UNIVERSITÀ
POLITECNICA
DELLE MARCHE

UNIVERSITÀ POLITECNICA DELLE MARCHE

FACOLTÀ DI INGEGNERIA
Corso di Laurea Triennale in Ingegneria Elettronica

Tesi di Laurea

Sviluppo di firmware per l'acquisizione di dati da sensori inerziali basati su piattaforma SensorTile STLCS02V1

Firmware development for data acquisition from inertial sensors based on the SensorTile STLCS02V1 platform

Candidato:
Matteo Orlandini

Relatore:
Prof. Giorgio Biagetti

Correlatore:
Prof. Paolo Crippa

Anno Accademico 2018-2019

Indice

Introduzione

Il progetto nasce con lo scopo di sviluppare un firmware in linguaggio C per acquisire dati da dei sensori di movimento presenti sulla piattaforma SensorTile STLCS02V1. A tal fine vengono configurati il microcontrollore e i sensori presenti sulla scheda per trasmettere tramite Bluetooth i dati.
La scelta della tecnologia Bluetooth deriva dal fatto che il lavoro effettuato ha come scopo il controllo del movimento di pazienti in strutture ospedaliere tramite dispositivi indossabili. È dunque necessaria una tecnologia non invasiva e a basso consumo energetico per garantire il funzionamento del dispositivo per lunghi periodi di tempo.
L'obiettivo del progetto è la trasmissione di pacchetti Bluetooth contenenti dati provenienti dai sensori inerziali. Le misurazioni ottenute dai vari componenti sono stati fuse per ottimizzare il consumo energetico e le prestazioni. Per ricevere i pacchetti Bluetooth è stato scritto, usando la libreria bluepy, un programma in Python che permette di salvare i valori rilevati. In particolare vengono calcolati, facendo data processing a bordo della scheda, e trasmessi gli angoli di beccheggio e rollio ricavati dal giroscopio e dall'accelerometro.
I sensori usati usano tecnologia MEMS, permettono dunque un uso non invasivo e un consumo energetico estremamente ridotto. Queste due prerogative sono molto importanti nel progetto sviluppato perché soddisfano i requisiti fondamentali.
Durante il progetto è stato studiato, modificato e ampliato il firmware open source reso disponibile da STMicroelectronics e compatibile con il SensorTile. Per poter sfruttare il firmware in modo corretto sono stati consultati i vari manuali [3], [10] e [11] dell'azienda produttrice. Anche il protocollo Bluetooth e l'implementazione di algoritmi di sensor fusion sono stati oggetto di studio.
L'elaborato propone un'introduzione alla tecnologia Bluetooth e Bluetooth Low Energy, una descrizione della piattaforma SensorTile e dei sensori inerziali usati nel progetto, una esposizione del programma in Python, una descrizione del firmware studiato ed implementato, un'analisi delle prestazioni del codice sviluppato in termini di consumi energetici e un confronto della precisione tra i dati grezzi dei sensori e i dati ricavati usando un algoritmo di sensor fusion.

Capitolo 1

Introduzione al Bluetooth

1.1 Bluetooth e Bluetooth Low Energy

La tecnologia Bluetooth è un sistema di comunicazione senza fili a corto raggio inteso a rimpiazzare i dispositivi elettronici con connessione via cavo. Le caratteristiche principali del Bluetooth sono la robustezza, il basso consumo e il basso costo.

Ci sono due forme di sistemi Bluetooth: Basic Rate (BR) e Low Energy (LE). Hanno in comune la ricerca del dispositivo, la costruzione e il meccanismo di connessione. Il sistema Basic Rate ha una estensione chiamata Enhanced Data Rate (EDR) che permette di aumentare il bit rate del sistema BR. Il Bluetooth LE include caratteristiche per lavorare con prodotti che richiedono un consumo di corrente, una complessità e un costo minori rispetto il BR. É implementato per applicazioni con minor data rate.

I dispositivi che implementano entrambi i sistemi possono comunicare con altri apparati che supportano sia il Bluetooth BR sia LE. In alcuni casi è supportato solo uno dei due sistemi, quindi i dispositivi che implementano entrambi possono supportare la maggior parte delle situazioni di lavoro.

Il sistema centrale Bluetooth consiste di un host e di uno o più controller. Osservando la figura 1.1 si possono definire l'host come un'entità logica che contiene tutti i layer sopra la Host Controller Interface (HCI) e il controller come un'entità logica che si compone di tutti i livelli al di sotto di HCI.

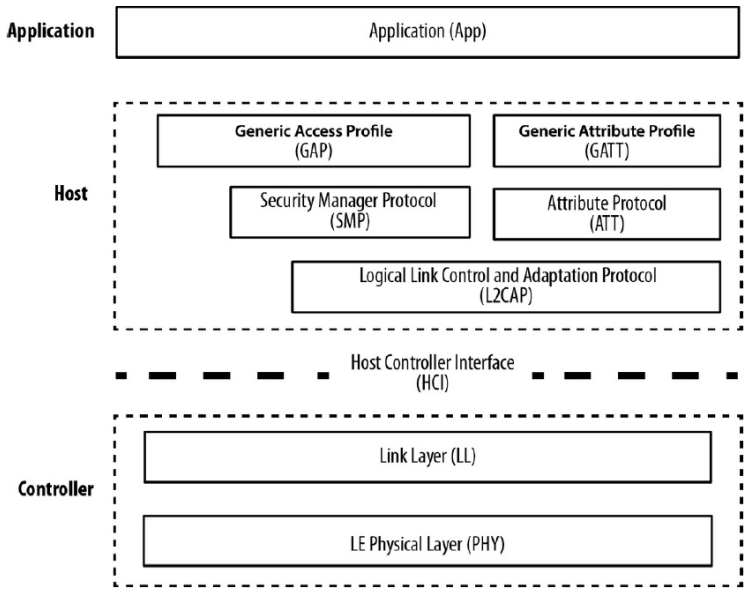


Figura 1.1: Stack protocollare Bluetooth Low Energy [14]

Nella configurazione Basic Rate, i layer inclusi nel livello host sono:

- Logical Link Control and Adaptation Protocol (L2CAP);
- Service Discovery Protocol (SDP)
- Attribute Protocol (ATT)
- Generic Attribute Profile (GATT)
- Generic Access Profile (GAP)

I layer del livello controller sono:

- Radio Frequency (RF)
- Base Band (BB)
- Link Manager Protocol (LMP)

Per il Bluetooth Low Energy i layer del livello host sono gli stessi del Basic Rate tranne per il layer Security Manager Protocol (SMP) che rimpiazza il Service Discovery Protocol (SDP). I layer del livello controller sono:

- Physical Layer (PHY)
- Link Layer (LL)

1.2 Panoramica delle operazioni Basic Rate e Enhanced Data Rate

La trasmissione radio Basic Rate / Enhanced Data Rate (BR/EDR) opera nella banda a 2.4 GHz. Il sistema impiega un ricetrasmettitore in modalità Frequency Hopping Spread Spectrum (FHSS) per ridurre l'interferenza e il fading. Questa tecnica consiste nel variare la frequenza di trasmissione a intervalli regolari in maniera pseudocasuale attraverso un codice prestabilito. La radio Basic Rate opera con modulazione in frequenza binaria per minimizzare la complessità del ricetrasmettitore. Il bit rate supportato dal Basic Rate è di 1 Megabit al secondo (1 Mb/s), mentre con l'Enhanced Data Rate si arriva fino a 3 Mb/s.

Durante le tipiche operazioni il canale radio è condiviso da un gruppo di dispositivi che sono sincronizzati con un clock comune e in modalità frequency hopping. Il dispositivo che fornisce il clock è chiamato *master*, tutti gli altri sincronizzati al clock del master sono gli *slaves*. Un gruppo di apparati sincronizzati formano una piconet.

Gli elementi di una piconet usano uno specifico modello per effettuare il frequency hopping determinato con un algoritmo insieme al master sfruttando, in modo pseudo-random, 79 frequenze separate da 1 MHz nella banda 2.4 GHz. Possono essere escluse alcune frequenze che sono usate da dispositivi interferenti. La tecnica frequency hopping permette al Bluetooth di coesistere con sistemi vicini anche se lavorano nella stessa banda.

Il canale fisico è suddiviso in unità temporali chiamate slot. Il dato è trasmesso tra due dispositivi Bluetooth in pacchetti posizionati in questi slot. Quando le circostanze lo permettono, è possibile assegnare un numero di slot consecutivi a un singolo pacchetto. Il cambiamento della frequenza ha luogo tra la trasmissione o la ricezione dei pacchetti. La tecnologia Bluetooth fornisce una trasmissione full duplex attraverso l'uso di uno schema Time-Division Duplex (TDD).

Sopra il canale fisico ci sono livelli di collegamento e canali con i relativi protocolli di controllo. La gerarchia, ordinata dal basso verso l'alto, è canale fisico, collegamento fisico, trasporto logico, collegamento logico e canale L2CAP.

Nel canale fisico, si forma un collegamento fisico che fornisce il trasporto di pacchetti bidirezionali tra i dispositivi master e slave. Dato che il canale fisico può includere più slaves, ci sono restrizioni su quali dispositivi possano formare un collegamento fisico. Questo si forma tra ogni slave e il master, ma non tra gli slaves di una piconet.

Il collegamento fisico viene utilizzato come trasporto per uno o più collegamenti logici, il cui traffico viene multiplexato sul collegamento fisico occupando gli slot assegnati da una funzione di pianificazione nel Resource Manager.

Il Link Manager Protocol (LMP) è un protocollo di controllo per la banda base e i layer fisici che viene trasferito su collegamenti logici insieme ai dati dell'utente. I dispositivi attivi in una piconet hanno un trasporto logico che è utilizzato per portare le segnalazioni del protocollo LMP.

La funzione Link Manager utilizza LMP per controllare il funzionamento dei dispositivi nella piconet e fornire servizi per gestire i livelli inferiori (layer radio e layer baseband).

Sopra il livello della banda base, il layer L2CAP fornisce un'astrazione basata sul canale per applicazioni e servizi. Effettua la segmentazione e il riassemblaggio di dati applicativi e il multiplexing e de-multiplexing di più canali su un collegamento logico condiviso.

1.3 Panoramica delle operazioni Bluetooth Low Energy

Come per la radio BR/EDR, anche la radio LE opera nella banda 2.4 GHz con una ricetrasmissione frequency hopping, supportando il bit rate di 1 Megabit al secondo (1 Mb/s).

Il Bluetooth LE usa due tipi di accesso multiplo: Frequency Division Multiple Access (FDMA) e Time Division Multiple Access (TDMA). Nello schema FDMA sono usati 40 canali fisici separati da 2 MHz, 3 canali sono usati per advertising e 37 sono usati per i dati. La TDMA è sfruttata quando un dispositivo trasmette ad un predeterminato intervallo temporale e il corrispondente apparato risponde dopo un periodo prestabilito.

Nella figura 1.2 viene presentata la suddivisione dei canali nel Bluetooth Low Energy.

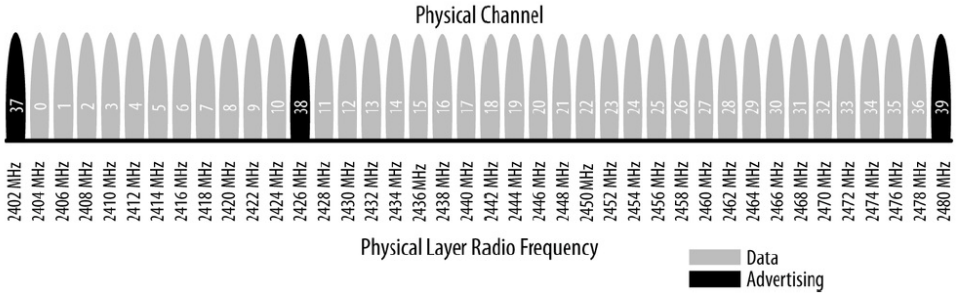


Figura 1.2: Suddivisione dei canali del Bluetooth LE [14]

Il canale fisico è suddiviso in unità temporali chiamate eventi. I dati sono trasmessi in pacchetti posizionati in questi eventi che si dividono in advertising e connection.

I dispositivi che trasmettono i pacchetti di advertising sono chiamati *advertisers*, quelli che il ricevono senza intenzioni di connettersi al dispositivo di advertising sono chiamati *scanners*. Ciascun dispositivo Bluetooth è identificato da un Bluetooth device address, cioè un numero di 48 bit (6 byte). Può essere di due tipi: pubblico, se è un indirizzo fisso, o random, se cambia ad ogni avvio dell'applicazione. Ogni dispositivo advertiser trasmette, ad intervalli regolari che vanno dai 20 ms ai 10.24 secondi pacchetti di grandezza fino a 31 byte contenenti: il nome, l'indirizzo, la classe del dispositivo, la lista dei servizi offerti

e altre informazioni (es. marca). Questa operazione è detta advertising. In base al tipo di pacchetto, lo scanner può fare una richiesta all'advertiser, seguita eventualmente da una risposta da parte di quest'ultimo. Il canale di advertising cambia al successivo pacchetto trasmesso nello stesso evento. L'advertiser può terminare un evento in qualsiasi momento, all'inizio del successivo evento di advertising viene usato il primo canale fisico come mostrato in figura 1.3

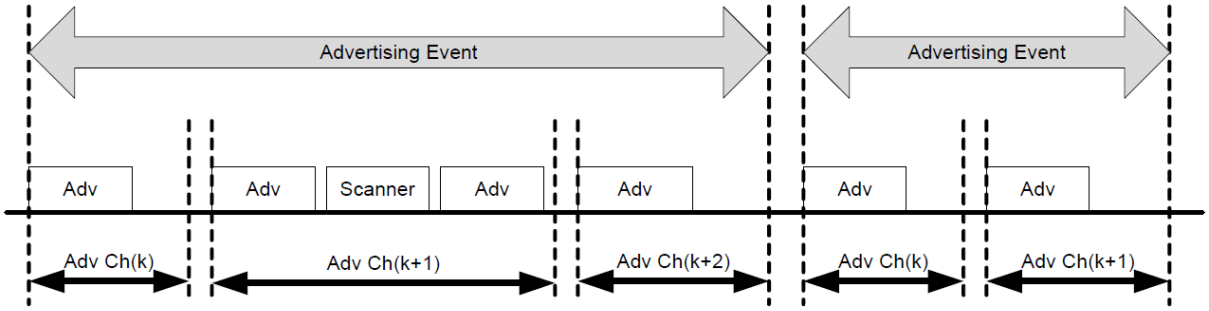


Figura 1.3: Eventi advertising [1]

I dispositivi che necessitano di connettersi con un altro dispositivo vengono chiamati *iniziatori*. Se l'advertiser sta usando un evento di advertising, un iniziatore può effettuare una richiesta di connessione usando lo stesso canale in cui è stato ricevuto il pacchetto. L'evento è così terminato e la connessione inizia se l'advertiser riceve e accetta la richiesta mandata. Una volta che la connessione è stabilita, l'iniziatore diventa il master e l'advertiser è lo slave. Gli eventi connection sono usati per mandare pacchetti di dati tra master e slave nello stesso canale. Il master inizia e finisce ogni evento di connessione.

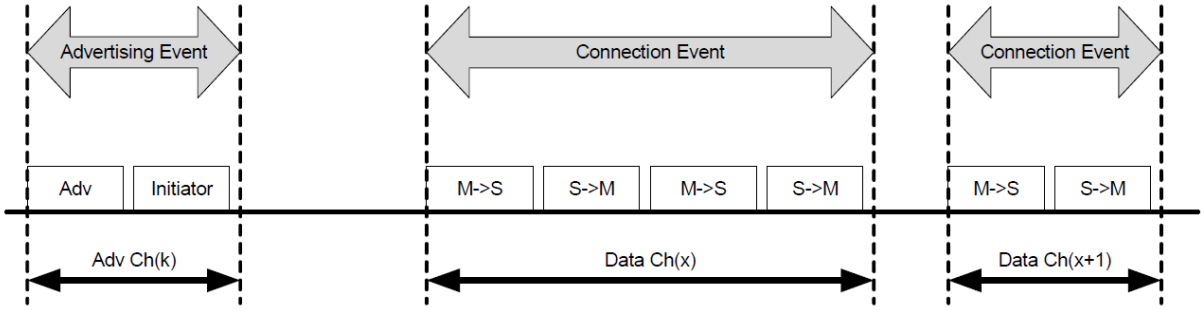


Figura 1.4: Eventi connection [1]

I dispositivi in una piconet usano la tecnica FHSS (Frequency Hopping Spread-Spectrum) che consiste nel commutare il canale di trasmissione con una frequenza di 1600 volte al secondo. L'algoritmo usato per cambiare le frequenze nel Bluetooth LE, condiviso tra il trasmettitore dati ed il ricevitore, è pseudo random nello scegliere le 37 frequenze disponibili, alcune delle quali possono essere escluse per evitare disturbi da altri dispositivi nella stessa area, permettendo inoltre di ridurre al minimo la probabilità di interferenza.

Nella richiesta di connessione, il master invia dei dati chiamati *connection parameters*: essi sono un set di parametri che determinano quando e come il central (master) ed il peripheral (slave) scambieranno i dati durante la connessione. Questi parametri vengono sempre stabiliti dal central, ma la periferica può inviare una *connection parameter update request*, ovvero un pacchetto di dati contenente dei "suggerimenti" di possibili parametri da utilizzare. Questi parametri sono:

- *Connection Interval*: determina quanto spesso il central chiederà il dato alla periferica. Lo standard, infatti, stabilisce un intervallo di possibili valori compreso tra 7,5 ms e 4 s. La periferica suggerisce un ulteriore intervallo, intrinseco al connection interval, compreso tra due valori in ms (i cui estremi sono stabiliti dai parametri MIN_CONN_INTERVAL e MAX_CONN_INTERVAL), ma è il central ad adottarne uno tra i possibili.

- *Slave latency*: è il numero di richieste di lettura dei dati consecutive inviate dal central alla periferica che verranno ignorate. Il suo valore è uno dei possibili compresi nel range tra 0x0000 (cioè ad ogni richiesta di lettura la periferica risponde con l'invio dei dati) e 0x01F3 (499, cioè alla 500esima richiesta di lettura la periferica risponderà con l'invio dei dati, trascurando tutte le precedenti 499). Ciò consente alla periferica di rimanere in modalità sleep per un tempo lungo se non ha dei dati aggiornati da inviare, permettendo così di risparmiare energia.
- *Connection supervision timeout multiplier*: poiché i devices coinvolti nella connessione non sono in grado di capire quando questa viene persa, è necessario che passi un tempo "abbastanza" lungo, chiamato timeout, affinché avvenga il trasferimento dei dati tra i due dispositivi prima di presupporre che la connessione sia stata persa. Ha un range compreso tra 100 ms e 32 secondi (3200 ms). È chiaro inoltre che il timeout dovrà essere maggiore del connection interval.

Sopra il canale fisico sono presenti le connessioni, i canali e i relativi protocolli di controllo. La gerarchia, ordinata dal basso verso l'alto, è canale fisico, collegamento fisico, trasporto logico, collegamento logico e canale L2CAP.

All'interno del canale fisico, si forma un collegamento fisico tra master e ogni slave. Non sono supportati collegamenti fisici tra gli slaves in una piconet. A differenza del Bluetooth BR, agli slaves non è permesso avere collegamenti fisici con più di un master alla volta. Inoltre non è permesso lo scambio di ruoli tra master e slave nella comunicazione.

Il collegamento fisico è usato per trasportare uno o più collegamenti logici che supportano il traffico asincrono. Il traffico sui collegamenti logici viene multiplexato sul collegamento fisico assegnato da una funzione di pianificazione nel Resource Manager.

Il Link Layer Protocol (LL) è un protocollo di controllo per il layer fisico e quello di collegamento, è trasportato attraverso i collegamenti logici insieme ai dati. I dispositivi attivi in una piconet hanno un trasporto logico di connessione asincrono (LE ACL), usato per trasportare le segnalazioni del protocollo LL.

La funzione Link Layer usa il protocollo LL per controllare le operazioni dei dispositivi in una piconet e fornisce servizi per gestire i layer architetturali più bassi (PHY e LL).

Come nel caso BR/EDR, sopra il link layer, L2CAP fornisce un canale di astrazione per applicazioni e servizi. Effettua la frammentazione e de-frammentazione dei dati delle applicazioni e multiplexing e de-multiplexing di più canali su un collegamento logico condiviso. L2CAP ha un canale di controllo del protocollo che è trasmesso nel trasporto logico.

Oltre a L2CAP, LE fornisce due livelli di protocollo aggiuntivi che si trovano sopra di esso. Il Security Manager (SMP) utilizza un canale L2CAP fisso per implementare le funzioni di sicurezza tra i dispositivi e il protocollo Attribute (ATT) fornisce un metodo per comunicare piccole quantità di dati su un canale L2CAP fisso. Il protocollo Attribute viene anche utilizzato dai dispositivi per determinare i servizi e le capacità di altri apparati, può essere utilizzato anche su BR / EDR.

1.4 Architettura host

1.4.1 Channel Manager

Il Channel Manager è responsabile di creare, gestire e chiudere canali L2CAP per il trasporto dei protocolli di servizio e per lo stream di dati. Usa il protocollo L2CAP per operare con il channel manager del dispositivo accoppiato per connettere gli endpoints. Interagisce con il proprio link manager per creare nuovi collegamenti logici, se necessario, e per configurarli per fornire la quality of service (QoS) richiesta in relazione al tipo di dato trasportato.

1.4.2 L2CAP Resource Manager

Il blocco L2CAP Resource Manager è responsabile della gestione dell'ordinamento dei frammenti dei pacchetti in banda base e della pianificazione dei canali L2CAP per garantire che questi non abbiano l'accesso negato al canale fisico a causa dell'esaurimento delle risorse del controller. Ciò è necessario perché l'architettura non presuppone che un controller abbia un buffering illimitato o che l'HCI abbia a disposizione una larghezza di banda infinita.

I gestori delle risorse L2CAP possono anche svolgere attività di controllo della conformità del traffico, assicurandosi che le domande presentino dati entro i limiti delle impostazioni negoziate.

1.4.3 Security Manager Protocol

Il protocollo Security Manager (SMP) è utilizzato per generare chiavi di crittografia e di identità. Il protocollo opera su un canale L2CAP dedicato. Il blocco SMP gestisce anche l'archiviazione delle chiavi di crittografia e di identità ed è responsabile della generazione e del riconoscimento di indirizzi casuali di dispositivi noti. Si interfaccia direttamente con il controller per fornire chiavi memorizzate durante le procedure di crittografia o associazione.

Questo blocco viene utilizzato solo nei sistemi LE. Funzionalità simili nel sistema BR / EDR sono contenute nel blocco Link Manager del controller. La funzionalità SMP è presente negli host su sistemi LE per ridurre i costi di implementazione dei soli controller LE.

1.4.4 Attribute Protocol

Il protocollo ATT definisce due ruoli: *server* e *client*. Il server è il dispositivo che accetta comandi e richieste in arrivo dal client e a cui invia risposte, indicazioni e notifiche. Il client, invece, è il dispositivo che avvia comandi e richieste verso il server e può ricevere risposte, indicazioni e notifiche inviate dal server. Il protocollo ATT consente a un server di esporre un set di attributi a un client che sono accessibili usando questo protocollo.

Un attributo è composto da quattro parti: *handle*, *tipo*, *valore* e *permessi*. La figura 1.5 mostra una rappresentazione logica di un attributo.

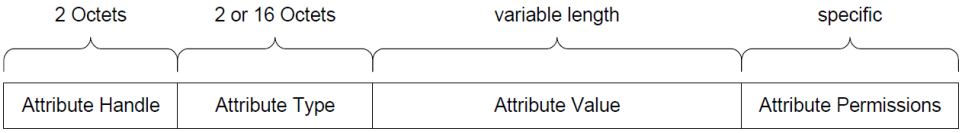


Figura 1.5: Rappresentazione logica di un attributo [1]

L'*handle* è un indice corrispondente a un attributo specifico. Il *tipo* dell'attributo è un UUID (universally unique identifier) che specifica ciò che esso rappresenta. Il *valore* è il dato descritto dal tipo dell'attributo e indicizzato dall'handle. Gli attributi vengono ordinati in modo crescente secondo l'handle. I valori di handle possono iniziare con qualsiasi valore compreso tra 0x0001 e 0xFFFF. Sebbene siano ordinati in modo crescente, potrebbero esserci delle lacune tra un handle e il suo successivo. Le *autorizzazioni* vengono utilizzate dal server per determinare se è consentito l'accesso in lettura o in scrittura per un determinato attributo e sono stabilite dal profilo GATT.

