

STM32 GameBoy

Marco Cutecchia, Edoardo Marangoni

{marco.cutecchia, edoardo.marangoni1}@studenti.unimi.it

6 ottobre 2022

1 Introduzione

Il GameBoy è una console portatile rilasciata da Nintendo all'inizio degli anni 90 che diede inizio al grandissimo successo dei videogiochi tascabili. La console vendette più di 118 milioni di unità nel mondo e divenne un fenomeno culturale ricordato ancora oggi.

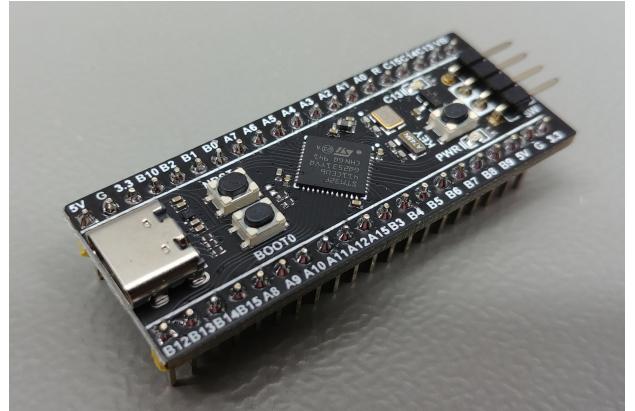
Lo scopo del progetto è quello di costruire un'imitazione del GameBoy in grado di giocare tutti i videogiochi originali, ma con una serie di miglioramenti grazie ai componenti hardware più moderni. Tra le modifiche al dispositivo originale che abbiamo implementato troviamo il caricamento dei giochi tramite microSD, invece che con cartucce, e l'utilizzo di uno schermo a colori e retroilluminato, al contrario dell'originale schermo a scala di grigi che diventava impossibile da vedere sotto scarsa luce.

2 Hardware

Le componenti hardware utilizzate sono tre: una scheda basata sul SoC STM32F411CEU6 (Fig. 1b), uno schermo TFT da 2.4" basata sul controllore ILI9341 (Fig. 2b) e infine una PCB per la plancia di gioco clone dell'originale utilizzata nel GameBoy (Fig. 3). Non è stato necessario comprare un lettore di schede microSD perché il display scelto ne integra uno al suo interno.

Oltre a questi componenti attivi sono stati utilizzati un interruttore SPDT, dei pulsanti in plastica e i rispettivi gommini da posizionare sopra la PCB dei tasti, delle millefori (insieme a stagno, cavi e pin) per costruire i collegamenti in modo da minimizzare lo spazio utilizzato e infine un case di GameBoy dove è stato alloggiato il tutto.

3V3	VB
G	C13
5V	C14
B9	C15
B8	R
B7	A0
B6	A1
B5	A2
B4	A3
B3	A4
A15	A5
A12	A6
A11	A7
A10	B0
A9	B1
A8	B2
B15	B10
B14	3V3
B13	G
B12	5V



(a) Pinout del microcontrollore STM32F411CEU6.

(b) La scheda WeAct Black Pill V2.0.

La componente centrale del progetto è il microcontrollore STM32F411CEU6, prodotto da ST e montato sulla board WeAct Black Pill V2.0. Il microcontrollore monta un core ARM Cortex-M4 con clock massimo di 100MHz (overclockato a 140). Questo microcontrollore offre 34 pin GPIO che verranno utilizzati per collegare il microcontrollore a schermo e pad; monta 512KiB di memoria flash e 128KiB di SRAM, necessari per l'esecuzione del software che abbiamo utilizzato.

Lo schermo che abbiamo utilizzato, basato sul controller ILI9341, è uno schermo LCD da 2.4", con una risoluzione di 320x240 e controllo individuale dei pixel, dotato di pin per il controllo dello schermo e della microSD che si può inserire nella stessa board.

