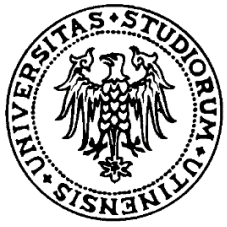


UNIVERSITÀ DEGLI STUDI DI UDINE

DIPARTIMENTO POLITECNICO DI INGEGNERIA E ARCHITETTURA
Corso di Laurea in Ingegneria Elettronica

Tesi di Laurea

STUDIO E SVILUPPO DI UN'APPLICAZIONE CLIENT-SERVER BASATA SU ACCELEROMETRI PER DISPOSITIVI IOT

Relatore:
Prof. Pier Luca Montessoro

Laureando:
Breda Stefano

ANNO ACCADEMICO 2018/2019

INDICE

INTRODUZIONE

Uno degli obbiettivi di questa tesi è quello di riuscire a ricavare i dati di un accelerometro attraverso una piattaforma Iot.
Come dispositivo per misurare l'accelerazione è stato scelto il LIS3DH, da cui saranno ricavate le informazioni base di funzionamento.
Per leggere i valori del sensore si è optato per l'Orangepi Pc 2, un minicomputer a basso costo.
Di questa piattaforma dovranno essere analizzati i componenti base, in modo da collegare correttamente i due sistemi.
L'Orangepi, essendo entrato sul mercato soltanto negli ultimi anni, presenta numerose problematiche legate al sotware di base, che devono essere risolte al fine di poter leggere dati dal sensore.
Altro punto fondamentale di questo progetto è lo sviluppo di un codice di programmazione in grado di rilevare le informazioni dell'accelerometro, quindi è necessaria una trattazione dettagliata dei vari passaggi seguiti per svolgere questo compito.
Infine, si illustrerà il problema legato alla trasmissione di tali dati in una comunicazione Client-Server, per renderli disponibili, o utilizzabili esternamente al sistema sensore-piattaforma.

CAPITOLO 1

1.FISICA E STRUTTURA DEGLI ACCELEROMETRI

1.1 ACCELEROMETRI

L'accelerometro è uno strumento che consente la misura dell'accelerazione calcolando il rapporto tra la forza applicata ad un oggetto e la sua massa.

I principali componenti sono una massa e una molla. Nella realtà è necessario considerare l'effetto ammortizzante dovuto all'ambiente nel quale tale oggetto è stato inserito, per questo motivo alla struttura complessiva deve essere incluso anche uno smorzatore. Questo costituisce il cosiddetto sistema massa-molla-smorzatore.

1.2 MICRO-ACCELEROMETRI

I micro-accelerometri hanno dimensioni molto ridotte rispetto agli accelerometri normali; è necessario trovare delle strutture che riescano a adattarsi perfettamente a tali condizioni.
La loro composizione, come riporta "Mems and Microsystems Design and Manufacture", è possibile descriverla nel seguente modo : "un piccolo raggio di silicio con una massa attaccata (chiamata generalmente massa sismica) costituiscono il sistema massa-molla; l'aria presente nella zona circostante rappresenta l'effetto di smorzamento mentre la struttura supportante la massa si comporta come una molla".
I micro-accelerometri possono essere di diverso tipo: piezoelettrici, piezoresistivi, capacitivi … e così via, in dipendenza dall'oggetto che misura l'accelerazione.

Di seguito viene riportato un breve esempio di accelerometro piezoresistivo[1]. In questa tipologia la molla è rappresentata da un supporto piano a cui viene attaccata una massa: quando viene applicata una forza la massa si deforma. La presenza di un piezoresistore permette di rilevare tale deformazione e calcolare l'accelerazione correlata.

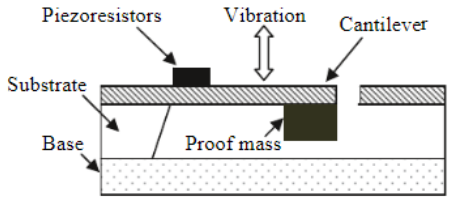


Fig.1.1: Struttura micro-accelerometro piezoresistivo.
Tratto da: Albarbar, A. & Mekid, Samir & Starr, Andrew & Robert, Pietruszkiewicz. (2008) : "Suitability of MEMS Accelerometers for Condition Monitoring: An experimental study", p.786

La scelta di uno di questi accelerometri dipende dalla tipologia di misurazione che dobbiamo andare ad operare. Per elevati valori di accelerazione si utilizzano, ad esempio, i piezoresistivi e i piezoelettrici, mentre per basse accelerazioni quelli capacitivi.

[1] Tratto dal libro Mems and Microsystem Design and Manufacture di Tai-Ran Hsu

1.3 MEMS

Con la sigla MEMS (Micro Electro Mechanical System) si intende indicare un insieme di dispostivi miniaturizzati integranti strutture di tipo meccanico, elettrico o elettronico; in "Mems and Microsystems Design and Manufacture" sono descritti come oggetti contenenti un certo numero di componenti, i quali possono assumere dimensioni che vanno da qualche micrometro a qualche millimetro.
I MEMS si dividono in due categorie, micro-sensori e micro-attuatori, a seconda del compito svolto. Nel caso dei micro-sensori avviene la trasformazione da una variabile fisica ad un'altra mentre nel caso di attuatori l'uscita prodotta è un'azione fisica.

