_del_norm  = ctk.CTkImage(Image.open(os.path.join(img_path, "Del_norm.png")),         size=(50, 50))
            self.img_del_hover = ctk.CTkImage(Image.open(os.path.join(img_path, "Del_hover.png")),        size=(50, 50))
            self.img_run_norm  = ctk.CTkImage(Image.open(os.path.join(img_path, "Run_norm.png")),         size=(50, 50))
            self.img_run_hover = ctk.CTkImage(Image.open(os.path.join(img_path, "Run_hover.png")),        size=(50, 50))
        except Exception as e:
            print(f"Errore caricamento icone: {e}")
            self.img_usb_norm = self.img_usb_hover = None
            self.img_ble_norm = self.img_ble_hover = None
            self.img_del_norm = self.img_del_hover = None
            self.img_run_norm = self.img_run_hover = None

        self.grid_columnconfigure(0, weight=1)

        # --- Barra superiore: tasti funzione C64 + pulsante joystick + editor testo ---
        self.top_bar = ctk.CTkFrame(self, fg_color=C64_BLUE_DARK, border_width=2, border_color=C64_BLUE_DARK)
        self.top_bar.grid(row=0, column=0, padx=20, pady=10, sticky="ew")
        self.top_bar.grid_columnconfigure(0, weight=1)
        self.top_bar.grid_columnconfigure(12, weight=1)

        # Tasti funzione F1-F8 + CLR (i tasti pari del C64 si ottengono con SHIFT+dispari)
        keys = ["F1", "F3", "F5", "F7", "F2", "F4", "F6", "F8", "CLR"]
        for i, k in enumerate(keys):
            ctk.CTkButton(self.top_bar, text=k, width=50, height=35,
                          fg_color=C64_BLUE_LIGHT, text_color=C64_BLUE_DARK,
                          font=("Arial", 14, "bold"),
                          command=lambda x=k: self.enqueue_key(x)).grid(row=0, column=i+1, padx=4, pady=10)

        # Pulsante joystick: mostra la porta attiva (1/2), giallo quando connesso
        self.btn_joy = ctk.CTkButton(self.top_bar, text="", width=35, height=35,
                                     fg_color=JOY_COLOR_OFF, text_color=C64_BLUE_DARK,
                                     font=("Arial", 16, "bold"), command=self.toggle_joy)
        self.btn_joy.grid(row=0, column=len(keys)+1, padx=4)

        # Pulsante per aprire l'editor testo multiriga
        ctk.CTkButton(self.top_bar, text="TXT", width=50, height=35,
                      fg_color="#2E8B57", text_color="white",
                      font=("Arial", 14, "bold"), command=self.open_txt_editor).grid(
                      row=0, column=len(keys)+2, padx=4)

        # --- Barra intermedia: casella testo + pulsanti reset/format ---
        self.mid_frame = ctk.CTkFrame(self, fg_color="transparent")
        self.mid_frame.grid(row=1, column=0, padx=20, pady=5, sticky="ew")
```

```
self.mid_frame.grid_columnconfigure(0, weight=1)

# Campo testo per inserimento/incolla righe BASIC singole
self.paste_area = ctk.CTkEntry(self.mid_frame,
                               placeholder_text="Incolla comandi testo qui...",
                               height=40, font=("Consolas", 14),
                               fg_color="#101040", text_color=C64_BLUE_LIGHT)
self.paste_area.grid(row=0, column=0, padx=(0, 20), sticky="ew")
self.paste_area.bind("<Return>", lambda e: self._send_and_refocus())

ctk.CTkButton(self.mid_frame, text="SOFT RESET", width=130, height=40,
              fg_color="#FF0000", text_color="white", font=("Arial", 14, "bold"),
              command=self.reset_cmd).grid(row=0, column=1, padx=5)
ctk.CTkButton(self.mid_frame, text="RDISK FORMAT", width=130, height=40,
              fg_color="#FF0000", text_color="white", font=("Arial", 14, "bold"),
              command=self.format_flash).grid(row=0, column=2, padx=5)

# --- Area file: lista PC (sinistra), pulsanti centrali, lista XIAO (destra) ---
self.file_frame = ctk.CTkFrame(self, fg_color=C64_BLUE_DARK, border_width=2, border_color=C64_BLUE_DARK)
self.file_frame.grid(row=2, column=0, padx=20, pady=10, sticky="nsew")
self.grid_rowconfigure(2, weight=1)
self.file_frame.grid_columnconfigure((0, 2), weight=1)
self.file_frame.grid_rowconfigure(1, weight=1)

self.pc_list_frame = ctk.CTkScrollableFrame(self.file_frame, fg_color="#101040",
                                            label_text="PC LOCAL DATA FOLDER",
                                            label_text_color="#404000")
self.pc_list_frame.grid(row=1, column=0, padx=15, pady=15, sticky="nsew")

self.xiao_list_frame = ctk.CTkScrollableFrame(self.file_frame, fg_color="#101040",
                                              label_text="XIAO REMOTE FOLDER",
                                              label_text_color="#404000")
self.xiao_list_frame.grid(row=1, column=2, padx=15, pady=15, sticky="nsew")