SD_SCK	
SD_DO	
SD_DI	
SD_SS	3.3V
LCD_D1	5V
LCD_D0	GND
LCD_D7	LCD_RD
LCD_D6	LCD_WR
LCD_D4	LCD_RS
LCD_D5	LCD_CS
LCD_D3	LCD_RST
LCD_D2	F_CS



(a) Pinout dello schermo ILI9341.

(b) Lo schermo ILI9341.

L'ultimo componente hardware necessario è la plancia di gioco: per questo utilizziamo una board costruita appositamente per il form factor del GameBoy originale con 12 tasti. Ne abbiamo utilizzati solo 8: frecce direzionali, A, B, SELECT e START.

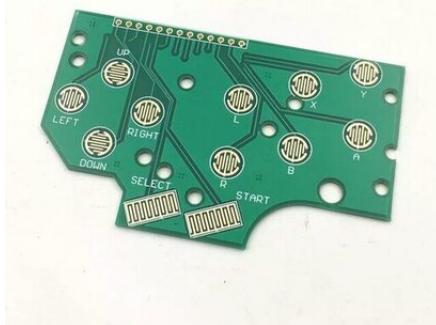


Figura 3: La PCB GB DMG.

2.1 Materiali

Nome	Modello	Costo unitario	Unità	Costo
Guscio esterno	GB DMG-01 Shell	9.90	1	9.90
Plancia di gioco	GB DMG-01 PCB	1.18	1	1.18
Bottoni	GB DMG-01 Buttons	2.95	1	2.95
Schermo	ILI9341 2.4"	6.44	2	12.88
Microcontrollore	STM32 F411CEU6	7.03	1	7.03
Interruttore	SS12D00 4mm	0.30	1	0.30
Millefori	50x70mm	3.50	1	3.50
Cavi e stagno	-	-	1	2.00
Totale				39.74€

Tabella 1: Materiali utilizzati per la costruzione del progetto. I costi indicati provengono da negozi online come Amazon, eBay e Aliexpress

2.2 Schema di collegamento

Lo schema dei collegamenti è mostrato in Fig. 4.

