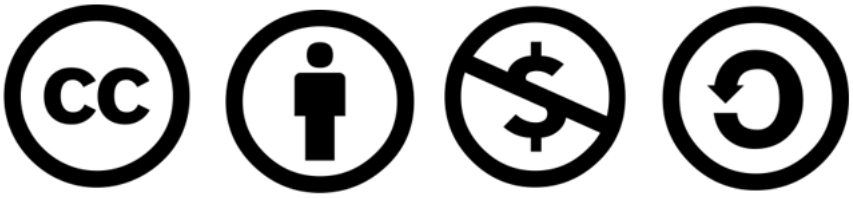


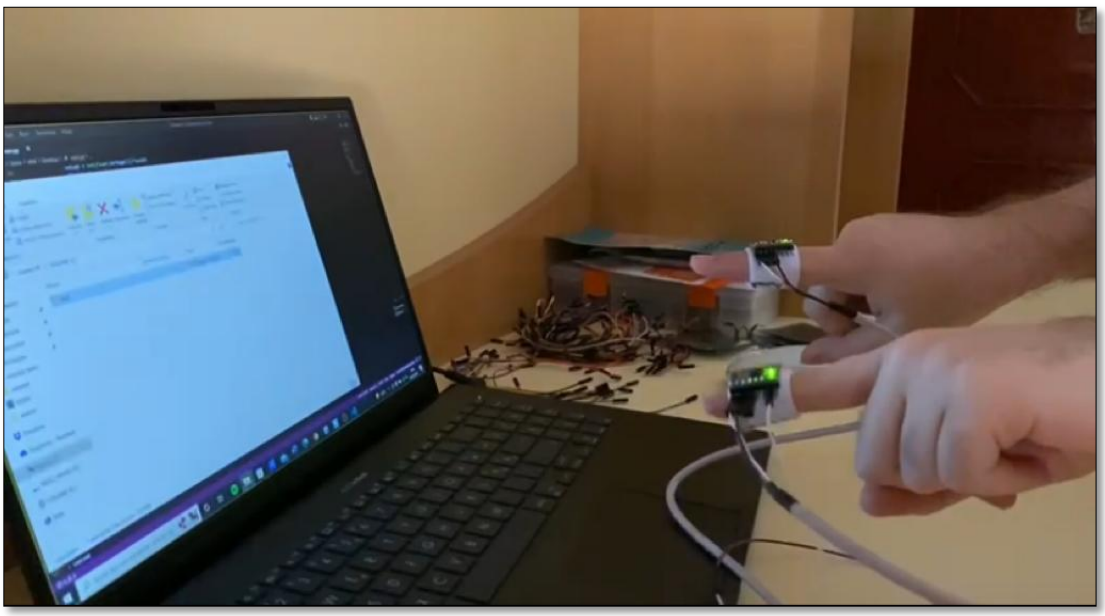
Air Mouse – Fingers

AIR MOUSE – FINGERS

Sistemi Embedded

Final Project

Based on STM32F401RET6



GRUPPO

Vito Turi	0622701795
Ferdinando Sica	0622701794
Camilla Spingola	0622701698
Mattia Marseglia	0622701697

C.d.L. Ingegneria Informatica Magistrale

A.A. 2021/2022

Sommario

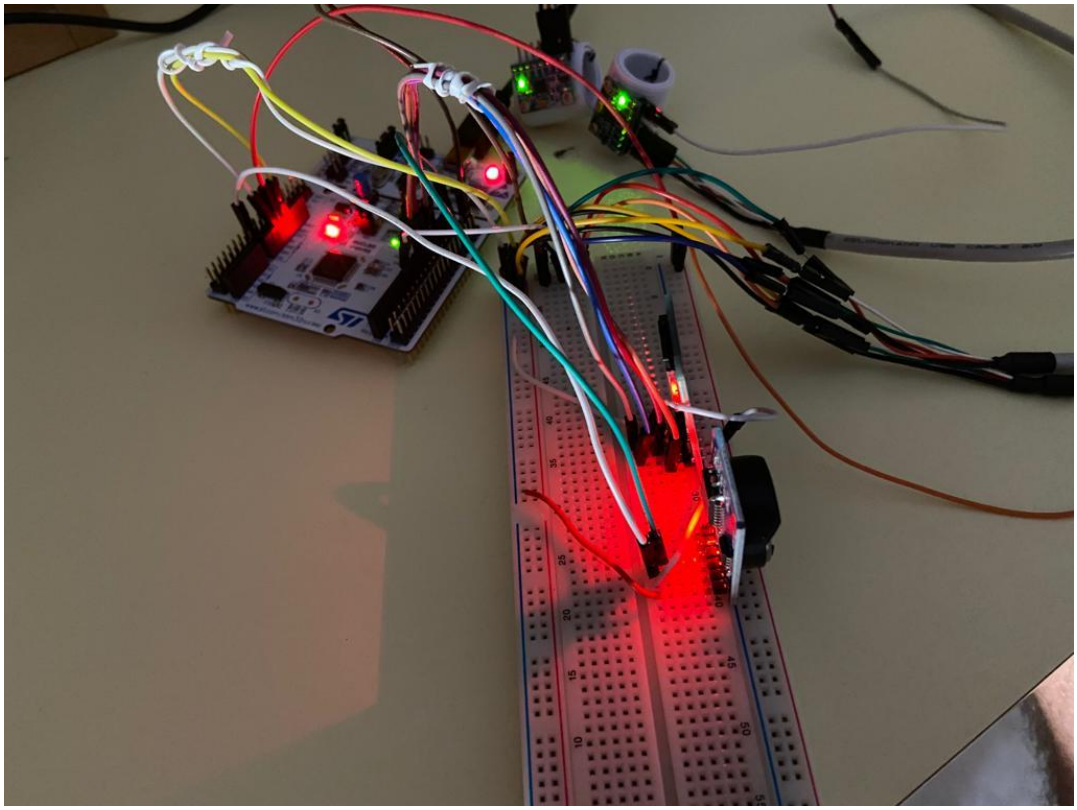
1 Introduzione

1.1 Scopo del documento

Questo documento si pone l'obiettivo di descrivere lo sviluppo del progetto finale del corso di Sistemi Embedded, che ha riguardato la realizzazione di un Air Mouse, utilizzabile tramite le dita di una mano.
Il presente documento viene allegato al progetto finale come documentazione ufficiale, al fine di offrire una descrizione completa delle specifiche funzionali del sistema, nonché della componentistica hardware utilizzata e dell'architettura software implementata.

1.2 Scopo del progetto

Il progetto finale del corso di Sistemi Embedded ha riguardato la realizzazione di un dispositivo Air Mouse controllabile con le mani, in grado di interpretare la maggior parte dei comandi eseguibili attraverso un comune mouse (movimento, click, double click etc.…), che analizzeremo successivamente.



2 Hardware utilizzato

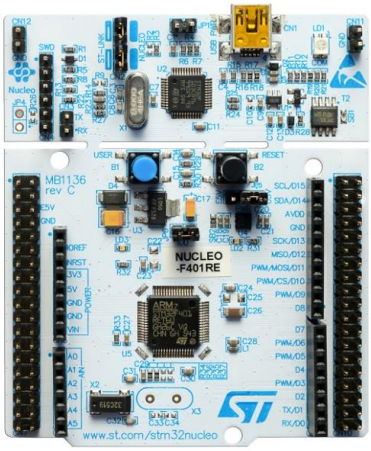
2.1 Descrizione della board

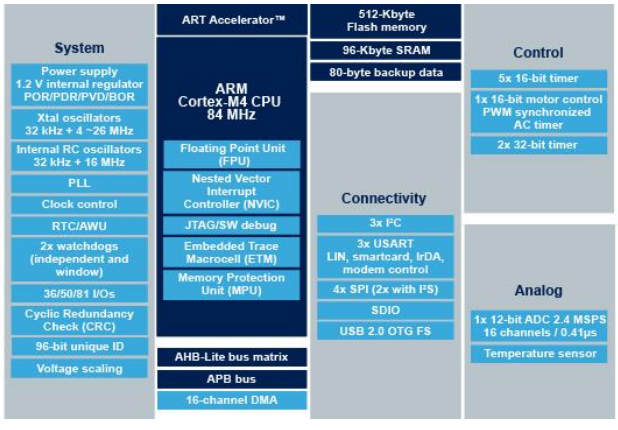
Per la progettazione e la realizzazione dell'Air Mouse, è stata utilizzata la board STM32F401RET6 della serie Nucleo. I dispositivi STM32F401xD/xE sono basati sul core RISC a 32 bit ARMCortex-M4 ad alte prestazioni che opera a una frequenza fino a 84 MHz. Il suo core Cortex-M4 è dotato di un'unità a virgola mobile (FPU) a precisione singola che supporta tutte le istruzioni di processing dei dati e tutti i tipi di dato a singola precisione ARM. Implementa inoltre un set completo di istruzioni DSP e un'unità di protezione della memoria (MPU) che migliora la sicurezza delle applicazioni. STM32F401xD/xE incorpora memorie embedded ad alta velocità (512 Kbyte di memoria Flash, 96 Kbyte di SRAM) e una vasta gamma di I/O e periferiche avanzate collegate a due bus APB, a due bus AHB e a una matrice bus multi-AHB a 32 bit.
Tutti i dispositivi offrono un ADC a 12 bit, un RTC a bassa potenza, 6 timer a 16 bit per uso generico, tra cui un timer PWM per il controllo del motore, due timer a 32 bit per uso generale. Sono inoltre dotati di interfacce di comunicazione standard e avanzate.

Caratteristiche

- Core: CPU Arm32-bit Cortex-M4 con FPU, acceleratore adattivo in tempo reale (ART Accelerator™) che consente l'esecuzione dello stato di attesa 0 dalla memoria Flash, frequenza fino a 84 MHz, unità di protezione della memoria, 105 DMIPS / 1,25 DMIPS / MHz (Dhrystone 2.1) e istruzioni DSP
- Memoria
    - fino a 512 Kbyte di memoria Flash
    - fino a 96 Kbyte di SRAM
- Gestione del clock, del reset e dell'alimentazione
    - Alimentazione dell'applicazione e I/O da 1,7 V a 3,6 V
    - POR, PDR, PVD e BOR
    - Oscillatore a cristallo da 4 a 26 MHz
    - RC interno a 16 MHz
    - Oscillatore 32 kHz per RTC con calibrazione
    - RC interno a 32 kHz con calibrazione
- Consumo energetico
    - Funzionamento: 146 µA/MHz (periferica spenta)
    - Stop (Flash in modalità Stop, fast wake-up time): 42 µA Typ @ 25C; 65 µA max @25 °C
    - Stop (Flash in modalità Deep power down, fast wake-up time): fino a 10 µA @ 25 °C; 30 µA max @25 °C
    - Standby: 2,4 µA @ 25 °C / 1,7 V senza RTC; 12 µA @85 °C @1.7 V
    - alimentazione per RTC: 1 µA @25 °C

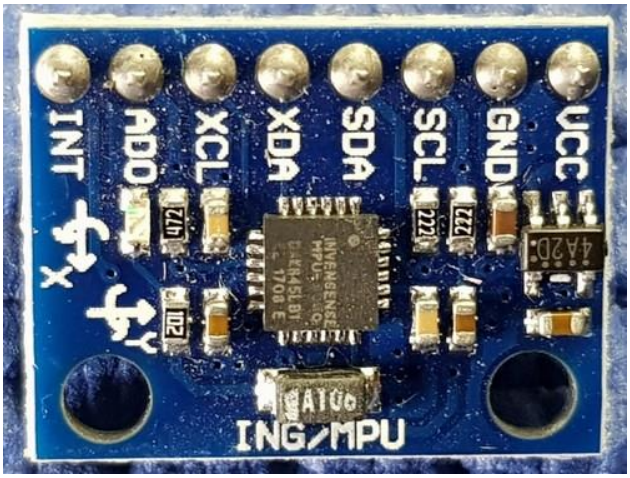
- Convertitore A/D 1×12 bit, 2.4 MSPS: fino a 16 canali
- DMA per uso generico: controller DMA a 16 flussi con FIFO e supporto burst
- Fino a 11 timer: fino a sei timer a 16 bit, due timer a 32 bit fino a 84 MHz, ciascuno con un massimo di quattro IC/OC/PWM o un contatore di impulsi e un ingresso encoder in quadratura (incrementale), due timer watchdog (indipendenti e finestrato) e un timer SysTick
- Modalità di debug
    - Interfacce SWD (Serial Wire Debug) e JTAG
    - Cortex-M4 Embedded Trace Macrocell
- Fino a 81 porte I/O con capacità di interruzione
    - Fino a 78 I/O veloci fino a 42 MHz
    - Tutte le porte I/O sono tolleranti a 5 V
- Fino a 12 interfacce di comunicazione
    - Fino a 3 Interfacce 12C (SMBus/PMBus)
    - Fino a 3 USART (2 x 10,5 Mbit/s, 1 x 5,25 Mbit/s), interfaccia ISO 7816, LIN, IrDA, controllo modem)
    - Fino a 4 SPI (fino a 42Mbit/s a fCPU= 84 MHz), SPI2 e SPI3 con muxed full-duplex I2S per ottenere la precisione della classe audio attraverso l'audio interno PLL o il clock esterno
    - Interfaccia SDIO
    - Connettività avanzata: USB 2.0 full speed device/ host / controller OTG con PHY su chip
- Unità di calcolo CRC
- ID univoco a 96 bit
- RTC: precisione al di sotto del secondo, calendario hardware
- Tutti i pacchetti (WLCSP49, LQFP64/100, UFQFPN48, UFBGA100) sono ECOPACK2