# Colonna centrale con i pulsanti di trasferimento
self.trans_mid = ctk.CTkFrame(self.file_frame, fg_color="transparent")
self.trans_mid.grid(row=1, column=1)

# Pulsante: flash totale via USB (mklittlefs + esptool)
self.btn_usb = ctk.CTkButton(self.trans_mid, text="", image=self.img_usb_norm,
                             width=50, height=50, fg_color="transparent",
                             hover_color=C64_BLUE_DARK, command=self.usb_upload)
self.btn_usb.pack(pady=10)
self.btn_usb.bind("<Enter>", lambda e: self.btn_usb.configure(image=self.img_usb_hover))
self.btn_usb.bind("<Leave>", lambda e: self.btn_usb.configure(image=self.img_usb_norm))

# Pulsante: upload singolo file via BLE
self.btn_ble = ctk.CTkButton(self.trans_mid, text="", image=self.img_ble_norm,
                             width=50, height=50, fg_color="transparent",
                             hover_color=C64_BLUE_DARK, command=self.ble_upload)
self.btn_ble.pack(pady=10)
self.btn_ble.bind("<Enter>", lambda e: self.btn_ble.configure(image=self.img_ble_hover))
self.btn_ble.bind("<Leave>", lambda e: self.btn_ble.configure(image=self.img_ble_norm))

# Pulsante: avvia programma selezionato nella lista XIAO
self.btn_run = ctk.CTkButton(self.trans_mid, text="", image=self.img_run_norm,
                             width=50, height=50, fg_color="transparent",
                             hover_color=C64_BLUE_DARK, command=self.run_cmd)
self.btn_run.pack(pady=10)
self.btn_run.bind("<Enter>", lambda e: self.btn_run.configure(image=self.img_run_hover))
self.btn_run.bind("<Leave>", lambda e: self.btn_run.configure(image=self.img_run_norm))

# Pulsante: elimina file selezionato dalla flash XIAO
self.btn_del = ctk.CTkButton(self.trans_mid, text="", image=self.img_del_norm,
                             width=50, height=50, fg_color="transparent",
                             hover_color=C64_BLUE_DARK, command=self.delete_cmd)
self.btn_del.pack(pady=10)
self.btn_del.bind("<Enter>", lambda e: self.btn_del.configure(image=self.img_del_hover))
self.btn_del.bind("<Leave>", lambda e: self.btn_del.configure(image=self.img_del_norm))
```



```
    # --- Barra di stato inferiore ---
    self.status_bar = ctk.CTkFrame(self, height=40, fg_color="#202020")
    self.status_bar.grid(row=3, column=0, sticky="ew")

    # Percentuale avanzamento trasferimento file
    self.lbl_percent = ctk.CTkLabel(self.status_bar, text="0%",
                                    text_color=C64_BLUE_LIGHT,
                                    font=("Courier New", 18, "bold"), width=80)
    self.lbl_percent.pack(side="left", padx=20, pady=5)

    # Stato connessione BLE
    self.lbl_status = ctk.CTkLabel(self.status_bar, text="OFFLINE",
                                   text_color="#FF4444", font=("Arial", 12, "bold"))
    self.lbl_status.pack(side="left", padx=10, pady=5)

    ctk.CTkButton(self.status_bar, text="LIST REFRESH", width=100, height=28,
                  fg_color=C64_BLUE_DARK, text_color="white", font=("Arial", 12, "bold"),
                  command=self.manual_refresh).pack(side="right", padx=10)
    ctk.CTkButton(self.status_bar, text="RECONNECT", width=100, height=28,
                  fg_color="#D35400", text_color="white", font=("Arial", 12, "bold"),
                  command=self.force_reconnect).pack(side="right", padx=10)

    # Binding globali
    self.bind_all("<Button-1>", self.check_deselect)  # Deseleziona cliccando fuori
    self.bind("<Key>", self.on_key)                   # Tastiera fisica → BLE

# ============================================================================
# JOYSTICK USB (polling continuo via pygame)
# ============================================================================

def joystick_poll_worker(self):
    """
    Thread che monitora il joystick USB collegato al PC.
    Invia lo stato degli assi e dei pulsanti via BLE ogni volta che cambia.
    Il valore joystick è un byte con i bit attivi bassi (convenzione C64):
      bit 0 = su, bit 1 = giù, bit 2 = sinistra, bit 3 = destra, bit 4 = fire
    """
    pygame.init()
    pygame.joystick.init()
    active_joy = None
    while True:
        time.sleep(0.05)
        pygame.event.pump()
        count = pygame.joystick.get_count()
        if count > 0:
            if active_joy is None:
                active_joy = pygame.joystick.Joystick(0)
                self.enqueue_raw(bytes([0x79, 0x01]))   # Notifica al firmware: JS USB connesso
                self.after(0, lambda: self.btn_joy.configure(text=self.joy_port, fg_color=JOY_COLOR_ON))