1.4.5 AMP Manager Protocol

AMP Manager è un livello che utilizza L2CAP per comunicare con un altro AMP Manager su un dispositivo remoto. È responsabile della scoperta degli AMP remoti e della loro disponibilità. Queste informazioni vengono utilizzate per impostare e gestire i collegamenti fisici AMP. Utilizza un canale di segnalazione L2CAP dedicato per comunicare con gli AMP manager remoti.

1.4.6 Generic Attribute Profile

Il blocco GATT (Generic Attribute Profile) si serve del protocollo ATT per trasportare i dati sotto forma di comandi, richieste, risposte, indicazioni, notifiche e conferme tra dispositivi. Quando si richiedono i dati sotto forma di notifiche, il server può inviare il valore di un attributo in qualsiasi momento. Il profilo GATT definisce un framework che specifica le procedure e i formati dei servizi e le loro caratteristiche utilizzando il protocollo ATT. I comandi e le richieste agiscono sui valori memorizzati negli attributi sul server. La figura 1.6 mostra la gerarchia del profilo GATT.

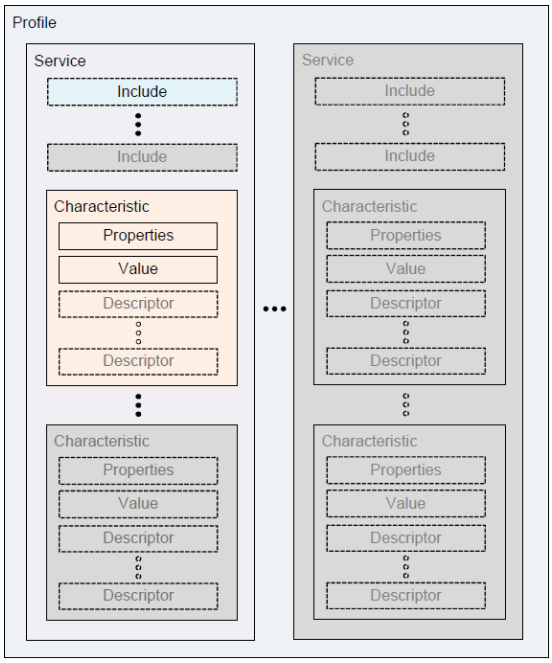


Figura 1.6: Gerarchia del profilo GATT [1]

Un *servizio* è una raccolta di dati e comportamenti associati per realizzare una particolare funzione o caratteristica. Nel GATT, una definizione di servizio può contenere servizi di riferimento, caratteristiche obbligatorie e caratteristiche opzionali.

Esistono due tipi di servizi: primari e secondari. Un servizio primario espone la funzionalità principale utilizzabile di questo dispositivo. Può essere incluso da un altro servizio. Un servizio secondario è un servizio a cui si intende fare riferimento solo da un servizio primario, un altro servizio secondario o altra specifica di livello superiore. L'inclusione è un metodo per riferire la definizione di un servizio esistente ad uno che deve essere definito. In questo caso, l'intera definizione del servizio incluso, che continua ad esistere come fosse indipendente, diventa parte della nuova definizione del servizio. Non ci sono limiti per il numero di definizioni di inclusioni.

Una dichiarazione di servizio è un attributo con il tipo impostato sull'UUID per ≪Servizio primario≫ (0x2800) o ≪Servizio secondario≫ (0x2801). Il campo valore può essere un UUID a 16 o a 128 bit, nel primo caso è un UUID Bluetooth, mentre nel secondo è l'UUID del servizio. Le autorizzazioni di attributo devono essere di sola lettura e non richiedono autenticazione o autorizzazione.

Attribute Handle	Attribute Type	Attribute Value	Attribute Permission
0xNNNN	0x2800 – UUID for «Primary Service» OR 0x2801 for «Secondary Service»]	16-bit Bluetooth UUID or 128-bit UUID for Service	Read Only, No Authentication, No Authorization

Figura 1.7: Dichiarazione di un servizio [1]

Una caratteristica è un attributo utilizzato in un servizio con proprietà e informazioni su come viene visualizzato o rappresentato il valore e su come accedervi. Nel GATT, una caratteristica è definita dalla sua definizione. Questa contiene una dichiarazione, proprietà, un valore e può contenere descrittori che descrivono il valore o consentono la configurazione del server rispetto alla caratteristica.

Una dichiarazione di una caratteristica è un attributo con il tipo impostato sull'UUID ≪Caratteristica≫ (0x2803) e il valore dell'attributo diviso in proprietà, handle e UUID. Le autorizzazioni devono essere leggibili e non richiedono autenticazione o autorizzazione.

Attribute Handle	Attribute Type s	Attribute Value			Attribute Permissions
0xNNNN	0x2803–UUID for «Characteristic»	Character-istic Proper-ties	Character-istic Value Attribute Handle	Characteris-tic UUID	Read Only, No Authentication, No Authorization

Figura 1.8: Dichiarazione di una caratteristica [1]

Il campo valore di una caratteristica è disponibile solamente in lettura ed è illustrato nella figura di seguito.

Attribute Value	Size	Description
Characteristic Properties	1 octets	Bit field of characteristic properties
Characteristic Value Handle	2 octets	Handle of the Attribute containing the value of this characteristic
Characteristic UUID	2 or 16 octets	16-bit Bluetooth UUID or 128-bit UUID for Characteristic Value

Figura 1.9: Campo value nella dichiarazione della caratteristica [1]

- Il campo proprietà determina come è possibile utilizzare il valore o come accedere ai descrittori delle caratteristiche.
- Il campo handle contiente l'handle dell'attributo in cui è presente il valore della caratteristica.
- Il campo UUID è un UUID Bluetooth a 16 bit o UUID generico a 128 bit che descrive il tipo di valore della caratteristica.

I *descrittori* delle caratteristiche vengono utilizzati per contenere informazioni correlate al valore della caratteristica. Il profilo GATT definisce un set standard di descrittori che possono essere utilizzati da profili di livello superiore. Ogni descrittore di caratteristiche è identificato dall'UUID.

Il *Client Characteristic Configuration Descriptor* (CCCD) è un descrittore di caratteristiche facoltativo che definisce come la caratteristica possa essere configurata da uno specifico client. Il valore deve essere persistente tra le connessioni per i dispositivi collegati e deve essere impostato sul valore predefinito per ogni connessione con dispositivi non collegati.

Un client può scrivere nel CCCD per controllare la configurazione della caratteristica. Possono essere richieste autenticazione e autorizzazione dal server per scrivere nel descrittore. La dichiarazione del Client Characteristic Configuration deve essere leggibile e scrivibile.

Il descrittore è contenuto in un attributo. Il campo tipo deve essere impostato sull'UUID come ≪Client Characteristic Configuration≫ (0x2902). Il CCCD agisce come un interruttore, abilitando o disabilitando gli aggiornamenti del campo "value" del "characteristic value" della caratteristica in cui si trova. Il suo valore è un bitfield a due bit, uno corrispondente alle notifiche e l'altro alle indicazioni.

Attribute Handle	Attribute Type	Attribute Value	Attribute Permissions
0xNNNN	02902 – UUID for «Client Characteristic Configuration»	Characteristic Configuration Bits	Readable with no authentication or authorization. Writable with authentication and authorization defined by a higher layer specification or is implementation specific.

Figura 1.10: Dichiarazione *Client Characteristic Configuration* [1]

I *Characteristic Configuration Bits* possibili sono elencati in tabella:

Configurazione	Valore	Descrizione
Notifiche	0x0001	Il campo value sarà notificato
Indicazioni	0x0002	Il campo value sarà indicato

Tabella 1.1: Definizione dei *Characteristic Configuration Bits* [1]

1.4.7 Generic Access Profile

Il blocco GAP (Generic Access Profile) rappresenta la funzionalità di base comune a tutti i dispositivi Bluetooth come le modalità e le procedure di accesso utilizzate dal trasporto, dai protocolli e dai profili applicativi. I servizi GAP comprendono il rilevamento dei dispositivi, le modalità di connessione, la sicurezza, l'autenticazione, i modelli di associazione e il rilevamento dei servizi.

1.5 Architettura controller

1.5.1 Device Manager

Il Device Manager è il blocco funzionale in banda base che controlla il comportamento generale del dispositivo Bluetooth. È responsabile di tutte le operazioni del sistema Bluetooth che non sono direttamente correlate al trasporto di dati, come ricercare la presenza di dispositivi Bluetooth nelle vicinanze, connettersi ad altri apparati o rendere il dispositivo locale rilevabile o collegabile.

Richiede l'accesso al supporto di trasporto dal Baseband Resource Controller per svolgere le sue funzioni.

Controlla anche il comportamento del dispositivo locale tramite una serie di comandi HCI, come la gestione del nome locale del dispositivo ed eventuali chiavi di collegamento memorizzate.

1.5.2 Link Manager

Il Link Manager è responsabile della creazione, modifica e rilascio dei collegamenti logici e, se necessario, dei relativi trasporti logici associati, nonché dell'aggiornamento dei parametri relativi ai collegamenti fisici tra i dispositivi. Riesce a svolgere queste funzioni comunicando con il relativo Link Manager nei dispositivi Bluetooth remoti utilizzando il Link Management Protocol (LMP) nel BR / EDR e il protocollo Link Layer (LL) in LE.

Effettua un controllo generale del collegamento e il trasporto di attributi come l'abilitazione della crittografia sul trasporto logico, l'adattamento della potenza di trasmissione in BR / EDR sul collegamento fisico o regolazione delle impostazioni di QoS in BR / EDR per un collegamento logico.

1.5.3 Baseband Resource Manager

Il Baseband Resource Manager è responsabile di tutti gli accessi al supporto radio. Ha due funzioni principali. Al suo centro c'è un pianificatore che gestisce il tempo necessario sui canali fisici a tutte le

entità che hanno negoziato un "contratto di accesso". L'altra funzione principale è quella di negoziare l'accesso con queste entità, ha quindi il compito di fornire un QoS necessario per offrire a un'applicazione utente la prestazione prevista.

Il contratto di accesso e la funzione di pianificazione devono tenere conto di qualsiasi comportamento che richiede l'uso del controller primario. Ciò include, ad esempio, il normale scambio di dati tra dispositivi connessi tramite collegamenti e trasporti logici, nonché l'uso del mezzo radio per effettuare connessioni, rendere il dispositivo rilevabile o collegabile o per eseguire letture da portanti non usate durante il frequency hopping.

In alcuni casi, nei sistemi BR / EDR la pianificazione di un collegamento logico comporta la sua modifica in un canale fisico diverso da quello precedentemente utilizzato. Quando i canali fisici non sono allineati nel time slot, il Baseband Resource Manager tiene conto anche del tempo di riallineamento tra le slot sul canale fisico originale e quelle sul nuovo canale fisico. In alcuni casi gli slot saranno naturalmente allineati a causa dello stesso clock del dispositivo utilizzato come riferimento per entrambi i canali fisici.

1.5.4 Link Controller

Il Link Controller è responsabile della codifica e della decodifica dei pacchetti Bluetooth dal payload dei dati, cioè la parte di dati trasmessi effettiva che è destinata all'utilizzatore, e dei parametri relativi al canale fisico, al trasporto logico e al collegamento logico.

Esegue la segnalazione del protocollo link control in BR / EDR e Link Layer in LE (in stretta collaborazione con la funzione di pianificazione del Resource Manager), che viene utilizzato per comunicare i segnali di controllo del flusso e di riconoscimento e richiesta di ritrasmissione. L'interpretazione di questi segnali è una caratteristica del trasporto logico associato al pacchetto in banda base. Il controllo della segnalazione del link control è normalmente associato al pianificatore del Resource Manager.

1.5.5 PHY

Il blocco PHY è responsabile della trasmissione e della ricezione di pacchetti di informazioni sul canale fisico. Un percorso di controllo tra la banda di base e il blocco PHY consente al blocco di banda di base di controllare il timing e la portante del blocco PHY. Trasforma uno stream di dati da e verso il canale fisico e la banda base nei formati richiesti.

Capitolo 2

I sensori inerziali

2.1 Introduzione ai sensori MEMS

In meno di 20 anni, la tecnologia MEMS (Micro Electro-Mechanical Systems) è passata da un'interessante argomento di ricerca in ambito accademico a parte integrante di molti prodotti di uso comune. Ma come con la maggior parte delle nuove tecnologie, l'implementazione pratica della tecnologia MEMS ha richiesto del tempo. Nei primi sistemi è stato utilizzato un approccio multi-chip con l'elemento di sensing e l'elettronica di condizionamento del segnale su chip differenti. Sebbene questo approccio sia più semplice dal punto di vista del processo, presenta molti svantaggi:

- l'area complessiva del silicio è generalmente più ampia,
- i moduli multi chip richiedono fasi di assemblaggio aggiuntive,
- la resa è generalmente inferiore per i moduli multi chip,
- sono necessari segnali più grandi dal sensore per superare la capacità parassita delle interconnessioni tra i chip e i campi elettromagnetici interferenti richiedono una struttura del sensore più grande,
- per ospitare due chip è necessario un package di dimensioni maggiori.

L'integrazione è la soluzione più economica e ad alte prestazioni, quindi si è perseguito un approccio integrato in cui l'elettronica di condizionamento del sensore e del segnale si trova su un chip. La progettazione meccanica di sistemi meccanici microscopici richiede innanzitutto una comprensione del comportamento meccanico dei vari elementi utilizzati. Mentre le regole di base della dinamica sono ancora valide nel mondo miniaturizzato, molti dei materiali utilizzati in queste strutture non sono ben caratterizzati meccanicamente. Sarà ora analizzato il funzionamento dei vari tipi di sensori MEMS usati nel progetto.
Gli accelerometri fanno parte dei sistemi MEMS, sono costituiti da due condensatori collegati in un half-bridge: un'accelerazione muove la massa sensibile che costituisce una delle armature dei condensatori facendone variare la capacità. Le capacità di questi condensatori è dell'ordine del pF, mentre la massima variazione di capacità è dell'ordine di 10-100 fF. Lo sbilanciamento dell'half-bridge può essere misurato integrando la carica accumulata dai condensatori quando questi sono sottoposti a dei brevi impulsi di tensione.
Il giroscopio è un sensore che misura il moto rotazionale. L'operazione di calcolo della velocità angolare in un giroscopio avviene grazie a piccole masse (si parla di dimensioni microscopiche, tra 1 e 100 micrometri) che si muovono in funzione di cambiamenti nella velocità angolare. Queste minime variazioni vengono convertite in tensioni elettriche, amplificate e elaborate opportunamente.
La maggior parte dei magnetometri MEMS sfrutta la forza di Lorentz per tradurre l'informazione di intensità di campo magnetico in un segnale meccanico, e una lettura capacitiva per trasdurre il segnale meccanico in un segnale elettrico. Il dispositivo è costituito da quattro molle ancorate al substrato ad un'estremità, ed alla massa mobile all'altra estremità, in modo da permettere alla massa mobile il solo movimento traslazionale in una sola direzione. La particolare struttura di questo frame permette l'inserimento, in spazi appositamente progettati, di elettrodi fissi (statori); gli affacciamenti tra questi e il rotore (frame centrale) creano un set di condensatori di capacità variabile, utilizzati per trasdurre il segnale

meccanico (spostamento della massa mobile) in segnale elettrico (variazione di capacità). All'interno delle molle viene iniettata una corrente sinusoidale, e se il dispositivo è immerso in un campo magnetico che presenta delle componenti lungo l'asse Z (perpendicolari al piano della struttura) si genera una forza che agisce in modo ortogonale sia sul campo magnetico sia alla corrente distribuita su entrambe le molle. Questa è la forza di Lorentz ed è definita, per una particella di carica q che si muove con velocità v all'interno di un campo magnetico B, come:

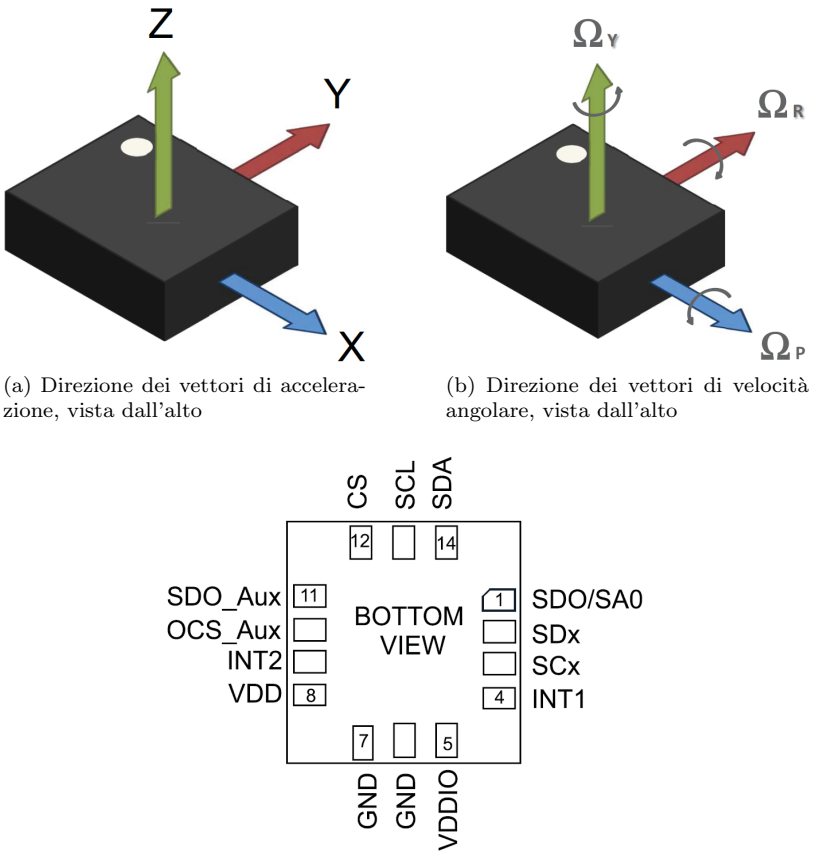
$$\vec{F} = q \cdot \vec{v} \times \vec{B}$$

Come per tutte le nuove tecnologie, sia i progettisti che gli utenti dei dispositivi MEMS hanno una curva di apprendimento da superare. Lo sforzo è utile, poiché i sensori MEMS di ultima generazione ad alte prestazioni e basso costo hanno consentito nuovi prodotti innovativi in moltissimi mercati.

2.2 LSM6DSM

2.2.1 Descrizione

L'LSM6DSM è un sistema integrato che presenta un accelerometro e un giroscopio digitale a 3 assi ad alte prestazioni.
La modalità risparmio energetico è in grado di ridurre i consumi fino a 0,65 mA in modalità ad alte prestazioni, combinando funzionalità a basso consumo sempre attive a una precisione superiore per un'esperienza di rilevamento del movimento ottimale per il consumatore grazie a prestazioni a bassissimo rumore sia per il giroscopio che per l'accelerometro.
LSM6DSM è in grado di rilevare l'orientamento, il movimento e i gesti al fine di potenziare gli sviluppatori e i consumatori di applicazioni con caratteristiche e capacità che sono più sofisticate del semplice orientamento dei loro dispositivi in verticale e orizzontale.
Gli eventi di interrupt consentono il rilevamento del movimento in modo efficiente e affidabile, implementando il riconoscimento hardware di eventi di caduta libera, orientamento 6D, rilevamento di clic e doppio clic, attività o inattività ed eventi di riattivazione.
LSM6DSM è stato progettato per implementare funzionalità hardware come il riconoscimento di movimenti significativi, l'inclinazione, il contapassi, il timestamp e per supportare l'acquisizione dei dati di un magnetometro esterno con correzione hironing (hard e soft). Offre flessibilità hardware per collegare i pin a sensori esterni per espandere le proprie funzionalità.
Ha una memoria FIFO di 4 kbyte con allocazione dinamica di dati significativi (ad esempio sensori esterni e timestamp) che consentono un risparmio energetico complessivo del sistema.
Come l'intero set di sensori MEMS, LSM6DSM sfrutta i robusti e maturi processi di produzione interni già utilizzati per la realizzazione di accelerometri e giroscopi micro elettro-meccanici. I vari elementi di rilevamento sono prodotti utilizzando processi specializzati, mentre le interfacce IC sono sviluppate con tecnologia CMOS che consentono la progettazione di un circuito dedicato che viene tagliato per adattarsi meglio alle caratteristiche del sensore.
LSM6DSM è disponibile in un package LGA (land grid array) 2,5 x 3,0 x 0,83 mm per soluzioni ultracompatte.



(a) Direzione dei vettori di accelerazione, vista dall'alto

(b) Direzione dei vettori di velocità angolare, vista dall'alto

(c) Descrizione dei pin, vista dal basso

Figura 2.1: Connessione e descrizione dei pin di LSM6DSM [8]

2.2.2 Funzionalità

Modalità operative Nel LSM6DSM, l'accelerometro e il giroscopio possono essere accesi o spenti indipendentemente l'uno dall'altro e possono avere un output data rate e modalità di alimentazione diversi. LSM6DSM ha tre modalità operative disponibili:

- solo accelerometro attivo e giroscopio spento
- solo giroscopio attivo e accelerometro spento
- entrambi i sensori accelerometro e giroscopio attivi con ODR indipendente

L'accelerometro viene attivato dalla modalità power-down scrivendo nei bit [7:4] (ODR_XL) nel registro CTRL1_XL (0x10), mentre il giroscopio viene risvegliato scrivendo nei bit [7:4] (ODR_G) in CTRL2_G (0x11).

Modalità di funzionamento del giroscopio Il giroscopio può essere configurato in quattro diverse modalità operative: spegnimento, bassa potenza, modalità normale e modalità ad alte prestazioni. La selezione della modalità operativa dipende dal valore del bit 7 (G_HM_MODE) in CTRL7_G (0x16). Se G_HM_MODE è impostato su "0", la modalità ad alte prestazioni è valida per tutti gli ODR (da 12,5 Hz a 6,66 kHz). Per abilitare la modalità a basso consumo e normale, il bit G_HM_MODE deve essere impostato su "1". La modalità low-power è disponibile per ODR inferiori (12,5, 26, 52 Hz), mentre nella modalità normale per ODR pari a 104 e 208 Hz.

Modalità di funzionamento dell'accelerometro L'accelerometro può essere configurato in quattro diverse modalità operative: spegnimento, bassa potenza, modalità normale e modalità ad alte prestazioni.

La modalità operativa selezionata dipende dal valore del bit 4 (XL_HM_MODE) in CTRL6_C (0x15). Se è impostato su '0', la modalità ad alte prestazioni è valida per tutti gli ODR (da 12,5 Hz a 6,66 kHz). Per abilitare la modalità a basso consumo e normale, il bit XL_HM_MODE deve essere impostato su "1". La modalità a bassa potenza è disponibile per ODR inferiori (1.6, 12.5, 26, 52 Hz), mentre la modalità normale è disponibile per ODR pari a 104 e 208 Hz.

FIFO La presenza della memoria FIFO consente un elevato risparmio energetico per il sistema perché il processore non necessita di dati continui dal sensore, ma può riattivarsi solo quando necessario ed espellere i dati significativi dalla FIFO.
LSM6DSM incorpora i dati nella FIFO da 4 kbyte per memorizzare i seguenti dati:

- giroscopio
- accelerometro
- sensori esterni
- contapassi e timestamp
- temperatura

La scrittura nella FIFO può essere configurata per essere attivata da:

- segnale pronto per accelerometro o giroscopio; in questo caso l'ODR deve essere inferiore o uguale a entrambi gli ODR dell'accelerometro e del giroscopio;
- segnale "data-ready" del sensore hub;
- segnale di rilevamento di uno stemp.

