

Università degli Studi di Salerno
DIEM, a.a. 2021-22
Corso: SISTEMI EMBEDDED
Professore: Vincenzo Carletti

Air-Mouse by Group 6

*Enrico Maria Di Mauro
Allegra Cuzzocrea
Andrea De Gruttola
Salvatore Grimaldi*

0612701706
0612701707
0612701880
0612701742

e.dimauro5@studenti.unisa.it
a.cuzzocrea2@studenti.unisa.it
a.degruttola@studenti.unisa.it
s.grimaldi29@studenti.unisa.it



Sommario

Introduzione	2
Requisiti	2
Scelte progettuali	3
Funzionamento.....	5
Architettura Hardware.....	7
STM32F401RE	7
Giroscopio-Accelerometro GY-521 MPU6050.....	8
Real Time Clock DS3231.....	8
Micro SD Reader.....	9
Schema Elettrico.....	10
Architettura Software.....	11
Libreria RTC	13
Libreria Giroscopio-Accelerometro.....	14
Libreria SDCard	15
Modulo FATFS	15
Libreria HID	17
Libreria Parametri	19
Threads.....	20
Thread di acquisizione del movimento	21
Thread di rilevazione delle gesture	22
Thread di calibrazione	22
Thread di log.....	23
Procedura di calibrazione	24
Software Supplementare	25
Monitor Seriale	25
EMLearn	26
Interfacce software per la comunicazione utilizzate.....	27
Risultato Finale	28

Introduzione

Il presente report consiste nella descrizione della progettazione e dell'implementazione di un **Air-Mouse**, realizzato nell'ambito del corso di **Embedded Systems**, tenuto dal professore Vincenzo Carletti presso l'Università degli Studi di Salerno. Il report è allegato al progetto in qualità di **documentazione ufficiale**, al fine di fornire un'accurata presentazione delle specifiche funzionali del sistema, delle scelte progettuali fatte, delle componenti hardware adoperate e dell'architettura software realizzata.

Air-Mouse è un mouse intelligente capace di inviare **input HID**¹ ad un computer attraverso una tecnologia **motion-sensing**. In particolare, il device realizzato fornisce all'utente la possibilità di controllare il cursore, fare click e doppio click, fare zoom-in e zoom-out, effettuare lo scrolling e attivare la shortcut ALT+TAB. Il controllo avviene attraverso semplici ed intuitivi movimenti aerei (*gesture*) della mano su cui l'utente *indossa* il dispositivo. La rilevazione ed il riconoscimento delle gesture sono effettuati da una **rete neurale** ad hoc.

Requisiti

Il progetto consiste nella realizzazione di un *Air-Mouse* dotato di:

- **2 Accelerometri-Giroscopi**
- **1 Real Time Clock**
- **1 SD Card Module**

L'interfaccia utilizzata per collegare il dispositivo al PC è **UART** (USB).

Gli obiettivi del progetto sono:

- *Progettare pattern aerei* per riprodurre: click, double-click, zoom-in, zoom-out, scroll-up, scroll-down e alt-tab
- Utilizzare mano e dita per riprodurre i movimenti del mouse
- *Calibrare* l'*Air-Mouse* con software apposito
- Progettare un *protocollo di comunicazione* tra STM32-Air-Mouse e PC
- Progettare interfacce con i bus
- Conservare in un *log* tutti gli eventi dell'*Air-Mouse* attraverso:
 - *Timestamp* dall'RTC
 - Scrittura su una SD Card di una rappresentazione (codificata) di tutti gli eventi
- Utilizzare differenti *Task* per differenti attività schedulandoli in **FreeRTOS**²

¹ HID (Human Interface Device) è un tipo di dispositivo informatico che utilizza uno standard di comunicazione per ricevere input dagli esseri umani e fornire output agli stessi.

² FreeRTOS è un particolare sistema operativo real-time per microcontrollori.

Scelte progettuali

Le principali scelte progettuali adottate sono:

1. Dato che è stata prevista una **fase di calibrazione opzionale** per la quale è necessario mostrare a schermo i comandi utilizzabili (impostazione delle dead-zones, dell'angolo corrispondente alla velocità massima, ecc.) dall'utente, **si è deciso di affidarsi ad una connessione USB ed al protocollo UART per l'invio seriale di testo dalla scheda al PC e viceversa.**
2. È necessario adoperare un software che funga da *monitor seriale in grado di leggere i dati trasmessi tramite protocollo UART dalla scheda al PC*, e viceversa, col fine di mostrare all'utente un apposito menù per la calibrazione.
3. **Per l'elaborazione degli input HID-compliant sono state utilizzate librerie standard di STM per l'HID.** L'alternativa sarebbe stata l'invio di input tramite protocollo UART dalla scheda al PC: in questo caso software dedicato si sarebbe dovuto occupare della loro trasformazione in input HID-compliant. *La scelta effettuata assicura maggiore portabilità* poiché non è necessario effettuare un'apposita configurazione software sulla macchina a cui è connessa la scheda: è sufficiente comunicare con i driver HID inclusi nella maggior parte dei SO, senza che vi sia un programma (da realizzare appositamente) in esecuzione sul PC che si occupi della traduzione degli input ricevuti tramite UART in input HID-compliant. L'unico aspetto negativo (dovuto, in realtà, alla presenza di un'unica porta USB sulla scheda adoperata) della scelta effettuata consiste nella necessità di realizzare una nuova interfaccia USB tra la scheda e il PC attraverso la costruzione di un collegamento fisico tra alcuni pin della scheda stessa e la parte terminale di un cavo USB.
4. **Ad ogni avvio del dispositivo l'utente può scegliere se utilizzare una configurazione di default oppure realizzare una nuova calibrazione,** il che garantisce grande *flessibilità, comodità d'uso* e possibilità di *customizzazione*. Lo spegnimento dell'*Air-Mouse* implica la perdita delle informazioni associate all'eventuale calibrazione effettuata. In una futura release non si esclude di implementare il mantenimento delle informazioni delle calibrazioni, così da assicurare ulteriore versatilità.
5. Un giroscopio-accelerometro è stato collocato sulle dita medio e anulare e viene utilizzato per **comandare lo spostamento del cursore sullo schermo.** L'altro giroscopio-accelerometro è stato posizionato in corrispondenza del dito indice e viene usato (insieme a quello sulle altre due dita) **per rilevare le gesture.** Si è scelto di collocare il sensore del movimento sulle dita medio e anulare perché in seguito a vari test questa si è rivelata una posizione abbastanza

stabile da permettere letture angolari consistenti per il posizionamento del cursore sullo schermo, e allo stesso tempo sufficientemente flessibile per permettere di rilevare i movimenti dovuti alla flessione delle dita, necessari per l'esecuzione delle gesture che coinvolgono entrambi i sensori.

6. **La velocità e la collocazione sullo schermo del cursore sono regolate dall'inclinazione della mano rispetto alla posizione angolare di default o a quella rilevata in fase di calibrazione (se svolta).** Questo permette di gestire il movimento del cursore senza la necessità di spostare la mano dalla sua posizione neutrale (mano parallela al pavimento con dita rivolte verso lo schermo), poiché si agisce soltanto sugli *assi di inclinazione*. Avere una *posizione neutrale di riferimento*, inoltre, è particolarmente vantaggioso per il processo di rilevazione delle gesture, poiché garantisce che i valori letti dai giroscopi-accelerometri quando l'utente realizza una certa gesture siano estremamente simili a quelli appresi dalla rete neurale in fase di addestramento (a patto che il movimento venga realizzato in modo consono dall'utente).
7. **È stato scelto un sistema di posizionamento relativo** per i vantaggi che questo comporta in termini di *rapidità di calibrazione e portabilità*. Nel caso si fosse adoperato un *sistema di posizionamento assoluto*, sarebbe stato necessario fissare i margini massimi di mobilità in funzione dei vertici dello schermo del PC. Inoltre, non si sarebbe potuta utilizzare alcuna calibrazione di default.
8. **Per il rilevamento delle gesture è stato scelto di utilizzare una rete neurale** piuttosto che un sistema di approssimazioni basato sull'algebra lineare. In questo modo si ha un grande vantaggio in termini di *accuratezza* nella rilevazione dei movimenti e maggiore *flessibilità*.
9. È stato scelto di **dividere il programma principale in thread**, data la natura delle operazioni che è necessario effettuare in tempo reale. Il programma necessita di un'elevata *responsività* per la gestione in contemporanea del movimento del cursore, della rilevazione ed esecuzione delle gesture, della scrittura dei log e, all'occorrenza, della procedura di calibrazione. L'uso di un sistema operativo multi-threaded è stata quindi una scelta naturale per soddisfare le specifiche richieste.
10. **Per la scrittura dei log è stato scelto di aggiornare un unico file presente sulla scheda sd** invece di creare un file separato per ogni rilevamento di una gesture. In questo modo tutte le informazioni vengono catalogate in un *unico punto* aumentandone la *leggibilità*.