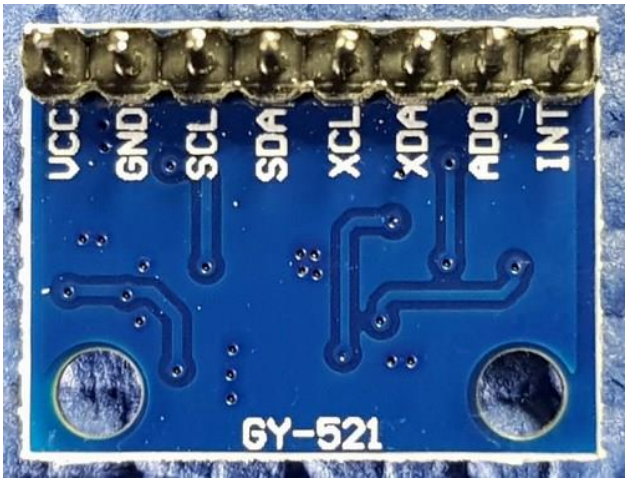


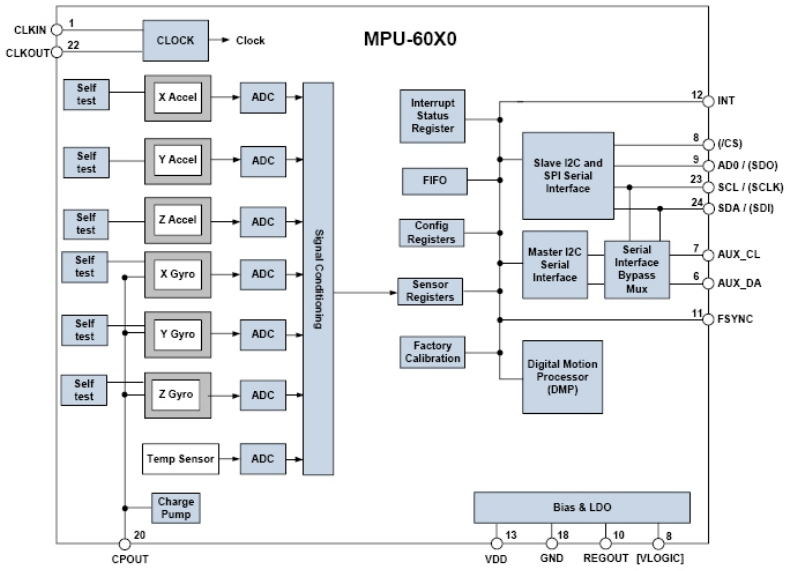


2.2 Descrizione dei sensori

Modulo GY-521 MPU-6050







Il modulo GY-521 è costruito per gestire il sensore MPU-6050 che combina al suo interno più dispositivi, tra i quali notiamo:

- un accelerometro a 3 assi
- un giroscopio a 3 assi
- un sensore di temperatura
- un oscillare di clock interno
- un processore di movimento digitale

Il modulo è di piccole dimensioni: 21.2 x 16.4 x 3.3 mm, è dotato in hardware di conversione analogico-digitale per ciascun canale, e consente l'acquisizione dai canali x, y, z. L'uscita può essere interfacciata con il bus I2C.

Accelerometro a 3 assi

L'accelerometro a 3 assi interno all'MPU-6050 è in grado di rilevare l'angolo di inclinazione o l'inclinazione lungo gli assi x, y, z utilizzando una tecnologia del sistema elettromeccanico (Micro Electro-Mechanical Systems MEMS).
L'accelerazione lungo gli assi è ricavata dalla massa in movimento c, rilevata con la tecnologia del sistema elettromeccanico (MEMS). Per la digitalizzazione dei valori viene utilizzato un ADC a 16 bit. Gli intervalli di uscita a fondo scala possibili sono: +/- 2g, +/- 4g, +/- 8g, +/- 16g.
In posizione normale, cioè quando il dispositivo è posizionato su una superficie piana, i valori sono 0 g sull'asse x, 0 g sull'asse y e +1 sull'asse z.

Giroscopio a 3 assi

L'MPU-6050 è costituito da un giroscopio a 3 assi in grado di rilevare la velocità di rotazione lungo l'asse x, y, z con la tecnologia del sistema micro-elettromeccanico (MEMS). Quando il sensore viene ruotato su qualsiasi asse, viene prodotta una vibrazione, grazie all'effetto Coriolis viene rilevata dal MEMS. L'ADC a 16 bit viene utilizzato per digitalizzare la tensione per campionare ciascun asse. Gli intervalli di uscita a fondo scala sono: +/- 250, +/- 500, +/- 1000, +/- 2000. La velocità angolare viene misurata lungo ciascun asse in gradi al secondo.

Oscillatore di clock interno

L'MPU ha uno schema di temporizzazione flessibile, che consente di utilizzare una varietà di sorgenti di clock interne o esterne per i circuiti sincroni interni. Questa circuiteria sincrona include il segnale di condizionamento e gli ADC, il DMP e i vari circuiti e registri di controllo. Un PLL su chip offre flessibilità negli input consentiti per generare il clock.
Le fonti interne consentite per generare il clock interno sono:

- Un oscillatore di rilassamento interno
- Qualunque giroscopio X, Y o Z (oscillatori MEMS con una variazione di ± 1% rispetto alla temperatura)

Digital motion processor (DMP)

Il processore di movimento digitale incorporato viene utilizzato per eseguire calcoli sui dati del sensore. I valori possono essere letti dai registri dmp. I valori grezzi possono essere letti direttamente dall'accelerometro e dai registri del giroscopio. Il chip MPU-6050 include anche un buffer FIFO da 1024 byte dove i dati possono essere posizionati e letti. L'MPU-6050 funge sempre da slave quando è connesso alla scheda con i pin SDA e SCL collegati al bus I2C. Ma ha anche il proprio controller I2C ausiliario che gli consente di agire come un master su un altro dispositivo (come per esempio un magnetometro). Il DMP integrato può essere programmato con il firmware per eseguire calcoli complessi con i valori del sensore.
L'AD0 seleziona tra gli indirizzi I2C 0x68 e 0x69 consentendo a 2 di questi sensori di essere connessi nello stesso progetto agli stessi pin senza creare sovrapposizione tra i dati. La maggior parte delle schede come la GY-521 ha una resistenza pull down o pull up per mantenere il valore predefinito, nel nostro caso al valore 0x68.

Caratteristica sensore

Alimentazione a 5Vcc

Fondo scala giroscopio selezionabile tra ± 250, ± 500, ± 1000, e ± 2000°/sec

Fondo scala accelerometro selezionabile tra ± 2g, ± 4g, 8g ± e ± 16g

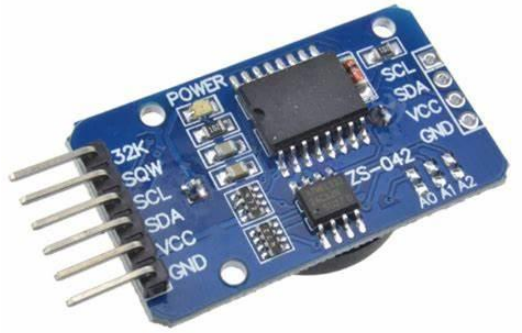
Interfaccia I2C

Assorbimento inferiore a 10mA durante la misura

Descrizione dei Pin

Pin	Descrizione	Funzione
1	VCC	Alimentazione, è possibile alimentare il modulo direttamente a +5V
2	GND	Terminale di massa
3	SCL	Serial clock
4	SDA	Serial data
5	XDA	Serial data per il collegamento a sensori esterni
6	XCL	Serial clock per il collegamento a sensori esterni
7	ADO	Indirizzo slave I2C LSB (AD0)
8	INT	Interrupt digital output (totem pole or open-drain)

Real Time Clock ds3231



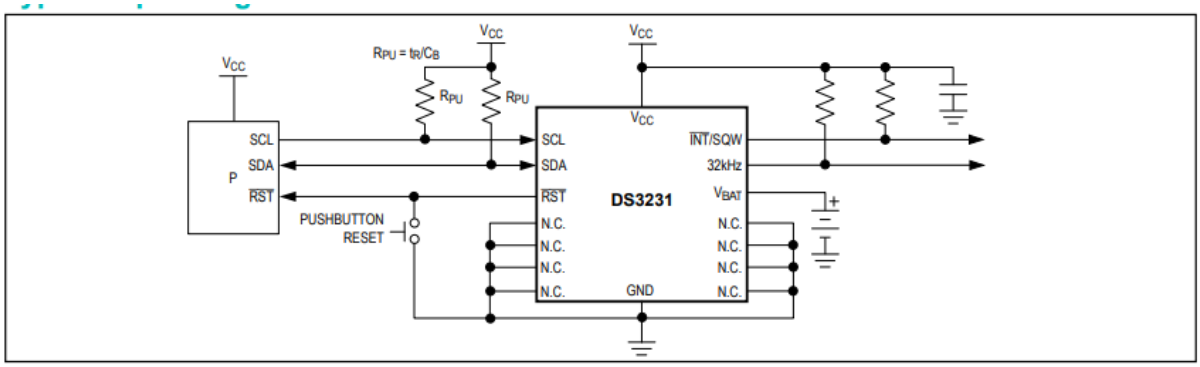
Il DS3231 è un Real Time Clock estremamente preciso e a basso consumo, con un TCXO (temperature compensated crystal oscillator) integrato. Il dispositivo incorpora un ingresso batteria e mantiene un cronometraggio accurato anche quando l'alimentazione principale al dispositivo viene interrotta. L'RTC mantiene secondi, minuti, ore, giorni, mese e anno. La data relativa alla fine del mese viene regolata automaticamente per i mesi con meno di 31 giorni, comprese le correzioni per l'anno bisestile. L'orologio funziona nel formato 24 ore o 12 ore con un indicatore AM/PM. Sono forniti due allarmi programmabili e un'uscita ad onda quadra programmabile. Indirizzi e dati vengono trasferiti in serie attraverso un bus bidirezionale I2C.

Un riferimento di tensione di precisione con compensazione della temperatura e un circuito comparatore controlla lo stato di VCC per individuare interruzioni di corrente, per fornire un'uscita di ripristino e per passare automaticamente all'alimentazione di riserva quando necessario.