            # Leggi assi e pulsanti, costruisci il byte di stato
            val = 0xFF
            if active_joy.get_axis(1) < -0.5: val &= ~1    # Su
            if active_joy.get_axis(1) >  0.5: val &= ~2    # Giù
            if active_joy.get_axis(0) < -0.5: val &= ~4    # Sinistra
            if active_joy.get_axis(0) >  0.5: val &= ~8    # Destra
            if any(active_joy.get_button(b) for b in range(min(active_joy.get_numbuttons(), 4))):
                val &= ~16                                 # Fire

            # Invia solo se il valore è cambiato (ottimizzazione banda BLE)
            if val != self.last_joy_val:
                self.enqueue_raw(bytes([0x77, val]))
                self.last_joy_val = val
        else:
            if active_joy is not None:
                self.enqueue_raw(bytes([0x79, 0x00]))   # Notifica al firmware: JS USB scollegato
                active_joy = None
                self.after(0, lambda: self.btn_joy.configure(text="", fg_color=JOY_COLOR_OFF))

# ============================================================================
# GESTIONE BLE
```

```
# ========================================================================

def ble_worker(self):
    """
    Thread dedicato all'invio dei pacchetti BLE.
    Consuma la coda ble_queue e invia ogni pacchetto alla caratteristica GATT.
    Usare una coda garantisce che i pacchetti siano inviati in ordine
    e che il thread UI non si blocchi mai in attesa del BLE.
    """
    while True:
        data = self.ble_queue.get()
        if self.client and self.client.is_connected:
            try:
                future = asyncio.run_coroutine_threadsafe(
                    self.client.write_gatt_char(CHAR_UUID, data, response=True),
                    self.loop)
                future.result()
            except:
                pass
        self.ble_queue.task_done()

def enqueue_raw(self, data):
    """Inserisce un pacchetto grezzo (bytes) nella coda BLE."""
    self.ble_queue.put(data)

def enqueue_key(self, k):
    """Invia la pressione di un tasto tramite la mappa CIA1."""
    if k in KEY_MAP:
        self.enqueue_raw(bytes(KEY_MAP[k]))

def on_key(self, event):
    """
    Intercetta i tasti fisici della tastiera del PC e li invia al C64 via BLE.
    Ignorato se il focus è sulla casella testo (l'utente sta digitando localmente).
    """
    if not self.client:
        return
    if self.focus_get() == self.paste_area._entry:
        return
    key = event.keysym
    if len(event.char) == 1 and event.keysym not in ["Return", "BackSpace", "Tab", "Escape"]:
        key = event.char
    self.enqueue_key(key)

# ========================================================================
# CONVERSIONE TESTO → PETSCII
# ========================================================================

# Codici mnemonici VICE → valori PETSCII corrispondenti.
# Usati nei listati esportati da VICE per rappresentare caratteri di controllo
# e colori con sequenze leggibili come {down}, {rvon}, {red}, ecc.
VICE_MAP = {
    '{down}': 17, '{up}': 145, '{left}': 157, '{rght}': 29,
    '{home}': 19, '{clr}': 147, '{inst}': 148, '{del}': 20,
    '{rvon}': 18, '{rvof}': 146, '{blk}': 144, '{wht}': 5,
    '{red}': 28, '{cyn}': 159, '{pur}': 156, '{grn}': 30,
    '{blu}': 31, '{yel}': 158, '{f1}': 133, '{f2}': 137,
    '{f3}': 134, '{f4}': 138, '{f5}': 135, '{f6}': 139,
    '{f7}': 136, '{f8}': 140, '{return}': 13, '{space}': 32,
    '{orng}': 129, '{brn}': 149, '{lred}': 150, '{gry1}': 151,
    '{gry2}': 152, '{lgrn}': 153, '{lblu}': 154, '{gry3}': 155,
    '{pi}': 255, '{pound}': 92,
}

# Abbreviazioni BASIC C64: ogni keyword viene sostituita con la sequenza
# di byte PETSCII corrispondente all'abbreviazione da tastiera.
# Sul C64 le abbreviazioni usano SHIFT+lettera (es. fO = FOR, nE = NEXT).
# I byte > 127 rappresentano SHIFT+lettera (codice = 128 + ASCII maiuscolo).
# Le keyword più lunghe sono cercate per prime per evitare falsi positivi
# (es. 'print#' deve essere trovato prima di 'print').
```