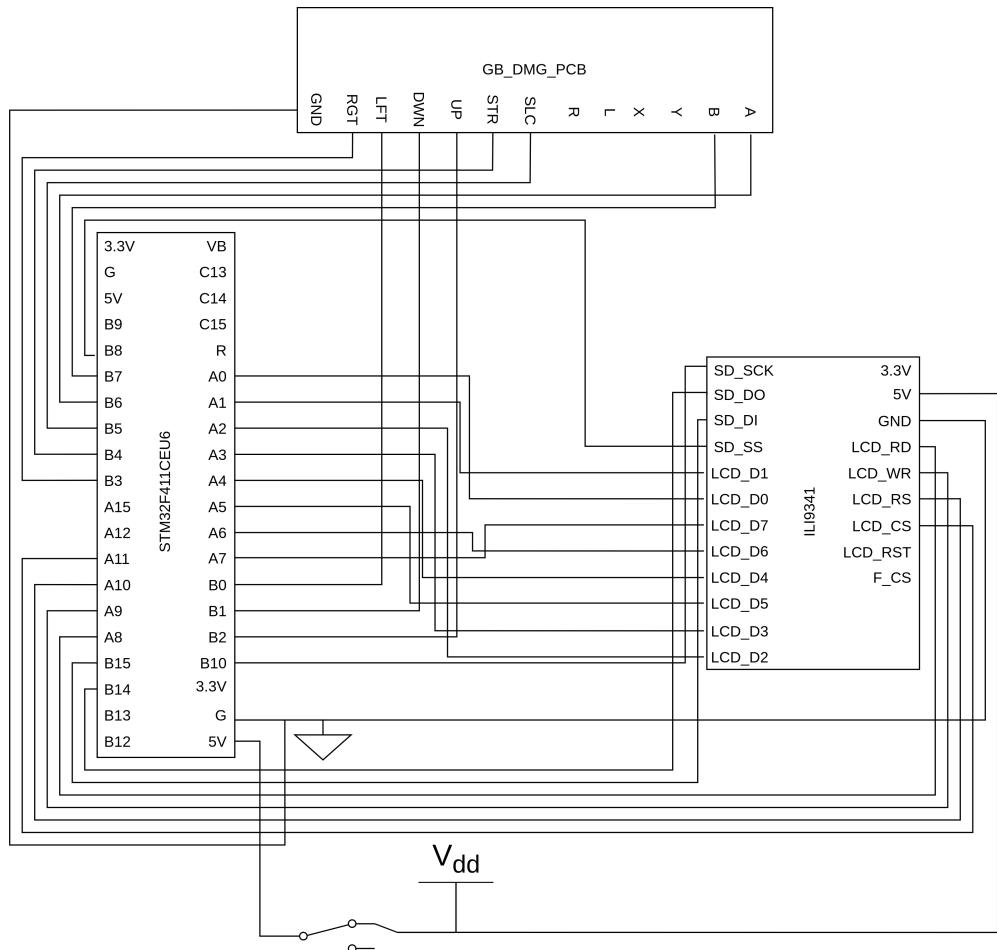


Figura 4

3 Software

Il software necessario per il progetto si divide nei seguenti moduli: gestione della scheda microSD, emulatore, interfaccia con lo schermo e una GUI per la selezione dei giochi.

3.1 Emulatore

Il modo in cui il nostro progetto permette di eseguire videogiochi originali del GameBoy è tramite l'emulazione. Un emulatore è un software che simula il processore ed eventuali altre periferiche in modo di eseguire programmi scritti per una certa architettura su un'altra architettura potenzialmente diversa. Scrivere un emulatore richiede uno studio attento dell'architettura che si vuole emulare ed è al di fuori dello scopo del corso, per questo motivo abbiamo deciso di riutilizzare il codice di un emulatore già esistente e open source.

L'emulatore che abbiamo scelto è peanutGB, scritto in C e progettato per essere facilmente portato su diverse piattaforme. Questo emulatore è distribuito in un singolo file sorgente, non richiede dipendenze e separa efficacemente il cuore dell'emulatore dalle interfacce d'input e output. Per integrare questo emulatore nel nostro progetto è stato dunque necessario implementare degli *hooks* come per esempio `gb_rom_read`, `gb_cart_ram_write` oppure `gb_lcd_draw_line`.

Il porting di questo emulatore per il microcontrollore non ha richiesto particolari modifiche. Nonostante questo, in generale, emulare una altra architettura è una operazione piuttosto costosa dal punto di vista delle performance, e il nostro caso non fa eccezione: allo scopo di misurare le performance del nostro progetto, utilizziamo la misura di *fotogrammi per secondo* (FPS).

Il GB originale aggiorna il proprio schermo circa 60 volte al secondo, dunque idealmente per avere una emulazione a piena velocità il nostro microcontrollore dovrebbe produrre 60 FPS. Le performance variano in base alla complessità dei giochi: la maggior parte dei giochi hanno un *framerate* intorno a 55 FPS, con qualche calo occasionale a 45 FPS nei momenti più concitati. Casi “patologici” sono alcuni dei giochi più complessi rilasciati verso il fine vita della console hanno framerate più bassi, intorno a 35 FPS; in generale, tuttavia, la gran parte dei giochi rimane comunque giocabile.

Per non compromettere le performance ulteriormente, abbiamo deciso di rinunciare all'audio: seppur il DMA ci avrebbe permesso di mandare l'audio a uno speaker senza intaccare troppo le performance, emulare il processore audio del GameBoy e generare i samples da mandare via PWM sarebbe stato molto dispendioso.

3.1.1 Colori

Originariamente, il GB disegnava esclusivamente in *grayscale*: ogni oggetto (sfondo, dialoghi, sprites, transizioni...) è assegnato a uno dei 4 *layer* che il GB offre, che viene ulteriormente diviso in 4 *variables*, come mostrato in Tabella 2. Con il rilascio del *Game Boy Color*, *Nintendo* preparò una serie di *palette* di colori per permettere ai giochi rilasciati in bianco e nero di diventare a colori. Queste palette sono semplicemente quattro colori in formato BGR565 per ognuno dei quattro layer supportati.