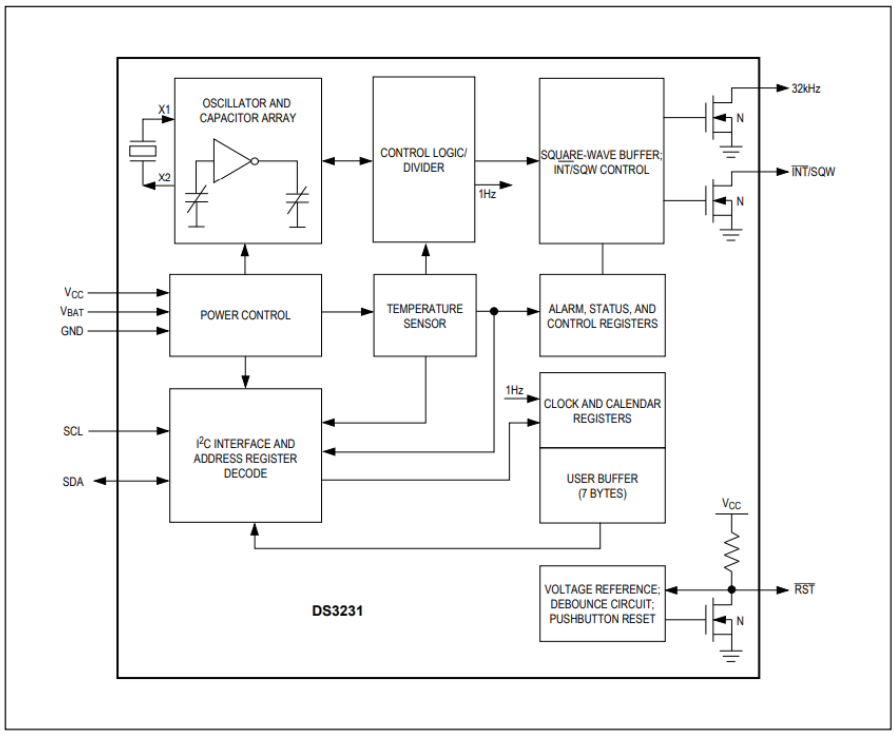
Features

- Highly Accurate RTC Completely Manages All
- Timekeeping Functions
- Real-Time Clock Counts Seconds, Minutes, Hours,
- Date of the Month, Month, Day of the Week, and Year, with Leap-Year Compensation Valid Up to 2100
- Accuracy ±2ppm from 0°C to +40°C
- Accuracy ±3.5ppm from -40°C to +85°C
- Digital Temp Sensor Output: ±3°C Accuracy
- Register for Aging Trim
- RST Output/Pushbutton Reset Debounce Input
- Two Time-of-Day Alarms
- Programmable Square-Wave Output Signal
- Simple Serial Interface Connects to Most Microcontrollers
    - Fast (400kHz) I2C Interface
- Battery-Backup Input for Continuous Timekeeping
    - Low Power Operation Extends Battery-Backup Run Time
    - 3.3V Operation
- Operating Temperature Ranges: Commercial (0°C to +70°C) and Industrial (-40°C to +85°C)

Operating circuit



Block Diagram

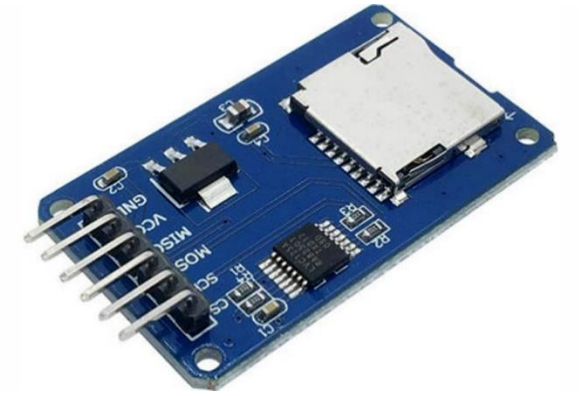


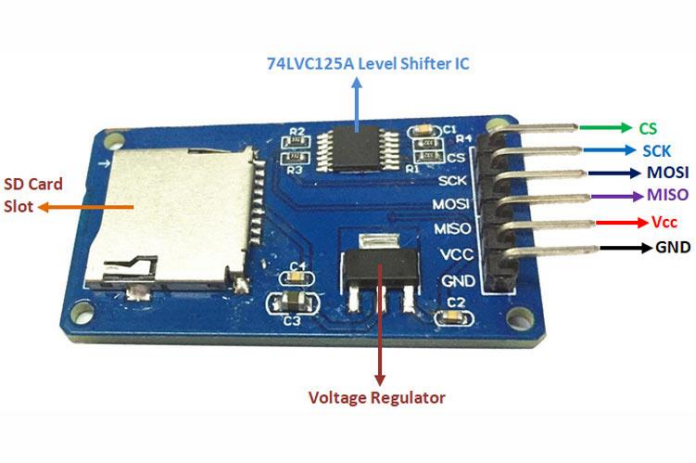
Descrizione Pin

PIN	NAME	FUNCTION
1	32kHz	32kHz Output. This open-drain pin requires an external pullup resistor. When enabled, the output operates on either power supply. It may be left open if not used.
2	V_{CC}	DC Power Pin for Primary Power Supply. This pin should be decoupled using a 0.1µF to 1.0µF capacitor. If not used, connect to ground.
3	$\overline{INT}$/SQW	Active-Low Interrupt or Square-Wave Output. This open-drain pin requires an external pullup resistor connected to a supply at 5.5V or less. This multifunction pin is determined by the state of the INTCN bit in the Control Register (0Eh). When INTCN is set to logic 0, this pin outputs a square wave and its frequency is determined by RS2 and RS1 bits. When INTCN is set to logic 1, then a match between the timekeeping registers and either of the alarm registers activates the $\overline{INT}$/SQW pin (if the alarm is enabled). Because the INTCN bit is set to logic 1 when power is first applied, the pin defaults to an interrupt output with alarms disabled. The pullup voltage can be up to 5.5V, regardless of the voltage on V_{CC}. If not used, this pin can be left unconnected.
4	$\overline{RST}$	Active-Low Reset. This pin is an open-drain input/output. It indicates the status of V_{CC} relative to the V_{PF} specification. As V_{CC} falls below V_{PF}, the $\overline{RST}$ pin is driven low. When V_{CC} exceeds V_{PF}, for t_{RST}, the $\overline{RST}$ pin is pulled high by the internal pullup resistor. The active-low, open-drain output is combined with a debounced pushbutton input function. This pin can be activated by a pushbutton reset request. It has an internal 50kΩ nominal value pullup resistor to V_{CC}. No external pullup resistors should be connected. If the oscillator is disabled, t_{REC} is bypassed and $\overline{RST}$ immediately goes high.
5–12	N.C.	No Connection. Must be connected to ground.
13	GND	Ground
14	V_{BAT}	Backup Power-Supply Input. When using the device with the V_{BAT} input as the primary power source, this pin should be decoupled using a 0.1µF to 1.0µF low-leakage capacitor. When using the device with the V_{BAT} input as the backup power source, the capacitor is not required. If V_{BAT} is not used, connect to ground. The device is UL recognized to ensure against reverse charging when used with a primary lithium battery. Go to www.maximintegrated.com/qa/info/ul.
15	SDA	Serial Data Input/Output. This pin is the data input/output for the I^2C serial interface. This open-drain pin requires an external pullup resistor. The pullup voltage can be up to 5.5V, regardless of the voltage on V_{CC}.
16	SCL	Serial Clock Input. This pin is the clock input for the I^2C serial interface and is used to synchronize data movement on the serial interface. Up to 5.5V can be used for this pin, regardless of the voltage on V_{CC}.

Modulo adattatore per scheda Micro SD

Le schede SD o Micro SD sono ampiamente utilizzate in varie applicazioni, come la registrazione dei dati, la visualizzazione dei dati e molte altre. I moduli Micro SD Card Adapter rendono possibile accedere a queste schede SD con facilità. Il modulo Micro SD Card Adapter è dotato di un'interfaccia SPI e di un regolatore di tensione da 3,3 V integrato per fornire una corretta alimentazione alla scheda SD.





Caratteristiche e specifiche del modulo adattatore per scheda Micro SD

Questa sezione menziona alcune delle caratteristiche e delle specifiche del modulo adattatore per schede Micro SD.

- Tensione di funzionamento: 4,5 V - 5,5 V CC
- Requisito attuale: 0,2-200 mA
- Regolatore di tensione di bordo da 3,3 V
- Supporta il file system FAT
- Supporta micro SD fino a 2GB
- Supporta Micro SDHC fino a 32GB

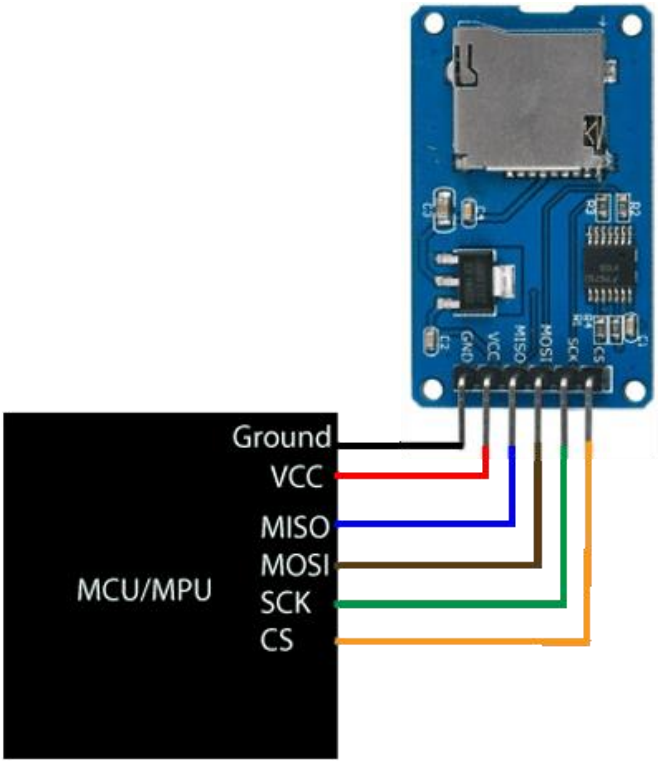
Configurazione Pin del modulo adattatore per scheda Micro SD

Il modulo contiene 6 pin per l'alimentazione e la comunicazione con il controllore. La tabella seguente descrive il tipo di pin e il ruolo di ciascun di ciascuno di essi sul modulo.

Tipo di pin	Descrizione pin
GND	Terra
VCC ·	Ingresso tensione
MISO	Master In Slave Out (SPI)
MOSI ·	Master Out Slave In (SPI)
SCK ·	Orologio seriale (SPI)
CS	Chip Select (SPI)

Collegamento del modulo adattatore per scheda Micro SD a un MCU/MPU

Un modulo adattatore micro-SD card può essere facilmente collegato a un MCU/MPU. Poiché il modulo comunica tramite il protocollo SPI, è necessario collegare MISO, MOSI, SCK e CS del modulo agli MCU.



3 Software utilizzato

3.1 StmCubeIDE

STM32CubeIDE è uno strumento di sviluppo multi-OS all-in-one, che fa parte dell'ecosistema software STM32Cube.

STM32CubeIDE è una piattaforma di sviluppo C/C++ avanzata con funzionalità di configurazione periferica, generazione di codice, compilazione di codice e debug per microcontrollori e microprocessori STM32. Si basa sul framework Eclipse / CDT™ e sulla toolchain GCC per lo sviluppo e GDB per il debug. Permette l'integrazione delle centinaia di plugin esistenti che completano le funzionalità dell'IDE Eclipse.

STM32CubeIDE integra le funzionalità di configurazione e creazione di progetti STM32 di STM32CubeMX per offrire un'esperienza utente all-in-one e risparmiare tempo di installazione e sviluppo. Dopo la selezione di un MCU o MPU STM32 vuoto, o di un microcontrollore o microprocessore preconfigurato dalla selezione di una scheda o dalla selezione di un esempio, viene creato il progetto e generato il codice di inizializzazione. In qualsiasi momento durante lo sviluppo, l'utente può tornare all'inizializzazione e alla configurazione delle periferiche o del middleware e rigenerare il codice di inizializzazione senza alcun impatto sul codice utente.

STM32CubeIDE include analizzatori di compilazione e stack che forniscono all'utente informazioni utili sullo stato del progetto e sui requisiti di memoria.

STM32CubeIDE include anche funzionalità di debug standard e avanzate, tra cui visualizzazioni di registri core CPU, memorie e registri periferici, nonché clock variabile in live, interfaccia Serial Wire Viewer o analizzatore di guasti.