Inoltre, ogni dato può essere memorizzato ad una velocità decimata rispetto all'ODR FIFO ed è configurabile dall'utente, impostando i registri FIFO_CTRL3 (0x08) e FIFO_CTRL4 (0x09). I fattori di decimazione disponibili sono 2, 3, 4, 8, 16, 32.
La soglia programmabile può essere impostata in FIFO_CTRL1 (0x06) e FIFO_CTRL2 (0x07) utilizzando i bit [10: 0] (FTH). Il flag dei bit si setta quando il numero di byte scritti in FIFO dopo il successivo la scrittura successiva è maggiore o uguale al livello di soglia.
Per monitorare lo stato, i registri dedicati FIFO_STATUS1 (0x3A), FIFO_STATUS2 (0x3B), FIFO_STATUS3 (0x3C), FIFO_STATUS4 (0x3D) possono essere letti per rilevare eventi di sovraccarico, memoria piena, vuota, lo stato della soglia e il numero di campioni non letti memorizzati nella memoria. Per generare interrupt dedicati sui pin INT1 e INT2 di questi eventi, si devono configurare i registri INT1_CTRL (0x0D) e INT2_CTRL (0x0E).
Il buffer FIFO può essere configurato secondo cinque diverse modalità:

- Modalità bypass, la FIFO non è operativa e rimane vuota.
- Modalità FIFO, i dati dai canali di uscita vengono memorizzati nella FIFO fino a quando non è pieno.
- Modalità continua, fornisce un aggiornamento continuo della FIFO: quando arrivano nuovi dati, i dati più vecchi vengono eliminati.
- Modalità continua - FIFO, il comportamento FIFO cambia in base all'evento trigger rilevato in uno dei seguenti registri di interruzione FUNC_SRC1 (53h), TAP_SRC (1Ch), WAKE_UP_SRC (1Bh) e D6D_SRC (1DH). Quando il bit di trigger selezionato è uguale a '1', la FIFO funziona in modalità FIFO. Quando il bit di trigger selezionato è uguale a '0', la FIFO funziona in modalità continua.
- Modalità bypass - continua, la memorizzazione della misurazione dei dati all'interno della FIFO funziona in modalità continua quando si innesca la selezione in uno dei seguenti registri di interruzione FUNC_SRC1 (0x53), TAP_SRC (0x1C), WAKE_UP_SRC (0x1B) e D6D_SRC (0x1D) sono uguali a 1, altrimenti il contenuto viene resettato (modalità bypass).

Ogni modalità è selezionata dai bit [2:0] (FIFO_MODE_) nel registro FIFO_CTRL5 (0x0A). Per garantire la corretta acquisizione dei dati durante l'attivazione e la disattivazione della modalità FIFO, il primo campione acquisito deve essere scartato.

2.2.3 Caratteristiche meccaniche

@ Vdd = 1.8 V, T = 25 °C

Parametro	Condizioni	Min.	Tipico	Max.	Unità
Range accelerometro			± 2		g
			± 4		
			± 8		
			± 16		
Range giroscopio			± 125		dps
			± 250		
			± 500		
			± 1000		
			± 2000		
Sensibilità accelerometro	FS =± 2		0.061		mg/LSB
	FS =± 4		0.122		
	FS =± 8		0.244		
	FS =± 16		0.488		
Sensibilità giroscopio	FS =± 125		4.375		mdps/LSB
	FS =± 250		8.75		
	FS =± 500		17.50		
	FS =± 1000		35		
	FS =± 2000		70		
Sensibilità accelerometro vs. temperatura	da -40° a +85°		± 0.01		%/°C
Sensibilità giroscopio vs. temperatura	da -40° a +85°		± 0.007		%/°C
Output data rate accelerometro			1.6[1]		Hz
			26		
			52		
			104		
			208		
			416		
			833		
			1666		
			3332		
			6664		
Output data rate giroscopio			26		
			52		
			104		
			208		
			416		
			833		
			1666		
			3332		
			6664		
Range operativo di temperatura		-40		+85	°C

Tabella 2.1: Caratteristiche meccaniche LSM6DSM [8]

1. Questo ODR è disponibile quando l'accelerometro è in modalità low-power.

2.2.4 Caratteristiche elettriche

@ Vdd = 1.8 V, T = 25 °C

Parametro	Condizioni	Min.	Tipico	Max.	Unità
Tensione di alimentazione		1.71	1.8	3.6	V
Consumo di corrente giroscopio e accelerometro in high-performace	ODR = 1.6 KHz		0.65		mA
Consumo di corrente giroscopio e accelerometro in normal mode	ODR = 208 Hz		0.45		mA
Consumo di corrente giroscopio e accelerometro in low-power	ODR = 52 Hz		0.29		mA
Consumo di corrente accelerometro in high-performance	ODR < 1.6 KHz		150		μA
	ODR $\geq$ 1.6 KHz		160		μA
Consumo di corrente accelerometro in normal mode	ODR = 208 Hz		85		μA
Consumo di corrente accelerometro in low-power mode	ODR = 52 Hz		25		μA
	ODR = 12.5 Hz		9		μA
	ODR = 1.6 Hz		4.5		μA
Consumo di corrente giroscopio e accelerometro in power-down			3		μA
Range operativo di temperatura		-40		+85	°C

Tabella 2.2: Caratteristiche elettriche LSM6DSM [8]

2.3 LSM303AGR

2.3.1 Descrizione

LSM303AGR è un prodotto system-in-package ad altissime prestazioni e bassissima potenza con un sensore digitale di accelerazione lineare e un sensore magnetico.
Il fondo scala per la misura dell'accelerazione è di ± 2g / ± 4g / ± 8g / ± 16g, mentre per il campo magnetico presenta un range di ± 50 gauss.
LSM303AGR include un bus seriale con interfaccia I^2C che supporta le modalità a 100 kHz, 400 kHz, 1 MHz e 3,4 MHz e una interfaccia seriale SPI.
Il sistema può essere configurato per generare un segnale di interrupt per individuare la caduta libera, la rilevazione di movimento e del campo magnetico.
I blocchi magnetici e accelerometrici possono essere abilitato o spenti separatamente.
LSM303AGR è disponibile in un package land grid array (LGA) e garantisce operazioni su un intervallo di temperatura esteso da -40 ° C a +85 ° C.

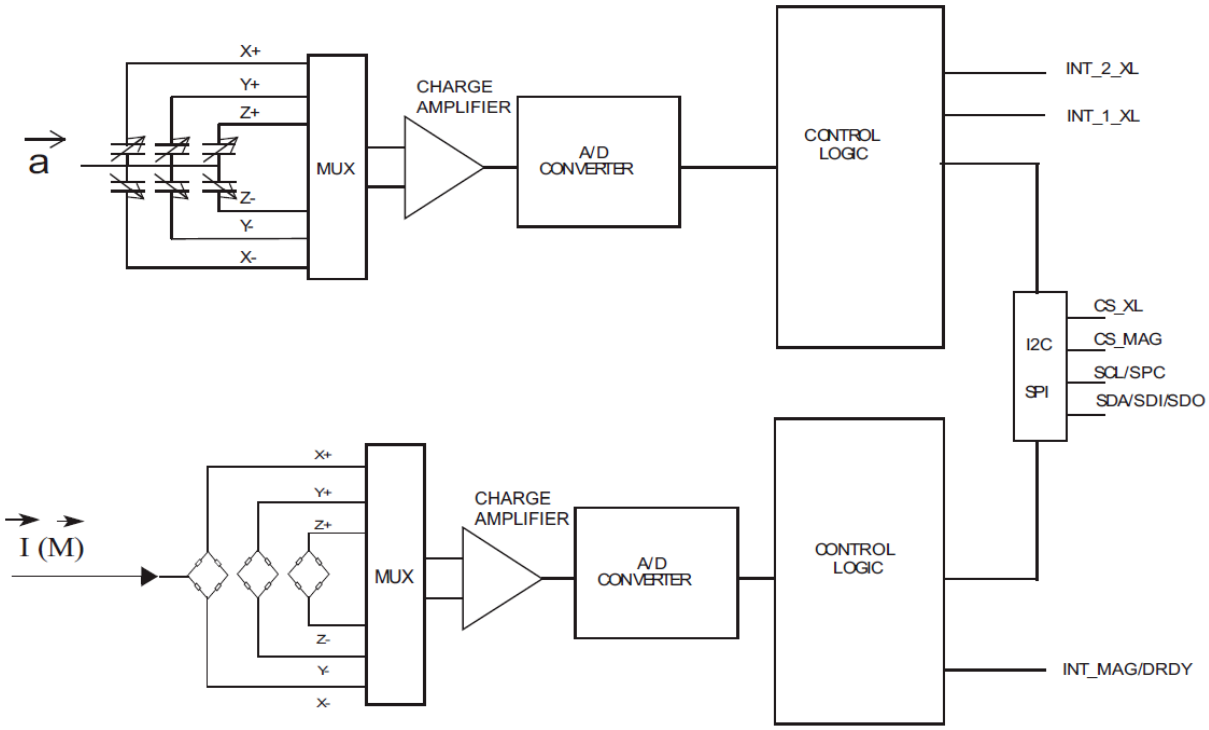
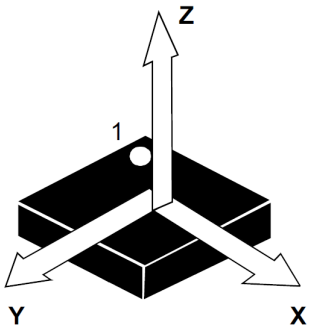
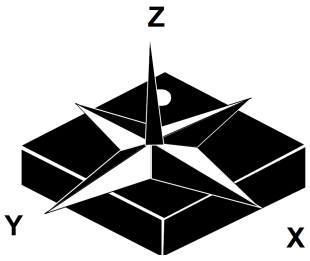


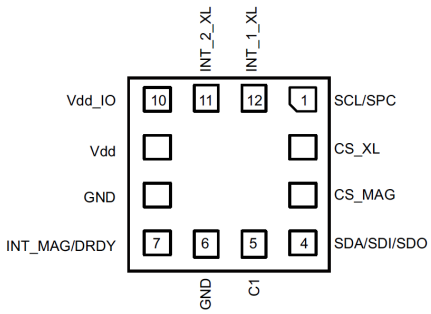
Figura 2.2: Diagramma a blocchi LSM303AGR [7]



(a) Direzione dei vettori di accelerazione, vista dall'alto



(b) Direzione dei vettori di campo magnetico, vista dall'alto



(c) Descrizione dei pin, vista dal basso

Figura 2.3: Connessione e descrizione dei pin LSM303AGR [7]

2.3.2 Funzionalità

Modalità di funzionamento dell'accelerometro L'accelerometro LSM303AGR offre tre diverse modalità operative: alta risoluzione, normale e bassa potenza.
La tabella seguente riepiloga come selezionare le diverse modalità operative.

Modalità	**Bit LPen**[1]	**Bit HR**[2]	**BW [Hz]**
Bassa potenza	1	0	ODR/2
Normale	0	0	ODR/2
Alta risoluzione	0	1	ODR/9
Non permesso	0	0	-

Tabella 2.3: Selezione delle varie modalità operative [7]

1. Il bit 3 (LPen) di CTRL_REG1 (0x20) abilita la modalità low-power se è a 1, viceversa si seleziona la modalità normale.
2. Il bit 3 (HR) di CTRL_REG4 (0x23) seleziona la modalità operativa normale o alta risoluzione.

Modalità operativa [Hz]	**Consumo di corrente [μA]**		
	Bassa potenza output a 8 bit	**Normale output a 10 bit**	**Alta risoluzione output a 12 bit**
1	3.7	3.7	3.7
10	4.4	5.4	5.4
25	5.6	8	8
50	7.7	12.6	12.6
100	11.7	22	22
200	20	40	40
400	36	75	75
1344	-	185	185
1620	102	-	-
5376	186	-	-

Tabella 2.4: Consumo di corrente dell'accelerometro LSM303AGR [7]

La funzione di riconoscimento di attività o inattività consente di ridurre il consumo energetico del solo accelerometro, quando questa funzionalità è attivata, l'accelerometro è in grado di passare automaticamente alla frequenza di campionamento di 10 Hz e si sveglia non appena si verifica un evento di interrupt, aumentando la velocità dell'output data rate e la larghezza di banda.
Con questa funzione, il sistema può essere commutato in modo efficiente dalla modalità di risparmio energetico alle massime prestazioni, o viceversa, in base agli eventi di posizionamento e accelerazione selezionabili dall'utente, quindi garantendo risparmio energetico e flessibilità.
La funzione si attiva scrivendo la soglia desiderata nel registro ACT_THS_A (0x3E), il filtro passa-alto viene abilitato automaticamente.

Registro	**Valore del LSB**
ACT_THS_A	Fonda scala / 128 [mg]
ACT_DUR_A	8/ODR [s]

Tabella 2.5: Registri di controllo [7]

Quando l'accelerazione diventa inferiore alla soglia per almeno la durata $\frac{8 \cdot ACT\_DUR+1}{ODR}$, i bit [7:4] (ODR) di CTRL_REG1_A (0x20) sono bypassati (Inattività) e impostato internamente l'output data rate pari a 10 Hz, ma il contenuto dei bit ODR di CTRL_REG1_A non viene modificato.
Quando l'accelerazione diventa maggiore della soglia, contenuta in ACT_THS_A, CTRL_REG1_A viene ripristinato immediatamente (Attività).
Una volta abilitata la funzione di rilevamento attività / inattività, verrà applicata al pin INT_2 impo-

stando il bit 3 (P2_ACT) di CTRL_REG6_A (0x25) a 1.
Per disabilitare la funzione di rilevamento attività o inattività, occorre impostare il contenuto del registro ACT_THS_A (0x3E) a 0x00.

Modalità di funzionamento del magnetometro Il magnetometro LSM303AGR offre due diverse modalità di funzionamento: alta risoluzione e bassa potenza. Le tabelle 2.6 e 2.7 riassumono i valori di rumore RMS del magnetometro e il consumo di corrente nelle diverse configurazioni.
Quando il filtro passa-basso è abilitato, la larghezza di banda viene ridotta, mentre le prestazioni del rumore vengono migliorate senza alcun aumento del consumo energetico.

Filtro L.P.	**Alta risoluzione**		**Bassa potenza**	
	BW [Hz]	**Rumore RMS [mg]**	**BW [Hz]**	**Rumore RMS [mg]**
disabilitato	ODR/2	4.5	ODR/2	9
abilitato	ODR/4	3	ODR/4	6

Tabella 2.6: Rumore RMS nelle varie modalità operative [7]

ODR[Hz]	**Consumo di corrente (μA)**			
	alta risoluzione OFF_CANC[1]=0	**basso consumo OFF_CANC=0**	**alta risoluzione OFF_CANC=1**	**basso consumo OFF_CANC=1**
10	100	25	120	50
20	200	50	235	100
50	475	125	575	235
100	950	250	1130	460

Tabella 2.7: Consumi nelle varie modalità operative [7]

1. Nel registro CFG_REG_B_M (0x61) il bit 1 (OFF_CANC) disabilita la cancellazione dell'offset se è a zero, viceversa se è settato.
LSM303AGR offre una modalità di misurazione singola sia in alta risoluzione che a bassa potenza.
La modalità di misurazione singola è abilitata scrivendo i bit MD [1: 0] su '01' in CFG_REG_A_M (0x60). In questa modalità, una volta eseguita la misurazione, il pin DRDY va alto, i dati sono disponibili nel registro di uscita e LSM303AGR è automaticamente configurato in modalità inattiva impostando il bit 1 (MD1) a 1.
La misurazione singola è indipendente dall'ODR programmato, dipende dalla frequenza con cui i bit MD [1: 0] vengono scritti dal microcontrollore.
La massima frequenza ODR ottenibile nella misurazione in modalità singola è riportata nella tabella seguente.

ODR massimo	**Modalità**
100 Hz	Alta risoluzione
150 Hz	Bassa potenza

Tabella 2.8: Massimo ODR nella modalità singola misura [7]

Cancellazione dell'offset del magnetometro La cancellazione dell'offset è il risultato dell'esecuzione di un set e del reset nel sensore magnetico. La tecnica di annullamento dell'offset è definita come segue:

$$H_{out} = \frac{H[n] + H[n-1]}{2}$$

dove H[n] e H[n-1] sono due misurazioni consecutive del campo magnetico, una dopo un set e una dopo un reset.
La cancellazione dell'offset è abilitata settando a uno il bit 1 (OFF_CANC)) e il bit 4 (OFF_CANC_ONE_SHOT) in modalità misurazione singola, nel registro CFG_REG_B_M (0x61).

La cancellazione dell'offset viene gestita automaticamente dal dispositivo in modalità continua, mentre deve essere gestita dall'utente in modalità di misurazione singola mediando due misurazioni consecutive H[n] e H[n-1].

Interrupt del magnetometro In LSM303AGR la generazione del segnale di interruzione del magnetometro si basa sul confronto tra dati e una soglia programmabile.
Per abilitare l'interrupt, nel registro INT_CTRL_REG_M (0x63) il bit 0 (IEN) deve essere settato.
L'utente può selezionare l'asse in cui è abilitato l'interrupt impostando correttamente i bit i bit [5:7] (XIEN, YIEN e ZIEN) in INT_CTRL_REG_M (0x63).
Il valore di soglia può essere programmato impostando i registri INT_THS_L_REG_M (0x65) e INT_THS_H_REG_M (0x66). La soglia è espressa in valore assoluto come un numero senza segno a 15 bit. La soglia ha la stessa sensibilità dei dati.
Quando i valori superano la soglia positiva o negativa, viene generato il segnale di interrupt e le informazioni sul tipo di interruzione vengono memorizzate nel registro INT_SOURCE_REG_M (0x64). In particolare, quando i dati superano la soglia positiva, uno tra i bit [7:5] (P_TH_S_axis) è settato a 1, viceversa uno tra i bit [4:2] (N_TH_S_axis) è portato a 1. Se i dati magnetici si trovano tra le soglie positiva e negativa, non viene emesso alcun segnale di interruzione.

Compensazione hard iron La distorsione hard iron si verifica quando un oggetto magnetico viene posizionato vicino al magnetometro e appare come una distorsione permanente nelle uscite del sensore.
La sua correzione consiste nel compensare i dati magnetici dalla distorsione. L'operazione è definita come segue:

$$H_{out} = H_{read} - H_{HI}$$

dove H_{read} sono i dati generici non compensati del campo magnetico, così come sono letti dal sensore, H_{HI} è il campo di distorsione hard iron e H_{out} è il dato compensato.
Il calcolo del campo hard iron dovrebbe essere eseguito da un processore esterno, in modo da compensare i dati magnetici misurati.
LSM303AGR offre la possibilità di archiviare dati dell'hard iron all'interno di sei registri dedicati da 0x45 a 0x4A. Ogni registro contiene otto bit in modo che possano essere espressi come 16 bit in complemento a due.
I registri 0x46,0x48 e 0x4A, OFFSET_axis_REG_H, contengono gli MSB dei dati hard-iron per ognuno dei tre assi, mentre i registri 0x45, 0x47 e 0x49 OFFSET_axis_REG_L contengono gli LSB.
I dati hard iron hanno lo stesso formato e peso dei dati di uscita. I valori memorizzati nei registri dedicati vengono automaticamente sottratti dai dati di output.

FIFO Il buffer FIFO può essere usato solo dall'accelerometro. LSM303AGR incorpora una FIFO di 32 livelli per ciascuno dei tre canali di uscita, X, Y e Z. Ciò permette un consistente risparmio energetico per il sistema, poiché il processore non deve eseguire continuamente il polling dei dati dal sensore, ma può svegliarsi solo quando necessario ed espellere i dati significativi dalla FIFO.
Per abilitare la memoria, il bit 6 (FIFO_EN) in CTRL_REG5_A (0x24) deve essere settato a 1.
Questo buffer può funzionare in base alle seguenti diverse modalità: modalità Bypass, FIFO, streaming e stream-to-FIFO. Ogni modalità è selezionata dai bit [1: 0] (FM) nel registro FIFO_CTRL_REG_A (0x2E).
In modalità Bypass la FIFO non è operativa e per questo motivo rimane vuota.
Nella modalità FIFO il buffer continua a riempire i dati dai canali dell'accelerometro X, Y e Z fino a quando non è pieno, cioè quando vengono memorizzati 32 campioni. Se la memoria è piena smette di raccogliere dati dai canali di input e il contenuto rimane invariato.
In modalità Stream, la FIFO continua a riempire i dati dai canali dell'accelerometro X, Y e Z, fino a quando il buffer non è pieno, a quel punto l'indice si riavvia dall'inizio e i dati precedenti vengono sostituiti da quelli correnti. I valori più vecchi continuano a essere sovrascritti fino a quando un'operazione di lettura libera gli slot della FIFO.
In modalità Stream-to-FIFO, il buffer inizia a funzionare in Stream e passa alla modalità FIFO quando si verifica l'interrupt selezionato. Quando viene configurato un evento di interrupt sul pin INT_1_XL, la FIFO opera in streaming se il valore del pin è uguale a 0 e funziona in modalità FIFO se INT_1_XL è a 1. La commutazione delle modalità viene eseguita in modo dinamico in base al valore del pin.

2.3.3 Caratteristiche meccaniche

@ Vdd = 2.5 V, T = 25 °C

Parametro	Condizioni	Min	Tipico	Max	Unità
Range accelerometro			± 2		g
			± 4		
			± 8		
			± 16		
Range magnetometro			± 49.152		gauss
Sensibilità accelerazione	FS=±2 g,alta risoluzione	-7%	0.98	+7%	mg/LSB
	FS=±4 g, alta risoluzione	-7%	1.95	+7%	
	FS=±8 g, alta risoluzione	-7%	3.9	+7%	
	FS=±16 g, alta risoluzione	-7%	11.72	+7%	
	FS=±2 g, normale	-7%	3.9	+7%	
	FS=±4 g, normale	-7%	7.82	+7%	
	FS=±8 g, normale	-7%	15.63	+7%	
	FS=±16 g, normale	-7%	46.9	+7%	
	FS=±2 g, low power	-7%	15.63	+7%	
	FS=±4 g, low power	-7%	31.26	+7%	
	FS=±8 g, low power	-7%	62.52	+7%	
	FS=±16 g, low power	-7%	187.58	+7%	
Sensibilità magnetometro		-7%	1.5	+7%	mgauss/ LSB
Sensibilità accelerometro vs. temperatura			0.01		%/°C
Sensibilità magnetometro vs. temperatura			±0.03		%/°C
Range operativo di temperatura		-40		+85	°C

Tabella 2.9: Caratteristiche meccaniche LSM303AGR [7]

2.3.4 Caratteristiche elettriche

@ Vdd = 2.5 V, T = 25 °C

Parametro	Condizioni	Min	Tipico	Max	Unità
Tensione di alimentazione		1.71	1.8	3.6	V
Consumo di corrente accelerometro, magnetometro in power-down	ODR 50 Hz, modalità normale		12.6		μA
	ODR 50 Hz, modalità normale		3.7		
	ODR 50 Hz, low-power		7.7		
Consumo di corrente magnetometro in alta risoluzione, accelerometro in power-down	ODR = 20 Hz		200		μA
Consumo di corrente magnetometro in low power, accelerometro in power-down	ODR = 20 Hz		50		μA
Consumo di corrente			2		μA

in power-down					
Range operativo di temperatura		-40		+85	°C

Tabella 2.10: Caratteristiche elettriche LSM303AGR [7]

2.4 Confronto tra LSM6DMS e LSM303AGR

Di seguito è proposto un confronto tra gli accelerometri dei sensori presentati nei capitoli 2.2 e 2.3. Entrambi hanno lo stesso range di alimentazione, da 1,7 V a 3.6 V, lo stesso fondo scala che va da ±2 g a ±16 g, la stessa variazione percentuale di sensibilità alla temperatura (0.01%/°C) e lo stesso range operativo di temperatura da -40°C a +85°C. Il sensore LSM6DMS ha però diversi parametri più performanti rispetto a LSM303AGR, ciò è dovuto al fatto che quest'ultimo ha un magnetometro al suo interno la cui realizzazione ha penalizzato le caratteristiche dell'accelerometro. Nella tabella riportata di seguito viene illustrato un breve confronto tra i parametri dei due sensori.