Funzionamento

L'*Air-Mouse* è dotato di 2 porte USB:

- una per il riconoscimento della scheda come dispositivo HID Composito, ossia come l'insieme di mouse e tastiera, in modo da garantire l'invio degli input al PC
- una per la comunicazione testuale tramite UART

Una volta collegata l'USB relativa all'invio di dati HID-compliant è possibile utilizzare l'*Air-Mouse* con i parametri preconfigurati. Nel caso in cui si voglia procedere con una calibrazione, invece, è necessario collegare entrambe le USB ed avviare un monitor seriale in grado di mostrare a schermo le istruzioni inviate tramite il protocollo UART. Facendo ciò è possibile impostare i seguenti parametri muovendo opportunamente la mano:

- **Velocità massima** del cursore (default=6, valore minimo=3, valore massimo=10)
- **Zona morta**: l'inclinazione che si deve superare per poter determinare l'effettivo movimento del cursore (default=10°, valore minimo=0°, valore massimo \cong 180°)
- **Angolazione massima** della mano corrispondente alla velocità massima del cursore (default=90°, valore minimo=zona morta, valore massimo \cong 180°)
- **Threshold di accelerazione** per l'attivazione delle gesture (default=4, valore minimo=3, valore massimo=5)
- **Data e ora** del RTC

La rilevazione delle gesture richiede l'utilizzo di entrambi i giroscopi-accelerometri. In particolare, la fase di lettura degli input, i quali verranno successivamente analizzati per essere ricondotti ad una specifica gesture, comincia quando i giroscopi-accelerometri subiscono un'accelerazione il cui valore supera la soglia di threshold. Le gesture utilizzabili sono:

- **Click sinistro del mouse**: abbassamento del dito indice



- **Doppio click sinistro del mouse**: doppio abbassamento del dito indice



- **Click destro del mouse:** abbassamento delle dita medio e anulare



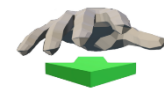
- **Zoom-in:** cerchio orario con la mano



- **Zoom-out:** cerchio antiorario con la mano



- **Scroll-up:** spostamento della mano in avanti



- **Scroll-down:** spostamento della mano in alto



- **Alt-Tab:** spostamento della mano a destra



Al rilevamento di una gesture viene automaticamente aggiornato il file di log presente sulla microSD alloggiata nell'sdcard reader, il quale comunica con la scheda STM32 con il protocollo spi. All'interno del file vengono salvate le informazioni rilevanti per ottenere uno *storico temporale* di quanto accade. A tale scopo vengono salvati il nome della gesture rilevata ed il timestamp relativo al momento di attivazione. Per il timestamp è necessario leggere il real time clock.

Architettura Hardware

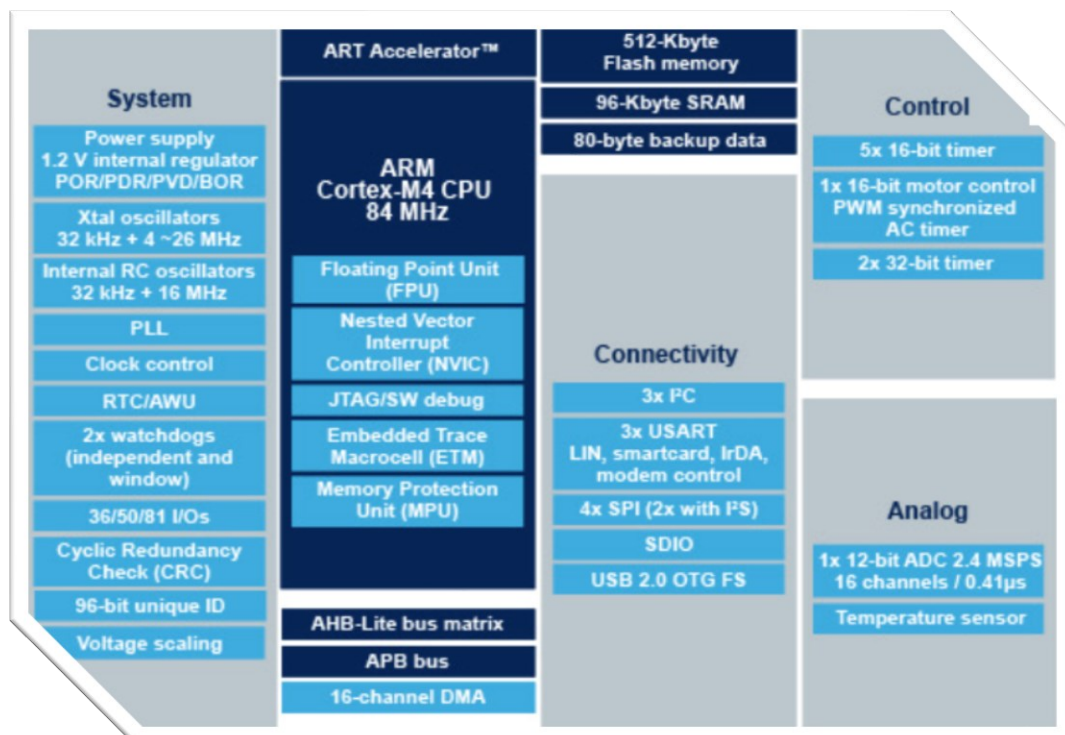
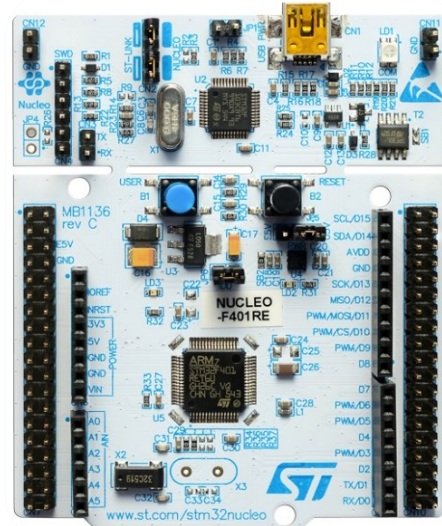
STM32F401RE

Il dispositivo **STM32F401RE** è basato su un high-performance *ARM Cortex-M4 32-bit RISC* core che opera a 84 MHz e che fornisce una Floating Point Unit (FPU) che supporta tutte le istruzioni ed i tipi di dato ARM.

Il Cortex-M4 core implementa anche un full set di istruzioni per il Digital Signal Processing (DSP) ed una Memory Protection Unit (MPU) che garantisce maggiore sicurezza.

Inoltre, la scheda incorpora delle high-speed embedded memories (512 Kbytes di Flash memory, 96 Kbytes di SRAM) ed un elevato numero di pin di I/O e di periferiche connesse a due Advanced Peripheral Bus (APB), due Advanced High-performance Bus (AHB) bus ed una bus matrix multi-AHB a 32bit.

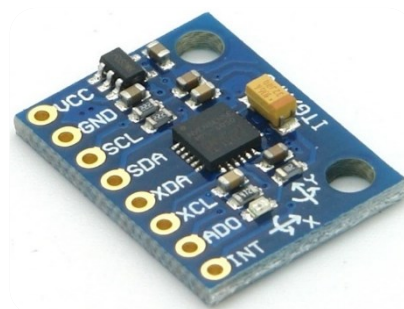
Il device offre:



Giroscopio-Accelerometro GY-521 MPU6050

Il sensore **GY-521 MPU-6050** contiene, in un singolo integrato, sia un *accelerometro* sia un *giroscopio* MEMS a 3 assi.

Il suo funzionamento è molto accurato grazie al fatto che contiene per ciascun canale un *convertitore analogico / digitale a 16-bit*. Di conseguenza riesce a catturare contemporaneamente i valori degli assi X, Y e Z.

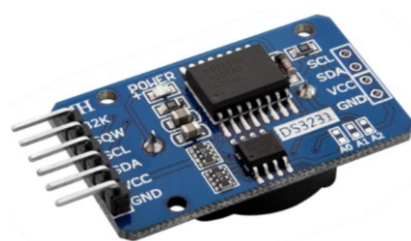


Caratteristiche:

- Convertitore AD a 16 bit Integrato
- Chip: MPU-6050
- Alimentazione: 3-5v (regolatore low dropout integrato)
- Comunicazione I2C
- Range di misura giroscopio: ± 250 , 500, 1000 e 2000 $^{\circ}/s$
- Range di misura accelerometro: +2, +4, +8, +16 g
- Dimensioni: 21.2 x 16.4 x 3.3 mm

Real Time Clock DS3231

Il **DS3231** è un *RTC I2C* a basso costo ed estremamente preciso grazie all'utilizzo di un *oscillatore al quarzo*. Il dispositivo incorpora un ingresso per una batteria in modo da non perdere la sincronizzazione dell'orario in mancanza di alimentazione diretta dalla scheda.



Il DS3231 *mantiene aggiornata l'informazione relativa alla data e all'ora*, conservando in memoria secondi, minuti, ore, giorno, mese ed anno. La data si regola automaticamente al cambio del mese, anche quando quest'ultimo è composto da meno di 31 giorni. L'orologio può operare sia nel formato 12 che 24 ore utilizzando l'indicatore AM/PM.

È fornita, inoltre, un'onda quadra programmabile in uscita. Il dispositivo è dotato anche di un sensore di temperatura.