Caratteristiche

- Integrazione dei servizi da STM32CubeMX: STM32 microcontroller, microprocessore, piattaforma di sviluppo ed esempio di selezione del progetto Pinout, clock, periferiche e middleware. Creazione del progetto e generazione del codice di inizializzazione. Software e middleware completati con pacchetti di espansione STM32Cube avanzati.
- Basato su Eclipse/CDT™, con supporto per componenti aggiuntivi Eclipse, toolchain GNU C/C++ for Arm e debugger GDB
- Serie STM32MP1: Supporto per progetti OpenST Linux.
- Altre funzionalità di debug avanzate, tra cui: analisi dei core della CPU, analisi delle periferiche, visualizzazione della memoria (variabili), analisi del sistema e tracciamento in tempo reale (SWV), strumento di analisi dei guasti della CPU e supporto di debug compatibile con RTOS, incluso Azure.
- Supporto per sonde di debug ST-LINK (STMicroelectronics) e J-Link (SEGGER)
- Progetto di importazione da Atollic True STUDIO e AC6 System Workbench per STM32 (SW4STM32)
- Supporto multi-OS: Windows, Linux e macOS, solo versioni a 64 bit

3.2 FreeRTOS

FreeRTOS è un sistema operativo in tempo reale (RTOS) a ingombro ridotto, portatile , preventivo e open source per microcontrollori . È stato portato su oltre 40 architetture diverse. Creato nel 2003 da Richard Barry e dal Team FreeRTOS, è oggi tra i più utilizzati nel mercato dei sistemi operativi in tempo reale.
Il numero di attività in esecuzione contemporaneamente e la loro priorità sono limitate solo dall'hardware. La pianificazione è un sistema di accodamento basato su Semafori e Mutex . Si basa sul modello Round-Robin con gestione delle priorità. Progettato per essere molto compatto, consiste solo di pochi file in linguaggio C e non implementa alcun driver hardware.
I campi di applicazione sono piuttosto ampi, in quanto i principali vantaggi di FreeRTOS sono l'esecuzione in tempo reale, il codice open source e le dimensioni molto ridotte. Viene quindi utilizzato principalmente per sistemi embedded che hanno vincoli di spazio per il codice, ma anche per sistemi di elaborazione video e applicazioni di rete che hanno vincoli di tempo reale.
Il suo design è volutamente minimalista per poter essere installato su piccoli impianti. L'immagine binaria del core pesa tra 4 KB e 9 KB (4,3 KB compilati su ARM7). Il kit minimale ha solo poche funzioni per gestire le attività e la memoria. Inoltre, sono implementate code , semafori e mutex.
Il kernel di versione 7.3.0 è composto di soli cinque file codificati in linguaggio C. Le sezioni in Assembler consentono di garantire la compatibilità di FreeRTOS con le diverse architetture hardware.

Essendo un sistema operativo puramente orientato al tempo reale fornito senza un'applicazione, spesso funge da base per lo sviluppo di API specifiche e/o proprietarie. Viene quindi utilizzato in un'ampia varietà di campi in cui il vincolo del tempo reale è forte, come ad esempio: dispositivi di sorveglianza medica, strumenti di controllo sismico e ambientale o persino il controllo di dispositivi industriali e robot. Le sue funzionalità di gestione delle attività garantiscono l'integrità dei dati raccolti in tempo reale utilizzando i più alti livelli di priorità.

4 Specifiche del sistema

4.1 Specifiche funzionali

L'Air Mouse è grado di interpretare i comandi basilari di un comune mouse. Più nello specifico, il progetto richiedeva di implementare varie funzionalità:

- riprodurre un click singolo attraverso un determinato movimento delle dita (acquisendo i dati dagli accelerometri).
- riprodurre un doppio click singolo attraverso un determinato movimento delle dita (acquisendo i dati dagli accelerometri).
- riprodurre i movimenti del mouse attraverso l'acquisizione dei dati dagli accelerometri.
- riprodurre uno zoom in attraverso un determinato movimento delle dita (acquisendo i dati dagli accelerometri).
- riprodurre uno zoom out attraverso un determinato movimento delle dita (acquisendo i dati dagli accelerometri).
- riprodurre un ALT-TAB attraverso un determinato movimento delle dita (acquisendo i dati dagli accelerometri).
- Registrare ogni evento (es.: movimenti) tramite la l'acquisizione del timestamp dall' RTC e la sua scrittura sulla SD card con una rappresentazione del tipo: from:(x,y,z); to:(x,y ,z); accel: (x,y,z); giroscopio: (x,y,z) ecc.
- Utilizzare task diversi per attività diverse (log, input dei sensori, ecc.), programmato in FreeRTOS (eventualmente con vincoli in tempo reale).
- progettare un protocollo di comunicazione adeguato dal mouse STM32-Air al PC.
- Calibrare l'Air Mouse.

4.2 Specifiche non funzionali

Il software realizzato deve essere in grado di ottenere un funzionamento del mouse tale da:

- essere **Corretto** e **Verificabile**, rispetto ai requisiti funzionali elencati precedentemente.
- essere **Pronto** e **Fluido** nel rispondere ai movimenti delle dita.

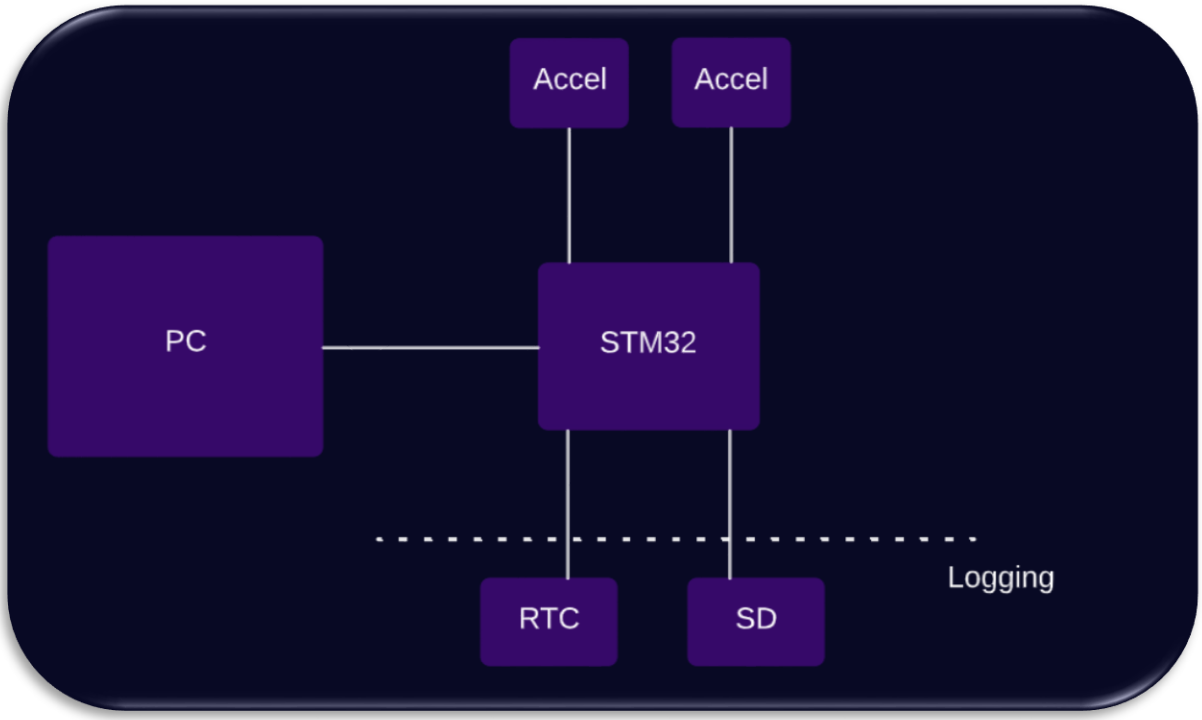
Anche se per questo progetto non sono requisiti obbligatori è comunque buona norma cercare di realizzare un software modulare e facilmente manutenibile.

5 Progettazione

5.1 Schema elettrico

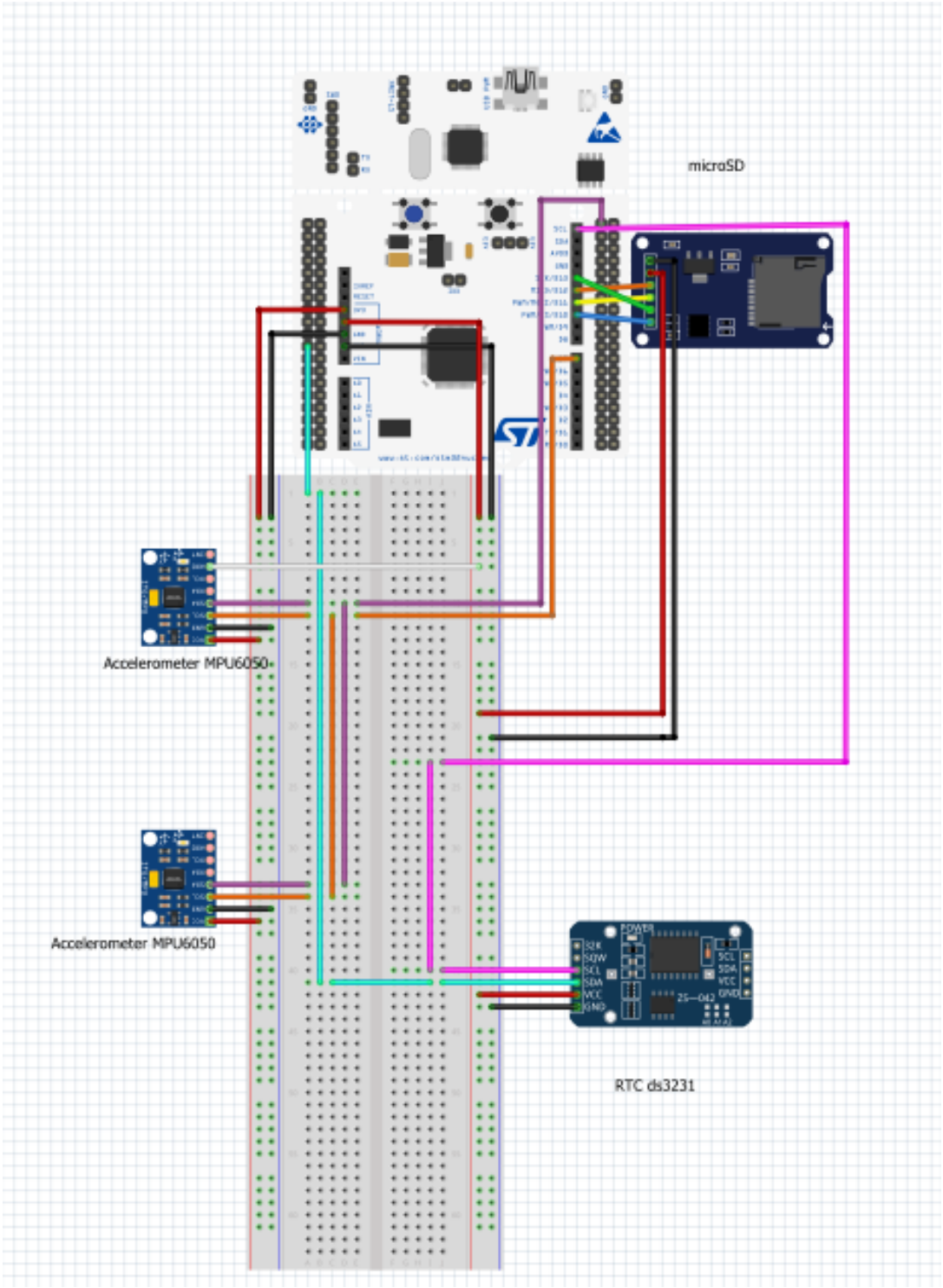
Di seguito vengono riportati degli schemi di interesse per il progetto realizzato. Viene dapprima riportato uno schema semplificativo delle connessioni effettuate, e successivamente viene riportato lo schema dei collegamenti elettrici realizzati.

Visione High Level



Ciò che si vede nell'immagine sopra riportata è una visione di alto livello dello schema hardware. Come già precedentemente detto, i sensori utilizzati sono due accelerometri, un Real Time Clock e un Modulo SD. Dunque, dal punto di vista hardware è stato necessario l'interfacciamento della scheda e quindi del processore con tali periferiche esterne. La scheda a sua volta sarà connessa al PC al quale, non appena il codice sarà stato caricato, trasmetterà i dati ottenuti, che si ripercuoteranno, tramite l'utilizzo di apposite librerie python, in movimenti del mouse. I due accelerometri comunicheranno i valori rilevati, ovvero i valori di accelerazione misurati lungo i tre assi, i quali, a seguito di un'elaborazione, saranno tradotti in movimenti del mouse. Gli altri due sensori, ovvero l'RTC e l'SD saranno invece funzionali allo sviluppo di un file di log, salvato su una scheda SD contenuta all'interno del modulo SD. Nello specifico il Real Time Clock sarà necessario per l'ottenimento di un timestamp, così che sul file si potranno documentare tutti i movimenti registrati e i rispettivi valori associati, collocandoli temporalmente.