Parametro	**LSM6DMS**	**LSM303AGR**
Sensibilità massima [mg/LSB]	0.488 ±1%	187.58 ±7%
Output data rate massimo [Hz]	6664	5376
Consumo minimo di corrente[1] [μA]	4.5	3.7
Consumo massimo di corrente[1] [μA]	650	186

Tabella 2.11: Confronto tra LSM6DMS e LSM303AGR

1. Relativo al solo accelerometro, con giroscopio o magnetometro in power down.
Il consumo di corrente è migliore nel LSM303AGR ed è descritto in modo dettagliato nella tabella 2.4. Questo risparmio di energia però viene penalizzato da una sensibilità e da un output data rate massimo raggiungibile molto minore rispetto a LSM6DSM.

Capitolo 3

La piattaforma SensorTile

La piattaforma SensorTile STLCS02V1 è prodotta da STMicroelectronics ed è una scheda di sviluppo che include sensori e un modulo Bluetooth versione 4.2. SensorTile ha una grandezza di circa 1 cm^2, questo permette di utilizzarla come smart watch per via delle ridotte dimensioni. La scheda comprende un microcontrollore STM32L476, un microfono, sensori di movimento (accelerometro, giroscopio, magnetometro), sensori ambientali (termometro e barometro) e un modulo Bluetooth per trasmettere i dati ricevuti dalle varie periferiche. Per caricare e debuggare il firmware è stato usato un STM32F429I-DISC0 opportunamente collegato alla scheda SensorTile. Di seguito verranno descritti i componenti usati durante il progetto.

3.1 STM32L476JGY6

Il microcontrollore STM32L476JGY6 è un Arm® Cortex® M4 a 32 bit. I dispositivi STM32L476xx sono microcontrollori ultra-low-power basati su un set di istruzioni RISC a 32 bit e con un core che opera a frequenze fino a 80 MHz. Il core Cortex-M4 è dotato di una Floating Point Unit (FPU) a precisione singola che supporta tutte le istruzioni e i tipi di elaborazione dei dati a precisione singola Arm. Implementa inoltre un set completo di istruzioni DSP e una Memory Protection Unit (MPU) che migliora la sicurezza delle applicazioni. Le caratteristiche principali sono di seguito elencate:

- Consumi:
  - Alimentazione compresa tra 1.7 e 3.6 V
  - 30nA in shutdown
  - 120nA in stanby
  - 100 μA/MHz in run mode
- Sorgenti di clock:
  - Fino a 80 MHz di clock
  - Oscillatore al cristallo da 4 Mhz a 48 MHz
  - Oscillatore al cristallo a 32 KHz per il real time clock (RTC)
  - Oscillatori interni low-power a 32 KHz per RTC
  - Oscillatori interni multi velocità da 100 KHz a 48 MHz
  - 3 PLL
- Periferiche:
  - Fino a 114 GPIO
  - 16 timers
- Memorie:

  - 1 MB di flash
  - 128 KB di SRAM
- Periferiche analogiche:
  - 3 ADC a 12 bit
  - 2 canali DAC di output a 12 bit
  - 2 amplificatori operazioni con una PGA built-in
  - 2 comparatore ultra low power
- 20 interfacce di comunicazione:
  - USB 2.0
  - 2 SAIs (serial audio interface)
  - 3 I^2C
  - 5 USARTs
  - 1 LPUART
  - 3 SPI a tre fili (e una a quattro fili)
  - CAN bus e interfaccia SDMMC
  - SWPMI
  - IRTIM (Infrared interface)

3.2 I sensori della piattaforma

Il SensorTile presenta due sensori inerziali: LSM6DSM e LSM303AGR, descritti rispettivamente nei capitoli 2.2 e 2.3. Inoltre ha al suo interno un sensore MEMS di pressione e temperatura LPS22HB. Come riportato nel datasheet [6], LPS22HB è un sensore di pressione piezoresistivo ultracompatto che funge da barometro digitale. Il dispositivo comprende un sensing element e un'interfaccia che comunica tramite I^2C o SPI. L'elemento sensibile, che rileva la pressione assoluta, è costituito da una membrana sospesa prodotta utilizzando un processo dedicato sviluppato da ST. LPS22HB è disponibile in uno package LGA forato per consentire a una pressione esterna di raggiungere l'elemento sensibile. Viene garantito il funzionamento in un intervallo di temperatura che si estende da -40 ° C a +85 ° C.
Le sue applicazioni tipiche sono:

- altimetro e barometro per dispositivi portatili,
- applicazioni GPS,
- stazioni meteo,
- sport watches.

Le caratteristiche più importanti del sensore di pressione e temperatura sono elencate di seguito:

- range di pressione assoluta compreso tra 260 e 1260 hPa,
- consumo di corrente di 3μA,
- output dei dati di pressione a 24 bit,
- output dei dati di temperatura a 16 bit,
- ODR compreso tra 1 e 75 Hz,
- presenta una memoria FIFO,
- tensione di alimentazione compresa tra 1.7 e 3.6 V.

3.3 BlueNRG-MS

La scheda BlueNRG è basata su un Cortex® M0 a 32 bit e implementa al suo interno lo stack Bluetooth Low Energy: GAP, GATT, SM, L2CAP, LL. Il BlueNRG-MS è un processore di rete Bluetooth Low Energy a bassissima potenza, conforme alla specifica Bluetooth v4.2. La memoria flash non volatile installata sul dispositivo permette l'aggiornamento on-field dello stack Bluetooth. BlueNRG-MS consente alle applicazioni di soddisfare i severi requisiti di corrente di picco delle normali batterie a bottone. La modalità di sleep a bassissima potenza e i tempi di transizione molto brevi tra le modalità operative consentono un consumo di corrente medio molto basso, con conseguente aumento della durata della batteria. BlueNRG-MS offre la possibilità di interfacciarsi con microcontrollori esterni tramite l'interfaccia SPI.
Il modulo integra sia un regolatore di tensione low dropout (LDO) sia un convertitore DC-DC step-down, uno di questi due può essere utilizzato per alimentare i circuiti interni BlueNRG-MS.
Le applicazioni di maggiore utilizzo del dispositivo sono lo sport, il controllo remoto di apparati, l'automazione industriale e domestica, l'assisted living, gli smartwatch e le periferiche PC e smartphone.
Le sue caratteristiche sono:

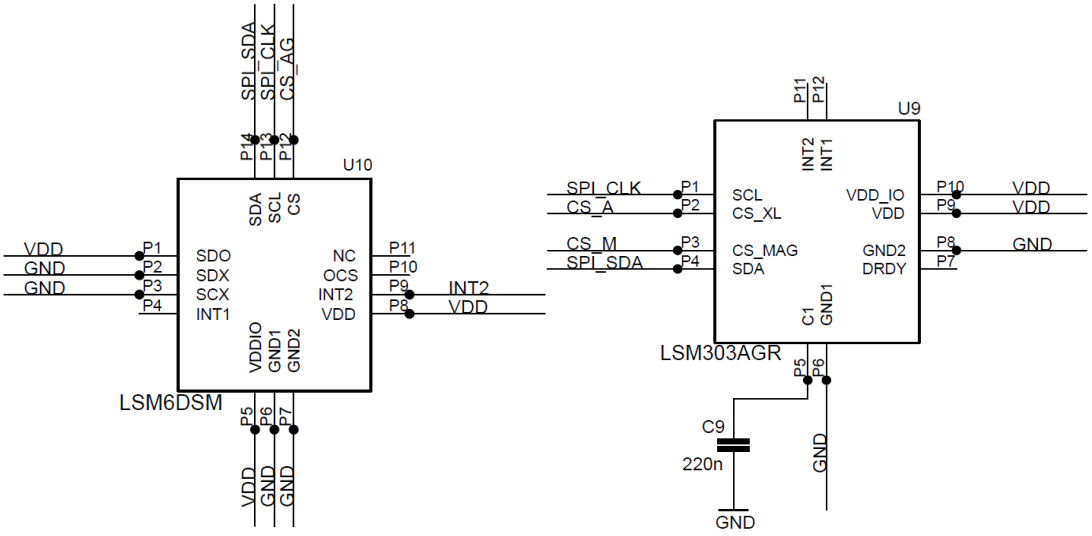
- Consumi di corrente nella modalità usata nel progetto, cioè con il convertitore DC-DC attivo:

Parametro	**Modalità**	**Condizioni**	**Valore**	**Unità di misura**
Corrente di alimentazione	Ricevitore	Alta potenza	7.7	mA
		Standard	7.3	mA
	Trasmissione standard	+5 dBm	11	mA
		0 dBm	8.2	mA
		-2 dBm	7.2	mA
		-6 dBm	6.7	mA
		-9 dBm	6.3	mA
		-12 dBm	6.1	mA
		-15 dBm	5.9	mA
		-18 dBm	5.8	mA
	Trasmissione alta potenza	+8 dBm	15.1	mA
		+4 dBm	10.9	mA
		+2 dBm	9	mA
		-2 dBm	8.3	mA
	standby		1.4	μA
	sleep	32 kHz XO ON (RAM2 OFF)	1.7	μA
		32kHz XO ON (RAM2 ON)	2.4	μA
		32 kHz RO ON (RAM2 OFF)	2.8	μA
		32 kHz RO ON (RAM2 ON)	3.5	μA

Tabella 3.1: Consumi di corrente BlueNRG [5]

- Clock:
  - oscillatore al cristallo a 16 o 32 MHz
  - ring oscillator a 12 MHz
  - oscillatore al cristallo a 32 KHz
  - ring oscillator a 32 KHz

3.4 Schema circuitale



(a) LSM6DSM (b) LSM303AGR

Figura 3.1: I due sensori inerziali [12]

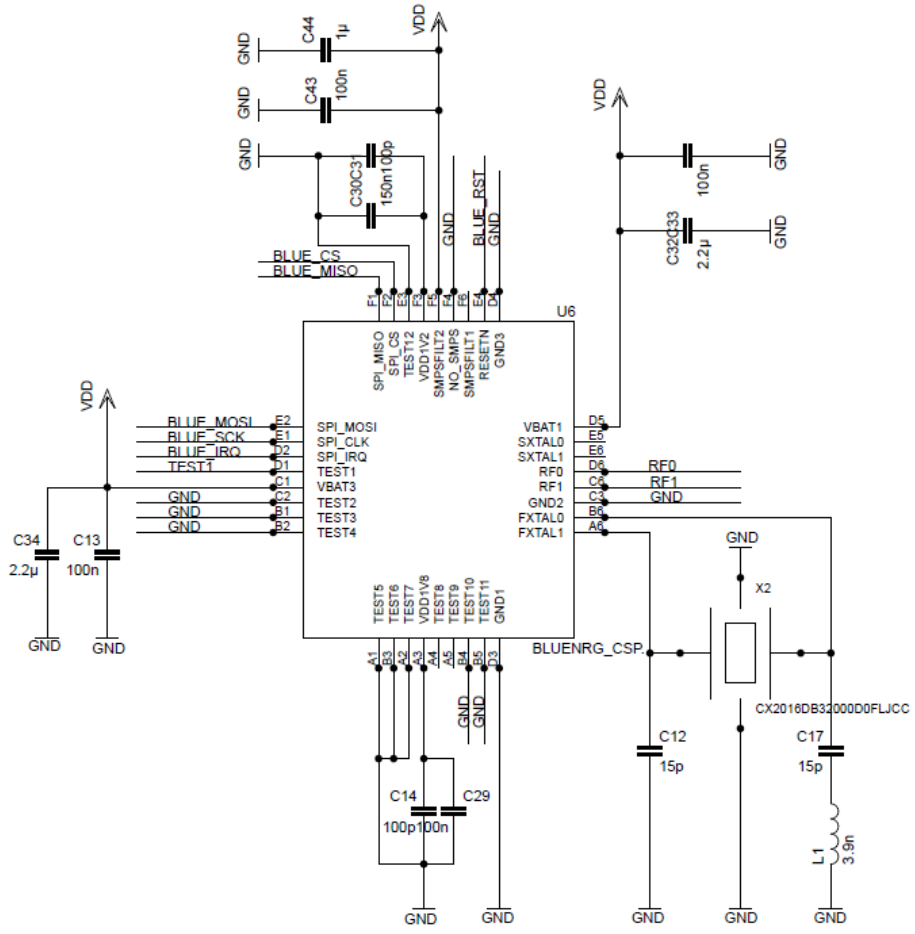


Figura 3.2: BlueNRG [12]

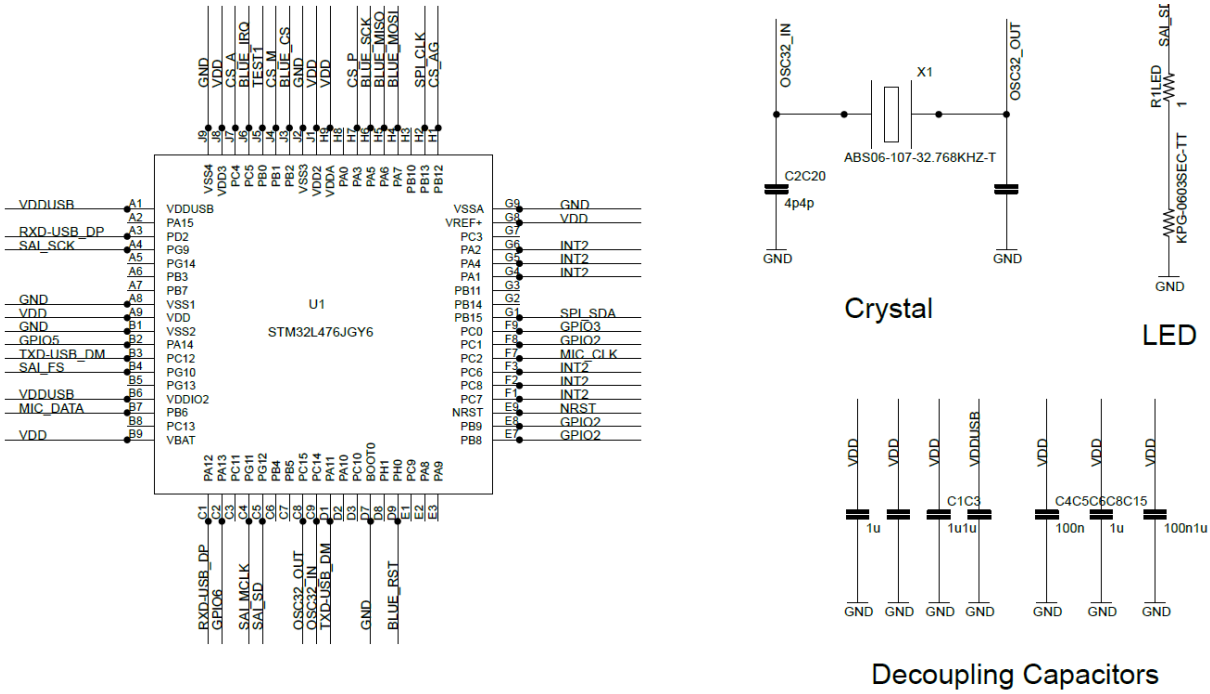


Figura 3.3: STM32L476JGY6 [12]

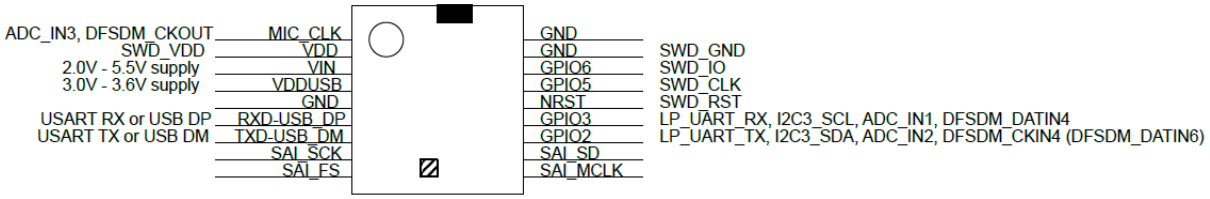


Figura 3.4: SensorTile [12]

Capitolo 4

Programma in Python

Nel progetto è stato scritto un programma in Python per ricevere le notifiche Bluetooth. È stata usata la libreria bluepy che permette l'accesso ai dispositivi Bluetooth Low Energy da Python. Al momento funziona solo su Linux e dispositivi Debian x86. Il programma ricerca in loop il dispositivo SensorTile tramite il suo MAC address (c0:86:1d:31:45:48). Appena lo trova si connette e chiede all'utente quali notifiche abilitare tra le seguenti:

- accelerometro, magnetometro e giroscopio,
- temperatura e pressione,
- dati sensor fusion,
- pitch e roll.

I dati ricevuti dalle caratteristiche con notifiche abilitate vengono salvati in dati file txt formattati in modo tabellare per facilitare l'analisi dei risultati con Matlab.

4.1 Connessione

All'inizio del programma viene creato un oggetto *scanner* che viene utilizzato per cercare dispositivi LE che trasmettono dati di advertising. Con il metodo *withDelegate(delegate)* si memorizza un riferimento a un oggetto *delegate* che viene chiamato quando vengono ricevuti messaggi asincroni, come le notifiche o dati di advertising. *Scan(timeout)* cerca i dispositivi per un intervallo temporale in secondi specificato nella variabile timeout. Durante questo periodo, verranno effettuate chiamate all'oggetto *delegate*. Al termine del timeout, la scansione si interrompe e il metodo restituisce un elenco di oggetti *ScanEntry* per tutti i dispositivi rilevati durante tale periodo. Gli oggetti *ScanEntry* hanno un attributo *addr* che contiene il MAC address del dispositivo. Si cerca dunque un oggetto che abbia questo uguale alla variabile *devAddr*, impostata nel programma pari a c0:86:1d:31:45:48, cioè l'indirizzo del SensorTile. Se la ricerca è andata a buon fine significa che la scheda è accesa e quindi si esce dal loop che ricerca i dispositivi Bluetooth disponibili. Infine, si crea un oggetto *peripheral* specificandone direttamente il MAC address, quando viene stabilita la connessione, i servizi e le caratteristiche offerti da quel dispositivo possono essere letti o scritti.

```
[...]
while True:
  try:
    #finchè il sensor tile è spento faccio una ricerca del dispositivo
    while (SensorTile_state == 0):
      print("Sto cercando: {},  indirizzo: {}".format(devAddr, addrType))
      #wthDelegate(delegate) è un metodo che immagazzina un oggetto "delegate"
      scanner = Scanner().withDelegate(ScanDelegate())
      #timeout per la scansione = 10 secondi
      timeout=10.0
      #scansiona i dispositivi per 10 secondi
      devices = scanner.scan(timeout)
```

```
13      for dev in devices:
14        #cerco il dispositivo con mac-address uguale a c0:86:1d:31:45:48
15        if (dev.addr == devAddr):
16          #SensorTile è acceso
17          SensorTile_state = 1
18    #creo un oggetto "Peripheral" ed effettuo una connessione al dispositivo indicato in
      devAdd (c0:86:1d:31:45:48)
19    conn = Peripheral(devAddr, addrType)
20    print("Connesso a: {}".format(devAddr))
21    [...]
```

4.2 Abilitazione notifiche

Una volta connesso al dispositivo, si richiede all'utente quali notifiche abilitare. Per lavorare con i servizi e le caratteristiche si è fatto riferimento al manuale utente [10]. Le caratteristiche che contengono dati provenienti dai sensori devono utilizzare un UUID come il seguente: XXXXXXXX-0001-11e1-ac36-0002a5d5c51b La prima parte dell'UUID ha bit impostati su "1" per ogni funzione esportata dalle caratteristiche. Nel caso di più funzioni mappate in una singola caratteristica, i dati devono essere nello stesso ordine della maschera di bit. La maschera delle funzionalità (feature mask) è un campo di bit che fornisce informazioni sulle funzioni esportate dalla scheda. La tabella 4.1 elenca la mappatura tra bit e funzioni.

bit	**funzionalità**
0	Pedometro
1	Rilevazione gesti dai dati MEMS
2	Rilevazione gesti usando sensori di prossimità
3	Indicazione su come l'utente sta trasportando il dispositivo
4	Attività
5	Bussola
6	Intensità del movimento
7	Sensor Fusion
8	Sensor Fusion Compact
9	Caduta libera
10	Evento dell'accelerometro
11	Direzione corrente per l'algoritmo di beamforming
12	SD log
13	Motore passo passo
14	Motore DC - Non ancora formalizzato
15	Sensore CO
16	Temperatura 2
17	Batteria
18	Temperatura 1
19	Umidità
20	Pressione
21	Magnetometro
22	Giroscopio
23	Accelerometro
24	Sensore di luminosità
25	Sensore di prossimità
26	Livello del microfono
27	Audio ADPCM
28	Direzione di arrivo
29	Stato degli switch
30	Sync ADPCM
31	Analog - Non ancora formalizzato

Tabella 4.1: Feature mask bit [10]

Il formato dei dati è riportato nella tabella 4.2:

Lunghezza	Nome
2 byte	Timestamp
>1 byte	Primo dato
>1 byte	Secondo dato
...	...

Tabella 4.2: Formato dei dati [10]

I primi 2 byte vengono utilizzati per inviare un timestamp. Ciò è particolarmente utile per riconoscere qualsiasi perdita di dati. Poiché la lunghezza massima del pacchetto BLE è di 20 byte, la dimensione massima di un campo dati è 18 byte.
I passi seguiti per abilitare le notifiche sono i seguenti e sono riferiti alla caratteristica del pitch e roll, in quanto le operazioni sono uguali per ogni caratteristica:

1. si ottengono le caratteristiche con il metodo *getCharacteristics* che restituisce un elenco contenente oggetti *characteristic* della periferica. I parametri della funzione sono *startHnd = 1* e *endHnd = 0xFFFF*, necessari per ottenere tutte le caratteristiche con handle compreso tra i due valori, e l'*UUID* della caratteristica da cercare, nel caso del pitch e roll l'uuid è *00ee0000-0001-11e1-ac36-0002a5d5c51b*. È possibile notare che la prima parte dell'uuid (00ee0000) non corrisponde a nessuna delle funzionalità elencate nella tabella 4.1. Il pitch e roll non era infatti stato previsto da STMicroelectronics tra le funzionalità del SensorTile ed è stato implementato nel corso del progetto come descritto nella sezione 6.1.8.

2. *getHandle()* restituisce il valore intero a 16 bit utilizzato per identificare la caratteristica nel protocollo GATT, cioè l'handle. Questo è utile per distinguere tra notifiche da caratteristiche diverse

3. si ottene un oggetto *Descriptor* con *getDescriptors(forUUID = 0x2902)*. In particolare si vuole cercare il descrittore con handle *0x2902*, cioè il CCCD (Client Characteristic Configuration Descriptor), con il quale si possono abilitare le notifiche

4. l'utente, a questo punto, sceglie quali notifiche abilitare

5. Il metodo *write(notify_enable)* scrive *notify_enable = 0x0100*, in little endian, nel campo value del CCCD, indicando che le notifiche sono abilitate, quindi il server può inviare notifiche al client.