Caratteristiche:

- Accuratezza $\pm 2\text{ppm}$ da 0°C a $+40^{\circ}\text{C}$
- Accuratezza $\pm 3.5\text{ppm}$ da -40°C a $+85^{\circ}\text{C}$

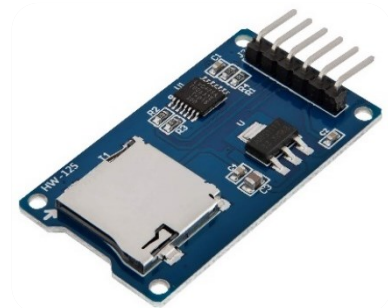
- Ingresso per la batteria di backup per il continuo aggiornamento di data e ora
- Range di temperature operativa da 0°C a +70°C
- Basso consumo energetico
- Real-Time Clock per il conteggio di secondi, minuti, ore, giorno, mese ed anno con compensazione dell'anno bisestile fino al 2100
- 2 allarmi giornalieri
- Onda quadra in output programmabile
- Interfaccia I2C veloce (400kHz)
- Tensione di alimentazione 2.3-5.5 V
- Sensore digitale di temperatura con accuratezza di $\pm 3^{\circ}\text{C}$

Micro SD Reader

Il modulo **Micro SD Reader** consente di accedere in lettura e scrittura al *File System* di una scheda *micro-SD*, interfacciandosi al contempo con STM32F401RE, attraverso il protocollo *SPI*, come una normale periferica.

Caratteristiche:

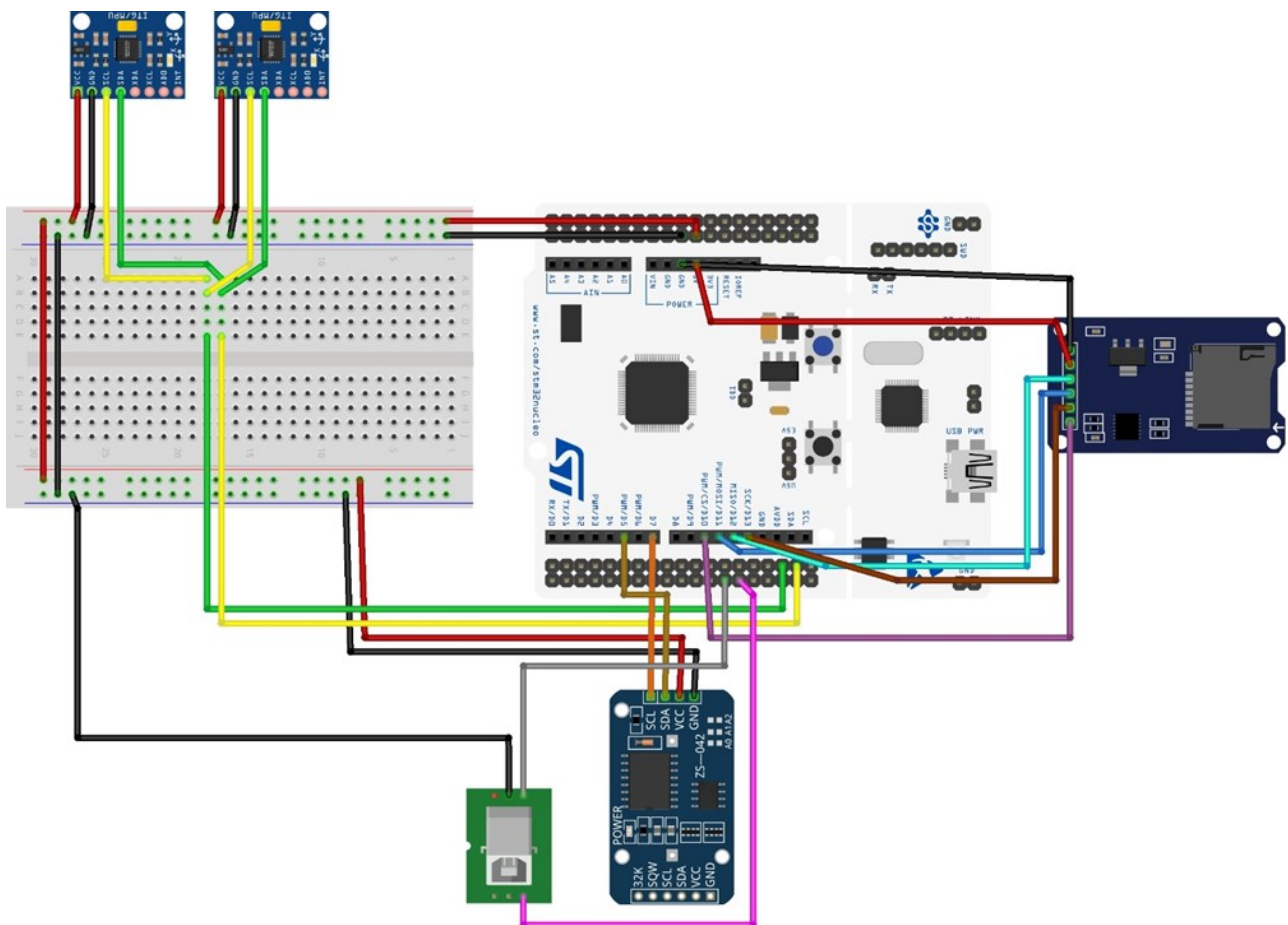
- Alimentazione: 3.3-5V
- Protocollo di comunicazione seriale: SPI
- Compatibile con SD Card fino a 16GB
- Basso consumo energetico



Schema Elettrico

Di seguito lo **schema elettrico del progetto**, realizzato con *Fritzing*. Tra i vari collegamenti ci sono:

- I due giroscopi e accelerometri GY-521, connessi allo stesso bus I2C1 tramite la breadboard;
- Il Real Time Clock DS3231, connesso al bus I2C3 direttamente sulla STM32;
- Il lettore di schede MicroSD, connesso direttamente ai pin MOSI e MISO della scheda, per il bus SPI;