Visione low Level



Quella sopra riportata è la visualizzazione basso livello dello schema hardware realizzato. A tale scopo è stato necessario anche l'impiego di una breadboard. La descrizione delle connessioni per ciascun sensore è riportata di seguito, in forma testuale, all'interno della tabella dei pin. È però necessario porre attenzione su alcune connessioni. Sia il Real Time Clock che i due Accelerometri necessitano di canali I2C per la comunicazione con la scheda. Dunque, in tale fase è stato necessario realizzare uno schema di protocollo che consentisse l'acquisizione dei dati da tutti i sensori senza creare delle sovrapposizioni. A tale scopo si è sfruttata la disponibilità della scheda di più di un canale I2C, oltre che la possibilità, associata ai soli accelerometri, di poter mettere a punto uno shift del registro attraverso cui effettuare la comunicazione del master con gli accelerometri. Dunque, un canale I2C, nello specifico I2C3, è stato destinato ai due accelerometri, mentre il canale I2C1 è stato destinato al Real Time Clock. Questo spiega i collegamenti effettuati per tali sensori e anche il motivo

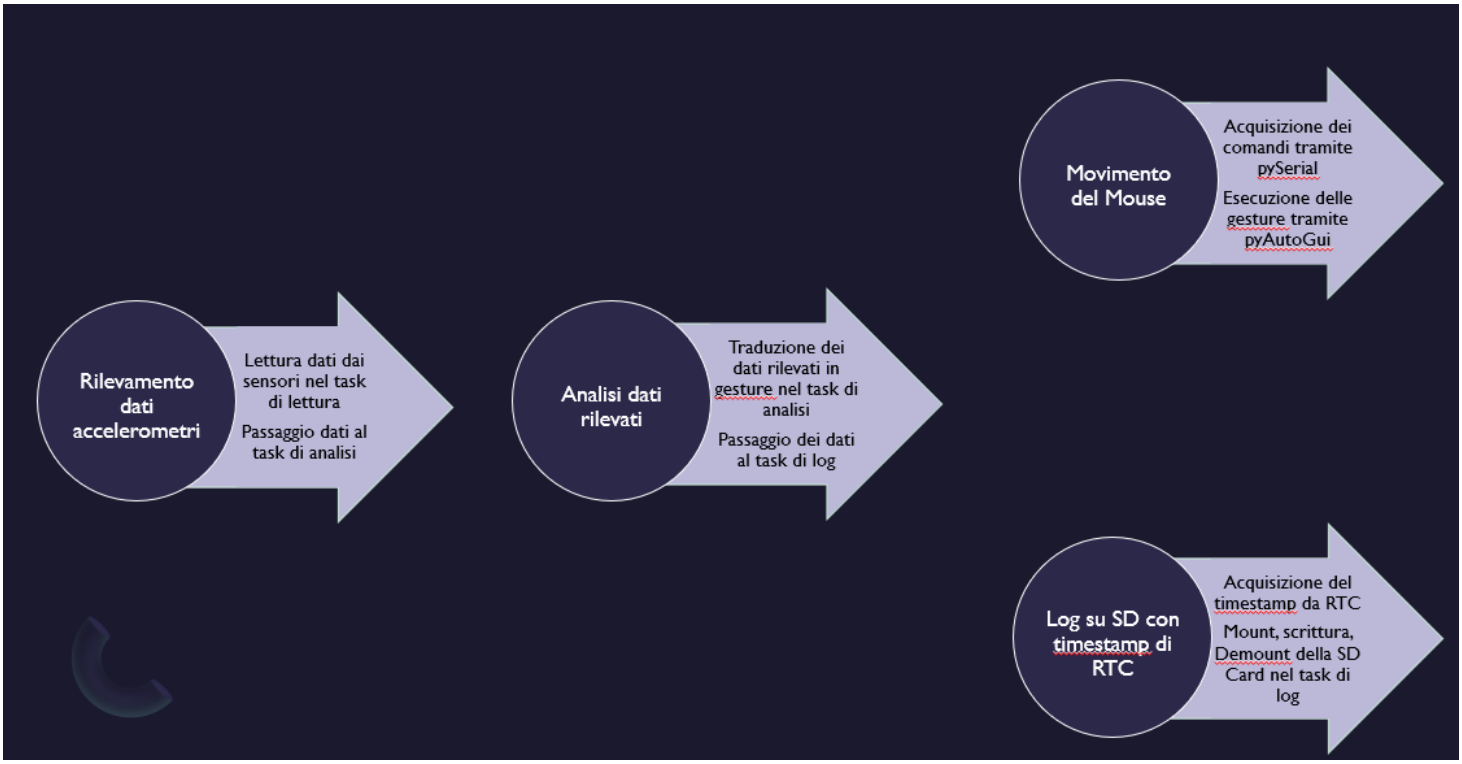
per cui i due pin di SCL e di SDA degli accelerometri siano in serie. Affinché non ci fossero sovrapposizioni tra i valori comunicati dai due diversi accelerometri, per uno dei due è stato effettuato lo shift del registro sopra descritto, in modo che uno dei due fosse identificato dal registro 0x68 mentre l'altro dal registo 0x69. Per effettuare tale shift è stato necessario, per l'accelerometro in questione, realizzare una connessione tra il pin AD0 e l'alimentazione a 5Volt, così come si vede in figura. Il lettore della microSD invece sfrutta una connessione SPI tramite i collegamenti visibili in figura.
Sono state inoltre rispettate le specifiche dei datasheet di ciascun sensore relativamente alla tensione di alimentazione, per cui alcuni risultano essere alimentati a 5Volt altri invece a 3,3Volt.

Tabella dei Pin

Porta	Pin	Link
A	5	SCK Pin (Micro SD Card Module)
A	6	MISO Pin (Micro SD Card Module)
A	7	MOSI Pin (Micro SD Card Module)
A	8	SCL Pins (Two Accelerometers GY-521 MPU6050)
B	6	CS Pin (Micro SD Card Module)
B	7	SDA Pin (Real Time Clock DS3231)
B	8	SCL Pin (Real Time Clock DS3231)
C	9	SDA Pins (Two Accelerometers GY-521 MPU6050)

5.2 Architettura software

Di seguito è riportata una visione di alto livello della struttura software del sistema progettato. Nello specifico già ad alto livello è possibile individuare dei task che successivamente sono stati tradotti in task veri e propri di freeRTOS. I task individuabili sono tre. Il primo è sicuramente quello orientato al rilevamento dei dati dai sensori, in particolare al rilevamento dei valori di accelerazione lungo gli assi registrati dai due accelerometri. Tali dati, presi in forma bruta, vengono poi analizzati da un secondo task, il quale in tale fase di analisi si occupa di discriminare il tipo di variazione registrata associandola all'eventuale comando corrispondente. A questo punto si diramano poi due task la cui evoluzione è indipendente, a differenza di quelli appena descritti che invece prevedono una successione temporale. Tali due task sono uno relativo alla trascrizione del file di log, per il quale è necessario memorizzare l'operazione svolta e il momento di esecuzione di quest'ultima, e quello relativo all'esecuzione del comando richiesto dall'utente.



Per la compilazione del file di log si utilizza il modulo SD, attraverso cui si monta la micoSD, si aggiorna tale file già esistente e si fa il demount successivo. Al fine di avere le diverse operazioni correttamente documentate temporalmente si fa impiego del Real Time Clock, il quale mantiene il tempo dal primo settaggio.

Per far sì che il comando venga impartito si è fatto utilizzo di alcune librerie Python. Nello specifico il comando individuato viene acquisito da tale modulo attraverso pySerial, che si appoggia dunque alla linea seriale. L'esecuzione vera e propria del comando invece prevende l'impiego di pyAutoGui.

Il meccanismo di comunicazione tra i vari task, ad un livello poi più basso si tradurrà nell'impiego di queue per contenere i dati scambiati tra i vari task.

Moduli software

iIl Software di controllo è stato realizzato in linguaggio C. Non essendo stato utilizzato un linguaggio ad oggetti quello di seguito riportato non figura come un diagramma UML delle classi vero e proprio, ma più che altro come una descrizione visiva dei diversi moduli utilizzati all' interno del software, delle relazioni che intercorrono tra questi e della descrizione di eventuali strutture o tipi definite da noi stessi. Di seguito poi si procederà alla descrizione di ciascuno dei moduli.

Modulo SD card

- `void Mount_SD (const TCHAR* path)`, Cancella il vecchio puntatore all'oggetto del file system, e ne crea e registra uno nuovo per poter eseguire operazioni sulla memoria.
- `void Unmount_SD (const TCHAR* path)`, riesegue un mount (il quale cancella il vecchio puntatore) però impostando il nuovo puntatore a NULL, dunque senza crearne uno nuovo.
- `FRESULT Scan_SD (char* pat)`, il quale scorre i file presenti nella microSD passandoli alla scheda la quale tramite UART li trasferisce al computer
- `FRESULT Format_SD (void),` scorre tutti i file presenti nella microSD e ne esegue *l'unlink*, cancellandoli.
- `FRESULT Write_File (char *name, char *data)`, cerca il file (con il nome come primo parametro) nella microSD, successivamente lo crea se non è presente, lo apre in modalità scrittura e chiama l'effettiva f_write() la quale scrive all'interno del file il contenuto del secondo parametro, infine chiude il file.
- `FRESULT Read_File (char *name)`, cerca il file (con il nome come primo parametro) nella microSD, lo apre in modalità lettura se è presente e chiama l'effettiva f_read() la quale legge il contenuto del file, infine chiude il file.
- `FRESULT Create_File (char *name)`, il quale fa un check per verificare se esiste il file; quindi, prova a crearlo e successivamente lo chiude.
- `FRESULT Update_File (char *name, char *data)`, esegue le operazioni fatte dalla funzione Write_file(char*,char*), ma appendendo il secondo parametro invece di sovrascrivere il file.
- `FRESULT Remove_File (char *name)`, cerca il file (con il nome come primo parametro) nella microSD, successivamente lo rimuove se presente.
- `FRESULT Create_Dir (char *name)`, prova a creare la cartella tramite la funzione f_mkdir(char *).
- `void Check_SD_Space (void)`, controlla lo spazio libero nell microSD tramite funzione f_getfree(…) successivamente ricava lo spazio finale e trasmette su UART i due valori.

Accelerometro

- `void MPU6050_Init(I2C_HandleTypeDef *I2Chnd)`, Copia l'handle I2C CubeMX in una variabile locale.
- `void I2C_Read(uint8_t ADDR, uint8_t *i2cBif, uint8_t NofData, int r)`, legge dei dati tramite I2C (presenti nell'indirizzo passato come parametro) dagli accelerometri scrivendoli in un vettore passato come parametro; questa funzione è utilizzata ad altre funzioni presenti all'interno della libreria (quelle di lettura effettive).

- `void I2C_Write8(uint8_t ADDR, uint8_t data)`, scrive i dati tramite I2C negli accelerometri passati come parametro questa funzione è utile ad altre funzioni presenti all'interno della libreria (quelle di configurazione).
- `void MPU6050_Config(MPU_ConfigTypeDef *config)`, configura gli accelerometri con tutti i settaggi passati come parametro incluso scaling, filtri, etc….
- `uint8_t MPU6050_Get_SMPRT_DIV(void)`, esegue la get del divisore di frequenza di campionamento.
- `void MPU6050_Set_SMPRT_DIV(uint8_t SMPRTvalue)`, esegue il set del divisore di frequenza di campionamento.
- `uint8_t MPU6050_Get_FSYNC(void)`, esegue la get della sincronizzazione dei frame esterni.
- `void MPU6050_Set_FSYNC(enum EXT_SYNC_SET_ENUM ext_Sync)`, esegue il set della sincronizzazione dei frame esterni.
- `void MPU6050_Get_Accel_RawData(RawData_Def *rawDef, int r)`, questa funzione legge lo stato degli accelerometri dai registri e se può legge i valori di accelerometro lungo i vari assi immagazzinandoli in una struct.
- `void MPU6050_Get_Accel_Scale(ScaledData_Def *scaledDef)`, la quale chiama MPU6050_Get_Accel_RawData(RawData_Def *, int) e riscala i valori adeguatamente ai parametri settati salvandoli in una nuova struct (quella effettivamente usata dall'utente).
- `void MPU6050_Get_Accel_Cali(ScaledData_Def *CaliDef)`, Ottiene i dati di MPU6050_Get_Accel_Scale(ScaledData_Def *) ma li calibra.
- `void MPU6050_Get_Gyro_RawData(RawData_Def *rawDef)`, questa funzione legge lo stato degli accelerometri dai registri e se può legge i valori del giroscopio lungo i vari assi immagazzinandoli in una struct.
- `void MPU6050_Get_Gyro_Scale(ScaledData_Def *scaledDef)`, la quale chiama MPU6050_Get_Gyro_RawData(RawData_Def *, int) e riscala i valori in maniera corretta salvandoli in unna nuova struct (quella effettivamente usata dall'utente).
- `void _Accel_Cali(float x_min, float x_max, float y_min, float y_max, float z_min, float z_max)`, esegue il set dei bias utilizzati per la calibrazione.