```
BASIC_ABBREV = {
    'gosub':  [71, 79, 211], 'restore': [82, 69, 211], 'return': [82, 69, 212],
    'left$':  [76, 69, 198], 'right$':  [82, 201],     'print#': [80, 210],
    'step':   [83, 84, 197], 'str$':    [83, 84, 210], 'input#': [73, 206],
    'close':  [67, 76, 207], 'chr$':    [67, 200],     'spc(':   [83, 208],
    'tab(':   [84, 193],     'mid$':    [77, 201],     'abs':    [65, 194],
    'and':    [65, 206],     'asc':     [65, 211],     'atn':    [65, 212],
    'clr':    [67, 204],     'cmd':     [67, 205],     'cont':   [67, 207],
    'data':   [68, 193],     'def':     [68, 197],     'dim':    [68, 201],
    'end':    [69, 206],     'exp':     [69, 216],     'for':    [70, 207],
    'fre':    [70, 210],     'goto':    [71, 207],     'let':    [76, 197],
    'list':   [76, 201],     'load':    [76, 207],     'next':   [78, 197],
    'not':    [78, 207],     'open':    [79, 208],     'peek':   [80, 197],
    'poke':   [80, 207],     'print':   [63],          'read':   [82, 197],
    'rnd':    [82, 206],     'run':     [82, 213],     'save':   [83, 193],
    'sgn':    [83, 199],     'sin':     [83, 201],     'sqr':    [83, 209],
    'stop':   [83, 212],     'sys':     [83, 217],     'then':   [84, 200],
    'usr':    [85, 211],     'val':     [86, 193],     'verify': [86, 197],
    'wait':   [87, 193],
}

def build_petscii_line(self, text):
    """
    Converte una riga di testo in una sequenza di byte PETSCII pronti per
    essere iniettati nel buffer tastiera del C64.

    Trasformazioni applicate in ordine:
    1. Keyword BASIC (es. 'for' → [70, 207]) — solo fuori dalle virgolette
    2. Codici mnemonici VICE (es. '{rvon}' → 18)
    3. Conversione ASCII → PETSCII per le lettere
       (minuscole a-z → maiuscole PETSCII 65-90,
        maiuscole A-Z → caratteri grafici PETSCII 97-122)
    4. Aggiunta ':rem' se la lunghezza causa problemi BLE (workaround)
    5. Aggiunta Return finale (byte 13)

    Nota sul workaround BLE: pacchetti di esattamente 19 byte (18 dati + 1
    byte comando 0x81) causano corruzione dati per un limite del firmware.
    Aggiungere ':rem' allunga la riga di 4 byte, uscendo dalla lunghezza
    problematica. Il BASIC C64 ignora il contenuto dopo REM.
    """
    result    = bytearray()
    i         = 0
    text_lower = text.lower()
    in_quotes  = False  # Traccia se siamo dentro una stringa tra virgolette

    while i < len(text):
        found = False

        # 1. Cerca abbreviazione BASIC (solo fuori dalle virgolette)
        if not in_quotes:
            for kw in sorted(self.BASIC_ABBREV.keys(), key=len, reverse=True):
                if text_lower[i:i+len(kw)] == kw:
                    result.extend(self.BASIC_ABBREV[kw])
                    i += len(kw)
                    found = True
                    break
        if found:
            continue

        # 2. Cerca codice mnemonico VICE (case insensitive)
        for code, val in self.VICE_MAP.items():
            if text[i:i+len(code)].lower() == code:
                result.append(val)
                i += len(code)
                found = True
                break

        # 3. Carattere singolo → conversione ASCII/PETSCII
        if not found:
            c = text[i]
            if c == '"':
```

```
            in_quotes = not in_quotes   # Toggle stato stringa
        o = ord(c)
        if 'a' <= c <= 'z':
            result.append(o - 32)       # Minuscolo → PETSCII maiuscolo (65-90)
        elif 'A' <= c <= 'Z':
            result.append(o + 32)       # Maiuscolo → PETSCII grafico (97-122)
        else:
            result.append(o)            # Altri caratteri: invariati
        i += 1

    # 4. Workaround BLE: aggiunge ':rem' se la lunghezza è problematica
    if len(result) % 18 == 1:
        for c in ':rem':
            result.append(ord(c) - 32 if 'a' <= c <= 'z' else ord(c))

    result.append(13)   # 5. Return finale
    return bytes(result)

def send_paste_text(self):
    """
    Invia il contenuto della casella testo al C64 via BLE (comando 0x81).
    Supporta più righe separate da newline (tipico di un incolla da editor).
    Ogni riga viene convertita in PETSCII e inviata in chunk da 18 byte
    con un delay di 400ms tra un chunk e l'altro per dare tempo al C64
    di consumare il buffer tastiera (un carattere ogni 200ms).
    """
    t = self.paste_area.get()
    if not t:
        return
    def run():
        lines = t.split('\n')
        for line in lines:
            if not line:
                continue
            petscii = self.build_petscii_line(line)
            for i in range(0, len(petscii), 18):
                chunk = petscii[i:i+18]
                self.enqueue_raw(bytes([0x81]) + chunk)
                time.sleep(0.4)
        self.after(0, self._focus_paste)
    threading.Thread(target=run, daemon=True).start()
    self.paste_area.delete(0, 'end')