Abbiamo estratto alcune palette dei giochi più popolari e implementato questo meccanismo di conversione dei colori grayscale nel corrispondente colore della palette del layer del momento.

Layer	Descrizione di uso tipico	Variabili			
BG0	Sfondo	B1	B2	B3	B4
Win	Menu a finestra: stato, pausa, mappa, inventario, HUD	W1	W2	W3	W4
Obj0	Sprites principali	-	S2	S3	S4
Obj1	Transizioni, sprites secondari	-	P2	P3	P4

Tabella 2: Tabella dei colori per il GB (da [1]).

3.2 Scheda microSD

Fisicamente, la scheda microSD è inserita nello schermo ILI9341, che dispone di un'apposita porta; per interfacciarsi con essa, abbiamo deciso di formattare la scheda con il filesystem FAT32 e utilizzare due librerie: *cudeide-sd-card*, per la comunicazione a basso livello con la scheda microSD, e *FATFS* per l'interazione con il filesystem. Il nostro lavoro dunque è stato quello d'integrarle nel progetto e di scrivere delle piccole funzioni per collegare le due librerie.

La scelta di utilizzare *FatFS* è stata influenzata dal fatto che *ST* la consiglia, al punto che questa è disponibile da includere direttamente da *STM32CubeIDE*.

Dato che abbiamo riservato l'intera porta GPIOA all'utilizzo dello schermo, la comunicazione con la scheda microSD avviene via SPI sul bus SPI2, utilizzando i pin PB10, PB14 e PB15 che vengono collegati rispettivamente ai pin sullo schermo SD_SCK, SD_D0 e SD_DI. Il bus è configurato in modalità *Full Duplex Master*.

3.2.1 Caching dei blocchi

Abbiamo osservato che durante l'esecuzione la lettura dei blocchi della SD costituiva un collo di bottiglia per la performance, tanto da rendere impossibile l'utilizzo della console. Sfortunatamente la grande maggioranza dei giochi sono troppo grandi per essere caricati interamente in memoria RAM, con alcuni giochi che addirittura arrivano a MB di dimensione.

Per mitigare questo problema abbiamo implementato un meccanismo di caching dei dati letti dalla SD con una strategia “Least Recently Used”. Durante l'inizializzazione dell'emulatore viene creata in memoria una struct Cache contenente una linked list in cui vengono salvati i contenuti dei blocchi letti dalla SD. In questa struct vengono mantenuti anche i metadati necessari per ritrovare il blocco utilizzato più recentemente e meno recentemente; questi dati e la lista vengono aggiornati in base alle richieste, inserendo il blocco richiesto più recentemente in cima alla lista e contestualmente rimuovere quello richiesto meno recentemente.

Tramite misurazioni empiriche abbiamo ottenuto le performance migliori utilizzando una dimensione dei blocchi di 512 byte, riservando 20KB di memoria RAM per la cache (dunque fino a 40 blocchi caricati temporaneamente). Questo meccanismo di caching è un modulo fondamentale; senza di esso l'esecuzione dei giochi diventa talmente lenta al punto da essere inaccettabile.

3.3 Driver ILI9341

Il chip ILI9341 è un driver per display a cristalli liquidi a bassa risoluzione molto popolare grazie al suo basso costo. La comunicazione con questo driver può avvenire tramite due diverse interfacce: via SPI oppure tramite una interfaccia parallela ad 8 bit. L'interfaccia SPI richiede meno pin ma è anche più lenta nella comunicazione dato che utilizza un protocollo seriale, al contrario l'interfaccia parallela è più veloce ma richiede anche molti più pin. La decisione di quale interfaccia utilizzare è presa dal produttore del display che espone solamente alcuni dei pin del circuito integrato.

Nel nostro caso dobbiamo mostrare dei videogiochi sullo schermo e dunque abbiamo bisogno di aggiornare l'immagine sullo schermo molto velocemente (circa 60 volte al secondo), la scelta di un display che espone una interfaccia parallela è dunque obbligata.