RTC (RealTimeClock)

- ```
  typedef struct {
        int16_t seconds;
        int16_t minutes;
        int16_t hour;
        int16_t dayofweek;
        int16_t dayofmonth;
        int16_t month;
        int16_t year;} DS3231Time
  ```
  , è l'oggetto utilizzato per ottenere I valori delle letture dall'RTC
- `uint8_t decToBcd(int16_t val)`, Converti i normali numeri decimali in decimali codificati binary.
- `int16_t bcdToDec(uint8_t val)`, Converti numeri decimali codificati binari in normali numeri decimali.
- `void set_time (uint8_t sec, uint8_t min, uint8_t hour, uint8_t dow, uint8_t dom, uint8_t`
  ```

`month, uint8_t year)`, esegue il set del time sull'RTC con i parametri passati, scrivendoli tramite protocollo I2C all'indirizzo `0xD0 su 0x00`

- `void get_time (void)`, ottiene il tempo dall'RTC tramite una mem_read fatta all'indirizzo 0xD0 usato anche nella funzione di set, e scrivendo i valori nella struttura DS3231Time.
- `float get_temp (void)`, ottiene la temperatura dall'RTC tramite una mem_read fatta all'indirizzo 0xD0, 0x11 usato anche nella funzione di set, tornando il valore
- `void force_temp_conv (void)`, tale funzione legge il valore dello status register e controlla se il bit BYS è settato, tale bit serve a controllare se il dispositivo è impegnato nelle funzioni TCXO, dopo di che legge il valore del registro di controllo e setta il valore di tale registro forzando a 1 il valore del bit CONV. Dunque, tale funzione serve per forzare la conversione della temperatura.

Main

Il Main risulta pricipalmente un modulo di configurazione iniziale prima di far partire il kernel di freeRTOS, in particolare chiama:

- `int main(void)`, questa è la prima funzione chiamata all'avvio del programma, è importante per configurare tutto ciò che è necessario prima di far partire il kernel di freeRTOS; in particolare chiama:
  - `HAL_Init()`, per Configurare il prefetch della Flash, la cache per le istruzioni e la cache dei dati; per Impostare la priorità delle interrupt; Usare systick come fonte di base dei tempi e configurare tick a 1ms; configurare l'hardware a basso livello.
  - `SystemClock_Config()`, in modo da Configurare la tensione di uscita del regolatore interno principale, poi Inizializzare gli oscillatori RCC in base ai parametri specificati nella struttura RCC_OscInitTypeDef e inizializzare i clock dei bus CPU, AHB e APB.
  - `MX_GPIO_Init()`, inizializzare le porte di i/o `(GPIO Ports Clock Enable, Configure GPIO pin Output Level, Configure GPIO pin)`
  - `MX_SPI1_Init()`, per inizializzare la comunicazione tramite SPI (per il lettore della microSD)
  - `MX_USART2_UART_Init()`, per inizializzare la comunicazione UART con il PC
  - `MX_FATFS_Init()`, per inizializzare il file system in modo da poter usufruire della microSD
  - `MX_I2C1_Init()` e `MX_I2C3_Init()`, per inizializzare la comunicazione tramite I2C utilizzata poi dai due accelerometri e dal Real Time Clock.
  - `MPU6050_Init(&hi2c3)`, per inizializzare gli accelerometri i quali entrambi comunicheranno tramite l'handle di I2C3
  - `myMpuConfig`, per configurare gli accelerometri MPU 6050 (es. sensibilità, ecc..)
  - `MPU6050_Config(&myMpuConfig)`, per configurare gli accelerometri
  - `Mount_SD("/"), Format_SD(), Create_File("ADT.TXT"), Unmount_SD("/")`, funzioni per aggiungere il file ADT.txt che sarà utilizzato per il log
  - `osKernelInitialize()`, per inizializzare il kernel

  - `MX_FREERTOS_Init()`, per creare le due code e i quattro task che serviranno per il corretto funzionamento del kernel
  - `osKernelStart()`, per far partire il kernel dopo aver terminato le configurazioni

- `void SystemClock_Config(void)`, Configura la tensione di uscita del regolatore interno principale, Inizializza gli oscillatori RCC in base ai parametri specificati nella struttura *RCC_OscInitTypeDef*., Inizializza i clock dei bus CPU, AHB e APB.

FreeRTOS

- `void MX_FREERTOS_Init(void)`, all'interno di tale funzione vengono create le code *LtoAqueueHandle* e *AtoLogqueueHandle*, la prima destinata alla comunicazione tra il task di lettura dei valori dai sensori e quello di analisi di tali dati e la seconda destinata alla comunicazione tra il task di analisi e quello deputato alla scrittura dei file di log. In fase di creazione di tali code, i valori passati alla funzione *osMessageQueueNew* sono quelli relativi alla dimensione della coda, al tipo dei valori che conterrà e gli attributi definiti in fase di configurazione, tra cui il nome. Si controlla la corretta creazione delle code che potrebbe essere ostacolata da spazio insufficiente nella memoria heap. Si procede poi alla creazione dei task *defaultTaskHandle*, *LectureTaskHandle*, *AnalysisTaskHandle*, *LogTaskHandle*. Alla funzione *osThreadNew* deputata alla creazione di tali task, si passano come argomenti le funzioni associate ai task (rispettivamente *StartDefaultTask*, *LectureFunction*, *AnalysisFunction*, *LogFunction*), e gli attributi di questi ultimi.

- `void StartDefaultTask(void *argument)`, funzione associata al task di default, contenente le operazioni svolte da quest'ultimo ogni qual volta risulta essere attivo (non è stata utilizzata nella nostra applicazione).

- `void LectureFunction(void *argument)`, funzione associata al *LectureTask*. Tale funzione, ogni qual volta il task è in esecuzione si occupa di prelevare i valori rilevati dagli accelerometri, inserirli all'interno della struttura dati "*Datastruct*" il cui tipo è *myData* e di inviarli sulla coda *LtoAqueue*.