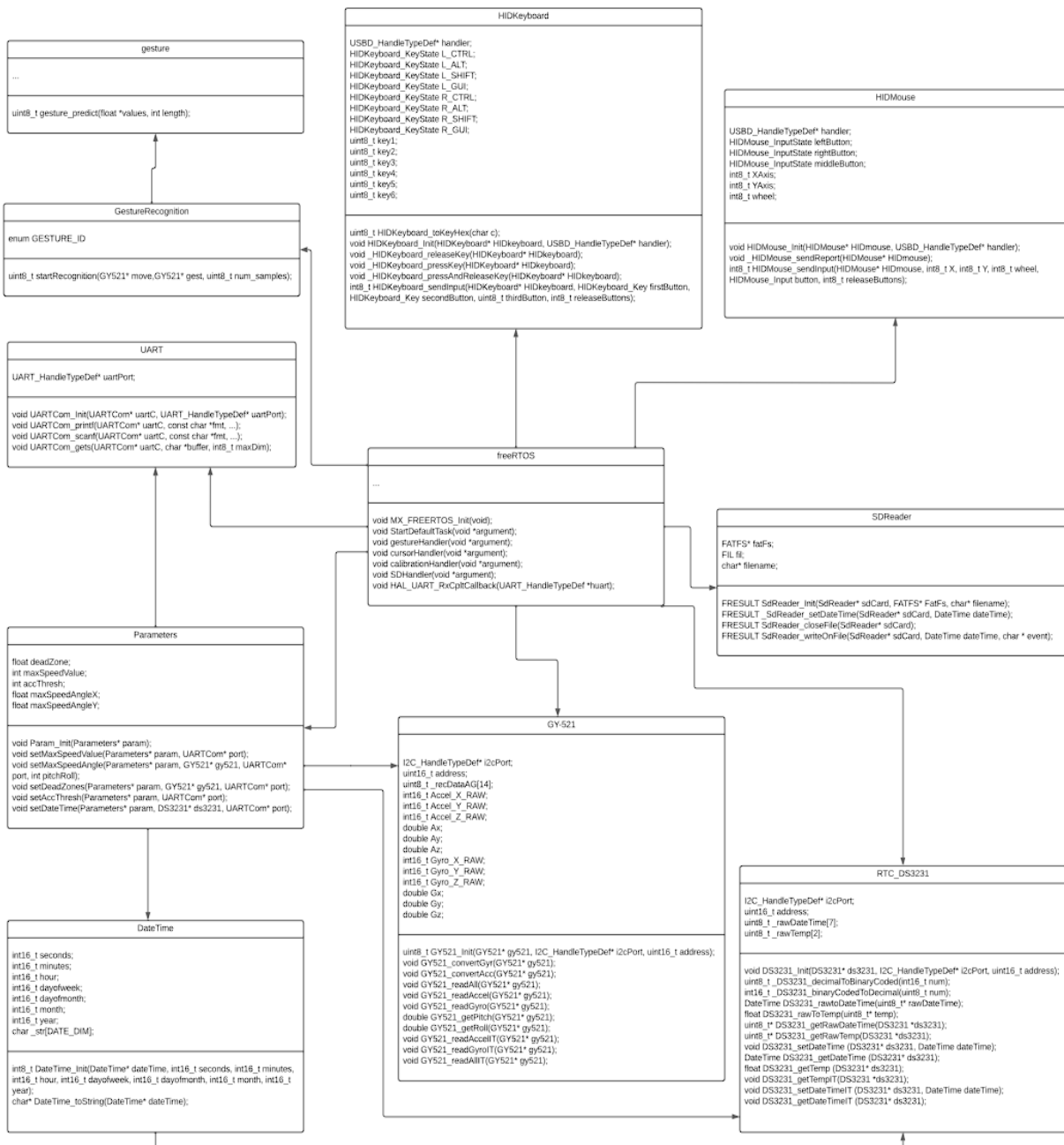


fritzing

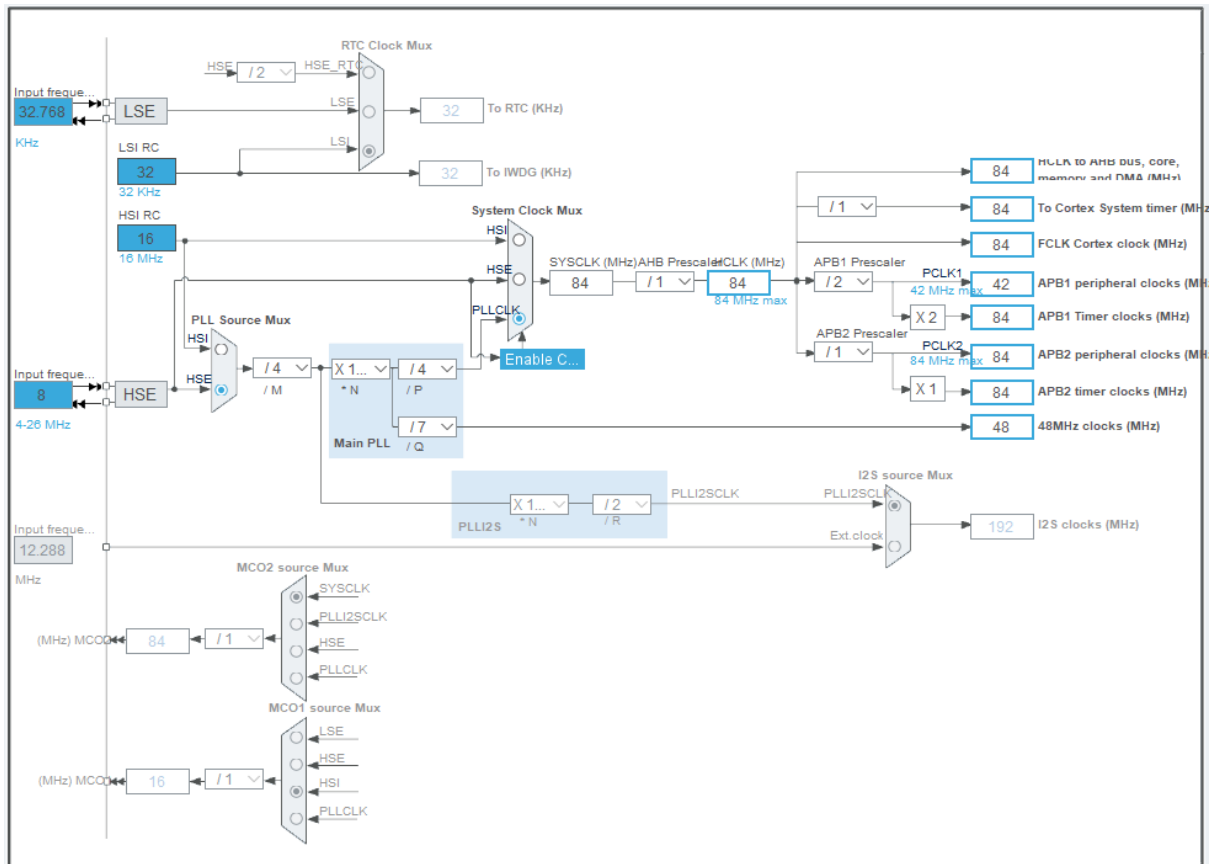
Architettura Software

Il codice presente sulla scheda STM è stato organizzato in maniera *modulare*, assegnando a ogni dispositivo una libreria per la gestione delle sue funzioni elementari e la comunicazione con la scheda. Sono inoltre presenti delle librerie relative alle funzioni più avanzate, come la *calibrazione* e la *comunicazione tra la scheda e il PC* a cui è connessa.

Sotto è riportato lo schema delle relazioni tra le librerie. La notazione adoperata è la seguente: se due librerie sono collegate da una freccia, allora la libreria da cui parte la freccia presenta almeno un'istanza di una struttura dati della libreria a cui punta tale freccia oppure usa delle sue funzioni.



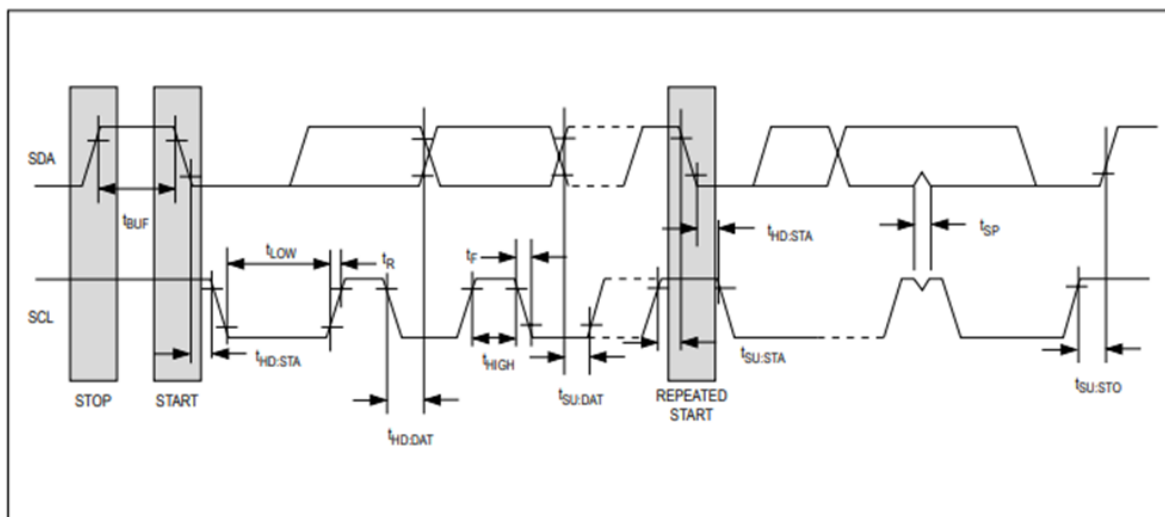
Inoltre, il **clock tree** della scheda STM32 è stato configurato nel seguente modo:



Libreria RTC

In questo modulo vengono gestite tutte le operazioni di lettura/scrittura del real time clock DS3231. Vengono utilizzate le funzioni dell'HAL per la comunicazione sul bus I2C utilizzato dal dispositivo, in modo da attenersi allo schema di trasferimento dati presente nel datasheet del DS3231.

Data Transfer on I2C Serial Bus



Il *datasheet* presenta anche lo schema dei *registri di memoria* e dei loro indirizzi, che vengono richiamati in maniera statica ad ogni lettura/scrittura nell'implementazione interna della funzione. L'*indirizzo I2C* del dispositivo, invece, viene passato in fase di inizializzazione assieme al numero della *porta I2C* che si intende usare: entrambi i valori vengono salvati nei relativi campi di una struttura dati. In questo modo è possibile usare la libreria per comunicare con più di un DS3231 nello stesso momento, a patto che ogni dispositivo abbia un indirizzo I2C differente. La struttura dati della libreria presenta anche dei campi per gli ultimi valori letti, sia per data ed ora che per i valori della temperatura. Inoltre, delle funzioni interne permettono di convertire opportunamente i valori grezzi letti dai registri. Tutte le funzioni di lettura e di scrittura sono presenti in duplice versione: una per la comunicazione in modalità di *polling* e l'altra tramite l'uso di *interrupt*.

Libreria Giroscopio-Accelerometro

Tale libreria è utile per la configurazione e l'uso del giroscopio e accelerometro GY-521. La maggior parte delle sue funzioni usa una *struttura dati di supporto* in cui vengono memorizzati, oltre all'indirizzo e alla porta I2C del dispositivo, i valori ottenuti dall'*ultima lettura dei registri* del giroscopio e dell'accelerometro. Nella struttura sono presenti dei campi sia per i dati *grezzi* sia per quelli *elaborati e convertiti* attraverso le operazioni aritmetiche indicate nel datasheet del dispositivo. Dato che nei registri del dispositivo i dati di lettura sono organizzati in maniera contigua e intervallati in alcuni casi dai valori del sensore di temperatura integrato (ridondanti e per questo da trascurare), è necessario prima di tutto memorizzare il contenuto dei registri in un *buffer di lettura* (un array allocato come campo della struttura dati del GY-521). Questo è ciò che avviene richiamando una delle funzioni di lettura (***GY521_readAll()***) della libreria, sia per i dati del giroscopio che per quelli dell'accelerometro e indipendentemente dalla modalità di trasmissione (con polling o con interruzioni). Una volta effettuata tale lettura è necessario chiamare un'altra funzione (***GY521_convertAcc()*** o ***GY521_convertGyr()***) della libreria per effettuare delle operazioni sui bit del buffer di lettura, concatenando e convertendo i dati nei valori finali sia in versione grezza che elaborata. Tali valori sono relativi alle letture sugli assi x, y e z del giroscopio e dell'accelerometro.