Sfortunatamente l'interfacciamento via SPI è molto più popolare di quello parallelo e le poche librerie compatibili con microcontrollori STM32 funzionano solamente con il primo tipo d'interfaccia. È stato dunque necessario scrivere un driver apposito leggendo il datasheet ufficiale[2] e il codice di driver per altri microcontrollori.

3.3.1 Ad alto livello

Lo schermo, nella versione con interfacciamento parallelo ad 8 bit, dispone per l'utilizzatore 5 pin di controllo: LCD_RD, LCD_WR, LCD_RS, LCD_CS, ed LCD_RST. Questi pin vengono utilizzati per modificare lo stato della scheda, come descritto nella documentazione ufficiale. Oltre a essi troviamo altri 8 pin (LCD_D0, LCD_D1, ..., LCD_D7) su cui passano i dati scambiati tra schermo e utilizzatore.

Il nostro driver implementa solamente un piccolo sottoinsieme delle operazioni supportate dal chip ILI9341, oltre a una funzione extra per inviare una immagine ingrandita. Quest'ultima funzione è implementata all'interno del driver per motivi di performance discussi più avanti.

Per una lista dettagliata delle operazioni implementate si rimanda al file sorgente al percorso `display/ili9341.h`.

3.3.2 Inizializzazione

La funzione d'inizializzazione è parametrizzata dagli identificatori dei pin GPIO del microcontrollore che vengono collegati ai pin della scheda: una volta inizializzata la struct che mantiene le corrispondenze di questi collegamenti, si procede a inizializzare i pin di controllo della scheda (LCD_RD, LCD_WR, LCD_RS, LCD_CS, ed LCD_RST) come pin di output tramite le funzioni messe a disposizione dalla HAL.

Diamo quindi uno stato iniziale alla scheda prima di mandare la sequenza d'inizializzazione al display: il pin LCD_CS è impostato come *attivo* (quindi *low*, il chip ILI9341 è attivo basso). Si impostano disabilitati i pin LCD_WR e LCD_RD.

Il pin LCD_RST provoca un reset dello stato interno del display quando è attivo, per questo motivo all'avvio attiviamo e disattiviamo questo pin per resettare la scheda a uno stato conosciuto. Questo pin dovrà rimanere disabilitato per tutta l'esecuzione (a meno che non si desideri resettare lo schermo).

Una volta che la scheda ha raggiunto uno stato conosciuto è necessario inviare la *sequenza d'inizializzazione*: questa è semplicemente una sequenza di comandi per attivare lo schermo (nello stato iniziale è spento in modalità *sleep*) e configurare parametri come il formato di scambio dei pixel e l'endianness. Descrivere nel dettaglio tutti i comandi mandati in questa sequenza sarebbe lungo e poco interessante, si rimanda dunque al codice ¹e al datasheet [2] per approfondimenti.

3.3.3 Disegnare sullo schermo

Disegnare sullo schermo vuol dire mandare dei comandi appositi al display per copiare un array di pixel, detto *framebuffer*, sopra il framebuffer interno del display a una determinata posizione. Nella nostra implementazione utilizziamo un singolo framebuffer globale sulla quale emulatore e GUI disegnano.

Prima di cominciare l'operazione d'invio è consigliato inviare inizialmente un comando *nop* per terminare qualunque potenziale comando rimasto in sospeso sul chip ILI9341. Questo caso può avvenire se il microcontrollore e il display non fossero più sincronizzati tra di loro.