# ============================================================================
# CONNESSIONE BLE E REFRESH
# ============================================================================

def run_async_loop(self):
    """Avvia l'event loop asyncio nel thread dedicato al BLE."""
    asyncio.set_event_loop(self.loop)
    self.loop.run_forever()

def start_ble_connection(self):
    """Avvia la connessione BLE in modo asincrono."""
    asyncio.run_coroutine_threadsafe(self.connect(), self.loop)

async def connect(self, silent=False):
    """
    Cerca il dispositivo BLE per nome e si connette.
    Attiva le notifiche per ricevere risposte dal firmware (es. esito delete).
    """
    if not silent:
        self.update_status("Searching...", "orange")
    device = await BleakScanner.find_device_by_name(DEVICE_NAME, timeout=5.0)
    if device:
        self.client = BleakClient(device)
        try:
            await self.client.connect()
            await self.client.start_notify(CHAR_UUID, self.on_data)
            self.update_status("CONNECTED", "green")
```



```
        except:
            self.update_status("FAILED", "red")
    else:
        self.update_status("NOT FOUND", "red")

def force_reconnect(self):
    """Disconnette e riconnette il BLE (utile in caso di perdita connessione)."""
    def run():
        self.update_status("RECONNECTING...", "orange")
        if self.client:
            try:
                asyncio.run_coroutine_threadsafe(self.client.disconnect(), self.loop)
            except:
                pass
        time.sleep(1)
        self.start_ble_connection()
    threading.Thread(target=run).start()

def initial_startup(self):
    """Avvio iniziale: refresh USB per ottenere la lista file + connessione BLE."""
    def run():
        self.perform_usb_refresh(auto_connect_ble=True)
    threading.Thread(target=run, daemon=True).start()

def manual_refresh(self):
    """Aggiorna la lista file PC e richiede la lista dalla XIAO via seriale."""
    self.update_pc_list()
    if not self.refreshing_usb:
        threading.Thread(target=self.perform_usb_refresh, daemon=True).start()

def perform_usb_refresh(self, auto_connect_ble=True):
    """
    Ottiene la lista file dalla XIAO tramite porta seriale.
    Il firmware, al boot, stampa la lista nel formato 'DIR:file1|file2|###'.
    Il refresh si attiva resettando la scheda via DTR/RTS (reset hardware).
    Attende fino a 6 secondi per ricevere i marcatori DIR: e ###.
    """
    self.refreshing_usb = True
    try:
        self.update_status("USB SCAN...", "orange")
        self.after(0, self.safe_ui_clear)
        with serial.Serial(COM_PORT, 115200, timeout=0.1) as ser:
            # Sequenza DTR/RTS per reset hardware ESP32
            ser.setDTR(False); ser.setRTS(True)
            time.sleep(0.15)
            ser.setDTR(True);  ser.setRTS(False)
            start_time, captured = time.time(), ""
            while (time.time() - start_time) < 6:
                if ser.in_waiting > 0:
                    chunk = ser.read(ser.in_waiting).decode('utf-8', errors='ignore')
                    captured += chunk
                    if "DIR:" in captured and "###" in captured:
                        self.xiao_files_raw = "DIR:" + captured.split("DIR:")[1].split("###")[0].strip()
                        self.after(0, self.safe_ui_refresh)
                        break
        if auto_connect_ble and not (self.client and self.client.is_connected):
            time.sleep(2.5)
            self.start_ble_connection()
        else:
            self.update_status("CONNECTED", "green")
    except:
        self.update_status("PORT BUSY", "red")
    finally:
        self.refreshing_usb = False

# ============================================================================
# COMANDI
# ============================================================================

def toggle_joy(self):
    """
```

```
    Alterna la porta joystick attiva (1 ↔ 2) e invia il comando 0x78 al firmware.
    La porta selezionata determina quale registro CIA1 viene usato
    ($DC00 per porta 2, $DC01 per porta 1).
    """
    self.joy_port = "1" if self.joy_port == "2" else "2"
    if self.btn_joy.cget("fg_color") == JOY_COLOR_ON:
        self.btn_joy.configure(text=self.joy_port)
    self.enqueue_raw(bytes([0x78]))

def format_flash(self):
    """Formatta l'intero filesystem LittleFS della XIAO (operazione irreversibile)."""
    if messagebox.askyesno("FORMAT", "Cancellare l'intero RDISK?"):
        def run():
            self.update_status("FORMATTING...", "orange")
            self.enqueue_raw(bytes([0xFF]))
            self.xiao_files_raw = ""
            time.sleep(5.0)
            self.manual_refresh()
        threading.Thread(target=run).start()