6. Il metodo *waitForNotifications (timeout)* blocca il programma fino a quando non viene ricevuta una notifica dalla periferica o fino allo scadere dell'intervallo specificato (in secondi) nella variabile *timeout = 1*. Se viene ricevuta una notifica, verrà chiamato il metodo *handleNotification()* dell'oggetto *delegate* e *waitForNotifications()* restituirà True. Se non viene ricevuto nulla prima che scada il timeout, questo restituirà False.

```
1     #ottengo una lista di oggetti "Characteristic" cercando tra le caratteristiche con
    handle compreso tra 0x00001 e 0xFFFF dalla periferica e con l'UUID specificato
2     [...]
3     ch_pitch_roll = conn.getCharacteristics (0X0001, 0XFFFF, "00ee0000-0001-11e1-ac36
    -0002a5d5c51b")[0]   #caratteristica pitch e roll
4     #ottengo l'handle della caratteristica usata per identificare la relativa
    caratteristica
5     #serve per capire quale caratteristica è relativa alla notifica ricevuta
6     [...]
7     handle_pitch_roll = ch_pitch_roll.getHandle() #handle caratteristica pitch e roll
8     #ottengo una lista di oggetti "Descriptor" con UUID relativo al CCCD (0x2902).
9     [...]
10    cccd_pitch_roll = ch_pitch_roll.getDescriptors(forUUID=0x2902)[0] #descrittore
    della caratteristica pitch e roll
11    [...]
12    scelta_pitch_roll = input ("Abilitare le notifiche della caratteristica del pitch e
     roll? (s/n) ")
13    [...]
```

```
      if (scelta_pitch_roll == "s"):
        #abilitazione notifiche pitch e roll
        cccd_pitch_roll.write(notify_enable)
      elif (scelta_acc_giro_magn == "n"):
        #disabilitazione notifiche pitch e roll
        cccd_pitch_roll.write(disable_notify_and_indication)
        print("Dati del pitch e roll non disponibili perchè le notifiche sono disattivate
    ")
      else:
        print ("Scelta non corretta")
      #se è stata abilitata almeno una notifica
      if (scelta_temperatura_pressione  == "s" or scelta_acc_giro_magn == "s" or
    scelta_sensor_fusion_compact == "s" or scelta_pitch_roll == "s"):
        #ciclo per la gestione delle notifiche
        try:
          while True:
            timeout_notification = 1.0
            if conn.waitForNotifications(timeout_notification):
              #chiamata della funzione handleNotification()
              continue
        except BTLEException as e:
          #azzero la variabile SensorTile_state perchè il Sensor Tile è disconesso
          SensorTile_state = 0
          print("Errore: ", e)
    finally:
    #disconnessione dal SensorTile
      conn.disconnect()
#premere CTRL + C per uscire dal ciclo while True in cui si ricevono le notifiche
except KeyboardInterrupt:
  print("Interruzione da tastiera")
```

4.3 Gestione notifiche

Per gestire i dati ricevuti si usa il metodo *handleNotification(self, cHandle, data)* i cui parametri *cHandle* e *data* indicano rispettivamente l'handle della caratteristica da cui provengono i dati e il valore ricevuto. Al fine di dividere il pacchetto si usa *unpack(format, string)* che frammenta la stringa secondo il formato specificato. Per decidere il modo in cui separare il dato si è usato nuovamente il manuale [10]. I dati per cui è possibile attivare le notifiche sono organizzati come riportato di seguito:

- Accelerometro
  I valori dei tre assi sono espressi in millesimi di g.

Byte	Descrizione
0	Timestamp
1	
2	Asse X (int16)
3	
4	Asse Y (int16)
5	
6	Asse Z (int16)
7	

Tabella 4.3: Formato dei dati accelerometro [10]

- Giroscopio
  I valori dei tre assi sono espressi in gradi al secondo.

Byte	Descrizione
0	Timestamp
1	
2	Asse X · 10 (int16)
3	
4	Asse Y · 10 (int16)
5	
6	Asse Z · 10 (int16)
7	

Tabella 4.4: Formato dei dati giroscopio [10]

- Magnetometro
  I valori dei tre assi sono espressi in millesimi di Gauss.

Byte	Descrizione
0	Timestamp
1	
2	Asse X (int16)
3	
4	Asse Y (int16)
5	
6	Asse Z (int16)
7	

Tabella 4.5: Formato dei dati magnetometro [10]

I dati dell'accelerometro, giroscopio e magnetometro sono inseriti in una sola caratteristica. L'organizzazione dei valori rispetta l'ordine dei mask bit, quindi, poiché i dati sono in little endian, viene ricevuto prima il timestamp, poi i byte dell'accelerometro, giroscopio e, infine, magnetometro.

- Pressione
  La pressione è espressa in millesimi di bar.

Byte	Descrizione
0	Timestamp
1	
2	Pressione · 100 (int32)
...	
5	

Tabella 4.6: Formato dei dati di pressione [10]

- Temperatura
  La temperatura è espressa in gradi centigradi.

Byte	Descrizione
0	Timestamp
1	
2	Temperatura · 10 (int16)
3	

Tabella 4.7: Formato dei dati di temperatura [10]

- Sensor fusion compact
  I dati sono i quaternioni calcolati dall'algoritmo di sensor fusion dei dati provenienti dai sensori MEMS, inviati come tre valori alla volta e già normalizzati. I quaternioni forniscono una notazione matematica conveniente per la rappresentazione di orientamenti e rotazioni di oggetti in tre dimensioni. Un quaternione è definito come una combinazione lineare espressa nella base $< 1, i, j, k >$: $\mathbf{q} = q_0 + q_1\mathbf{i} + q_2\mathbf{j} + q_3\mathbf{k}$, dove i coefficienti q_0, q_1, q_2, q_3 sono reali. In analogia con i numeri complessi, dove $c = a + jb$ è rappresentabile da una coppia di reali, (a, b), il generico quaternione è rappresentabile da una quadrupla di reali, (q_0, q_1, q_2, q_3). Una rotazione può essere rappresentata utilizzando la formula:
  $$\mathbf{q} = \cos\frac{\theta}{2} + (a_x\mathbf{i} + a_y\mathbf{j} + a_z\mathbf{k})\sin\frac{\theta}{2}$$
  dove θ è l'angolo di rotazione e il vettore (a_x, a_y, a_z) è un versore che rappresenta l'asse di rotazione. Per applicare le rotazioni a un punto $\mathbf{p}$ rappresentato dal vettore posizione $\mathbf{p} = (p_x, p_y, p_z)$, si calcola il prodotto $\mathbf{qpq}^*$. Il risultato è il nuovo vettore posizione (p'_x, p'_y, p'_z) del punto dopo la rotazione. L'asse e l'angolo θ possono essere ricavati dal quaternione $\mathbf{q} = q_0 + q_1\mathbf{i} + q_2\mathbf{j} + q_3\mathbf{k}$ usando le seguenti formule:
  $$\theta = 2\arccos q_0$$
  $$(a_x, a_y, a_z) = \frac{1}{sin\frac{\theta}{2}}(q_1, q_2, q_3)$$

Byte	**Descrizione**
0	Timestamp
1	
2	$Q_i \cdot 10000$ (int16)
3	
4	$Q_j \cdot 10000$ (int16)
5	
6	$Q_k \cdot 10000$ (int16)
7	
8	$Q_i \cdot 10000$ (int16)
9	
10	$Q_j \cdot 10000$ (int16)
11	
12	$Q_k \cdot 10000$ (int16)
13	
14	$Q_i \cdot 10000$ (int16)
15	
16	$Q_j \cdot 10000$ (int16)
17	
18	$Q_k \cdot 10000$ (int16)
19	

Tabella 4.8: Formato dei dati sensor fusion compact [10]

- Pitch e roll
  I dati sono espressi in radianti e moltiplicati per 8192 (2^{13}) nel firmware.

Byte	Descrizione
0	Timestamp
1	
2	Pitch· 8192 (int16)
3	
4	Roll · 8192 (int16)
5	

Tabella 4.9: Formato dei dati pitch e roll

I dati vengono infine scritti in un file txt in modo tabellare dopo essere stati convertiti. Ad esempio, i dati di beccheggio e rollio vengono divisi per 8192 e riportati in gradi.

```
class DefaultDelegate:
  def handleNotification(self, cHandle, data):
    DBG("Notification:", cHandle, "sent data", binascii.b2a_hex(data))
    [...]
    #se l'handle "cHandle" passato a handleNotification è quello corrispondente al sensor fusion compact
    if (cHandle == handle_pitch_roll):
      #salvataggio del pacchetto ricevuto in una variabile ausiliaria
      newvalue = data
      print("\t\tValore ricevuto caratteristica pitch e roll: ",str(binascii.hexlify(newvalue), 'ascii').upper())
      #scomposizione del pacchetto ricevuto in timestamp (2 byte) e altri 2 dati da 2 byte ciascuno secondo il file "Getting started with the BlueST protocol and SDK.pdf"
      timestamp4, pitch, roll = unpack('<Hhh', newvalue)
      #conversione da radianti a gradi
      pitch = pitch / 8192 * 180 / math.pi
      roll = roll / 8192 * 180 / math.pi
      print("\t\tTimestamp: {}\n\t\tPitch: {} °\n\t\tRoll: {} °\n\t\t".format(timestamp4, pitch, roll))
      #scrittura sul file che registra i dati del pitch e roll necessari a MATLAB
      try:
        file_pitch_roll =  open(nome_file_pitch_roll_matlab, 'a+')
        file_pitch_roll.write("{}\t{}\t{}\t{}\n".format(timestamp4, time_formato_matlab, pitch, roll))
        file_pitch_roll.close()
      except IOError:
        print ("Errore di I/O sul file.")
```

Capitolo 5

Struttura del firmware e configurazione dell'ambiente di sviluppo

5.1 Configurazione System Workbench for STM32

La scheda SensorTile contiene già un firmware funzionante al suo interno. L'azienda STMicroelectronics fornisce il programma caricato gratuitamente nel proprio sito. Per modificare il firmware si è scaricato il pacchetto di sviluppo FP-SNS-ALLMEMS1 che consente di collegare il nodo IoT a uno smartphone tramite BLE e utilizzare un'applicazione Android o iOS adatta, come l'app ST BLE Sensor, per visualizzare i dati dei sensori ambientali in tempo reale, dei sensori di movimento, il livello del microfono digitale e della batteria. L'ambiente di sviluppo utilizzato è "System Workbench for STM32", disponibile anche per Linux che è il sistema operativo usato durante il progetto. Per il corretto funzionamento dell'ambiente di sviluppo è necessario installare la libreria libncurser5 a 32 bit e la toolchain per dispositivi ARM. Una volta scaricato ed estratto il file "en.fp-sns-allmems1_firmware.zip", per compilare correttamente il programma occorre rinominare la libreria statica "libbluevoiceADPCM_200_CM4F_GCC_ot.a" presente nel percorso "STM32CubeFunctionPack_ALLMEMS1_V3.4.0/Middlewares/ST/STM32_BlueVoiceADPCM_Library" in "libBlueVoiceADPCM_200_CM4F_GCC_ot.a". Per compilare correttamente il progetto occorre anche impostare *STM32_SENSORTILE = 1* nelle impostazioni del preprocessore.
Per flashare il firmware si può usare lo script "CleanALLMEMS1_SW4STM32_ST.sh" contenuto nel percorso "/STM32CubeFunctionPack_ALLMEMS1_V3.4.0/Projects/Multi/Applications/ ALLMEMS1/SW4STM32/STM32L476RG-SensorTile". Per usare questo script occorre scaricare OpenOCD. Il contenuto dello script va modificato settando il percorso di installazione di OpenOCD, degli script per stm32 e togliendo il commento alla riga relativa a Linux che aggiunge la libreria OpenOCD. Al fine di caricare correttamente il firmware si deve cambiare "source [find interface/stlink-v2-1.cfg]" in "source [find interface/stlink-v2.cfg]" nel file di configurazione della board "nucleo_l476rg.cfg" presente nel percorso "/Ac6/plugins/fr.ac6.mcu.debug_2.1.3.201710240950/openocd/scripts/st_board/".

5.2 Inizializzazioni

Viene ora presentata la sequenza logica delle inizializzazioni delle varie periferiche delle scheda.

- RestartInBootLoaderMode
- SystemClock_Config(): configura il clock di sistema
- Sensor_IO_SPI_CS_Init_All(): configura tutti i pin di chip select
- InitTargetPlatform(TARGET_SENSORTILE): inizializza le varie periferiche della piattaforma SensorTile

  - USB (se abilitata)
  - Init_MEM1_Sensors(): inizializza e configura LSM6DSM, LSM303AGR (Magnetometro), LPS22HB e HTS221
  - Init_STC3115(): se la batteria è presente inizializza l'indicatore del livello di carica della batteria
  - Init_MEMS_Mics(): inizializza il microfono
- InitLicenseManager(): avvia il gestore delle licenze per l'algoritmo di sensor fusion
- Init_BlueNRG_Stack(): inizializza il processore di rete wireless a bassa energia Bluetooth BlueNRG
- Init_BlueNRG_Custom_Services(): inizializza i servizi del protocollo bluetooth
  - Add_HWServW2ST_Service(): aggiunge i servizi relativi all'hardware della scheda (sensori e algoritmo di sensor fusion)
  - Add_ConsoleW2ST_Service(): aggiunge i servizi della console utilizzando il profilo specifico del fornitore
  - Add_ConfigW2ST_Service(): aggiunge i servizi di configurazione (calibrazione, reset, eventi dell'accelerometro)
- InitHWFeatures(): legge l'output data rate di default dell'accelerometro
- InitTimers(): inizializza i timer
  - TIM4: usato per i sensori ambientali (temperatura e pressione) → 500ms / 2 Hz
  - TIM1: usato per i sensori di movimento
  - TIM5: usato per il microfono → 10ms

5.3 Main loop

Nel main loop viene gestita principalmente la parte Bluetooth, sia per quanto riguarda la connessione sia per la trasmissione dei dati.

- Led Management(): gestisce l'accensione dei led
- Handlers(): gestisce gli eventi di interrupt esterno (da sensori di movimento o dal Bluetooth) o dovuti ai timer
- setConnectable(): aggiorna i dati in advertising e rendere la scheda collegabile
- Send [SensorData] o Update [SensorData]: manda periodicamente dati ambientali, audio, di movimento, quaternioni, ecc...
- __WFI(): Wait For Interrupt events, attende eventi per provocano interrupt da periferiche come timer, Bluetooth, accelerometro.

Capitolo 6

Descrizione firmware

6.1 Inizializzazioni

6.1.1 Accensione del dispositivo

Nella funzione *main()* la variabile volatile *out_of_shutdown* è inizializzata a 0 e settata quando viene rilevata l'uscita dallo spegnimento.

```
int main(void)
{
  #ifdef STM32_SENSORTILE
  out_of_shutdown = 0;
  #endif /* STM32_SENSORTILE */
  HAL_Init();
  [...]
```

6.1.2 Configurazione flash, cache e interrupt

Viene poi chiamata la funzione *HAL_Init()* che configura il flash prefetch, le cache di istruzioni e dati, l'origine di base dei tempi, NVIC (Nested Vectored Interrupt Controller) e qualsiasi hardware di basso livello globale richiesto chiamando la funzione *HAL_MspInit ()* che è facoltativamente definita nel file utente *stm32l4xx_hal_msp.c*. La funzione *HAL_Init()* è chiamata dopo il reset e prima della configurazione del clock. Il dispositivo è settato di default nel seguente modo:

- Prefetch disabilitato
- Cache istruzioni abilitata
- Cache dati abilitata

Il timer di sistema (Systick) è usato come base dei tempi. La configurazione Systick è basata sul clock MSI (Multi Speed Oscillator), cioè sul clock usato dopo il reset del sistema, e la configurazione di NVIC è settata al gruppo di priorità 4. Una volta inizializzato il timer, il tick della base dei tempi inizia ad incrementarsi, la variabile che conta i tick è incrementata ogni millisecondo nel gestore di interruzioni *SysTick_Handler()*.

```
HAL_StatusTypeDef HAL_Init(void)
{
  [...]
  /* Set Interrupt Group Priority */
  HAL_NVIC_SetPriorityGrouping(NVIC_PRIORITYGROUP_4);

  /* Use SysTick as time base source and configure 1ms tick (default clock after Reset is
     MSI) */
  HAL_InitTick(TICK_INT_PRIORITY);

  /* Init the low level hardware */
  HAL_MspInit();

```

```
  /* Return function status */
  return HAL_OK;
}
```

6.1.3 Configurazione del clock

Successivamente la funzione *SystemClock_Config()* configura il clock di sistema. Con la funzione *HAL_PWR_EnableBkUpAccess()* si setta ad 1 il bit 8 (PWR_CR1_DBP) del registro PWR_CR1 per abilitare l'accesso al dominio di backup (registri RTC e registri RTC di backup), perché dopo il reset il dominio di backup è protetto contro possibili accessi in scrittura indesiderati. Si abilita l'oscillatore LSE (low-speed external) tramite la funzione *HAL_RCC_OscConfig* che inizializza gli oscillatori RCC (Reset and Clock Control) secondo i parametri definiti nella struct *RCC_OscInitTypeDef*. Nel caso in cui il segnale LSE non sia presente o è corrotto si abilita l'interrupt CSS (Clock Security System). Si abilita l'oscillatore MSI e si attiva un PLL (Phase Locked Loop) con MSI come sorgente usando la funzione *HAL_RCC_OscConfig*. Viene poi abilitata l'autocalibrazione dell'oscillatore MSI attraverso LSE tramite la funzione *HAL_RCCEx_EnableMSIPLLMode()* che setta il bit 2 (RCC_CR_MSIPLLEN) del registro RCC_CR. Viene selezionata l'uscita MSI come sorgente del clock della USB. Infine si seleziona la PLL come sorgente di clock del sistema e si configurano i prescaler dei clock HCLK (High-speed Clock), PCLK1 (Peripheral Clock) e PCLK2. I divisori del prescaler sono tutti settati a zero.

```
static void SystemClock_Config(void)
{
  [...]
  __HAL_RCC_PWR_CLK_ENABLE();
  HAL_PWR_EnableBkUpAccess();

  /* Enable the LSE Oscilator */
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_LSE;
  RCC_OscInitStruct.LSEState = RCC_LSE_ON;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK){
    while(1);
  }
  /* Enable the CSS interrupt in case LSE signal is corrupted or not present */
  HAL_RCCEx_DisableLSECSS();

  /* Enable MSI Oscillator and activate PLL with MSI as source */
  [...]
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK){
    while(1);
  }
  [...]
  if(HAL_RCCEx_PeriphCLKConfig(&PeriphClkInitStruct) != HAL_OK){
    while(1);
  }
  /* Enable MSI Auto-calibration through LSE */
  HAL_RCCEx_EnableMSIPLLMode();

  /* Select MSI output as USB clock source */
  PeriphClkInitStruct.PeriphClockSelection = RCC_PERIPHCLK_USB;
  PeriphClkInitStruct.UsbClockSelection = RCC_USBCLKSOURCE_MSI;
  HAL_RCCEx_PeriphCLKConfig(&PeriphClkInitStruct);

  /* Select PLL as system clock source and configure the HCLK, PCLK1 and PCLK2
  clocks dividers */
  [...]
  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_4) != HAL_OK){
    while(1);
  }
}
```

6.1.4 Controllo uscita dallo spegnimento

Si controlla se il sistema è ritornato dalla modalità di spegnimento ricorrendo al registro di backup RTC RTC_BKP31R per verificare l'uscita dallo sleep indipendentemente dal fatto che il flag di blocco

dell'arresto sia stato impostato dal software prima di entrare in modalità spegnimento. Si legge il registro e se è settato a 1 viene resettato e portata ad 1 la variabile *out_of_shutdown*.

```
#ifdef STM32_SENSORTILE
/* Check if the system was resumed from shutdown mode,
resort to RTC back-up register RTC_BKP31R to verify
whether or not shutdown entry flag was set by software
before entering shutdown mode.  */
if (READ_REG(RTC->BKP31R) == 1)
{
  /* reset back-up register */
  WRITE_REG( RTC->BKP31R, 0x0 );
  /* out of shutdown detected */
  out_of_shutdown = 1;
}
#endif /* STM32_SENSORTILE */
```

6.1.5 Inizializzazioni sensori

La funzione *InitTargetPlatform* inizializza tutte le funzionalità della piattaforma SensorTile. Tramite *Sensor_IO_SPI_CS_Init_All()* si configurano e si abilitano i pin di Chip Select. Con questa funzione vengono settati i pin di chip select di: LSM6DSM, LSM303AGR_X, LSM303AGR_M, LPS22HB. In particolare viene usata la velocità alta dei GPIO e non viene attivata la modalità pull-up o pull-down. Una volta abilitati i dispositivi viene acceso il LED1. Il pin "SAI_SD", in cui andrebbe connesso il led secondo lo schema elettrico mostrato in figura 3.3, durante lo svolgimento del progetto risulta non collegato. *Init_MEM1_Sensors()* inizializza le funzionalità dei sensori MEMS della piattaforma.

```
void InitTargetPlatform(TargetType_t BoardType)
{
  TargetBoardFeatures.BoardType = BoardType;
  /* Configure and disable all the Chip Select pins */
  Sensor_IO_SPI_CS_Init_All();
  [...]
  /* Initialize LED */
  BSP_LED_Init( LED1 );
  [...]
  /* Discovery and Intialize all the MEMS Target's Features */
  Init_MEM1_Sensors();
  [...]
}
```

6.1.6 Configurazione sensori

La funzione *Init_MEM1_Sensors()* inizializza i vari sensori usando *Sensor_IO_Write* che riceve come parametri l'handle del sensore, il registro in cui scrivere il dato, il puntatore al dato e il numero di byte da scrivere. Per tutti i sensori viene abilitato l'SPI a tre fili e disabilitata l'interfaccia I^2C, tranne per il sensore di umidità che lavora con protocollo I^2C. La procedura usata è la seguente:

1. *BSP_ACCELERO_Init* inizializza uno dei due sensori accelerometrici presenti. Inizialmente la funzione *BSP_LSM6DSM_ACCELERO_Init* prova ad inizializzare l'accelerometro LSM6DSM, se non vi riesce si inizializza LSM303AGR con *BSP_LSM303AGR_ACCELERO_Init*
   (a) Per inizializzare LSM6DSM:
      i. viene scritto 0x0C nel registro CTRL3_C (0x12) con la funzione *Sensor_IO_Write* per abilitare la modalità a 3 fili dell'SPI.
      ii. Si disabilita l'interfaccia I^2C usando la funzione *LSM6DSM_ACC_GYRO_W_I2C_DISABLE* che setta il bit 2 (I2C_disable) nel registro CTRL4_C (0x13).
      iii. Con la funzione *LSM6DSM_Sensor_IO_ITConfig()* si configura la linea dell'interrupt di LM6DSM sensibile al fronte di salita con priorità 8.
   (b) Per inizializzare LSM303AGR:
      i. si setta il bit 0 (SPI_ENABLE) nel registro CTRL_REG4 (0x23) di LSM303AGR per abilitare la modalità a 3 fili dell'SPI.