Prima di procedere con le letture, è importante inizializzare il dispositivo attraverso l'apposita funzione ***GY521_init()***, che si occupa di scrivere dei bytes su alcuni registri. Il registro più importante è quello che controlla lo stato di *sleep/awake* del sensore: è necessario che tutti i suoi bit siano posti a zero affinché il sensore entri in funzione e le letture di giroscopio e accelerometro abbiano effetto. Gli altri registri di inizializzazione servono rispettivamente per settare il *data rate* e per *configurare alcuni parametri* di accelerometro e giroscopio.

Libreria SDCard

Questa libreria contiene le funzioni necessarie alla *scrittura dei log* (in seguito all'esecuzione di una *gesture*) sulla scheda microSD inserita nel lettore, il quale comunica con la STM32 tramite il protocollo SPI. Prima di procedere con le scritture, è necessario chiamare la funzione di inizializzazione **SdReader_Init()**, la quale si occupa di montare il supporto e di effettuare la prima apertura del file dei log. Per il mount è richiesto un handler del filesystem FATFS, che viene passato come parametro ad SdReader_Init() e successivamente salvato in un campo della struttura dati della libreria. L'handler viene passato e non creato all'interno della funzione al fine di aumentare la *portabilità* e la *riusabilità* della libreria, ad esempio in contesti in cui è necessario usare lo stesso handler per gestire accessi al filesystem provenienti da dispositivi differenti.

Nell'implementazione interna della funzione che si occupa del *mount* si effettuano delle chiamate a funzione del modulo *FATFS* integrato nell'ambiente di ST, e lo stesso vale per tutte le altre operazioni di scrittura e lettura su file. Lo scopo della libreria ad alto livello **SDReader** è, pertanto, quello di offrire *un'interfaccia di uso facile e immediato* alle funzioni offerte dal modulo software del filesystem. È possibile usare le funzioni messe a disposizione dalla libreria per scrivere i log su file semplicemente passando una stringa contenente la descrizione dell'evento e una variabile di tipo *DateTime* per indicare la data e l'ora in cui è avvenuta la scrittura. Essendovi un unico file di log a cui si accede costantemente in scrittura (in *append*), il nome di tale file viene passato soltanto una volta durante l'inizializzazione; pertanto, nel caso in cui si volesse cambiare il riferimento al file sarebbe necessario usare la funzione **SdReader_closeFile()** per poi effettuare nuovamente l'inizializzazione passando una stringa con il nome del nuovo file.

Modulo FATFS

FatFs è una libreria software dal peso ridotto per microcontrollori e sistemi embedded che implementa il **supporto al filesystem FAT/exFAT**. È scritto completamente in ANSI C, *indipendente dalla piattaforma* sottostante e per questo facile da portare su hardware differenti. È spesso usato in sistemi embedded a *basso consumo* dove la memoria è limitata, dato che la libreria ha bisogno di poco spazio da allocare in RAM e sul disco. Nella versione più minimale, il codice usa soltanto dai 2 ai 10 kB di RAM.

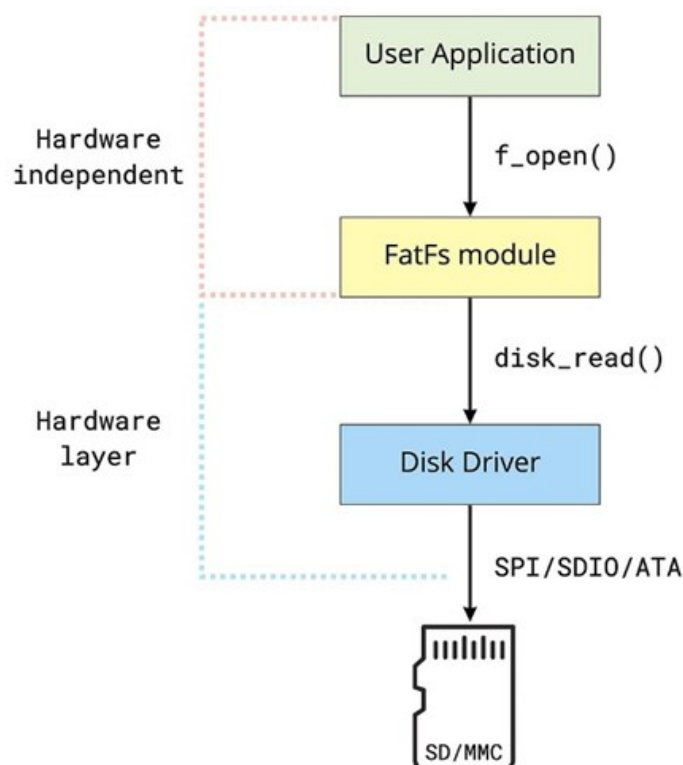
Il modulo *FatFs* separa logicamente l'astrazione della user app dal codice dipendente dalla piattaforma. L'architettura della libreria implica che il sistema possa avere *diversi dispositivi di storage con diversi driver dedicati*. Inoltre, è possibile importarla ed eventualmente modificarla nel contesto di un sistema operativo multi-threaded.

Nella sua implementazione minima, il livello del driver deve supportare almeno le tre seguenti interfacce:

- *disk_status* — restituisce lo stato del dispositivo (non inizializzato, mancante, protetto, pronto)
- *disk_initialize* — inizializza il disco fisico
- *disk_read* — legge un blocco dal disco fisico

FatFs fornisce varie funzioni del filesystem per le applicazioni, in particolare per l'accesso ai file:

- *f_open* — Apre/crea un file
- *f_close* — Chiude un file aperto
- *f_read* — Legge dal file
- *f_write* — Scrive sul file
- *f_seek* — Muove il puntatore del file
- *f_truncate* — Tronca la dimensione del file
- *f_sync* — Effettua il flush della cache
- *f_forward* — Indirizza i dati del flusso
- *f_expand* — Alloca un blocco contiguo al file
- *f_gets* — Legge una stringa
- *f_putc* — Scrive un carattere
- *f_puts* — Scrive una stringa
- *f_printf* — Scrive una stringa formattata
- *f_tell* — Restituisce la posizione attuale del puntatore del file
- *f_size* — Restituisce la dimensione del file



Libreria HID

Questa libreria permette alla scheda STM di *comunicare con il PC* al quale è connessa così da essere riconosciuta come *un dispositivo della classe HID*: una definizione supplementare della specifica USB che determina come i driver di questo tipo debbano estrarre i dati dai dispositivi USB.

Le librerie relative all'invio di report HID sono due: **HIDMouse** e **HIDKeyboard**.

La prima presenta una *struttura dati* con un *handler USB*, e tre variabili enumerative per ciascun tasto del mouse virtuale (sinistro, destro e centrale); sono inoltre presenti degli interi che rappresentano il valore di spostamento dell'asse x e dell'asse y e il numero di step effettuati dalla rotella del mouse. Alla funzione di inizializzazione **HIDMouse_Init()** vengono passati l'handler USB e le tre enumerazioni dei bottoni. Inoltre, all'interno della struttura dati, le tre enumerazioni vengono poste al valore che corrisponde allo stato "rilasciato" del bottone. Allo stesso modo le variabili della struttura relative allo spostamento sugli assi e alla rotella del mouse vengono poste a zero.

Le funzioni dedicate all'invio dei report HID sono due: la prima si chiama **_HIDMouse_sendReport()** ed effettua l'invio del report usando la funzione `USBD_CUSTOM_HID_SendReport` della libreria di STM per la gestione dei dispositivi HID, alla quale vengono passati come parametri l'handler USB e il *buffer* che si intende inviare. Tale buffer viene costruito all'interno della stessa **_HIDMouse_sendReport()**, concatenando il valore identificativo del mouse (come indicato nella tabella degli identificatori HID) e i valori presenti nella struttura dati attraverso delle operazioni su bit. In questo modo è possibile, quindi, specificare *in che modo l'input generato dalla scheda agisce sui tasti del mouse virtuale*, sugli spostamenti orizzontali e verticali e sulla rotella.