def reset_cmd(self):
    """
    Esegue un soft reset del C64 inviando il comando SYS 64738 via tastiera BLE.
    Equivalente alla pressione di RUN/STOP + RESTORE sul C64 originale.
    """
    if messagebox.askyesno("RESET", "Inviare SOFT RESET?"):
        def run():
            self.enqueue_key("Return")
            time.sleep(0.3)
            for c in "sys64738":
                if c in KEY_MAP:
                    self.enqueue_key(c)
                    time.sleep(0.1)
            self.enqueue_key("Return")
        threading.Thread(target=run).start()

def on_data(self, sender, data):
    """
    Riceve le notifiche BLE dal firmware.
    Attualmente gestisce le risposte al comando di eliminazione file (0xEE):
      'OK:nomefile'  → eliminazione riuscita
      'ERR:nomefile' → eliminazione fallita
    """
    try:
        msg = data.decode('utf-8', errors='ignore')
        if msg.startswith("OK:"):
            self.update_status(f"Eliminato: {msg[3:]}", "green")
        elif msg.startswith("ERR:"):
            self.update_status(f"Errore eliminazione: {msg[4:]}", "red")
    except:
        pass


# ============================================================================
# GESTIONE LISTE FILE
# ============================================================================

def safe_ui_clear(self):
    """Svuota la lista XIAO nell'interfaccia."""
    self.xiao_buttons = {}
    for w in list(self.xiao_list_frame.winfo_children()):
        w.destroy()

def safe_ui_refresh(self):
    """
    Popola la lista XIAO con i file ricevuti via seriale.
    I file con prefisso '-' (porta joystick 1) sono ordinati alfabeticamente
    ignorando il prefisso, in modo da apparire nella posizione corretta.
    """
    self.safe_ui_clear()
```



```
    clean = self.xiao_files_raw.replace("DIR:", "").strip()
    if not clean:
        return
    clean = "".join(c for c in clean if c.isprintable() or c == '|')
    entries = [f.strip() for f in clean.split("|") if f.strip()]
    entries.sort(key=lambda x: x.lstrip("-").lower())
    for ft in entries:
        btn = ctk.CTkButton(self.xiao_list_frame, text=ft, fg_color="transparent",
                            text_color=C64_BLUE_LIGHT, anchor="w", font=("Consolas", 16),
                            command=lambda x=ft: self.sel_xiao(x))
        btn.pack(fill="x")
        self.xiao_buttons[ft] = btn
        btn.bind("<Button-3>", lambda e, x=ft: self.show_context_menu(e, x))

def sel_xiao(self, ft):
    """Seleziona un file nella lista XIAO (evidenziazione visiva)."""
    self.deselect_all()
    self.selected_xiao_full_text = ft
    self.selected_xiao_name = ft.split("(")[0].strip()  # Rimuove la dimensione "(N bytes)"
    if ft in self.xiao_buttons:
        self.xiao_buttons[ft].configure(fg_color=C64_BLUE_LIGHT, text_color=C64_BLUE_DARK)

def update_pc_list(self):
    """Popola la lista PC con i file .prg e .raw presenti in DATA_DIR."""
    def build_pc():
        for w in list(self.pc_list_frame.winfo_children()):
            w.destroy()
        self.pc_buttons = {}
        if os.path.exists(DATA_DIR):
            files = [f for f in os.listdir(DATA_DIR)
                     if f.lower().endswith('.prg') or f.lower().endswith('.raw')]
            for f in sorted(files):
                size    = os.path.getsize(os.path.join(DATA_DIR, f))
                display = f"{f} ({size})"
                btn = ctk.CTkButton(self.pc_list_frame, text=display, fg_color="transparent",
                                    text_color=C64_BLUE_LIGHT, anchor="w", font=("Consolas", 16),
                                    command=lambda x=f: self.sel_pc(x))
                btn.pack(fill="x")
                self.pc_buttons[f] = btn
                btn.bind("<Button-3>", lambda e, x=f: self.show_pc_context_menu(e, x))
    self.after(0, build_pc)

def sel_pc(self, f):
    """Seleziona un file nella lista PC (evidenziazione visiva)."""
    self.deselect_all()
    self.selected_pc_file = f
    if f in self.pc_buttons:
        self.pc_buttons[f].configure(fg_color=C64_BLUE_LIGHT, text_color=C64_BLUE_DARK)


# ============================================================================
# TRASFERIMENTO FILE
# ============================================================================

def ble_upload(self):
    """
    Trasferisce il file selezionato dalla lista PC alla flash XIAO via BLE.
    Protocollo: 0xAA (apri) → 0xBB (chunk 20 byte) → 0xCC (chiudi).
    Il nome file viene normalizzato: minuscolo, max 16 caratteri base.
    La percentuale di avanzamento è mostrata nella barra di stato.
    """
    if not self.selected_pc_file:
        return
    fn = self.selected_pc_file
    def run():
        try:
            base, ext = os.path.splitext(fn)
            norm = base.lower()[:16] + ext.lower()
            with open(os.path.join(DATA_DIR, fn), "rb") as f:
                data = f.read()
            self.update_status(f"SENDING {norm}...")
```