Il protocollo della scheda prevede che il client, per disegnare sullo schermo, imposti inizialmente l'area sulla quale vuole disegnare. L'utente usa i due comandi COLUMN_ADDRESS_SET e PAGE_ADDRESS_SET: il primo comando è utilizzato per specificare le due colonne destra e sinistra che delimitano l'area in cui si vuole disegnare, e simmetricamente fa il secondo comando; entrambi i comandi richiedono quattro parametri da 8 bit ciascuno che indicano i 2 + 2 byte necessari per identificare righe e colonne. Una volta specificata l'area, l'utente utilizzerà il

¹L'invio di questa sequenza è fatto nella funzione ILI9341_SendInitializationSequence del file display/ili9341.c

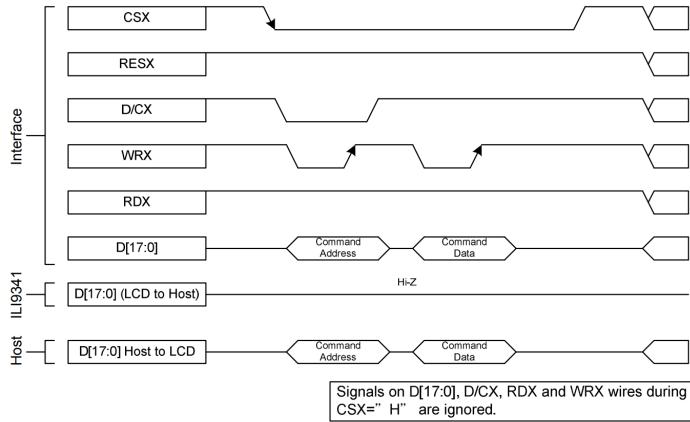


Figura 5: Rappresentazione dello stato dei pin di controllo sul display durante l'invio di un comando con 1 byte di dati

comando `MEMORY_WRITE` per inviare alla scheda i dati che descrivono il disegno. Ogni pixel dello schermo è descritto da due byte in formato BGR565 e verrà inviato seguendo il *write cycle* come descritto in Fig. 5.

Nell'implementazione abbiamo deciso di forzare l'utilizzo dei pin A0-A7 del microcontrollore per l'invio dei dati allo schermo (ossia devono essere collegati ai pin D0-D7 dello schermo): in questo modo possiamo scrivere direttamente sul registro `GPIOA->ODR` il byte da inviare. Possiamo dunque evitare l'utilizzo della HAL di ST e di operazioni di `mask` e `shift`: questo ci permette di ottenere prestazioni migliori evitando overhead non indifferenti.

Scaling I display basati su ILI9341 hanno risoluzione 320×240 e utilizzano 2 byte per ogni pixel. Facendo un breve calcolo dunque un framebuffer per l'intero schermo occuperebbe $320 \cdot 240 \cdot 2 = 153KB$; non potrebbe mai essere contenuto nei $128KB$ di SRAM del nostro microcontrollore. Per questo motivo abbiamo deciso di tenere in memoria un framebuffer più piccolo ma di effettuare uno scaling dell'immagine al momento dell'invio dei pixel al display.

L'algoritmo di scaling che abbiamo implementato cerca di scalare l'immagine prodotta da GUI ed emulatore senza sacrificare le performance ed è pertanto relativamente semplice: aggiungiamo un pixel ogni due, sia in verticale che in orizzontale; il nuovo pixel avrà come colore la media dei colori dei due pixel che lo hanno generato.

La dimensione del framebuffer in memoria è di appena 160×144 pixel, la stessa risoluzione del display del GameBoy, ma l'immagine scalata che viene mandata al display è più grande ed ha dimensioni 240×216 pixel, cioè uno scaling del 1.5x.

3.4 GUI



Figura 6

Abbiamo scritto una semplice GUI per permettere all’utente di scegliere i giochi disponibili sulla scheda SD (Fig. 6a) o, in caso non vi siano giochi disponibili sulla scheda SD (o non sia inserita) mostrare a schermo un messaggio di errore (Fig. 6b). Benché siano disponibili diversi framework per sviluppare GUI su ambienti embedded, abbiamo deciso di non utilizzarli implementando da zero il tutto, data la relativa semplicità dell’interfaccia che abbiamo immaginato.

All’avvio della console l’utente vede una lista di giochi presenti sulla scheda microSD. Utilizzando le frecce direzionali SU e GIU e il tasto START l’utente può selezionare il gioco da far partire. Per ritornare alla lista dei giochi l’utente può spegnere e riaccendere la console.