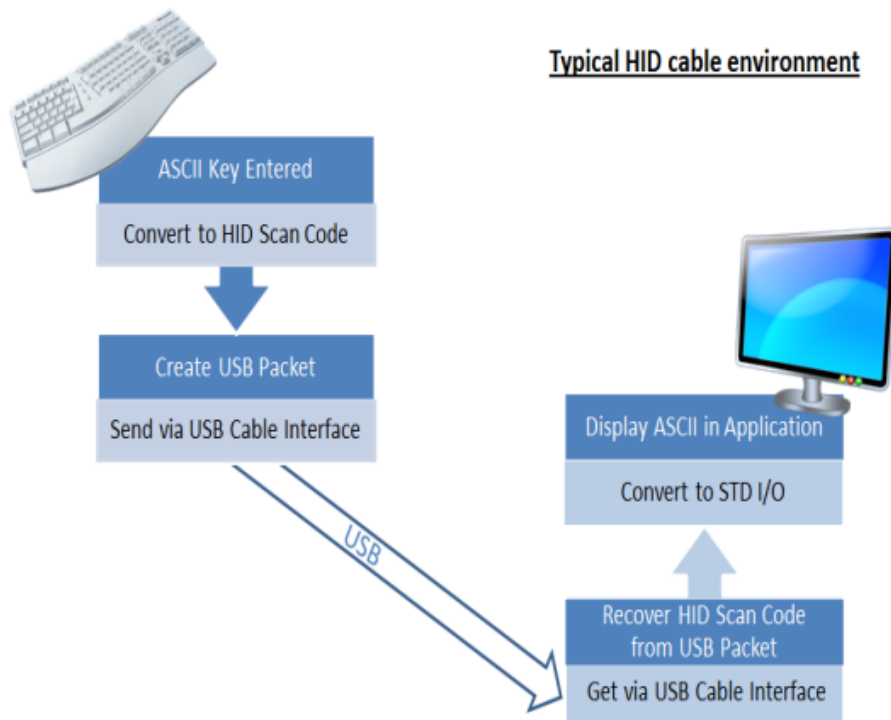
L'ultima funzione della libreria è **HIDMouse_sendInput()**, che si occupa di modificare i valori della struttura dati per poi richiamare **_HIDMouse_sendReport()** e finalizzare l'invio del report HID. **HIDMouse_sendInput()** è la funzione che viene richiamata esternamente alla libreria ogni qualvolta si desidera inviare un mouse-input, e prende come parametri, oltre all'istanza della struttura dati:

- i valori degli spostamenti e della rotella
- una variabile enumerativa che indica quale dei tre bottoni si intende premere (è possibile passare un valore nullo nel caso in cui non si abbia intenzione di premere alcun bottone)
- un intero che stabilisce se i bottoni devono essere premuti e poi rapidamente rilasciati oppure devono restare nello stato "premuto" anche dopo l'esecuzione della funzione.

La seconda libreria HID, **HIDKeyboard**, funziona in maniera simile per quanto concerne l'invio dei report. Anche in questo caso, infatti, è presente una *struttura dati con un handler USB e delle variabili enumerative* relative a dei tasti speciali della tastiera (CTRL, ALT, SHIFT ecc.). Sono inoltre presenti degli interi che possono essere usati per i restanti tasti della tastiera. È stato scelto un layout di questo tipo poiché in molti casi può essere desiderabile mandare dei keyboard-input in cui vengono premuti più tasti contemporaneamente; i casi più comuni prevedono proprio l'invio di uno o due tasti speciali mandati in contemporanea a un carattere normale della tastiera (ad esempio per richiamare *shortcut* o *hotkeys* nella maggior parte dei sistemi operativi). Anche in questa libreria sono presenti delle enumerazioni, oltre a quelle che identificano i caratteri speciali, che servono a indicare lo stato di un tasto: "premuto" o "rilasciato".

Nella funzione di inizializzazione **HIDKeyboard_Init()** tutti i caratteri speciali della struttura dati vengono messi nello stato "rilasciato" e tutti gli altri caratteri vengono posti a zero; viene inoltre passato l'handler USB e memorizzato nel relativo campo della struttura. La funzione di rilascio **_HIDKeyboard_releaseKey()**, invece, è identica alla **HIDKeyboard_Init()** nella parte in cui si modificano i valori della struttura: anche in questo caso vengono posti ai loro valori nulli. Tuttavia, non è necessario passare nuovamente l'handler, e alla fine della funzione si richiama la funzione interna della libreria **_HIDKeyboard_pressKey()** che si occupa di mandare in maniera effettiva i report.

Nella funzione di invio dei report, in maniera analoga a ciò che avviene nella libreria HIDMouse, si costruisce un buffer concatenando l'HID ID della tastiera ai valori presenti nella struttura dati con delle operazioni su bit. Alla fine si richiama la funzione **USBD_CUSTOM_HID_SendReport** passandole l'handler USB e il buffer precedentemente costruito. La funzione più complessa della libreria è **HIDKeyboard_sendInput()**, che si occupa di modificare i valori della struttura dati per poi chiamare le funzioni interne di invio dei report. All'interno di tale funzione ci sono due diverse funzioni di invio: una per premere e poi rilasciare rapidamente i tasti desiderati, e l'altra per premere i tasti senza il rilascio finale.



Libreria Parametri

La libreria **Parameters** viene utilizzata dal *thread della calibrazione* (vedi paragrafo relativo) per effettuare i processi di calibrazione relativi ai singoli comandi a disposizione dell'utente. Tale libreria presenta una *struttura dati* in cui vengono memorizzati i *parametri*, sia in seguito alle calibrazioni sia in fase di inizializzazione (in questo caso sono dei valori di *default*). La procedura di calibrazione verrà descritta in dettaglio nei paragrafi successivi.

Threads

I **thread** in cui è diviso il programma in esecuzione sulla STM32 sono i seguenti:

- ✓ **Thread di acquisizione del movimento:** legge costantemente dal sensore GY-521 dedicato al movimento (collocato su medio e anulare dell'utente) per modificare la posizione del cursore sullo schermo. Ha la priorità più alta, *Real Time*.
- ✓ **Thread di rilevazione delle gesture:** si occupa della rilevazione delle gesture attraverso delle letture continue di entrambi i sensori GY-521, di cui solo quello posizionato sul dito indice è impiegato esclusivamente per la pattern recognition. Come per il thread di acquisizione del movimento, ha priorità *Real Time*.
- ✓ **Thread di calibrazione:** è impiegato soltanto in fase di calibrazione per la comunicazione con il monitor seriale e l'invio e ricezione di testo per i comandi. Ha priorità *Normale*.
- ✓ **Thread di log:** si occupa di scrivere sul file di log ogni qualvolta venga effettuata una gesture, specificando il tipo della gesture e la data e l'ora in cui questa è stata eseguita. Ha priorità *Normale*.

È inoltre presente il *default thread di RTOS* che viene eseguito all'avvio del programma e quando la CPU è in *idle*.

Sono stati utilizzati dei **mutex** per la sincronizzazione tra i thread e una *queue* per la comunicazione.

1. Il primo mutex, ***mutexI2CHandle***, viene utilizzato in fase di calibrazione, per fare in modo che l'unico thread in esecuzione sia proprio quello che si occupa della fase di calibrazione.
2. Il secondo mutex, ***mutexUARTHandle***, è necessario affinché l'handler della porta UART sia gestito solamente da un thread in un dato istante; infatti, sia il thread dei log che quello della calibrazione possono accedere a tale risorsa.
3. Il terzo mutex, ***mutexGestureHandle***, fa in modo che il thread del movimento e quello delle gesture accedano in maniera mutualmente esclusiva all'handler della porta I2C, al fine di comunicare con i due sensori GY-521.
4. Il quarto mutex, ***mutexHIDHandle***, viene impiegato per sincronizzare i thread di rilevazione gesture e di acquisizione del movimento nell'accesso all'handler delle comunicazioni HID.

La queue ***SDQueueHandle*** è invece usata per la comunicazione tra il thread delle gesture e quello dei log, così che il thread dei log possa ottenere le informazioni relative all'ultima gesture che è stata eseguita.

Thread di acquisizione del movimento

Il thread più importante assieme al thread di rilevazione delle gesture, con cui condivide il livello di priorità *Real Time*, è il **thread di acquisizione del movimento** (implementato dalla funzione ***cursorHandler()***). Esso si occupa di effettuare letture continue dall'accelerometro-giroscopio GY-521 per poi elaborare i dati letti e ottenere i valori dell'inclinazione rispetto all'asse orizzontale e verticale.

Sia le letture che le successive elaborazioni sono effettuate chiamando il set di funzioni della libreria dedicata al GY-521. Una volta calcolati i *valori degli angoli*, vengono effettuati dei *controlli* per stabilire le azioni da effettuare; in particolare, bisogna controllare che *gli angoli orizzontali e verticali siano maggiori degli angoli delle rispettive zone morte* (impostati in fase di calibrazione). In tal caso si passa al controllo successivo:

- Se uno di tali angoli ha un valore superiore a quello indicato per l'angolo di velocità massima, allora viene salvato il valore della velocità massima nella variabile relativa al prossimo valore di spostamento (per l'asse x o per l'asse y) che verrà inviato come input al PC.
- Se invece l'angolo letto, sia nel caso dell'angolo orizzontale che per quello verticale, è compreso tra il valore indicato per la deadzone e quello dell'angolo di velocità massima, allora si chiama una funzione (***MAP()***) che effettua un mapping del valore dell'angolo in maniera tale da *assegnare alla variabile dello spostamento un valore proporzionale alla distanza che tale angolo ha rispetto alla deadzone e all'angolo di velocità massima*: se è più vicino alla deadzone, si avrà un valore di spostamento minore; più è vicino all'angolo di velocità massima e maggiore sarà il valore dello spostamento.

Una volta effettuati questi controlli ed assegnati i valori alle variabili di spostamento, si procede a chiamare la funzione della libreria **HIDMouse** (***HIDMouse_sendInput()***) per lo spostamento del cursore sullo schermo, passando le due variabili di spostamento come parametri.