2. Inizializzazione giroscopio (LSM6DSM) tramite la funzione *BSP_GYRO_Init*:
   (a) Viene settato il bit 3 (SIM - SPI Serial Interface Mode selection) del registro CTRL3_C (0x12) per abilitare la modalità dell'SPI a 3 fili.
   (b) Si disabilita l'interfaccia I^2C con *LSM6DSM_ACC_GYRO_W_I2C_DISABLE* settando il bit 2 (I2C_disable) nel registro CTRL4_C (0x13).
3. Inizializzazione magnetometro (LSM303AGR) tramite la funzione *BSP_MAGNETO_Init*:
   (a) si disabilita l'interfaccia I^2C settando il bit 5 (I2C_DIS) del registro CFG_REG_C (0x62) in modo da usare solo l'SPI
4. Inizializzazione sensore di umidità (HTS221) con *BSP_HUMIDITY_Init*:
   (a) Viene configurato l'I^2C con clock a 400KHz come specificato nella funzione *I2C_SENSORTILE_Init()*.
   (b) Viene abilitato l'interrupt dell'I^2C con priorità massima.
5. Si inizializza il sensore di temperatura con *BSP_TEMPERATURE_Init*. Si prova ad inizializzare prima il sensore HTS221, nel caso in cui non si riesce si inizializza LPS22HB.
   (a) Per inizializzare HTS221 si guardi il punto 4.
   (b) Per inizializzare LPS22HB:
      i. Con la funzione *BSP_LPS22HB_TEMPERATURE_Init* si setta il bit 0 (SIM - SPI Serial Interface Mode selection) del registro CTRL_REG1 (0x10) per abilitare la modalità SPI a 3 fili.
6. Inizializzazione sensore di pressione con *BSP_PRESSURE_Init*:
   (a) tramite la funzione *Sensor_IO_Write* setta il bit 0 (SIM - SPI Serial Interface Mode selection) del registro CTRL_REG1 (0x10) per scegliere l'SPI con 3 fili.
   (b) Viene disabilitata l'interfaccia I^2C resettando il bit 3 (I2C_DIS) del registro CTRL_REG2 (0x11).

```
static void Init_MEM1_Sensors(void)
{
  [..]
  /* Accelero */
  if (BSP_ACCELERO_Init( ACCELERO_SENSORS_AUTO, &TargetBoardFeatures.HandleAccSensor ) ==
     COMPONENT_OK){
    [...]
  /* Gyro */
  if(BSP_GYRO_Init( GYRO_SENSORS_AUTO, &TargetBoardFeatures.HandleGyroSensor )==
    COMPONENT_OK){
    [...]
  /* Magneto */
  if(BSP_MAGNETO_Init( MAGNETO_SENSORS_AUTO, &TargetBoardFeatures.HandleMagSensor )==
    COMPONENT_OK){
    [...]
  /* Humidity */
  if(BSP_HUMIDITY_Init( HUMIDITY_SENSORS_AUTO, &TargetBoardFeatures.HandleHumSensor )==
    COMPONENT_OK){
    [...]
  /* Temperature1 */
  if(BSP_TEMPERATURE_Init( TEMPERATURE_SENSORS_AUTO, &TargetBoardFeatures.
    HandleTempSensors[TargetBoardFeatures.NumTempSensors] )==COMPONENT_OK){
    [...]
  /* Pressure */
  if(BSP_PRESSURE_Init( PRESSURE_SENSORS_AUTO, &TargetBoardFeatures.HandlePressSensor )==
    COMPONENT_OK){
    [...]
}
```

In questa funzione, infine, vengono create delle strutture del seguente tipo in cui sono riportati i parametri del dispositivo. Di seguito si può osservare l'implementazione nel codice.

```
typedef struct
{
  /* Identity */
  uint8_t who_am_i;
  /* Configuration */
  uint8_t ifType;          /* 0 means I2C, 1 means SPI, etc. */
  uint8_t address;         /* Sensor I2C address (NOTE: Not a unique sensor ID). */
  uint8_t spiDevice;       /* Sensor Chip Select for SPI Bus */
  uint8_t instance;        /* Sensor instance (NOTE: Sensor ID unique only within its class
    ). */
  uint8_t isInitialized; /* Sensor setup done. */
  uint8_t isEnabled;       /* Sensor ON. */
  uint8_t isCombo;         /* Combo sensor (component consists of more sensors). */
  /* Pointer to the Data */
  void *pData;
  /* Pointer to the Virtual Table */
  void *pVTable;
  /* Pointer to the Extended Virtual Table */
  void *pExtVTable;
  } DrvContextTypeDef;
  [..]
}
```

Nel Board Support Package (BSP) si completa la configurazione dei registri che non sono usati nella funzione *Init_MEM1_Sensors()*. Sono ora analizzate le impostazioni dei sensori MEMS:

- Accelerometro LSM303AGR
  - Si setta il bit 7 (BDU - Block Data Update) del registro CTRL_REG4_A (0x23), questo non permette l'aggiornamente dei registri di output finché non sono letti MSB e LSB.
  - Si resettano i bit [1:0] del registro FIFO_CTRL_REG_A (0x2E) per selezionare la modalità bypass in cui la memoria FIFO non è usata.
  - Si setta l'output data rate a 100 Hz scrivendo 0101 nei bit [7:4] del registro CTRL_REG1_A (0x20).
  - Si imposta il fondo scala a ± 2 g scrivendo 00 nei bit [5:4] del registro CTRL_REG4_A (0x23).
- Magnetometro LSM303AGR
  - Si setta il bit 4 del registro CFG_REG_C (0x62) per evitare la lettura di dati errati. Infatti se la richiesta di lettura arriva durante un aggiornamento dei dati di output, si leggono parti alte e basse del stesso registro di output incoerenti.
  - L'output data rate è impostato a 100 Hz perché viene scritto 11 nei bit [2:1] del registro CFG_REG_A_M (0x60).

Si noti che il fondo scala del magnetometro è fisso e vale ± 50 Gauss.

- Accelerometro LSM6DSM
  - Si setta il bit 6 (BDU) del registro CTRL3_C (0x12), questo non consente l'aggiornamento dei registri di output finché non sono stati letti MSB e LSB.
  - Si scrive 0000 nei bit [6:3] del registro FIFO_CTRL5 (0x0A) per disabilitare la FIFO.
  - Si imposta l'output data rate a 104 Hz scrivendo 0100 nei bit [7:4] del registro CTRL1_XL.
  - Il fondo scala è a ± 2 g, infatti nel registro CTRL1_XL (0x10) vengono resettati i bit [3:2].
- Giroscopio LSM6DSM
  - L'output data rate è di 104 Hz, viene scritto 0100 nei bit [7:4] del registro CTRL2_G (0x11).
  - Il fondo scala è di 2000 dps, vengono settati i bit [3:2] del registro CTRL2_G (0x11).

6.1.7 Inizializzazione BlueNRG

Successivamente all'inizializzazione delle periferiche, nel main è settato il nome che assume il dispositivo quando viene fatta una scansione Bluetooth con la funzione *ReCallNodeNameFromMemory()* la quale assegna il nome "AM1V340" alla piattaforma SensorTile.
Dopo aver configurato il nodo Bluetooth, viene inizializzato lo stack BlueNRG con la funzione *Init_BlueNRG_Stack()*. *BNRG_SPI_Init* inizializza la comunicazione SPI con la scheda BlueNRG. In *BNRG_SPI_Init* si crea una struttura dati chiamata *SpiHandle* che contiene informazioni come l'indirizzo del registro dopo leggere o scrivere un dato, il modo in cui viene usata l'interfaccia (bidirezionale), la lunghezza del pacchetto e tutto ciò che è relativo al corretto funzionamento del dispositivo SPI. *BNRG_SPI_Init* richiama infine *HAL_SPI_Init(&SpiHandle)* che inizializza l'SPI secondo quanto specificato nella struttura creata in precedenza.
HCI_Init() inizializza l'interfaccia HCI del Bluetooth, crea infatti la testa di una lista di pacchetti di dati hci e aggiunge in coda i pacchetti liberi.
BlueNRG_RST() resetta il BlueNRG portando al livello logico basso il pin PH0 (D9) di STM32L476, chiamato nel codice GPIO_PIN_0. Si aspettano 5 millisecondi e infine viene riportato al livello alto il pin per riattivare il BlueNRG.
getBlueNRGVersion(&hwVersion, &fwVersion) elabora l'output della funzione hci_le_read_local_version, combinando i campi hci_revision e lmp_pal_subversion per ottenere la variabile *fwVersion* in forma esadecimale secondo la tabella 6.1:

FW version name	hci_version	lmp_pal_subversion	fwVersion
7.1a	0x07	0x0011	0x0711
7.1c	0x07	0x0013	0x0713
7.1e	0x07	0x0015	0x0715
7.2a	0x07	0x0021	0x0721
7.2c	0x07	0x0023	0x0723

Tabella 6.1: Versione del firmware Bluetooth [4]

Viene resettato nuovamente il BlueNRG.
aci_gatt_init() inizializza il server GATT sul dispositivo slave. Inizializza tutti i nodi attivi. Inoltre aggiunge il servizio GATT con le caratteristiche modificate dei servizi. Finché questa funzione non viene usata, il canale GATT non elaborerà alcun comando anche se la connessione è stata aperta. Questo comando deve essere dato prima di utilizzare una qualsiasi delle funzionalità GAP.
L'applicazione può settare un indirizzo random usando il comando *hci_le_set_random_address* dopo ogni reset. Se non viene settato un indirizzo random tramite questa funzione, la generazione è gestita autonomamente dallo stack Bluetooth. Viene chiamata la funzione *aci_gatt_update_char_value* per aggiornare il campo value della caratteristica che contiene il nome della scheda, cioe "AM1V340".
aci_gap_set_auth_requirement setta i requisiti di autenticazione del dispositivo. Questo comando deve essere eseguito quando il dispositivo non è connesso. In particolare viene impostata la protezione MITM (Man In The Middle, indica un attacco informatico in cui qualcuno segretamente ritrasmette o altera la comunicazione tra due parti che credono di comunicare direttamente tra di loro.). L'autorizzazione out of bound (OOB) è disabilitata, quindi i successivi 16 byte di OOB_Data saranno ignorati in ricezione. Durante il processo di accoppiamento la chiave di crittografia da utilizzare è compresa tra 7 e 16 byte, si usa il pin fisso settato pari a "123456". La procedura di connessione è effettuata in bonding, cioè se un dispositivo è collegato a un altro dispositivo, ad esempio un cardiofrequenzimetro e uno smartphone, è possibile crittografare la connessione senza scambiare informazioni sensibili sulla sicurezza.
aci_hal_set_tx_power_level(en_high_power, pa_level) setta il livello di potenza in uscita. Poiché *en_high_power* è uguale a 1 e *pa_level* è uguale a 4, secondo la tabella 6.2 la potenza in trasmissione è di -2.1 dBm.

EN_HIGH_POWER	PA_LEVEL	TX Power Level (dBm)
0	0	-18
0	1	-14.7
0	2	-11.4
0	3	-8.1

0	4	-4.9
0	5	-1.6
0	6	1.7
0	7	5.0
1	0	-15
1	1	-11.7
1	2	-8.4
1	3	-5.1
1	4	-2.1
1	5	1.4
1	6	4.7
1	7	8.0 (default)

Tabella 6.2: Livelli di potenza in trasmissione [3]

```
static void Init_BlueNRG_Stack(void)
{
  [...]
  /* Initialize the BlueNRG SPI driver */
  BNRG_SPI_Init();
  /* Initialize the BlueNRG HCI */
  HCI_Init();
  /* Reset BlueNRG hardware */
  BlueNRG_RST();
  /* get the BlueNRG HW and FW versions */
  getBlueNRGVersion(&hwVersion, &fwVersion);
  [...]
  /*Reset BlueNRG again otherwise it will fail */
  BlueNRG_RST();
  [...]
  ret = aci_gatt_init();
  [...]
  ret = hci_le_set_random_address(bdaddr);
  [...]
  ret = aci_gatt_update_char_value(service_handle, dev_name_char_handle, 0, 7/*strlen(
    BoardName)*/, (uint8_t *)BoardName);
  [...]
  ret = aci_gap_set_auth_requirement(MITM_PROTECTION_REQUIRED,
  OOB_AUTH_DATA_ABSENT, NULL, 7, 16, USE_FIXED_PIN_FOR_PAIRING, 123456, BONDING);
  [...]
  /* Set output power level */
  aci_hal_set_tx_power_level(1,4);
}
```

6.1.8 Creazione servizi e caratteristiche Bluetooth

Successivamente, nel main, tramite la funzione *Init_BlueNRG_Custom_Services()* vengono inizializzati tutti i servizi del protocollo Bluetooth.
Con *Add_HW_SW_ServW2ST_Service()* viene creato il servizio che contiene tutte le caratteristiche relative ai sensori disponibili nella piattaforma SensorTile. Questo servizio viene realizzato con la funzione *aci_gatt_add_serv* che ha in ingresso i seguenti quattro parametri:

Parametro	**Lunghezza**	**Descrizione**
Service_UUID_Type	1 byte	0x01: UUID a 16 bit 0x02 UUID a 128 bit
Service_UUID	2 o 16 byte	UUID a 16 o 128 bit
Service_Type	1 byte	0x01: servizio primario 0x02 servizio secondario

Max_Attribute_Records	1 byte	numero massimo di attributi che possono essere aggiunti nel servizio

Tabella 6.3: Parametri della funzione *aci_gatt_add_serv* [3]

Per creare il servizio in questione sono stati settati i parametri come di seguito:

- Service_UUID_Type = 0x02 → UUID a 128 bit
- Service_UUID = 00000000000111e19ab40002a5d5c51b
- Service_Type = 0x01 → servizio primario
- Max_Attribute_Records = 2+3·max_attr_records = 14 poichè max_attr_record è uguale a 4 → nel servizio possono essere aggiunti 14 attributi

Viene poi riempito il servizio con le varie caratteristiche relative ai sensori con la funzione *aci_gatt_add_char* che prende in ingresso nove parametri come illustrato nella tabella 6.4:

Parametro	**Lunghezza**	**Descrizione**
Service_Handle	2 byte	Handle del servizio in cui va inserita la caratteristica
Char_UUID_Type	1 byte	0x01: UUID a 16 bit 0x02:UUID a 128 bit
Char_UUID	2 o 16 byte	UUID a 16 o 128 bit
Char_Value_Length	2 byte	massima lunghezza del campo "value" della caratteristica
Char_Properties	1 byte	OR bit a bit dei valori delle proprietà della caratteristica (definito nel volume 3, sezione 3.3.3.1 delle specifiche Bluetooth 4.1)
Security_Permissions	1 byte	0x00: nessun permesso di sicurezza 0x01: autorizzazione richiesta per leggere 0x02: autorizzazione richiesta per leggere 0x04: il collegamento deve essere criptato per leggere 0x08: autorizzazione richiesta per scrivere 0x010: autorizzazione richiesta per scrivere 0x20: il collegamento deve essere criptato per scrivere
Gatt_Evt_Mask	1 byte	0x01: l'applicazione verrà notificata quando un client scrive nell'attributo 0x02: l'applicazione verrà notificata quando il server riceve una richiesta di scrittura 0x04: l'applicazione verrà notificata quando una richiesta di lettura di ogni tipo viene ricevuta
Encryption_Key_Size	1 byte	Minima lunghezza della chiave di criptazione per l'attributo. Range valido: da 7 a 16
isVariable	1 byte	0x00: l'attributo ha un campo value di lunghezza fissa 0x01: l'attributo ha un campo value di lunghezza variabile

Tabella 6.4: Parametri della funzione *aci_gatt_add_char* [3]

L'esempio riportato di seguito è relativo alla caratteristica del pitch e roll.

- Service_Handle = 12
- Char_UUID_Type = 0x02 → UUID a 128 bit
- Char_UUID = 00ee0000000111e1ac360002a5d5c51b
- Char_Value_Length = 2+2+2 → 2 byte per il timestamp, 2 byte rispettivamente per il valore del pitch e del roll
- Char_Properties = 0x10
- Security_Permissions = 0x00 → nessun permesso di sicurezza
- Gatt_Evt_Mask = 0x04 → l'applicazione verrà notificata quando una richiesta di lettura di ogni tipo viene ricevuta
- Encryption_Key_Size = 16
- isVariable = 0 → campo value di lunghezza fissa

Si noti che nel firmware ogni volta che viene aggiunta una caratteristica, viene aumentato l'handle relativo di due unità.
In modo del tutto analogo le funzioni *Add_ConsoleW2ST_Service()* e *Add_ConfigW2ST_Service()* aggiungono rispettivamente i servizi di debug e di configurazione e le relative caratteristiche. Il primo ha UUID 00000000-000E-11e1-9ab4-0002a5d5c51b e contiene due caratteristichecon UUID:

- 00000001-000E-11e1-ac36-0002a5d5c51b: usata per mandare stringhe di comando alla scheda e notificare l'utente riguardo il risultato
- 00000002-000E-11e1-ac36-0002a5d5c51b: usata per notificare l'utente riguardo un errore

Il secondo ha UUID 00000000-000F-11e1-9ab4-0002a5d5c51b e contiene la seguente caratteristica:

- 00000002-000F-11e1-ac36-0002a5d5c51b: usata per mandare comandi o dati ad una specifica feature.

```
static void Init_BlueNRG_Custom_Services(void)
{
  int ret;
  ret = Add_HW_SW_ServW2ST_Service();
  [...]
  ret = Add_ConsoleW2ST_Service();
  [..]
  ret = Add_ConfigW2ST_Service();
  [...]
}
```

6.1.9 Lettura output data rate dell'accelerometro

In seguito viene letto l'output data rate dell'accelerometro con la funzione *InitHWFeatures()*. Passando l'handle del dispositivo e il puntatore al dato dove è contenuto l'ODR a *BSP_ACCELERO_Get_ODR* viene letta la frequenza con cui l'accelerometro aggiorna i suoi dati in output. Se la lettura ha avuto successo la funzione ritorna COMPONENT_OK (0), altrimenti COMPONENT_ERROR (1).

```
void InitHWFeatures(void){
  /* Read the Default Output Data Rate for Accelerometer */
  BSP_ACCELERO_Get_ODR(TargetBoardFeatures.HandleAccSensor,&DefaultAccODR);
}
```

6.1.10 Settaggio fondo scala dell'accelerometro

Con la funzione *Set2GAccelerometerFullScale()* viene settato il full scale dell'accelerometro a ± 2 G. *BSP_ACCELERO_Set_FS_Value* riceve come parametri l'hande del dispositivo e il valore del full scale, in questo caso pari a 2.
BSP_ACCELERO_Get_Sensitivity ha come parametri l'handle dell'accelerometro e il puntatore dove il valore di sensitività è scritto, espresso in mg/LSB. Questo valore viene infine moltiplicato per la costante *FROM_MG_TO_G* definita nella libreria sensor_service.h e pari a 0.001, infatti per passare dai millesimi di G a G è necessario dividere per 1000 o moltiplicare per 0.001.

```
void Set2GAccelerometerFullScale(void)
{
  /* Set Full Scale to +/-2g */
  BSP_ACCELERO_Set_FS_Value(TargetBoardFeatures.HandleAccSensor,2.0f);
  /* Read the Acc Sensitivity */
  BSP_ACCELERO_Get_Sensitivity(TargetBoardFeatures.HandleAccSensor,&sensitivity);
  sensitivity_Mul = sensitivity* ((float) FROM_MG_TO_G);
}
```

6.1.11 Lettura sensitività dell'accelerometro

Viene letta una ulteriore volta la sensitività usando sempre la funzione *BSP_ACCELERO_Get_Sensitivity*, per poi immagazzinare il valore nella variabile *sensitivity_Mul*.

```
/* Read the Acc Sensitivity */
BSP_ACCELERO_Get_Sensitivity(TargetBoardFeatures.HandleAccSensor,&sensitivity);
sensitivity_Mul = sensitivity * ((float) FROM_MG_TO_G);
```

6.1.12 Inizializzazione timer Bluetooth

InitTimers() inizializza i timer necessari per mandare le informazioni tramite Bluetooth. I timer servono per i dati di accelerometro, giroscopio, magnetometro, sensor fusion, sensori ambientali e microfono. Per settare i vari contatori vengono definiti i relativi prescaler calcolati come

$$prescaler = \frac{F_{sistema}}{F_{desiderata}} - 1$$

dove $F_{sistema} = 4\ MHz$. *HAL_TIM_Base_Init* inizializza la base dei tempi dei timer secondo i parametri definiti in TIM_HandleTypeDef e inizializza l'handle associato. Di seguito sono elencate le impostazioni dei timer con i rispettivi prescaler, frequenza e valore da cui iniziano a contare.

- Timer 1 - Sensori di movimento

Prescaler	Frequenza	Periodo
400	10 KHz	65535

Tabella 6.5: Impostazioni Timer 1

- Timer 4 - Sensori ambientali

Prescaler	Frequenza	Periodo
2000	2 KHz	999

Tabella 6.6: Impostazioni Timer 4

- Timer 5 - Microfono

Prescaler	Frequenza	Periodo
400	10 KHz	499

Tabella 6.7: Impostazioni Timer 5

Con la funzione *HAL_TIM_OC_Init* si inizializzano i registri di Output Compare secondo gli specifici parametri definiti in TIM_HandleTypeDef e inizializza l'handle associato. Successivamente vengono specificati i valori dell'impulso che deve essere caricato nel registro di capture compare, questi valori sono definiti nella liberia main.h, secondo la seguente formula

$$pulse = \frac{F_{timer}}{F_{trasmissione}}$$

dove $F_{trasmissione}$ è la frequenza con cui vengono mandati i dati Bluetooth.
HAL_TIM_OC_ConfigChannel configura i canali dell'Output Compare dei timer in accordo con i parametri settati in TIM_OC_InitTypeDef. Di seguiti sono indicati i valori degli impulsi per il modulo capture compare, la frequenza del timer e quella di trasmissione per diversi dati.

- Timer 1 - Sensori di movimento

Dato	**Canale**	**Pulse**	**Frequenza timer**	**Frequenza trasmissione**
Motion FX - Sensor Fusion	Channel 1	100	10 KHz	100 Hz
Motion CP - Carry Position	Channel 2	200	10 KHz	50 Hz
Motion AR - Activity Recognition	Channel 3	625	10 KHz	16 Hz
Accelerometro Giroscopio Magnetometro Pitch e roll	Channel 4	100	10 KHz	100 Hz

Tabella 6.8: Impostazioni Capture Compare Timer 1

```
static void InitTimers(void)
{
  [...]
  /* Set TIM4 instance ( Environmental ) */
  [...]
  if(HAL_TIM_Base_Init(&TimEnvHandle) != HAL_OK) {
    [...]
  /* Set TIM1 instance ( Motion ) */
    [...]
  if(HAL_TIM_OC_Init(&TimCCHandle) != HAL_OK)
    [...]
  /* Code for MotionFX integration - Start Section */
  /* Output Compare Toggle Mode configuration: Channel1 */
  sConfig.Pulse = DEFAULT_uhCCR1_Val;
  if(HAL_TIM_OC_ConfigChannel(&TimCCHandle, &sConfig, TIM_CHANNEL_1) != HAL_OK)
    [...]
  /* Output Compare Toggle Mode configuration: Channel2 */
  sConfig.Pulse = DEFAULT_uhCCR2_Val;
  if(HAL_TIM_OC_ConfigChannel(&TimCCHandle, &sConfig, TIM_CHANNEL_2) != HAL_OK)
    [...]
  /* Code for MotionAR integration - Start Section */
  /* Output Compare Toggle Mode configuration: Channel3 */
  sConfig.Pulse = DEFAULT_uhCCR3_Val;
  if(HAL_TIM_OC_ConfigChannel(&TimCCHandle, &sConfig, TIM_CHANNEL_3) != HAL_OK)
    [...]
  /* Output Compare Toggle Mode configuration: Channel4 */
  sConfig.Pulse = DEFAULT_uhCCR4_Val;
  if(HAL_TIM_OC_ConfigChannel(&TimCCHandle, &sConfig, TIM_CHANNEL_4) != HAL_OK)
    [...]
  /* Set TIM5 instance ( Mic ) */
    [...]
  if(HAL_TIM_Base_Init(&TimAudioDataHandle) != HAL_OK) {
    [...]
```

```
  }
}
```

6.1.13 Conteggio dei tick del timer

Una volta terminate tutte le inizializzazioni, viene salvato nella variabile *ActivityTimeout_StartTime* il numero di tick contati fino a quel momento, convertito in millisecondi, usando la funzione *HAL_GetTick()*.

```
ActivityTimeout_StartTime = HAL_GetTick();
```

HAL_GetTick() ritorna la variabile *uwTick* che si incrementa ogni millisecondo.

```
__weak uint32_t HAL_GetTick(void)
{
  return uwTick;
}
```

6.2 Main loop

6.2.1 Lampeggio del led

Se nessun client si è connesso al Bluetooth, inizia il lampeggio del led tramite le due funzioni *LedOnTargetPlatform()* e *LedOffTargetPlatform()* che accendono e spengono il LED1.

```
/* Led Blinking when there is not a client connected */
if(!connected){
  if(!TargetBoardFeatures.LedStatus) {
    if(!(HAL_GetTick()&0x3FF)) {
      LedOnTargetPlatform();
    }
  }
  else {
    if(!(HAL_GetTick()&0x3F)) {
      LedOffTargetPlatform();
    }
  }
}
```

6.2.2 Controllo della connessione

Successivamente viene controllata la differenza temporale tra i tick contati alla fine delle inizializzazioni, salvata nella variabile *ActivityTimeout_StartTime*, e l'attuale valore dei tick contenuto nella variabile *ActivityTimeout_CurrTime* ed ottenuto usando la funzione già vista *HAL_GetTick()*. Se è abilitata la possibilità di mandare in shut down il dispositivo, cioè la variabile *ENABLE_SHUT_DOWN_MODE* è uguale a 1 e la differenza temporale è maggiore del valore *RANGE_TIME_WITHOUT_CONNECTED* definito nel main.h e pari a 1200000 millisecondi, cioè 20 minuti, la MCU va in power down tramite la funzione *MCU_PowerSave()*.

```
if((!connected) && (ENABLE_SHUT_DOWN_MODE))
{
  ActivityTimeout_CurrTime = HAL_GetTick();
  if( ActivityTimeout_CurrTime - ActivityTimeout_StartTime > RANGE_TIME_WITHOUT_CONNECTED
     )
  {
  MCU_PowerSave();
  }
}
else
{
  ActivityTimeout_StartTime = HAL_GetTick();
}
#endif
```

6.2.3 Modalità power save

Verrà ora analizzata la funzione *MCU_PowerSave()* che manda la MCU in shut down mode. I passaggi da compiere per mandare il microcontrollore in questa modalità sono elencati di seguito.