- `void AnalysisFunction(void *argument)`, funzione associata *all'AnalysisTask*, controlla che la coda *LtoAqueue* non sia vuota, e che quindi il task precedente gli abbia effettivamente mandato dei valori da analizzare. Procede dunque all'analisi di tali valori, dopo averli adeguatamente scalati e ne individua la funzionalità corrispondente che deve essere effettuata. Salva tali valori all'interno di una struttura "*all_dataStruct*" il cui tipo è *myData_2*, contenente sia i valori degli accelerometri che la gesture ad essi associata, e procede all'invio di tale struttura sulla coda *AtoLogqueueHandle*, trasmettendoli inoltre sulla huart, da cui saranno prelevati da pySerial.

- `void LogFunction(void *argument)`, funzione associata al *LogTask*, controlla che la coda *AtoLogqueueHandle* non sia vuota, e che quindi il task precedente gli abbia effettivamente mandato dei valori da trascrivere sul file di Log. Rileva il timestamp dall'RTC e scrive tutte le informazioni necessarie sul file contenuto nell'SD Card. Ogni volta si occupa di fare il "mount" e "l'unmount" dell'SD.

Descrizione dei task e delle code create

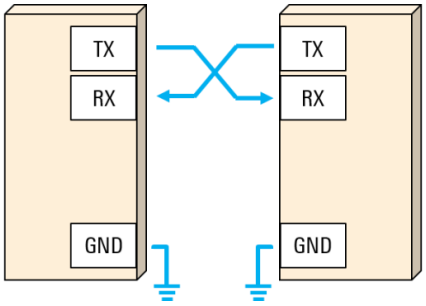
Per la realizzazione del software si è previsto l'impiego di FreeRTOS. Come precedentemente descritto, sono stati creati tre task diversi, a medesima priorità e con diverse funzionalità. I task implementati sono stati individuati a partire dalle

funzionalità principali che dovevano essere offerte dal nostro sistema. Dunque, è stato realizzato un primo task *LectureTask* deputato all'acquisizione dei dati dagli accelerometri, che ha una stack size di 128 word e la cui priorità è *RealTime*. Il secondo task *AnalysisTask* deputato a riconoscere, a seconda dei valori ottenuti dagli accelerometri, quale sia il movimento da compiere e inviare tale analisi alla uart in modo da comunicare poi con i moduli python, ha una stack size di 128 word e una priorità *RealTime*. Il terzo task *LogTask*, invece, è deputato alla scrittura del file di log sulla scheda SD e ha una stack size maggiore, pari a 2048 word, e una priorità *RealTime*. Dunque, il task di lettura è un task periodico. L'attività degli altri due task è invece condizionata alla presenza di valori da analizzare all'interno delle code. Nello specifico le code create sono state due: la prima *LtoAqueue* con una dimensione di 16 e predisposta a contenere valori del tipo myData, funge da connessione tra il task di lettura e quello di analisi, la seconda *AtoLogqueue* con una dimensione di 16 e predisposta a contenere valori del tipo my_Data2, funge da connessione tra il task di analisi e quello di log.

6 Interfaccia di comunicazione

6.1 Interfaccia UART

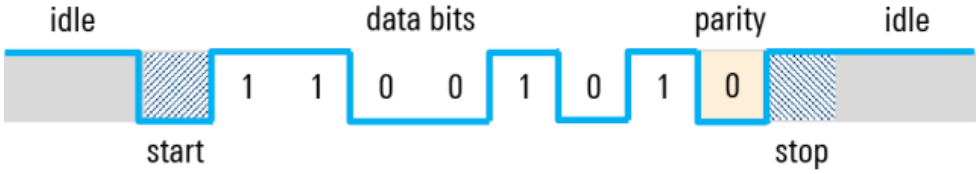
UART sta per Universal Asynchronous Receiver / Transmitter (ricevitore / trasmettitore asincrono universale) e definisce un protocollo, o un insieme di regole, per lo scambio di dati seriali tra due dispositivi. Il protocollo UART è molto semplice



e utilizza solo due fili fra il trasmettitore e il ricevitore per trasmettere e ricevere in entrambe le direzioni. Entrambe le estremità hanno anche una connessione a massa. La comunicazione UART può essere simplex (i dati sono inviati in una sola direzione), half-duplex (ogni lato parla ma solo uno alla volta) o full-duplex (entrambi i lati possono trasmettere contemporaneamente). I dati in un'interfaccia UART sono trasmessi sotto forma di frame. Il formato e il contenuto di questi frame sono brevemente descritti e spiegati.

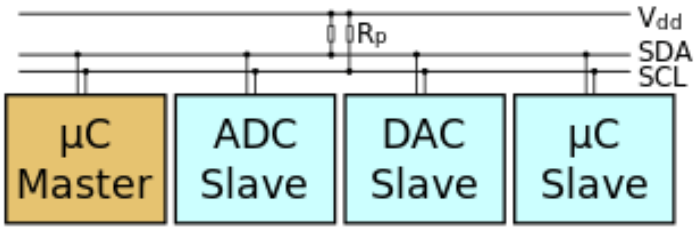
Uno dei grandi vantaggi del protocollo UART è che è asincrono: il trasmettitore e il ricevitore non condividono un segnale di clock comune. Anche se questo semplifica molto il protocollo, pone alcuni requisiti al trasmettitore e al ricevitore. Poiché non condividono un clock, entrambe le estremità devono trasmettere alla stessa velocità prestabilita per avere la stessa tempistica dei bit. I baud rate UART più comuni in uso oggi sono 4800, 9600, 19,2K, 57,6K e 115,2K. Oltre ad avere lo stesso baud rate, entrambi i lati di una connessione UART devono anche utilizzare la stessa struttura di frame e gli stessi parametri. Per poter meglio comprendere di seguito si riporta la struttura di un frame UART.

Formato dei frame



6.2 Interfaccia I2C

Il protocollo I2C (pronuncia I-quadro-C, in Inglese I-squared-C) è stato creato dalla Philips Semiconductors nel 1982; la

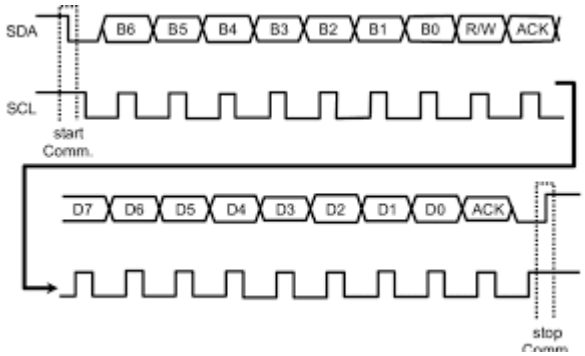


sigla, comunemente indicata anche con I2C, sta per Inter-Integrated Circuit. Il protocollo permette la comunicazione di dati tra due o più dispositivi I2C utilizzando un bus2 a due fili, più uno per il riferimento comune di tensione, come si vede nella figura riportata. Dunque, in tale protocollo le informazioni sono inviate serialmente usando una linea per i dati (SDA: Serial Data line) ed una per il Clock (SCL: Serial Clock line). Deve inoltre essere presente una terza linea: la massa, comune a tutti i dispositivi.

Sequenza di trasferimento dati

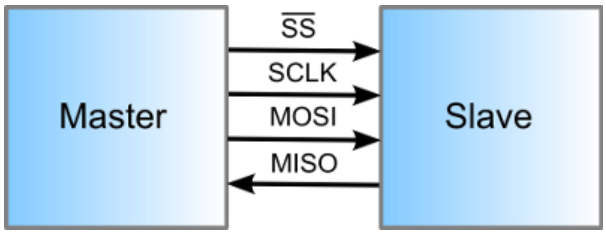
Nello scambio di tali messaggi si configurano due tipologie di attori, master e slave. Una sequenza elementare di lettura o scrittura di dati tra master e slave segue il seguente ordine, come riportato in figura:

1. Invio del bit di START (S) da parte del master
2. Invio dell'indirizzo dello slave (ADDR)7 ad opera del master
3. Invio del bit di Read (R) o di Write (W), che valgono rispettivamente 1 e 0 (sempre ad opera del master)
4. Attesa/invio del bit di Acknowledge (ACK)
5. Invio/ricezione del byte dei dati (DATA)