È importante specificare che, poco prima di ogni lettura del sensore, il mutex *mutexGestureHandle* (relativo alla risorsa costituita dall'handler della porta I2C) viene acquisito dal thread per poi essere rilasciato immediatamente dopo la fine della lettura. Alla fine del ciclo del thread viene inoltre effettuato un controllo sul mutex *mutexI2CHandle* che viene acquisito dal thread di calibrazione: se tale mutex è impegnato, si aspetta fino a quando non viene rilasciato; dopodiché viene acquisito e poi rilasciato in rapida successione.

All'inizio del thread viene effettuato un ulteriore controllo: se sia l'angolo orizzontale che quello verticale letti dal GY-521 hanno un valore uguale alle

loro rispettive letture avvenute nel ciclo precedente, si incrementa un *counter*. Quando tale counter raggiunge un *valore di soglia*, si effettua nuovamente l'inizializzazione del sensore (con la dovuta gestione del mutex relativo alla risorsa I2C). Valori degli angoli sempre uguali ad ogni ciclo, infatti, potrebbero indicare che il sensore GY-521 non sta funzionando in maniera corretta, e in tal caso può essere necessario effettuare una nuova inizializzazione.

Thread di rilevazione delle gesture

Il **thread di rilevazione delle gesture** effettua a sua volta delle letture continue dei sensori GY-521; inoltre, immediatamente dopo la lettura viene effettuato un controllo: se, per entrambi i sensori, la somma dei valori di accelerazione lungo l'asse x, y e z supera un certo *valore di soglia* (stabilito in fase di calibrazione e presente nella struttura dati della libreria delle impostazioni di calibrazione) allora viene richiamata la funzione (***startRecognition()***) della libreria delle gesture per la *pattern recognition* di un movimento del sensore. Il *controllo delle accelerazioni* serve per attivare la rilevazione di una gesture soltanto nel momento in cui viene percepito un *movimento deciso* da parte dei sensori: questo è utile per evitare di avviare la fase di riconoscimento dei movimenti nel caso in cui si intende soltanto muovere il cursore sullo schermo cambiando l'angolazione del sensore posto sulle dita medio e anulare.

A seconda del valore restituito da `startRecognition()` si procede ad eseguire le azioni richieste dalla relativa gesture identificata, richiamando le funzioni delle librerie `HIDMouse` e `HIDKeyboard`.

Come per il thread di acquisizione del movimento, vengono gestiti appropriatamente i mutex quando si accede alle risorse I2C ed è presente il controllo finale per la sincronizzazione con il thread di calibrazione.

Thread di calibrazione

Il **thread di calibrazione** (implementato dalla funzione ***calibrationHandler()***) gestisce la procedura di calibrazione, acquisendo innanzitutto il mutex legato all'handler della porta UART.

Dopo aver stampato sul monitor seriale un *menù contenente tutti i possibili comandi selezionabili dall'utente*, il thread rimane in attesa della lettura di una stringa inserita dall'utente nel monitor seriale e, usando una callback UART della libreria integrata nell'ambient di ST, procede soltanto quando riceve il carattere di ritorno (pressione del tasto *Invio* da parte dell'utente). Una volta ricevuta la stringa completa, lo stesso thread effettua delle comparazioni e, se tale stringa coincide con quella associata ad un comando, allora avvia la procedura di calibrazione per quel comando richiamando la relativa funzione della libreria di calibrazione, che salverà

poi i parametri desiderati nell'apposita struttura dati a cui accedono anche gli altri thread.

Al termine dell'esecuzione di ogni comando di calibrazione, il mutex della risorsa UART viene rilasciato.

Thread di log

Il **thread di log** (implementato dalla funzione ***SDHandler()***) effettua le *scritture sul file di log ad ogni esecuzione di una gesture*. Al suo avvio effettua il mount della scheda SD e l'apertura del file di log (un unico file in cui verranno memorizzati tutti i log in maniera sequenziale in append). Dopodiché, all'interno del suo ciclo aspetta che sia stato inserito un nuovo elemento nella queue condivisa con il thread di rilevazione delle gesture; quando ciò avviene, effettua una lettura dal real time clock DS3231 e scrive sul file di log l'evento della gesture con la data e l'ora in cui è stata eseguita. Dopo la scrittura vengono anche mandate via UART al PC delle informazioni testuali relative all'esito dell'operazione.

Come per il thread di acquisizione del movimento e di rilevazione delle gesture, è presente un controllo finale sul mutex acquisito dal thread di calibrazione, in modo tale che anche il thread di log resti bloccato finché tale mutex non viene rilasciato.

Procedura di calibrazione

La **calibrazione** viene effettuata attraverso il monitor seriale che riceve e manda dati testuali tramite la connessione UART della scheda, su USB. In questo modo *l'utente può visualizzare le comunicazioni della scheda e inviare a sua volta delle stringhe per selezionare uno specifico comando*. I comandi messi a disposizione sono i seguenti:

- **maxSpeed**: permette di impostare il valore della velocità massima del cursore.
- **deadzones**: configura i valori degli angoli entro i quali la posizione della mano dell'utente viene considerata come "ferma". Quando il sensore rileva angoli che superano questi valori, il movimento del cursore viene attivato e la sua velocità calcolata in maniera proporzionale all'angolo attuale.
- **accThresh**: viene usato per impostare la soglia dei valori di accelerazione letti dai sensori, oltre la quale viene avviato il processo di rilevazione delle gesture, attraverso la pattern recognition della rete neurale.
- **maxSpeedAngleX** e **maxSpeedAngleY**: impostano i valori degli angoli per i quali la velocità del cursore assume il valore massimo. Muovendo la mano oltre questi angoli, la velocità del cursore continua ad essere la velocità massima.
- **clock**: imposta la data, l'ora e il giorno della settimana del Real Time Clock.

Una volta che l'utente inserisce un comando, e questo viene correttamente riconosciuto come tale valutando la sua stringa, si avvia il processo di calibrazione relativo a tale comando.

La procedura di calibrazione dedicata è differente per ogni comando:

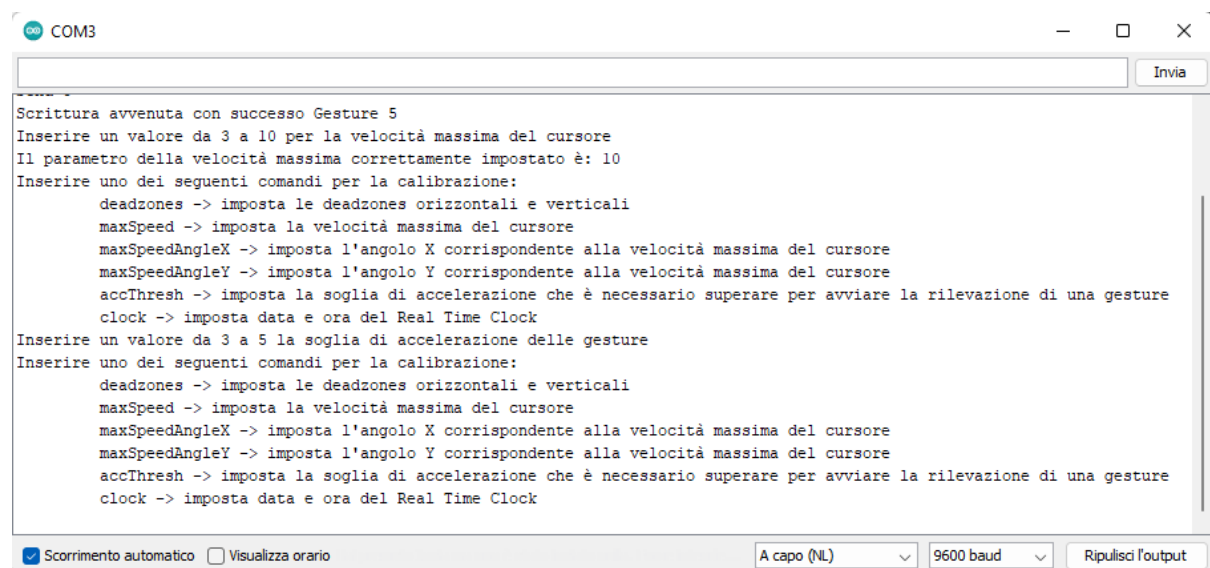
- **maxSpeed**: l'unica azione richiesta dall'utente è quella di inserire un *numero intero compreso tra 3 e 10* che verrà usato come valore per la velocità massima. Nel caso in cui l'utente immetta un valore invalido, viene visualizzato un messaggio di errore e si aspetta nuovamente l'inserimento di un valore corretto.
- **deadzones**: all'inizio della procedura si entra in una fase di attesa in cui si aspetta che l'utente inclini la mano oltre una certa soglia angolare per *5 secondi*; quando tale condizione viene rispettata, inizia l'effettiva fase di calibrazione, in cui l'utente deve restare fermo per qualche secondo con la mano inclinata secondo l'angolo che desidera impostare come deadzone. L'ultimo valore letto al termine dell'attesa sarà il parametro da memorizzare nella struttura dati della libreria di calibrazione.