3.4.1 Font PSF

Questo componente software scrive direttamente i valori dei pixel nel framebuffer globale. Per poter mostrare a video i nomi dei giochi disponibili è necessario implementare la risoluzione di font sullo schermo: per fare questo abbiamo implementato un piccolo renderer di font in formato *PC Screen Font* (PSF), un tipo di font utilizzato nei terminali Linux.

I font PSF codificano ogni *glifo* (ossia ogni simbolo, ad esempio una lettera) come bitmap. Nel font che abbiamo scelto, i simboli vengono codificati come 8 righe di 8 bit ciascuna: un bit $i = 0$ indica che per mostrare il glifo, il pixel corrispondente al bit i deve avere il colore di background dello schermo, mentre un bit $i = 1$ indica che il pixel deve avere il colore di foreground.

Assieme con queste bitmap, per implementare la GUI sono bastate delle funzioni che svolgono dei semplici calcoli sulle dimensioni del framebuffer (ossia calcolare i rettangoli in cui andare a dipingere i glifi).

```
// Character 48
Bitmap: -#####-- \
        ##---##- \
        ##---##- \
        ##-#-##- \
        #####-- \
        ##---##- \
        -#####-- \
        -----
Unicode: [00000030];
```

Tabella 3: Il glifo “0” nel font PSF koi8r per l’alfabeto cirillico.

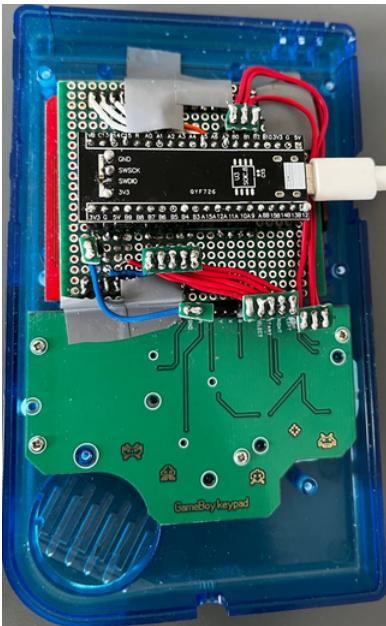
4 Assemblaggio

Abbiamo speso una buona parte del tempo totale della realizzazione del progetto a pianificare e tentare vari approcci per l’assemblaggio finale. Questa parte di svolgimento, infatti, si è rivelata più ostica del previsto, poiché l’insieme dei componenti e dei cavi non stava nel case del GB.

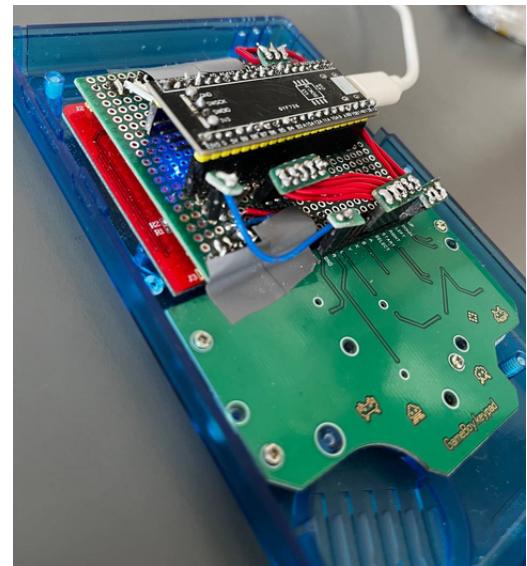
Essendo la PCB del gamepad un clone dell’originale del GameBoy, il case è costruito per alloggiarla comodamente. Sfortunatamente non si può dire lo stesso del nostro schermo e microcontrollore, essendo componenti completamente diversi da quelli trovati nel GB originale, in cui l’intera circuiteria è posizionata dietro lo schermo; in questo spazio il nostro display e microcontrollore occupano l’intero spazio verticale, lasciando poco spazio per i cavi.