```
            self.enqueue_raw(bytes([0xAA]) + norm.encode() + bytes([0x00]))
            for i in range(0, len(data), 20):
                self.enqueue_raw(bytes([0xBB]) + data[i:i+20])
                self.ble_queue.join()
                percent = int(((i+20) / len(data)) * 100)
                self.lbl_percent.configure(text=f"{int(percent)}%")
            self.enqueue_raw(bytes([0xCC]))
            self.update_status("FLASHING...")
            time.sleep(2.0)
            self.manual_refresh()
            self.lbl_percent.configure(text="0%")
            self.after(0, self.deselect_all)
        except:
            self.update_status("ERROR", "red")
    threading.Thread(target=run).start()

def usb_upload(self):
    """
    Flash totale della partizione LittleFS via USB (mklittlefs + esptool).
    Normalizza tutti i nomi file (minuscolo, max 16 caratteri base),
    crea l'immagine LittleFS con mklittlefs e la scrive con esptool.
    Più veloce del BLE per caricare molti file contemporaneamente.
    """
    if messagebox.askyesno("USB", "Flash totale con normalizzazione nomi?"):
        def run():
            tmp = os.path.join(os.path.dirname(DATA_DIR), "temp_usb_norm")
            try:
                self.update_status("NORMALIZING...")
                if os.path.exists(tmp):
                    shutil.rmtree(tmp)
                os.makedirs(tmp)
                for fn in os.listdir(DATA_DIR):
                    if fn.lower().endswith('.prg') or fn.lower().endswith('.raw'):
                        b, e = os.path.splitext(fn)
                        n = b.lower()[:16] + e.lower()
                        shutil.copy2(os.path.join(DATA_DIR, fn), os.path.join(tmp, n))
                subprocess.run([MKLITTLEFS, "-c", tmp, "-p", "256", "-b", "4096",
                                "-s", LITTLEFS_SIZE, "littlefs.bin"], check=True)
                subprocess.run([ESPTOOL, "--chip", "esp32s3", "--port", COM_PORT,
                                "--baud", "921600", "write_flash", LITTLEFS_OFFSET,
                                "littlefs.bin"], check=True)
                time.sleep(2.5)
                self.manual_refresh()
            except:
                self.update_status("USB ERROR", "red")
            finally:
                if os.path.exists(tmp):
                    shutil.rmtree(tmp)
                if os.path.exists("littlefs.bin"):
                    os.remove("littlefs.bin")
        threading.Thread(target=run).start()

def run_cmd(self):
    """
    Avvia il programma selezionato nella lista XIAO inviando il comando 0x7A.
    Il firmware cerca il file nella playlist e lo carica automaticamente.
    """
    if not self.selected_xiao_name:
        messagebox.showwarning("Attenzione", "Seleziona prima un file dalla lista XIAO.")
        return
    def run():
        name = self.selected_xiao_name
        self.enqueue_raw(bytes([0x7A]) + name.encode() + bytes([0x00]))
    threading.Thread(target=run).start()

def delete_cmd(self):
    """Elimina il file selezionato dalla flash XIAO (con richiesta di conferma)."""
    if not self.selected_xiao_name:
        messagebox.showwarning("Attenzione", "Seleziona prima un file dalla lista XIAO.")
        return
```



```
    if messagebox.askyesno("Confirm", f"Eliminare {self.selected_xiao_name}?"):
        def run():
            self.enqueue_raw(bytes([0xEE]) + self.selected_xiao_name.encode() + bytes([0x00]))
            self.xiao_files_raw = ""
            time.sleep(1.0)
            self.manual_refresh()
            self.after(0, self.deselect_all)
        threading.Thread(target=run).start()


# ================================================================================
# MENU CONTESTUALI
# ================================================================================

def show_pc_context_menu(self, event, f):
    """Menu contestuale (tasto destro) sulla lista PC: Copy e Flash all."""
    self.sel_pc(f)
    menu = tk.Menu(self, tearoff=0, bg="#101040", fg=C64_BLUE_LIGHT,
                   activebackground=C64_BLUE_DARK, activeforeground="white",
                   font=("Consolas", 12))
    menu.add_command(label=f"→  Copy {f}", command=self.ble_upload)
    menu.add_separator()
    menu.add_command(label="  Flash all", command=self.usb_upload)
    menu.tk_popup(event.x_root, event.y_root)

def show_context_menu(self, event, ft):
    """Menu contestuale (tasto destro) sulla lista XIAO: Run e Delete."""
    self.sel_xiao(ft)
    menu = tk.Menu(self, tearoff=0, bg="#101040", fg=C64_BLUE_LIGHT,
                   activebackground=C64_BLUE_DARK, activeforeground="white",
                   font=("Consolas", 12))
    menu.add_command(label="  Run",      command=self.run_cmd)
    menu.add_separator()
    menu.add_command(label="  Delete",   command=self.delete_cmd)
    menu.tk_popup(event.x_root, event.y_root)