- **accThresh**: bisogna inserire *un numero intero da 3 a 5*, e la procedura è analoga a quella del comando maxSpeed.
- **maxSpeedAngleX** e **maxSpeedAngleY**: procedura analoga a quella del comando deadzones, con una differenza per quanto riguarda maxSpeedAngleY: in questo caso gli step sono identici ma le inclinazioni della mano sono relative all'asse verticale (inclinazione della mano verso l'alto o verso il basso).

Software Supplementare

Monitor Seriale

Per la fase di calibrazione e in generale per la visualizzazione di *feedback* a schermo durante l'esecuzione del programma è stato impiegato il **monitor seriale** incluso nell'ambiente *Arduino IDE*, tuttavia, qualsiasi software in grado di leggere e scrivere dati da una porta seriale è adatto a tale scopo. L'utente può quindi visualizzare i messaggi e i menu che vengono stampati



The screenshot shows the 'COM3' Serial Monitor window in Arduino IDE. The text inside the window is as follows:

```

Scrittura avvenuta con successo Gesture 5
Inserire un valore da 3 a 10 per la velocità massima del cursore
Il parametro della velocità massima correttamente impostato è: 10
Inserire uno dei seguenti comandi per la calibrazione:
    deadzones -> imposta le deadzones orizzontali e verticali
    maxSpeed -> imposta la velocità massima del cursore
    maxSpeedAngleX -> imposta l'angolo X corrispondente alla velocità massima del cursore
    maxSpeedAngleY -> imposta l'angolo Y corrispondente alla velocità massima del cursore
    accThresh -> imposta la soglia di accelerazione che è necessario superare per avviare la rilevazione di una gesture
    clock -> imposta data e ora del Real Time Clock
Inserire un valore da 3 a 5 la soglia di accelerazione delle gesture
Inserire uno dei seguenti comandi per la calibrazione:
    deadzones -> imposta le deadzones orizzontali e verticali
    maxSpeed -> imposta la velocità massima del cursore
    maxSpeedAngleX -> imposta l'angolo X corrispondente alla velocità massima del cursore
    maxSpeedAngleY -> imposta l'angolo Y corrispondente alla velocità massima del cursore
    accThresh -> imposta la soglia di accelerazione che è necessario superare per avviare la rilevazione di una gesture
    clock -> imposta data e ora del Real Time Clock
  
```

At the bottom of the window, there are controls: a checked box for 'Scorrimento automatico', an unchecked box for 'Visualizza orario', a dropdown menu set to 'A capo (NL)', a dropdown menu set to '9600 baud', and a button labeled 'Ripulisce l'output'.

in tempo reale e scrivere nella barra di invio per inserire i comandi ed effettuare le procedure di calibrazione.

EMLearn

EMLearn un *framework per il machine learning su sistemi embedded*; il suo codice è scritto interamente in *Python* ma ha delle funzioni che permettono di generare delle librerie in C che implementano il funzionamento di una determinata **rete neurale**. Supporta anche alcuni metodi che usano unicamente operazioni sugli interi, per i microcontrollori sprovvisti di Floating Point Unit. La rete neurale che genera è descritta unicamente da un header file e non presenta allocazioni dinamiche; inoltre è già utilizzabile e testabile nell'ambiente Python in cui viene effettuato il training. Presenta i seguenti classificatori:

Supervised:

- *eml_trees*: sklearn.RandomForestClassifier, sklearn.ExtraTreesClassifier, sklearn.DecisionTreeClassifier
- *eml_net*: sklearn.MultiLayerPerceptron, Keras.Sequential with fully-connected layers
- *eml_bayes*: sklearn.GaussianNaiveBayes

Unsupervised / Outlier Detection / Anomaly Detection:

- *eml_distance*: sklearn.EllipticEnvelope (Mahalanobis distance)

Feature extraction:

- *eml_audio*: Melspectrogram

In seguito ad alcuni test, è stato scelto il classificatore **ExtraTreesClassifier**. È risultato essere il *compromesso migliore tra performance della rete* (esprese in accuracy sul validation set), *grandezza dell'header file generato e velocità di esecuzione della sua funzione predict()*. Tra i suoi parametri di configurazione ci sono:

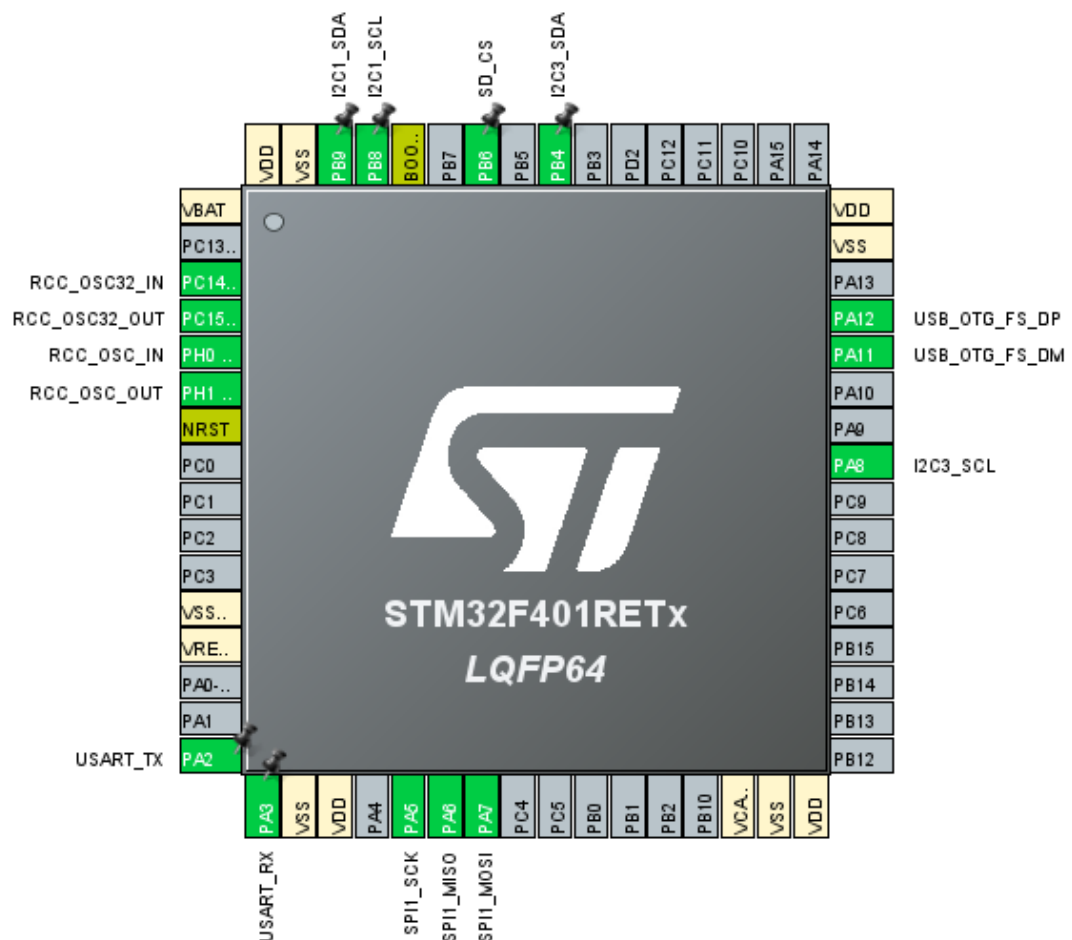
- *numero di stimatori*: 100
- *profondità massima della rete*: nessuna
- *stato random iniziale*: 2

Il *training set* che viene fornito alla rete è stato costruito manualmente registrando *100 letture* da entrambi i sensori GY-521; ogni lettura è costituita da *50 samples*, ed ogni sample è a sua volta formato da una serie di valori letti dal sensore posizionato sul dito indice (letture dell'accelerometro per l'asse x, y e z e letture del giroscopio per i medesimi assi) e dallo stesso set di valori letti dal secondo sensore, posizionato sulle dita medio e anulare.

Interfacce software per la comunicazione utilizzate

Sono state impiegate le seguenti interfacce software per la comunicazione tra i dispositivi:

- **I2C**: utilizzato dai sensori *GY-521* e dal *RTC DS3231*. È un protocollo *sincrono a bus open-collector* che prevede la presenza di almeno un *master* e uno *slave*; i due GY-521 sono stati collocati sullo stesso bus *I2C1*, mentre il RTC, avendo lo stesso indirizzo di uno dei due sensori, ha richiesto l'uso del bus *I2C3*.
- **SPI**: è stato utilizzato per il *lettore di microSD*. È un protocollo *sincrono, full-duplex e master slave*.
- **UART**: UART, o *universal asynchronous receiver-transmitter*, è uno dei protocolli *device-to-device* più usati. È stato utilizzato per la comunicazione tra la scheda *STM32* e il *monitor seriale* in esecuzione sul PC, per lo scambio *bidirezionale* di testo durante la fase di calibrazione.



Risultato Finale

Di seguito sono riportate delle immagini del risultato finale: a sinistra sono visibili i sensori per il movimento e per le gesture indossati sulla mano, mentre a destra è raffigurata la struttura completa con i vari collegamenti tra scheda STM32, breadboard e dispositivi.