Il nostro primo tentativo di utilizzare dei semplici cavetti per i collegamenti si è rivelato fallimentare: l’alto numero di collegamenti e il loro spessore (anche dopo averli accorciati) rendevano difficilissimo chiudere il case. Inoltre, anche dopo essere riusciti a chiudere il tutto, questi cavetti erano proni a staccarsi dai pin durante la sessione di gioco.

La soluzione è stata quella di riprodurre i collegamenti su una millefori, saldando dei cavetti piatti in stagno su di essa e utilizzando dei pin header femmina che fungono da alloggiamento per il microcontrollore. Questa millefori, tagliata per essere delle stesse dimensioni dello schermo, è stata progettata per essere saldata sui pin dello schermo, riducendo al minimo lo spessore totale.



(a)



(b)

Figura 7

Questa millefori espone dei pin maschio per l'alimentazione (5V e GND) e per i tasti (da PB0 a PB8), collegati direttamente ai pin del microcontrollore. Questi pin vengono collegati rispettivamente ai cavi dell'alimentazione e ai pin della PCB dei tasti.

In mezzo ai cavi che collegano batterie e microcontrollore troviamo un semplice interruttore, che viene posizionato sulla parte superiore del GameBoy. L'interruttore utilizzato è uno switch SPDT, dunque a singolo polo con doppia mandata, ma noi utilizziamo solamente una delle due mandate. La scelta di utilizzare uno switch a due mandate invece di uno con singola mandata è dovuto al fatto che i primi sono molto più economici e reperibili dei secondi. Lo switch e i cavi a cui è collegato sono stato saldati su un piccolo pezzo di millefori, tagliato per fare in modo che si incastri nelle incavature già presenti nel case del GameBoy.

4.1 Alimentazione e consumi

La scheda che abbiamo scelto contiene al suo interno un regolatore di tensione in input da +3.52V fino +5.25V: questo ci ha permesso di costruire un sistema di alimentazione molto semplice, composto banalmente da tre batterie AA (1.5V ciascuna) in serie per un totale di +4.5V in ingresso.

Abbiamo misurato che il nostro progetto consuma circa 116mAh. La console originale, invece, ne consuma tra i 70 e gli 80. Assumendo che le batterie in uso siano da 2500mAh, utilizzando la formula

$$\text{tempo} = \frac{\text{capacità batteria}}{\text{consumo}}$$

otteniamo $2500/116 \approx 21$ ore come autonomia per il nostro progetto, mentre per il GB è $2500/70 \approx 35$.

Di questi 116mAh il consumatore principale è lo schermo retroilluminato; infatti disattivando la retroilluminazione il consumo crolla a circa 43mAh. Di questi 43mAh circa 22mAh sono utilizzati dal microcontrollore e dalle sue periferiche attive mentre il rimanente è utilizzato dallo schermo non retroilluminato e dalla comunicazione con la scheda microSD.

5 Considerazioni finali

Con rammarico ci tocca ammettere di aver sottostimato la potenza necessaria per emulare questi videogiochi, e di aver dovuto di conseguenza effettuare dei compromessi significativi nella realizzazione finale; primo tra tutti l'assenza del suono. Con il senno di poi sicuramente sceglieremmo una scheda più potente, oppure opteremmo per emulare una console meno potente.

Nonostante questo, il fatto che molte persone a cui abbiamo fatto provare il progetto riuscissero a perdersi in lunghe sessioni di gioco in Tetris o Super Mario Land ci porta a dire di aver comunque raggiunto il nostro obiettivo.

Riferimenti bibliografici

- [1] *Coloring Gameboy – Layers* — *tirodvd.wordpress.com*. <https://tirodvd.wordpress.com/2015/01/27/coloring-gameboy-layers/>. [Accessed 23-Sep-2022].
- [2] Ilitek. *TFT LCD Single Chip Driver 240RGBx320 Resolution and 262K color - Specification*. 2011.
- [3] STMicroelectronics. *Developing applications on STM32Cube™ with FatFs*. 2019.