# ================================================================================
# EDITOR TESTO MULTIRIGA
# ================================================================================

def open_txt_editor(self):
    """
    Apre la finestra popup dell'editor testo per l'inserimento di listati BASIC.
    Se già aperta, la porta in primo piano senza creare duplicati.
    """
    if hasattr(self, '_txt_win') and self._txt_win.winfo_exists():
        self._txt_win.lift()
        return
    self._txt_win = ctk.CTkToplevel(self)
    self._txt_win.title("Editor Testo C64")
    self._txt_win.geometry("700x500")
    self._txt_win.configure(fg_color=C64_BLUE_BG)
    # Delay necessario: CTkToplevel impiega qualche ms a renderizzarsi
    self._txt_win.after(100, lambda: self._txt_win.lift())
    self._txt_win.after(150, lambda: self._txt_win.focus_force())

    self._txt_box = ctk.CTkTextbox(self._txt_win, font=("Consolas", 14),
                                   fg_color="#101040", text_color=C64_BLUE_LIGHT)
    self._txt_box.pack(fill="both", expand=True, padx=10, pady=10)

    btn_frame = ctk.CTkFrame(self._txt_win, fg_color="transparent")
    btn_frame.pack(fill="x", padx=10, pady=(0, 10))

    ctk.CTkButton(btn_frame, text="INVIA", width=120, height=35,
                  fg_color="#2E8B57", text_color="white", font=("Arial", 13, "bold"),
                  command=self._send_txt_editor).pack(side="left", padx=5)
    ctk.CTkButton(btn_frame, text="CANCELLA", width=120, height=35,
                  fg_color="#D35400", text_color="white", font=("Arial", 13, "bold"),
                  command=lambda: self._txt_box.delete("1.0", "end")).pack(side="left", padx=5)
    ctk.CTkButton(btn_frame, text="CHIUDI", width=120, height=35,
```

```
                fg_color=C64_BLUE_DARK, text_color="white", font=("Arial", 13, "bold"),
                command=self._txt_win.destroy).pack(side="right", padx=5)

    def _send_txt_editor(self):
        """
        Invia il contenuto dell'editor testo al C64 riga per riga via BLE.
        Identico a send_paste_text ma legge da CTkTextbox invece di CTkEntry.
        """
        t = self._txt_box.get("1.0", "end-1c")
        if not t.strip():
            return
        def run():
            lines = [l for l in t.split('\n') if l.strip()]
            for line in lines:
                petscii = self.build_petscii_line(line)
                for i in range(0, len(petscii), 18):
                    chunk = petscii[i:i+18]
                    self.enqueue_raw(bytes([0x81]) + chunk)
                    time.sleep(0.4)
        threading.Thread(target=run, daemon=True).start()


    # ================================================================================
    # UTILITÀ
    # ================================================================================

    def update_status(self, t, c=None):
        """Aggiorna il testo della barra di stato (con colore opzionale)."""
        self.lbl_status.configure(text=t)
        if c:
            self.lbl_status.configure(text_color=c)

    def deselect_all(self):
        """Rimuove la selezione da entrambe le liste file."""
        try:
            if self.selected_pc_file in self.pc_buttons:
                self.pc_buttons[self.selected_pc_file].configure(
                    fg_color="transparent", text_color=C64_BLUE_LIGHT)
            if self.selected_xiao_full_text in self.xiao_buttons:
                self.xiao_buttons[self.selected_xiao_full_text].configure(
                    fg_color="transparent", text_color=C64_BLUE_LIGHT)
        except:
            pass
        self.selected_pc_file = self.selected_xiao_full_text = self.selected_xiao_name = None

    def check_deselect(self, event):
        """Deseleziona tutto quando si clicca su un'area vuota della finestra."""
        if (event.widget and
                "button" not in str(event.widget).lower() and
                event.widget != self.paste_area and
                event.widget != self.paste_area._entry):
            self.deselect_all()
            self.focus_set()

    def _send_and_refocus(self):
        """Invia il contenuto della casella testo e rimette il focus su di essa."""
        self.send_paste_text()
        self.after(200, self._focus_paste)

    def _focus_paste(self):
        """Forza il focus sulla casella testo e posiziona il cursore alla fine."""
        self.paste_area._entry.focus_force()
        self.paste_area._entry.icursor("end")


# ================================================================================
if __name__ == "__main__":
    app = C64ControlApp()
    app.mainloop()
```