6. Attesa/invio del bit di Acknowledge (ACK)
7. Invio del bit di STOP (P) da parte del master



6.3 Interfaccia SPI

L'interfaccia SPI venne originariamente ideata dalla Motorola (ora Freescale) a supporto dei propri microprocessori e microcontrollori. A differenza dello standard I2C ideato dalla Philips, l'interfaccia SPI non è stata mai standardizzata; ciononostante è divenuta di fatto uno standard. Il non avere delle regole ufficiali ha portato all'aggiunta di molte caratteristiche ed opzioni che vanno opportunamente selezionate ed impostate al fine di permettere una corretta comunicazione tra le varie periferiche interconnesse.



L'interfaccia SPI descrive una comunicazione singolo Master - singolo Slave, ed è di tipo sincrono e full-duplex. Il clock viene trasmesso con linea dedicata (non necessariamente una trasmissione sincrona ha una linea dedicata al clock) ed è possibile sia trasmettere che ricevere dati in contemporanea. La figura sopra riportata rappresenta uno schema base di collegamento tra due periferiche che fanno uso dell'interfaccia SPI.

Dalla Figura è possibile subito notare quanto appena detto, ovvero che la comunicazione avviene generalmente tra un Master ed uno Slave. L'interfaccia presenta inoltre 4 linee di collegamento (esclusa la massa comunque necessaria), per cui lo standard SPI è anche noto come 4 Wire Interface.

È compito del Master avviare la comunicazione e fornire il clock allo Slave. La nomenclatura delle varie linee presenti nell'interfaccia SPI è normalmente la seguente:

MOSI : Master Output Slave Input

MISO : Master Input Slave Output

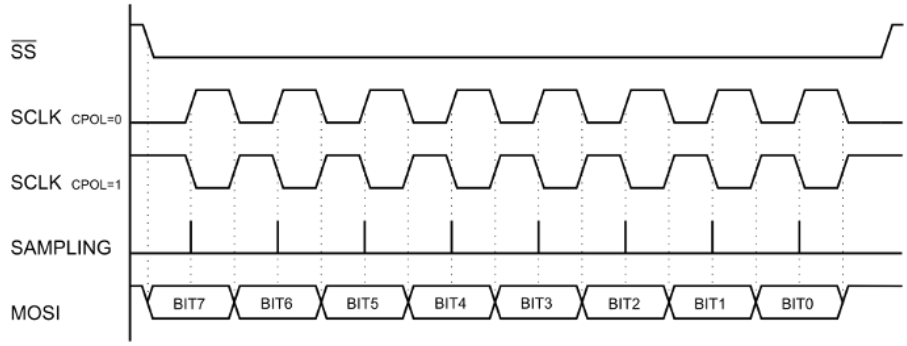
SCLK : Serial Clock (generato dal Master)

CS/SS : **C**hip **S**elect, **S**lave **S**elect, emesso dal master per scegliere con quale dispositivo slave vuole comunicare

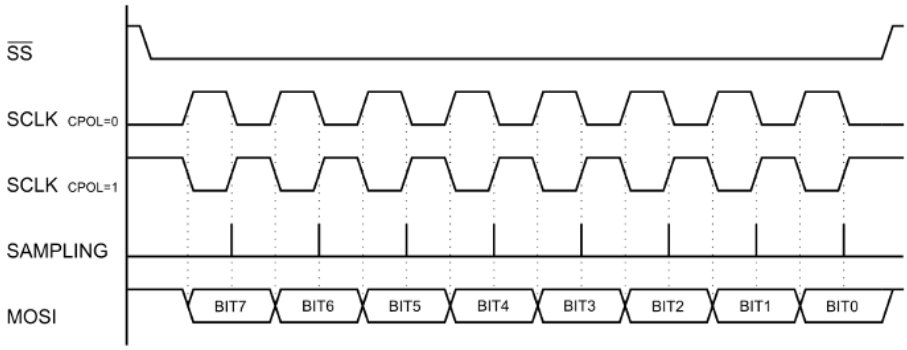
Sequenza di trasferimento dati

Vi sono due diverse modalità per il trasferimento dei dati:

CPHA=1:



CPOL=0:



7 Software a supporto

Affinché si potessero tradurre le elaborazioni fatte sui dati bruti dell'accelerometro in gesture e movimenti del mouse, sono state utilizzate alcune librerie esterne. Nello specifico il modulo pySerial è un modulo che incapsula l'accesso alla porta seriale, mentre pyAutoGui consente, attraverso script python di realizzare un controllo semplice della tastiera e del mouse.

7.1 Librerie esterne

PyAutoGui

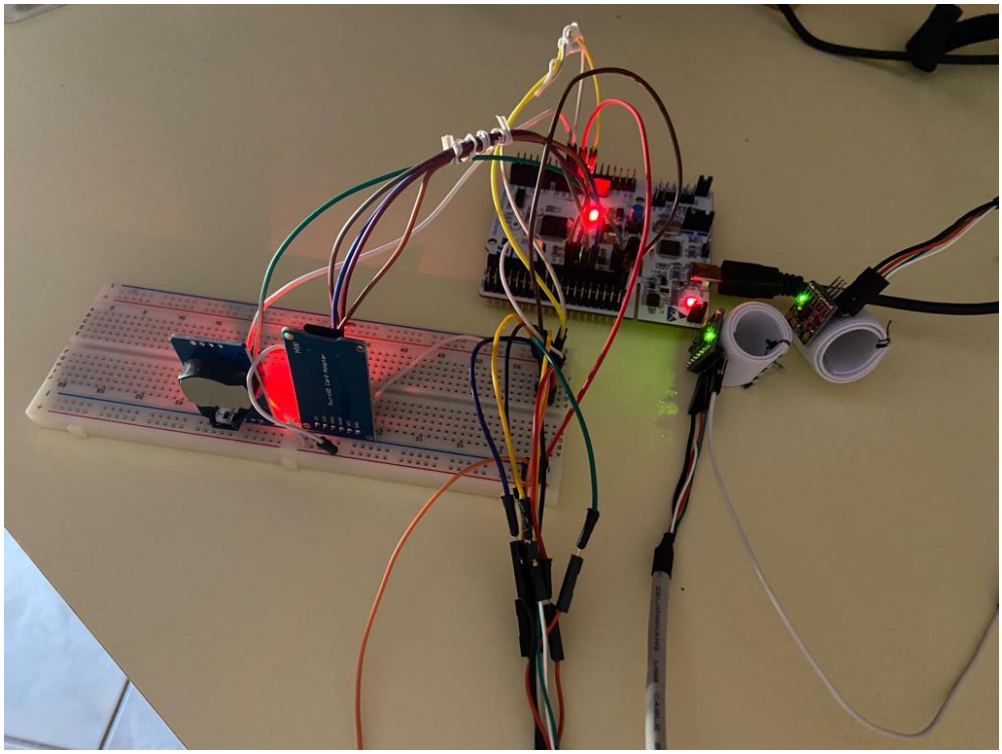
PyAutoGUI è un modulo Python di automazione GUI multipiattaforma per esseri umani, utilizzato per controllare a livello di codice il mouse e la tastiera. Mette a disposizione diverse funzionalità: movimento del mouse e realizzazione dei click anche in finestre di altre applicazioni, invio di sequenze di tasti alle diverse applicazioni, individuazione delle finestre delle applicazioni con possibilità di spostarle, ridimensionarle, ingrandirle, ridurle a icona o chiuderle. (Tali funzionalità attualmente sono integralmente supportate solo in windows). Consente inoltre la visualizzazione di finestre di avviso e di messaggio.

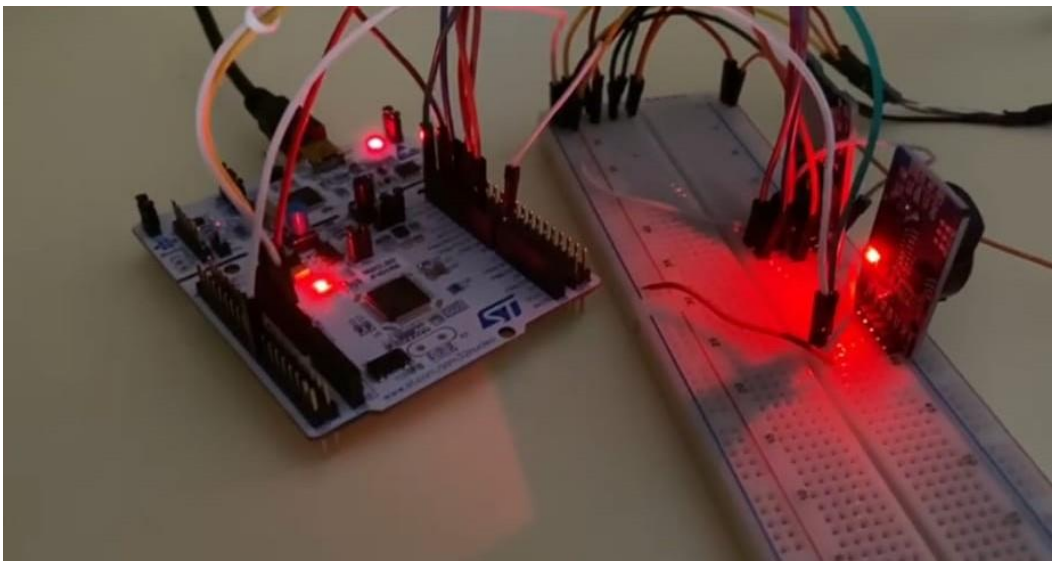
PySerial

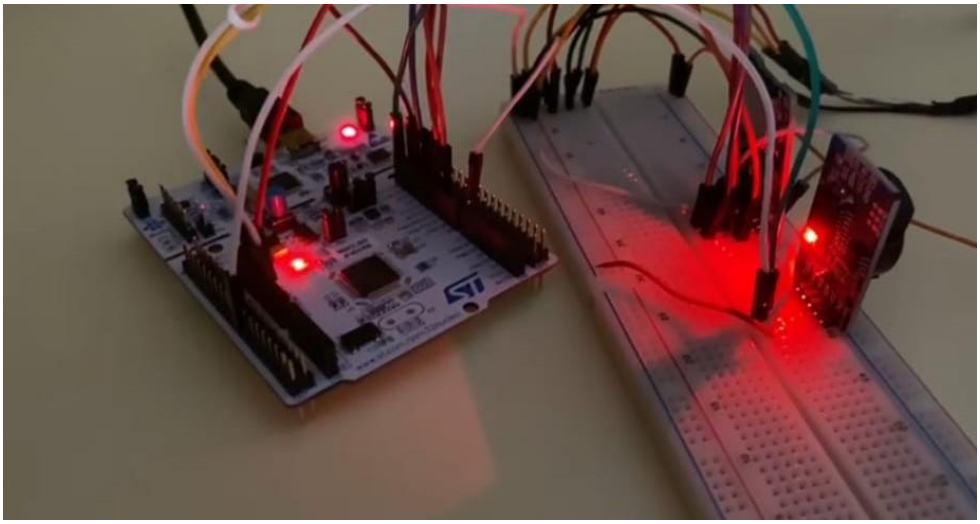
Questo modulo incapsula l'accesso per la porta seriale. Fornisce backend per Python in esecuzione su Windows, OSX, Linux, BSD (possibilmente qualsiasi sistema conforme a POSIX) e IronPython. Il modulo denominato "seriale" seleziona automaticamente il backend appropriato.

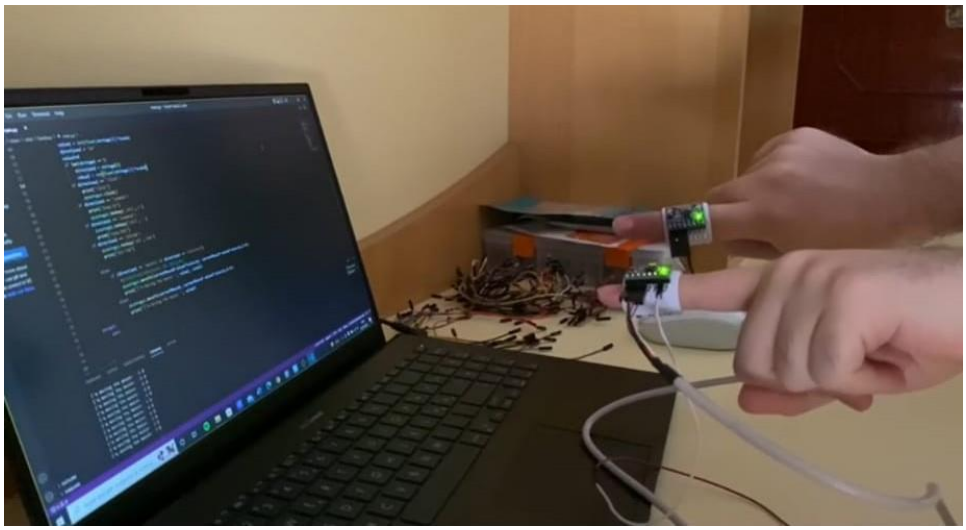
Tali due moduli di supporto sono stati necessari, in quanto, il modulo pySerial avendo accesso alla porta seriale ci ha consentito di realizzare la comunicazione con la scheda STM32 in modo da poter acquisire i comandi di movimento attraverso il protocollo di comunicazione UART. L'acquisizione di tali dati ne ha poi permesso l'elaborazione e la traduzione in movimenti del mouse o in esecuzione di comandi quali il click, zoom-in, zoom-out, alt-tab, etc.., attraverso l'impiego di PyAutoGui.

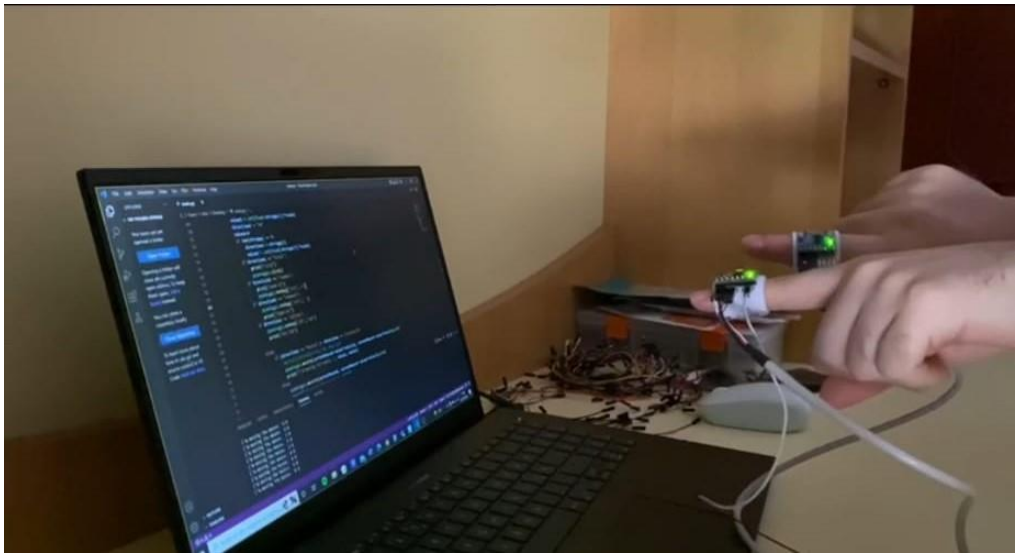
8 Foto







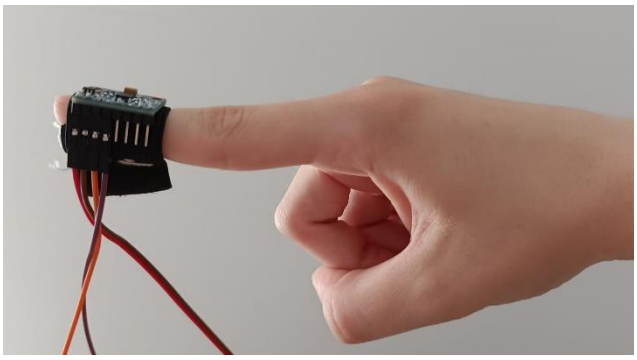




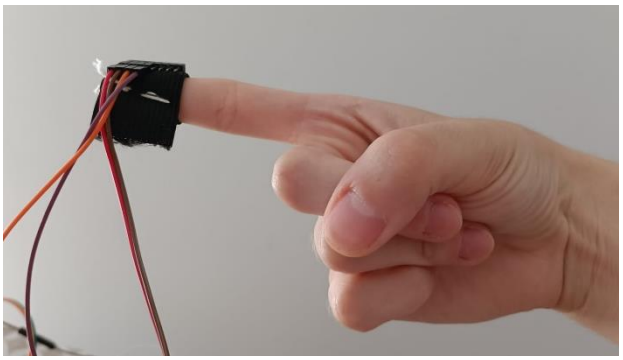
9 Istruzioni di utilizzo

La configurazione realizzata prevede l'impiego di due accelerometri, ciascuno dei due posizionato sul dito indice della mano. Sulla mano destra deve essere posizionato l'accelerometro il cui pin AD0 non è connesso all'alimentazione di 5Volt, e sarà l'accelerometro deputato allo spostamento del mouse (movimenti verso destra, sinistra, sopra, sotto o eventuali movimenti in obliquo), sulla mano sinistra invece dovrà essere fissato l'accelerometro con AD0 alimentato a 5Volt, che è deputato ad effettuare le gesture (click, zoom-in, zoom-out, alt-tab).
Si riportano di seguito delle immagini esplicative per l'esecuzione dei comandi.

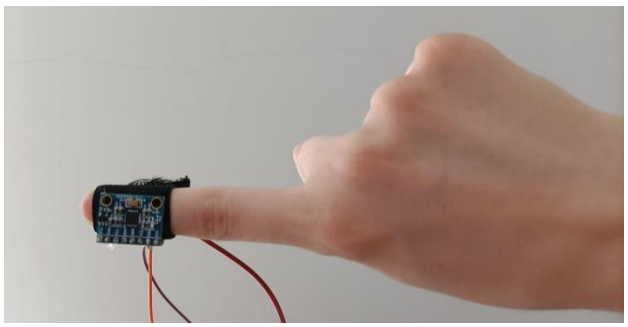
9.1 Mano Destra



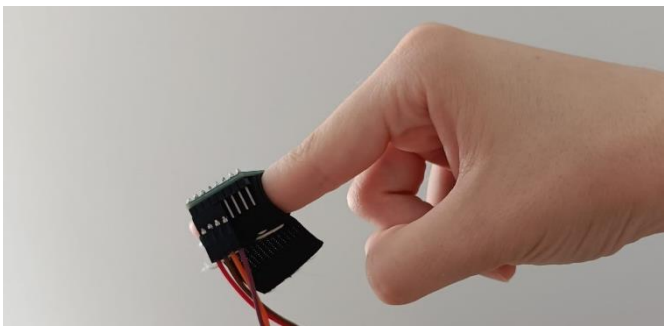
Nessun Movimento



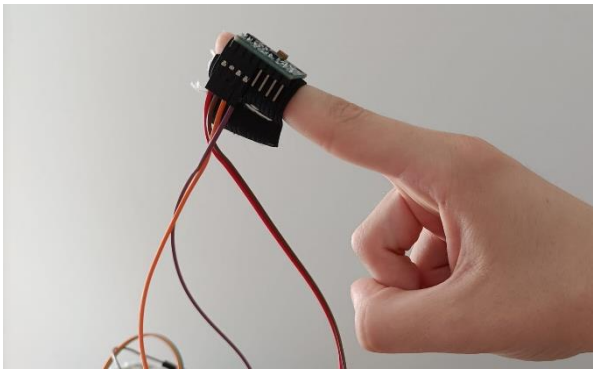
Movimento verso destra



Movimento verso sinistra



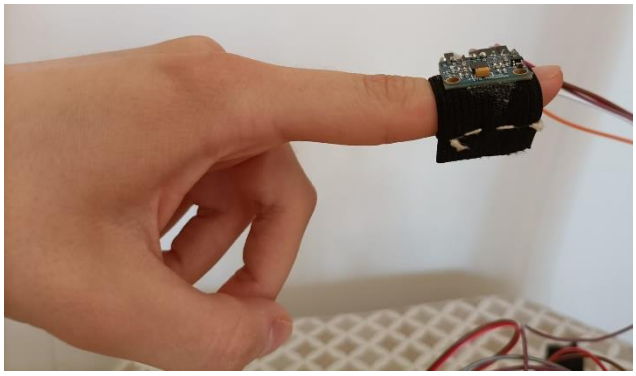
Movimento verso il basso



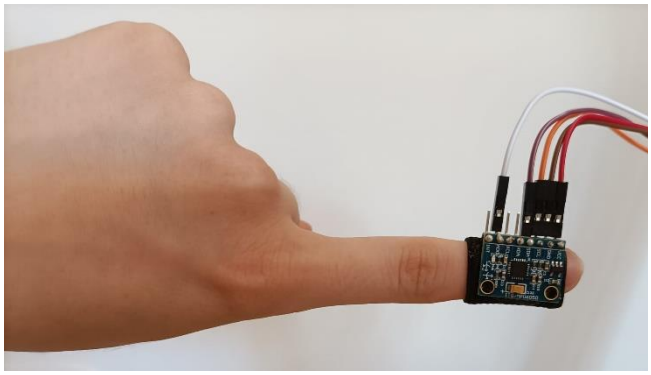
Movimento verso l'alto

I movimenti obliqui (destra-sotto, destra-sopra, sinistra-sotto, sinistra-sopra) possono essere effettuati inclinando il dito appositamente rispetto alle orientazioni sopra illustrate.

9.2 Mano Sinistra



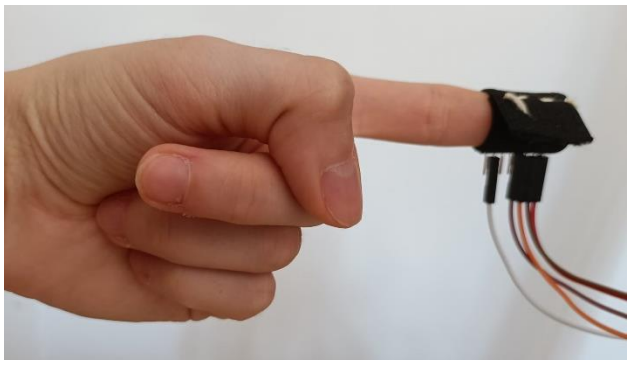
Nessun Movimento



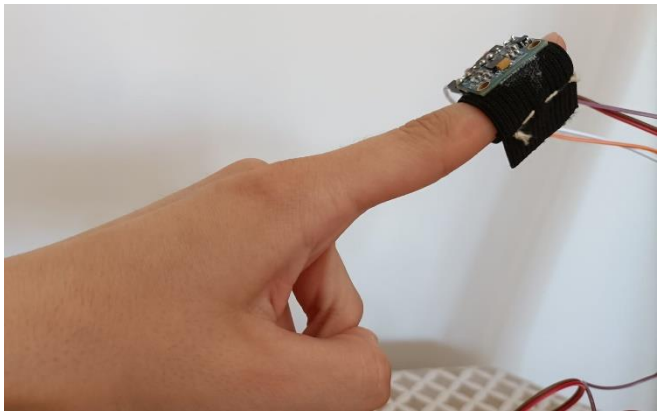
Zoom-out

Click

(Il doppio click può essere effettuato reiterando il movimenento di seguito)



Zoom-in



Alt-tab