1. Viene controllata la variabile WakeupSource, di default settata pari a ACC_DOUBLE_TAP.
   (a) Se WakeupSource è uguale a ACC_DOUBLE_TAP, definita pari a 0x40 nel main.h, viene abilitata la funzionalità dell'accelerometro LSM6DSM che rileva un doppio tocco sulla scheda.
   (b) Se WakeupSource è uguale a ACC_WAKE_UP, definita pari a 0x80 nel main.h, viene abilitato il rilevamento del risveglio, funzionalità disponibile solo per l'accelerometro LSM6DSM.
2. Vengono disabilitati giroscopio, magnetometro, il sensore di pressione e di temperatura e quello di umidità.
3. Tramite la funzione *aci_gap_set_non_discoverable()* viene settato il dispositivo Bluetooth in modalità non rilevabile. Questo comando disabilita i messaggi di advertising e porta il dispositivo in standby.
4. Viene portato ad uno il bit 0 (GPIOAEN) del registro RCC_AHB2ENR per abilitare il clock sulla porta A.
5. Vengono disabilitati le periferiche SPI e I^2C e i relativi interrupt.
6. Viene portato a zero il bit 24 (RCC_APB2ENR_DFSDM1EN) del registro RCC_APB2ENR per disabilitare il clock DFSDM (Digital Filter For Sigma-Delta Modulators).
7. Vengono disabilitati i timer 1, 2, 4 e 5 con la funzione *DeinitTimers()*.
8. Vengono disabilitate tutte le porte di GPIO tranne la porta A perché il pin di wake up è connesso a PA2. Questo passaggio viene realizzato richiamando *HAL_GPIO_Init* che prende come parametri GPIOx, dove x può essere A..H, che indica la periferica GPIO da disabilitare e il puntatore alla struttura GPIO_InitTypeDef che contiene le informazioni di configurazione per la specifica periferica.
9. Viene sospeso l'incremento dei tick usando la funzione *HAL_SuspendTick()*.
10. Viene disabilitato il Real Time Clock portando a zero il bit 15 (RTCEN) del registro RCC_BDCR.
11. Vengono settati a zero i bit 0, 1, 2, 3, 4, 5, 6, 7, 9 del registro RCC_CIER per disabilitare ogni sorgente di interrupt dei timer.
12. Viene fatto un clear delle sorgenti di interrupt esterno, tranne quello proveniente da PA2 in cui è connesso il pin di wake up dell'accelerometro.
    (a) Vengono portati a zero tutti i bit del registro EXTI_IMR1 tranne il 2 che rimane inalterato. Tutte le linee di interrupt esterno sono disabilitate ad eccetto della linea 2.
    (b) Vengono portati a zero tutti i bit del registro EXTI_IMR2 per disabilitare le linee 32...39 che possono generare interrupt esterno.
    (c) Vengono portati a zero tutti i bit del registro EXTI_RTSR1 tranne il 2 che rimane inalterato. Per tutte le linee di interrupt, ad eccezione della linea 2, vengono disabilitati i trigger per eventi sul fronte di salita.
13. Si disabilitano tutte le sorgenti di wake up e viene fatto un clear del flag di wake up.
14. Si abilita il wake up solamente sul pin PA2.
15. Viene settato ad uno il registro RTC_BKP31R per indicare al sistema, una volta risvegliato, che si era entrati in modalità shut down.
16. Con la funzione *HAL_PWREx_EnterSHUTDOWNMode()* si entra definitivamente in modalità shut down portando ad 1 il bit 2 (LPMS) del registro PWR_CR1 e settando il bit "SLEEPDEEP" del registro System Control Register (SCR).

6.2.4 Inizializzazione sensor fusion

Se il microcontrollore non è entrato in shut down mode e la variabile set_connectable è diversa da zero, si entra in una parte di codice in cui vengono inizializzate le funzionalità del sensor fusion con *MotionFX_manager_init()*. Si definiscono le orientazioni di accelerometro, giroscopio e magnetometro, successivamente tramite *MotionFX_manager_start_9X()* si abilita la funzionalità chiamata a 9 assi del sensor fusion, cioè si fondono insieme i dati dei tre assi dei tre sensori sopra indicati. Alla fine di queste inizializzazioni si esegue una calibrazione del magnetometro con *MagCalibTest()*. Si inizializzano anche altre librerie come, ad esempio, l'activity recognition e il BlueVoice, ma non sono trattate poiché queste funzionalità non sono state usate durante il progetto.

```
if(set_connectable){
  /* Code for MotionFX integration - Start Section */
  /* Initialize MotionFX library */
  if(TargetBoardFeatures.MotionFXIsInitalized==0)
  {
    MotionFX_manager_init();
    MotionFX_manager_start_9X();
    /* Enable magnetometer calibration */
    MagCalibTest();
  }
  [...]
```

6.2.5 Dispositivo in modalità collegabile

La funzione *setConnectable()* mette il dispositivo in modalità collegabile e aggiorna i dati di advertising. Si crea un vettore di 26 elementi chiamato *manuf_data* che contiene informazioni riguardo il Bluetooth come la potenza di trasmissione, il nome del dispositivo e il mac address. Gli indici 16, 17, 18 e 19 del vettore sono dedicati ai sensori e alle funzionalità della scheda, se questi sono abilitati viene scritto un numero per definire che sono in funzione. Sono inizialmente settati a zero tranne per l'indice 17 in cui viene scritto 0xE0 ed è dedicato all'accelerometro, giroscopio e magnetometro, sensori ambientali e informazioni sulla batteria.
La funzione *hci_le_set_scan_resp_data(0,NULL)* disabilita i response data, cioè il pacchetto di risposta inviato dalla periferica alla ricezione di richieste di scansione.
aci_gap_set_discoverable riceve undici parametri per settare il dispositivo in modalità rilevabile come indicato nella tabella 6.9. La scheda rimane in tale condizione finché non viene chiamato il comando *Aci_Gap_Set_Non_Discoverable*. I parametri *Adv_Interval_Min* e *Adv_Interval_Max* sono opzionali, se sono entrambi settati a zero, il GAP usa i valori di default per gli intervalli di advertising.

Parametro	Lunghezza	Descrizione
Advertising_Event_Type	1 byte	0x00: connessione e scansione abilitata 0x02: scansione abilitata, connessione disabilita 0x03: non collegabile e response data disabilitati
Adv_Interval_Min	2 byte	Intervallo di advertising minimo. Range: 0x0020 - 0x4000 Default: N = 0x0800 (1.28 s) Tempo = N * 0.625 ms Range tempo: 20 ms - 10.24 s.
Adv_Interval_Max	2 byte	Intervallo di advertising massimo. Range: 0x0020 - 0x4000 Default: N = 0x0800 (1.28 s) Tempo = N * 0.625 ms Range tempo: 20 ms - 10.24 s.
Address_Type	1 byte	0x00: indirizzo pubblico 0x01: indirizzo random

Adv_Filter_Policy	1 byte	0x00: Richiesta di scansione e di connessione permessa ad ogni dispositivo. 0x01: Richiesta di scansione permessa solo ai dispositivi in white list. Richiesta di connessione permessa a tutti. 0x02: Richiesta di scansione permessa a tutti. Richiesta di connessione permessa solo ai dispositivi in white list. 0x03: Richiesta di scansione e di connessione permessa solo ai dispositivi in white list.
Local_Name_Length	1 byte	Lunghezza del nome del disposivo espressa in numero di byte. Se è settata a 0x00, il parametro Local_Name non deve essere usato.
Local_Name	0-N byte	Nome del dispositivo
Service_UUID_Length	1 byte	Lunghezza dell'UUID del servizio di advertising.
Service_UUID_List	0-N byte	Lista degli UUID degli advertising data come specificato nel volume 3, sezione 11.1.1 delle specifiche GAP.
Slave_Conn_Interval_Min	2 byte	Intervallo di connessione minimo dello slave. Il connection interval è definito nel seguente modo: connIntervalmin = Slave_Conn_Interval_Min * 1.25ms Slave_Conn_Interval_Min range: 0x0006 - 0x0C80.
Slave_Conn_Interval_Max	2 byte	Intervallo di connessione massimo dello slave. Il connection interval è definito nel seguente modo: connIntervalmax = Slave_Conn_Interval_Max * 1.25ms Slave_Conn_Interval_Max range: 0x0006 - 0x0C80.

Tabella 6.9: Parametri della funzione *aci_gap_set_discoverable* [3]

I parametri sono settati nel seguente modo:

- Advertising_Event_Type = 0 → connessione e scansione abilitata.
- Adv_Interval_Min = 0 → 20 ms
- Adv_Interval_Max = 0 → 20 ms
- Address_Type = 1 → indirizzo random

- Adv_Filter_Policy = 0 → richiesta di scansione e connessione permessa ad ogni dispositivo
- Local_Name_Length = sizeof(localname)
- Local_Name = localname = "AM1V340"
- Service_UUID_Length = 0 → Nessun servizio disponibile in advertising.
- Service_UUID_List = NULL → Nessun servizio disponibile in advertising.
- Slave_Conn_Interval_Min = 0 → Lo struct che definisce il range del connection interval non è aggiunto negli advertising data.
- Slave_Conn_Interval_Max = 0 → Lo struct che definisce il range del connection interval non è aggiunto negli advertising data.

Il comando *aci_gap_update_adv_data* riceve come parametri la variabile che contiene i dati di advertising e la sua lunghezza. È usato per aggiornare i dati di advertising, se questo è lungo più di 31 byte la funzione è respinta e sono tenuti in considerazione i vecchi dati.

```
void setConnectable(void)
{
  char local_name[8] = {AD_TYPE_COMPLETE_LOCAL_NAME,NodeName[1],NodeName[2],NodeName[3],
    NodeName[4],NodeName[5],NodeName[6],NodeName[7]};
  uint8_t manuf_data[26] = {
    2,0x0A,0x00 /* 0 dBm */, // Trasmission Power
    //8,0x09,NAME_BLUEMS, // Complete Name
    8,0x09,NodeName[1],NodeName[2],NodeName[3],NodeName[4],NodeName[5],NodeName[6],
    NodeName[7], // Complete Name
    13,0xFF,0x01/*SKD version */,
    0x02,
    0x00 /* AudioSyncAudioData */,
    0xE0 /* ACC + Gyro + Mag + Environmental + Battery Info */,
    0x00 /*  Hardware Events + MotionFX + SD Card Logging */,
    0x00, /*  */
    0x00, /* BLE MAC start */
    0x00,
    0x00,
    0x00,
    0x00,
    0x00, /* BLE MAC stop */
  };
  [...]
  /* disable scan response */
  hci_le_set_scan_resp_data(0,NULL);
  aci_gap_set_discoverable(ADV_IND, 0, 0, STATIC_RANDOM_ADDR, NO_WHITE_LIST_USE, sizeof(
    local_name), local_name, 0, NULL, 0, 0);

  /* Send Advertising data */
  aci_gap_update_adv_data(26, manuf_data);
}
```

6.2.6 Trasmissione dei dati

Vengono successivamente mandati i dati che l'utente vuole leggere dal Bluetooth. Verrà analizzata la funzione che trasmette i dati di pitch e roll, poiché tutte le funzioni che implementano la trasmissione Bluetooth sono realizzate in modo analogo. La variabile che permette la chiamata alla funzione in questione è *SendPitchRoll* che viene settata ad uno quando il client abilita le notifiche per questa caratteristica.

```
if(SendPitchRoll) {
  SendPitchRoll=0;
  SendPitchRollData();
}
```

6.2.7 Calcolo del pitch e roll

Il rollio rappresenta l'oscillazione di un corpo intorno al proprio asse longitudinale, mentre il beccheggio rappresenta l'oscillazione intorno al proprio asse trasversale. Ipotizzando che gli assi X e Y del corpo si trovino sul piano orizzontale e l'asse Z sia perpendicolare a questo piano, una rotazione intorno all'asse X forma un angolo tra asse Y e il piano orizzontale. Questo angolo che si forma a causa della rotazione di X si chiama angolo di rollio. Una rotazione intorno all'asse Y forma un angolo tra asse X e il piano orizzontale. Questo angolo che si forma a causa della rotazione di Y si chiama angolo di beccheggio. È possibile calcolare questi angoli con i valori dei tre assi dell'accelerometro. Le formule che definiscono il rollio e il beccheggio sono rispettivamente:

$$Roll = \arctan\left(\frac{-a_x}{a_z}\right)$$

$$Pitch = \arctan\left(\frac{a_y}{\sqrt{a_x^2 + a_z^2}}\right)$$

Poiché la funzione arcotangente è definita solo tra $-\frac{\pi}{2}$ e $\frac{\pi}{2}$, nel programma viene usato il comando *atan2f* che ritorna l'arcotangente di un rapporto nel range $[-\pi, +\pi]$. Anche i dati di velocità angolare del giroscopio possono essere integrati per ricavare beccheggio e rollio. Le formule relative sono:

$$Roll(t) = Roll(t_0) + \int_{t_0}^{t} \omega_y(\tau) d\omega$$

$$Pitch(t) = Pitch(t_0) + \int_{t_0}^{t} \omega_x(\tau) d\omega$$

Considerando il dominio a tempo discreto ed indicando la differenza tra due istanti di campionamento come $T[n] - T[n-1]$ si può integrare con il metodo dei rettangoli nel seguente modo:

$$Roll[n] = Roll[n-1] + \omega_y[n] \cdot (T[n] - T[n-1])$$

$$Pitch[n] = Pitch[n-1] + \omega_x[n] \cdot (T[n] - T[n-1])$$

Il giroscopio tende ad avere un drift, una deriva, che nel tempo gli fa rilevare valori completamente sbagliati. L'accelerometro, invece, pur dando un valore fedele quando l'accelerazione è progressiva, soffre moltissimo le vibrazioni, durante le quali restituisce valori dell'angolo sbagliati ed eccessivi. Per questo motivo si è implementato un filtro complementare che combina i dati dai due sensori. Il progetto è incentrato sulla realizzazione di questo filtro sfruttando la capacità computazionale del microcontrollore e sulla valutazione dei consumi che comporta l'implementazione di questo sulla scheda. I filtri complementari gestiscono al loro interno filtri passa basso e passa alto contemporaneamente. I filtri passa basso filtrano i segnali con frequenza elevata (ad esempio l'accelerometro nel caso delle vibrazioni) e filtri passa alto che filtrano i segnali con bassa frequenza (ad esempio il drift del giroscopio). Combinando opportunamente questi filtri si ottiene un segnale più fedele alla realtà. La formula del filtro per un angolo generico è la seguente:

$$\theta[n] = \alpha \cdot (\theta[n-1] + \omega[n] \cdot (T[n] - T[n-1])) + (1 - \alpha) \cdot \theta_{acc}[n]$$

con θ_{acc} viene indicato il valore dell'angolo ricavato con i dati dell'accelerometro. Per un filtro passa basso del primo ordine si ricava α dalla formula $\alpha = \frac{\Delta t}{\tau + \Delta t}$ dove $\tau = 0.5s$ e $\Delta t = \frac{1}{104}s$ dato che l'accelerometro ha un output data rate di 104 Hz. Con questi valori si ottiene $\alpha \cong 0.018$.
Per un filtro passa alto la formula per ricavare α è $\alpha = \frac{\tau}{\tau + \Delta t} \cong 0.981$. Dal fatto che le costanti di tempo della componente passa alto e passa basso siano uguali si deduce che tutte le componenti in frequenza sono pesate allo stesso modo, dato che dove un filtro inizia a tagliare subentra l'altro.
Il problema nel quale si incorre applicando questo genere di filtro è che la transizione dell'angolo di rollio da $-\pi$ a π e viceversa non avviene in maniera netta come dovrebbe. Dalla formula del filtro si può notare che il rollio potrebbe eccedere i valori compresi tra $-\pi$ e π. Infatti il rollio calcolato dalle accelerazioni, tramite arcotangente, è compreso tra $-\pi$ e π ma quello calcolato dalle velocità angolari, tramite integrazione, potrebbe eccedere i valori limite. Per questi motivi il filtro complementare è stato

usato per calcolare le componenti sinusoidali e cosinusoidali di rollio e beccheggio, dalle quali sono stati ricavati gli angoli stessi:

$$\sin(Roll[n]) = \alpha \cdot \sin(Roll[n-1] + \omega_y \cdot (T[n] - T[n-1])) + (1-\alpha) \cdot (-a_x[n])$$

$$\cos(Roll[n]) = \alpha \cdot \cos(Roll[n-1] + \omega_y \cdot (T[n] - T[n-1])) + (1-\alpha) \cdot a_z[n]$$

$$\sin(Pitch[n]) = \alpha \cdot \sin(Pitch[n-1] + \omega_x \cdot (T[n] - T[n-1])) + (1-\alpha) \cdot a_y[n]$$

$$\cos(Pitch[n]) = \alpha \cdot \cos(Pitch[n-1] + \omega_x \cdot (T[n] - T[n-1])) + (1-\alpha) \cdot (\sqrt{a_x[n]^2 + a_z[n]^2})$$

$$Roll[n] = \arctan\left(\frac{\sin(Roll[n])}{\cos(Roll[n])}\right)$$

$$Pitch[n] = \arctan\left(\frac{\sin(Pitch[n])}{\cos(Pitch[n])}\right)$$

dove $\alpha = 0.981$.
Per ottenere i valori dell'accelerometro e del giroscopio si usa la funzione *BSP_ACCELERO_Get_Axes* e *BSP_GYRO_Get_Axes* che hanno come parametri i puntatori alla struct in cui sono memorizzati i tre assi. Le due funzioni restituiscono l'accelerazione espressa in mg e la velocità angolare espressa in mdps. Dato che non si può scrivere un float nel campo value della caratteristica, i valori trovati vengono moltiplicati per 8192 (2^{13}) e restituiti come un intero.

```
static void SendPitchRollData(void)
{
  SensorAxes_t ACC_Value;
  SensorAxes_t GYR_Value;
  SensorAxes_t MAG_Value;
  float deltaT = 0.01;     //10 ms
  static float pitch = 0; //pitch(t0) = 0
  static float roll = 0;  //roll(t0) = 0
  float cost = 0.981;

  /* Read the Acc values */
  BSP_ACCELERO_Get_Axes(TargetBoardFeatures.HandleAccSensor,&ACC_Value);

  /* Read the Gyro values */
  BSP_GYRO_Get_Axes(TargetBoardFeatures.HandleGyroSensor,&GYR_Value);

  // casting necessario per il corretto funzionamento della funzione atanf
  float accx = (float) ACC_Value.AXIS_X;
  float accy = (float) ACC_Value.AXIS_Y;
  float accz = (float) ACC_Value.AXIS_Z;

  // casting e ridimensionamento delle variabili
  float gyrx = (float) GYR_Value.AXIS_X;
  float gyry = (float) GYR_Value.AXIS_Y;

  // componenti sinusoidali e cosinusoidali del beccheggio
  float sin_pitch = cost * sinf(pitch + ( gyry * deltaT)) + (1 - cost) * accy;
  float cos_pitch = cost * cosf(pitch + ( gyry * deltaT)) + (1 - cost) * sqrt(accx * accx
     + accz * accz);

  // componenti sinusoidali e cosinusoidali del rollio
  float sin_roll = cost * sinf(roll + (gyrx * deltaT)) + (1 - cost) * (-accx);
  float cos_roll = cost * cosf(roll + (gyrx * deltaT)) + (1 - cost) * accz;
  //calcolo di rollio e beccheggio a partire dalle relative componenti
  //sinusoidali e  cosinusoidali, moltiplicato per 8192 perché
  //nel campo value della caratterisitca è ammesso solo un int
  pitch = atan2f (sin_pitch, cos_pitch) * 8192;
  roll = atan2f (sin_roll, cos_roll) * 8192;

  //aggiorno la caratteristica del pitch e roll
  PitchRoll_Update(&pitch, &roll);
}
```

6.2.8 Aggiornamento caratteristica pitch e roll

Per aggiornare la caratteristica che contiene i valori di pitch e roll viene usata la funzione *PitchRoll_Update*. Si crea un buffer di 6 byte che contiene tre dati da due byte ciascuno: il timestamp, il beccheggio e il rollio. Per ottenere il timestamp si usa *HAL_GetTick()* che ritorna i tick, espressi in millisecondi, contati dal timer. Questo valore viene poi shiftato verso destra di 3 bit (>>3), ciò implica che il timestamp si incrementa di una unità ogni 8 millisecondi.
Allo scopo di aggiornare la caratteristica ogni dieci millisecondi si è utilizzata la funzione *Aci_L2CAP_Connection_Parameter_Update_Request*, presente nel codice "sensor_service.c", che invia una richiesta di aggiornamento dei parametri di connessione L2CAP dallo slave al master. I parametri che la funzione riceve in ingresso sono i seguenti:

Parametro	**Lunghezza**	**Descrizione**
Connection_handle	2 byte	Handle della connessione a cui è richiesto l'aggiornamento dei parametri
Interval_min	2 byte	Minimo valore per l'intervallo di connessione, definito come: connIntervalMin = Interval Min · 1.25 ms
Interval_max	2 byte	Massimo valore per l'intervallo di connessione, definito come: connIntervalMax = Interval Max · 1.25 ms
Slave_latency	2 byte	Definisce il parametro di latenza dello slave (come numero di connessioni LL)
Timeout_multiplier	2 byte	Definisce il timeout della connessione nella seguente maniera: Timeout Multiplier · 10 ms

Tabella 6.10: Parametri della funzione *Aci_L2CAP_Connection_Parameter_Update_Request* [3]

I parametri sono settati nel seguente modo:

- Connection_handle = connection_handle → la variabie contiene l'handle della connessione
- Interval_min = 8 → Minimo intervallo di connessione: $1.25 \cdot 8 = 10ms$
- Interval_max = 8 → Massimo intervallo di connessione: $1.25 \cdot 8 = 10ms$
- Slave_latency = 0 → Latenza dello slave nulla
- Timeout_multiplier = 400 → $Timeout = Timeout\_multiplier \cdot 10ms = 400 \cdot 10ms = 4s$

Per aggiornare il campo value della caratteristica si usa *aci_gatt_update_char_value*, i cui parametri sono descritti nella tabella 6.11:

Parametro	**Lunghezza**	**Descrizione**
Serv_Handle	2 byte	Handle del servizio a cui appartiene la caratteristica
Char_Handle	2 byte	Handle della caratteristica
Val_Offset	1 byte	Offset da cui deve essere aggiornato il valore dell'attributo
Char_Value_Length	1 byte	Lunghezza del campo value in byte
Char_Value	1-N byte	Contenuto del campo value

Tabella 6.11: Parametri della funzione *aci_gatt_update_char_value* [3]

Per spiegare i parametri si prende come esempio la caratteristica del pitch e roll:

- Serv_Handle = → HWServW2STHandle = 12
- Char_Handle = → PitchRollCharHandle = 19
- Val_Offset = 0 → nessun offset

- Char_Value_Length = 6 → il valore da scrivere è lungo 6 byte.
- Char_Value = buff → il contenuto del campo value è un buffer di 6 byte, in cui sono contenuti tre dati da due byte: timestamp, pitch e roll.

```
tBleStatus PitchRoll_Update(float *pitchptr, float *rollptr)
{
  tBleStatus ret;
  uint8_t buff[PitchRollValueLength];

  int pitch = *pitchptr;
  int roll = *rollptr;

  STORE_LE_16(buff    ,(HAL_GetTick()>>3));
  STORE_LE_16(buff+2, pitch);
  STORE_LE_16(buff+4, roll);

  /* Apply Magneto calibration */
  //AXIS_X = Mag->AXIS_X - MAG_Offset.AXIS_X;
  //AXIS_Y = Mag->AXIS_Y - MAG_Offset.AXIS_Y;
  //AXIS_Z = Mag->AXIS_Z - MAG_Offset.AXIS_Z;

  ret = ACI_GATT_UPDATE_CHAR_VALUE(HWServW2STHandle, PitchRollCharHandle, 0,
    PitchRollValueLength, buff);
  [...]
}
```

Capitolo 7

Analisi delle prestazioni

7.1 Filtro complementare

I seguenti grafici illustrano il confronto tra i dati di beccheggio e rollio filtrati, in blu, e i dati ricavati semplicemente dalle formule in cui si usano i valori degli assi dell'accelerometro, in rosso. Si può notare come per i dati in cui è stato usato il filtro complementare le variazioni rapide sono più distinte, in quanto il giroscopio riesce a compensare le scarse capacità dell'accelerometro nel breve periodo.

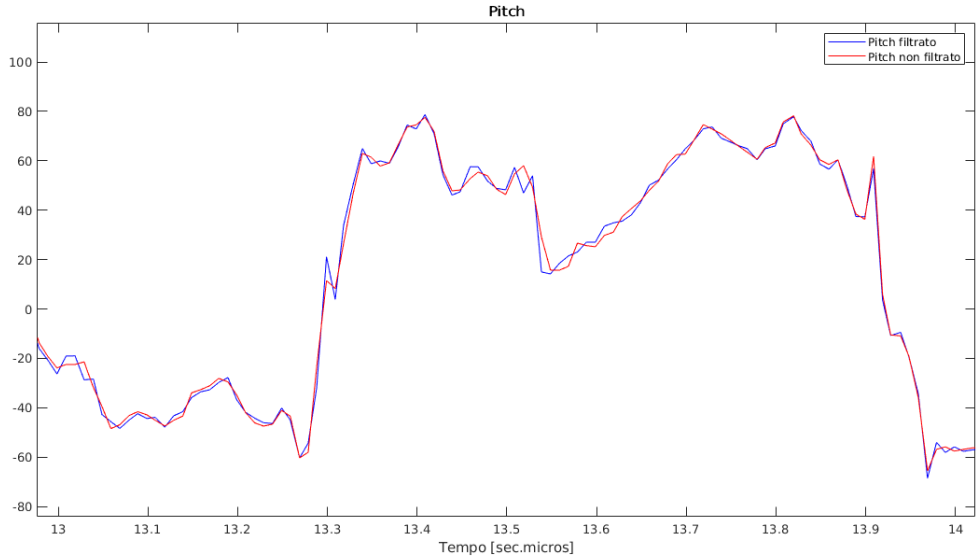


Figura 7.1: Pitch

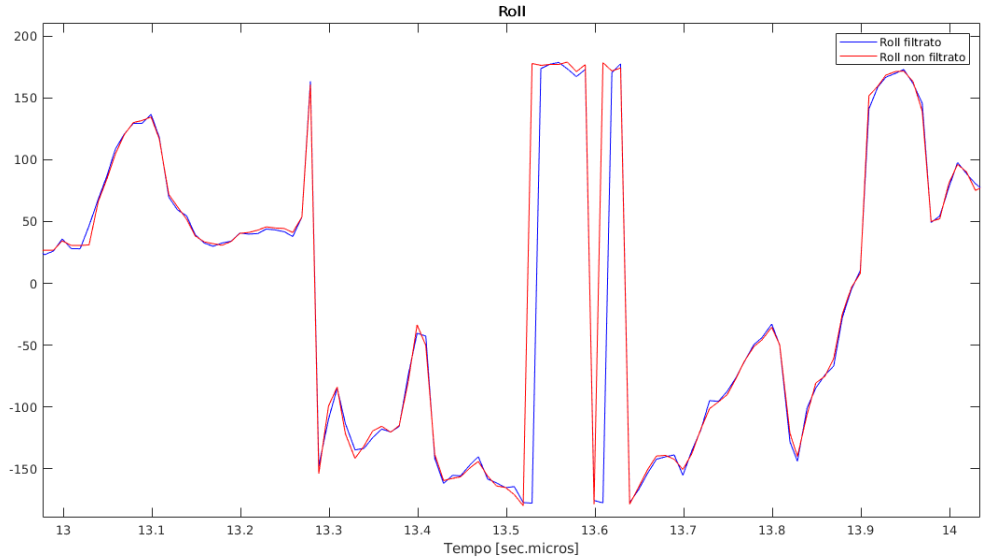


Figura 7.2: Roll

7.2 Consumi di corrente nelle varie modalità operative

Per misurare il consumo di corrente è stato usata la scheda INA226 che permette di misurare la corrente tramite una resistenza shunt del valore di 0.1 mΩ, in particolare si misura la tensione ai capi di questa resistenza e poi, tramite la legge di Ohm, si ricava la corrente. Il dispositivo opera con una tensione tra i 2.7 V e i 5.5 V e può misurare tra 0 V e 36 V, ha un'alta accuratezza con un errore percentuale massimo di 0.1 % e un offset massimo di 10 μV . Le altre caratteristiche elettriche sono riportate nella tabella 7.1:

Parametro	**Valore**	**Unità di misura**
Risoluzione ADC	16	bit
LSB	2.5	μV
Corrente minima misurabile	25	μA
Corrente massima misurabile	0.8192	A

Tabella 7.1: Caratteristiche INA226 [2]

Usa un'interfaccia I^2C per la configurazione e la lettura dei dati. All'interno del progetto è stato usato per misurare il consumo di corrente del SensorTile interfacciandolo con una scheda Raspberry che tramite un programma legge i valori di tensione.

7.2.1 Sleep

Quando il microcontrollore è in sleep la corrente media è di 370.1 μA. Per fare questa misura è stato forzato lo stato del micro usando la funzione *MCU_PowerSave()*.

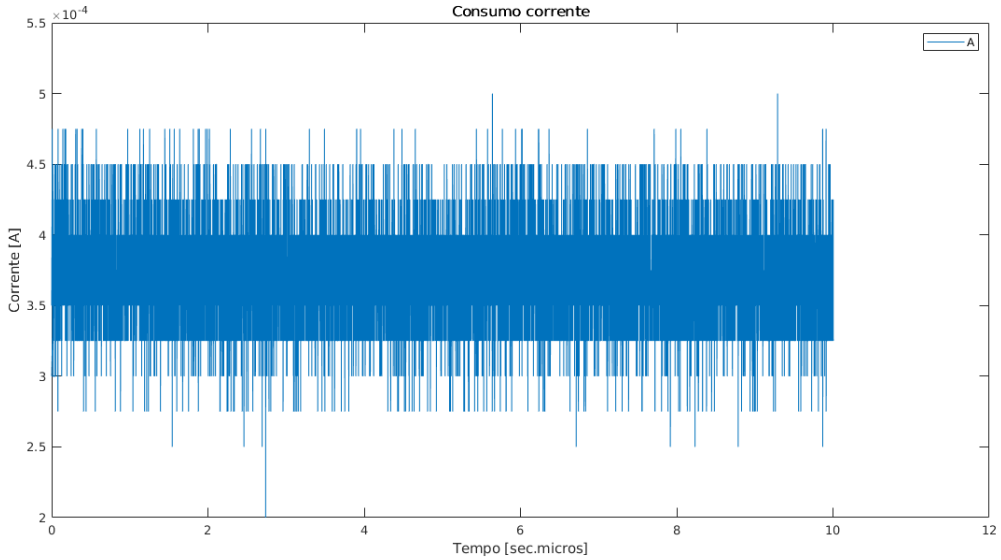


Figura 7.3: Consumo MCU in sleep

7.2.2 Connessione e advertising

La corrente media è di 12.2 mA. Nei primi 5 secondi circa il dispositivo è connesso al server, nei successivi vengono mandati dei dati di advertising.

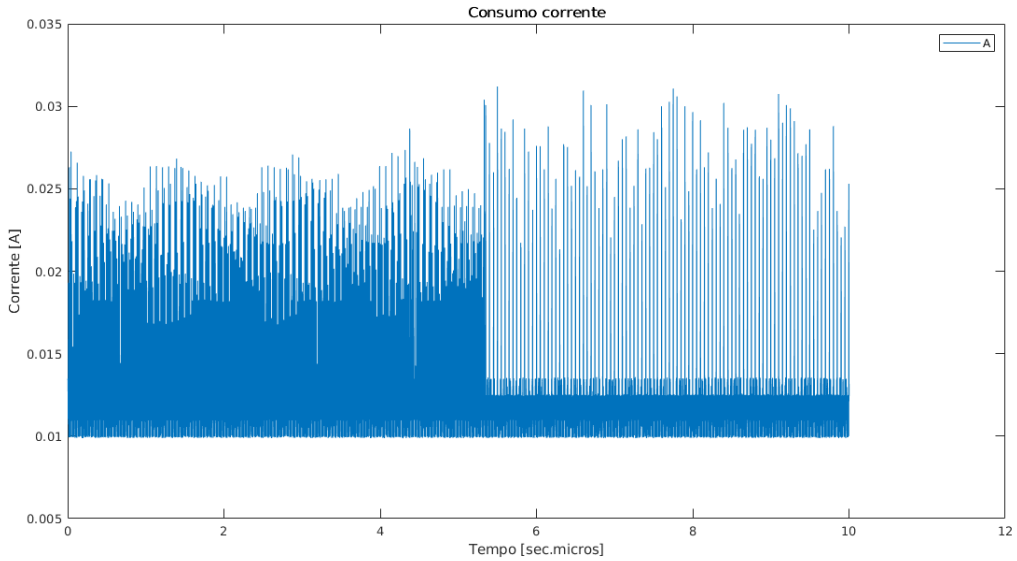


Figura 7.4: Consumo scheda connessa e in advertising

Nel successivo grafico si possono notare dei picchi di corrente ogni 50 ms, infatti i dati di advertising sono trasmessi alla frequenza di 20 Hz.

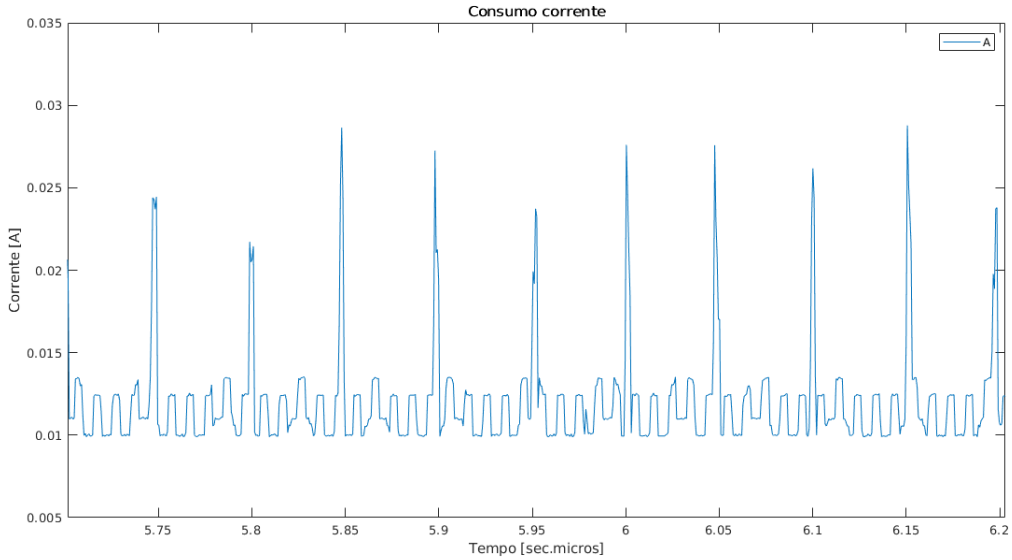


Figura 7.5: Dati di advertising

7.2.3 Streaming

La corrente media in streaming è di 15.1 mA.

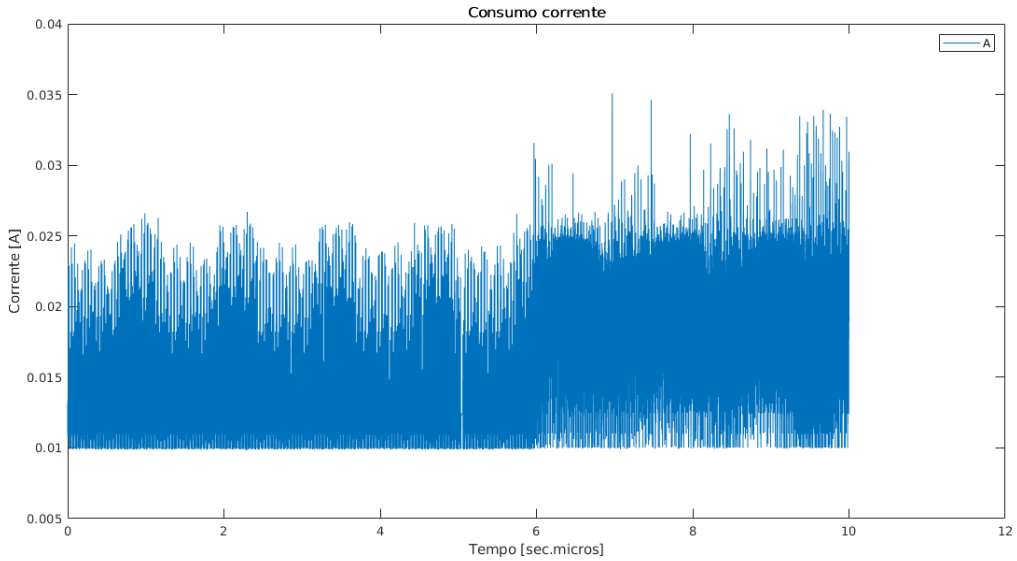


Figura 7.6: Consumo in streaming

Guardando il grafico ingrandito si possono notare dei picchi di corrente ogni 10 ms, questo è dovuto alla trasmissione dei dati via Bluetooth alla frequenza di 100 Hz.

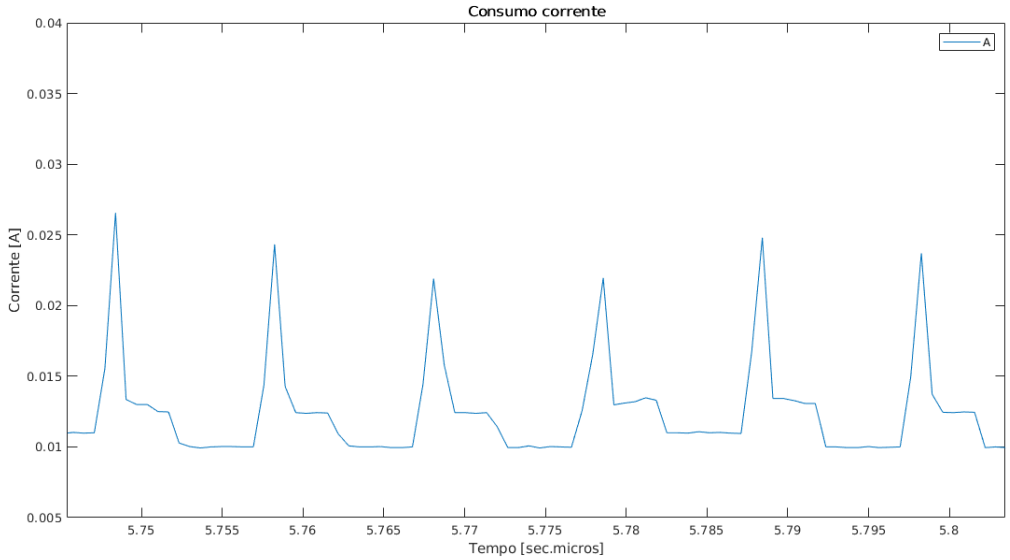


Figura 7.7: Streaming dei dati

7.3 Analisi dei consumi

Dai grafici si può constatare che durante la trasmissione dei dati è presente un offset di circa 10 mA. Questo è dovuto al fatto che il microcontrollore lavora sempre alla massima capacità, ciò è confermato dalla tabella 27 del datasheet [9] in cui è riportato che alla frequenza di clock di 80 MHz, con cache on e prefetch off, si ha un consumo tipico di 10.2 mA. Allo scopo di diminuire ulteriormente i consumi, è necessario spegnere il dispositivo quando non si stanno trasmettendo i dati.
È stato modificato opportunamente il firmware mandando il microcontrollore in sleep con *HAL_PWREx_EnableLowPowerRunMode()* prima del comando wait for interrupt (*__WFI()*). Questa è l'ultima funzione del main loop e fa sospendere l'esecuzione del processore fino a quando si verifica un interrupt. Una volta uscito da questa routine viene ripristinato lo stato con *HAL_PWREx_DisableLowPowerRunMode()*. Usando la modalità di risparmio energetico e abilitando lo stream dei dati Bluetooth si può notare che la trasmissione si interrompe dopo pochi secondi. Questo potrebbe essere dovuto a un riempimento del buffer di trasmissione.
Debuggando l'applicazione è stato notato che in questa parte di codice si settano i seguenti bit di flag:

- bit 0 (UIF, Update Interrupt Flag) del registro TIM1 status register (TIM1_SR), indica l'overflow del timer 1,
- bit 0 (UIF) del registro TIM4 status register (TIM4_SR), indica l'overflow del timer 4,
- bit 0 (UIF) del registro TIM5 status register (TIM5_SR), indica l'overflow del timer 5,
- bit 1 (TXBEF, Transmit buffer empty flag) del registro Single Wire Protocol Master Interface Interrupt and Status register (SWPMI_ISR), indica che il buffer di trasmissione è stato svuotato
- bit 6 (TXE, Transmit data register empty) del registro SWPMI_ISR, indica che il dato scritto in SWPMI_TDR è stato trasmesso e SWPMI_TDR può essere nuovamente riscritto,
- bit 7 (TCF, Transfer complete flag) del registro SWPMI_ISR, indica che sia la trasmissione sia la ricezione sono completate e SWP è settato nello stato "sospeso",
- bit 9 (SUSP, SUSPEND flag) del registro SWPMI_ISR, indica che il bus SWP è nello stato "sospeso" o "disattivato".

I flag dei timer 1, 4 e 5 sono abilitati perché configurati per dare la temporizzazione dell'invio dei dati tramite Bluetooth. I bit precedentemente citati portano all'uscita dalla funzione *__WFI()*.
Per misurare approssimativamente quanto tempo il micro passa nel Wait For Interrupt si fa accendere il led prima dell'abilitazione della modalità low power e si spegne subito dopo l'uscita dallo stato di risparmio energetico. Il rapporto tra la tensione nel regolare flusso del programma sul pin SAI_SD, in cui è mappato il led come mostrato in figura 3.3, e la tensione quando il pin è acceso indica la percentuale di tempo in cui il micro passa nel Wait For Interrupt. Se, ad esempio, la tensione misurata sul pin è nulla allora il programma non passa per niente nel Wait For Interrupt, viceversa, se la tensione tende al valore massimo misurabile, 0.71 V, allora il firmware si blocca molto tempo nel Wait For Interrupt. Per assicurarsi che la Serial Audio Interface sia effettivamente disabilitata sul pin SAI_SD, viene fatto il clear dei bit SAIXEN dei registri SAI_xCR1.
È stata misurata una tensione di 0.66 V che, rapportata a 0.71 V registrati quando il pin è sempre acceso, indica che il microcontrollore passa un intervallo pari a $\frac{0.66}{0.71} = 0.93 = 93\%$ del tempo totale utile nella funzione *__WFI()*.

```
CLEAR_BIT(SAI1_Block_A->CR1, SAI_xCR1_SAIEN);
CLEAR_BIT(SAI1_Block_B->CR1, SAI_xCR1_SAIEN);
CLEAR_BIT(SAI2_Block_A->CR1, SAI_xCR1_SAIEN);
CLEAR_BIT(SAI2_Block_B->CR1, SAI_xCR1_SAIEN);
//accensione led prima di andare in low power
LedOnTargetPlatform();
//Low power quando si aspetta un interrupt
HAL_PWREx_EnableLowPowerRunMode();
/* Wait for Interrupt */
__WFI();
//uscita dalla modalità low power quando è arrivato un interrupt
HAL_PWREx_DisableLowPowerRunMode();
//spegnimento led dopo essere uscito dal low power
LedOffTargetPlatform();
```

Capitolo 8

Conclusioni

I risultati del progetto hanno rispettato le aspettative previste. La ricezione dei dati Bluetooth, usando il programma in Python, avviene in modo corretto. La trasmissione dei dati di beccheggio e rollio opportunamente filtrati si verifica senza problemi: si possono notare infatti le differenze tra i dati filtrati e non filtrati. L'unico aspetto che non ha pienamente rispettato le previsione è il consumo energetico perché il dispositivo, per lavorare in modo efficiente, dovrebbe presentare un consumo di corrente minimo quando non si trasmettono dati.
Il lavoro svolto ha consentito di applicare quanto studiato nei diversi corsi. È stato interessante capire quali problematiche si presentano nell'effettiva realizzazione di un sistema complesso. Durante questo progetto, infatti, sono stati toccati diversi step che contribuiscono allo sviluppo di nuove tecnologie. Lo studio iniziale, le prime prove, l'implementazione di ulteriori miglioramenti e l'analisi delle prestazioni hanno caratterizzato un percorso svolto effettivamente completo.
L'attività svolta ha permesso una maggiore conoscenza in vari ambiti. Lo studio dello standard Bluetooth e lo sviluppo un firmware in linguaggio C hanno ampliato le conoscenze nelle telecomunicazioni e nei sistemi embedded. La realizzazione di questo progetto ha permesso di acquisire nozioni che non sono state trattate in nessun corso, come la scrittura con un più alto livello di programmazione come Python. Questa attività ha toccato molti argomenti e ne ha mostrato la loro stretta correlazione: le discipline trattate sono nell'ambito della sensoristica, dell'elettronica digitale, della realizzazione di sistemi embedded, delle telecomunicazioni e dell'Internet of Things.
Il lavoro ha stimolato l'apprendimento di tecnologie innovative, come il Bluetooth Low Energy, che possono essere sia motivo di ricerca in ambito accademico sia strumento per migliorare notevolmente la vita delle persone.
La tecnologia Bluetooth, argomento principale dell'elaborato, sta rivoluzionando la società sostituendo, o affiancando, le reti WIFI casalinghe ormai congestionate dai tanti dispositivi collegati. Il Bluetooth può controllare gran parte degli apparati dell'Internet of Things che oggi si connettono al WIFI, fornendo soluzioni destinate all'integrazione non solo in ambito domestico, ma anche aziendale ed industriale.

Bibliografia

[1] Ericsson, Lenovo, Intel Corporation, Microsoft Corporation, Motorola, Nokia Corporation e Toshiba Corporation. *Specification of the Bluetooth System*. Vol. 4.0. 2010. URL: http://www.bluetooth.com.

[2] Texas Instruments. *Datasheet INA226*. 2018.

[3] ST Microelectronics. *BlueNRG-MS Bluetooth® LE stack application command interface (ACI)*. UM1865. 2017.

[4] ST Microelectronics. *BlueNRG-MS radio stack images versions*. 2016.

[5] ST Microelectronics. *Datasheet BlueNRG-MS*. 2019.

[6] ST Microelectronics. *Datasheet LPS22HB*. 2017.

[7] ST Microelectronics. *Datasheet LSM303AGR*. 2018.

[8] ST Microelectronics. *Datasheet LSM6DSM*. 2017.

[9] ST Microelectronics. *Datasheet STM32L476xx*. 2018.

[10] ST Microelectronics. *Getting started with the BlueST protocol and SDK*. UM2496. 2018.

[11] ST Microelectronics. *Getting started with the STM32 ODE function pack for IoT node with BLE connectivity, digital microphone, environmental and motion sensors*. UM2059. 2018.

[12] ST Microelectronics. *STEVAL-STLCS02V1 Schematic diagram*. 2019.

[13] ST Microelectronics. *STM32L4x6 advanced ARM® based 32-bit MCUs*. RM0351. 2015.

[14] K. Townsend, C. Cufi, Akiba e R. Dividson. *Getting Started with Bluetooth Low Energy*. O'Reilly, 2014.