1.4 ACCELEROMETRI CAPACITIVI MEMS

Questa tecnologia di miniaturizzazione ha consentito la realizzazione degli accelerometri capacitivi MEMS, che permettono di misurare la forza dinamica in maniera piuttosto accurata. La struttura base che li compone è data da una massa centrale che, oscillando attorno ad una posizione di equilibrio, genera delle differenze capacitive; tali variazioni sono dell'ordine dei fF (Femto-Farad) e possono essere rilevate grazie ad appositi circuiti amplificatori di segnale.

Questo sensore Mems è costituito da due condensatori collegati attraverso un semi-ponte capacitivo[2] (in half-bridge) .

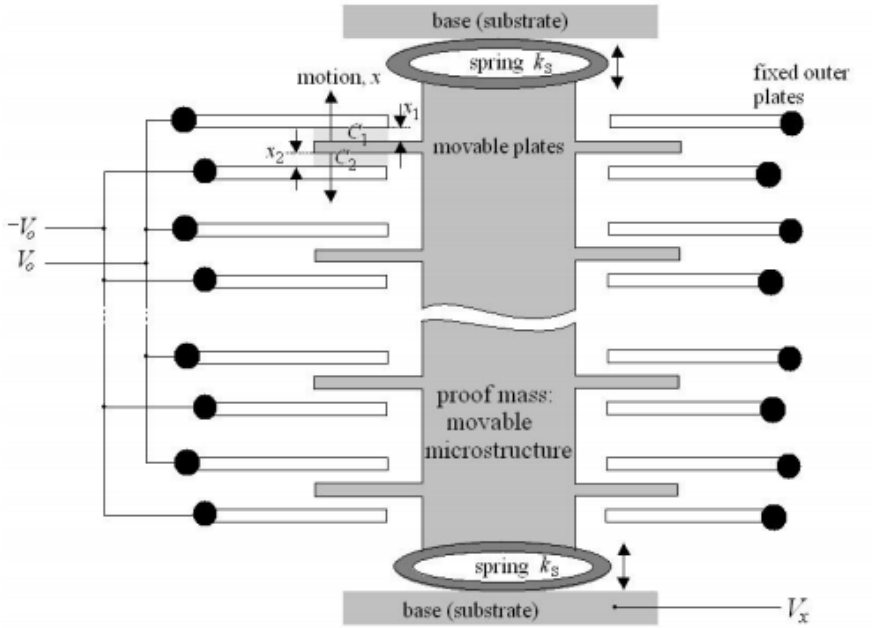


Fig.1.2: Struttura dell'accelerometro. Il movimento della massa centrale genera una variazione della distanza piedini-armature e produce una tensione in uscita.
Tratto da: Constantinescu, Florin & Gheorghe, Alexandru & Nitescu, Miruna. (2013): "A Capacitive Accelerometer Model", p.165

La struttura di un accelerometro capacitivo (figura 1.2) è data dall'alternanza di armature fisse e di piattini movibili; quest'ultimi sono attaccati ad un corpo centrale chiamato massa test.

[2] Descrizione riportata da Nano and micro-elettromechanical system

La massa test è libera di muoversi: definite C_1 e C_2 le capacità , x_1 e x_2 le distanze tra i piattini e le armature fisse, è possibile affermare che tra di loro esiste una relazione di diretta proporzionalità.
Di seguito sono presentati i passaggi principali, presenti in "Nano and Micro Elettromechanical System", per calcolare tale rapporto.
Supponendo che ogni struttura armatura-piattino-armatura si comporti come un condensatore a facce piane e parallele ideale (cioè senza effetti ai bordi), la generica capacità è esprimibile come:

$$C = \frac{\varepsilon A}{d} \tag{1.1}$$

dove ε è la permettività, d la distanza piattino-armatura e A l'area di sovrapposizione.
Supponendo un movimento del corpo centrale, definita x tale differenza di spostamento e definito l il valore di equilibrio per cui $x_1 = x_2$:

$$C_1 = \frac{\varepsilon A}{p_1} = \frac{\varepsilon A}{l+x} \qquad C_2 = \frac{\varepsilon A}{p_2} = \frac{\varepsilon A}{l-x} \tag{1.2}$$

dove p_1 e p_2 sono le nuove posizioni assunte rispetto alla situazione nominale.
La differenza di capacità risulta essere:

$$\Delta C = C_2 - C_1 = \frac{\varepsilon \cdot A \cdot x}{l^2 - x^2} \tag{1.3}$$

Da ciò si può dimostrare che esiste una relazione di diretta proporzionalità tra x e ΔC.
Dalla precedente equazione si ottiene che:

$$\Delta C x^2 - \varepsilon A x - \Delta C l^2 = 0 \tag{1.4}$$

Per piccole variazioni di C, si può trascurare il termine $\Delta C x^2$, ottenendo:

$$x \approx \frac{l^2}{\varepsilon A} \Delta C \tag{1.5}$$

La relazione 1.5 prova quanto affermato in precedenza.
Trovando una relazione tra x e l'accelerazione è possibile mettere a confronto quest'ultima e ΔC.
La massa test si comporta come una molla ideale: è possibile applicare la legge di Hook ,la quale afferma che:

$$F_m = k_m x \tag{1.6}$$

dove k_m è la costante della molla e x l'allungamento.
Dalla seconda legge del moto:

$$ma = F_m = k_m x \tag{1.7}$$

La seguente formula mette in stretta correlazione la variabile x all'accelerazione:

$$a = \frac{k_m}{m} x \tag{1.8}$$

Prendendo in considerazione quanto scritto prima, riguardo alla relazione tra l'allungamento e la differenza di capacità, si ottiene che:

$$a = -\frac{k_m l^2}{m\varepsilon A} \Delta C \tag{1.9}$$

Questa equazione permette sia di calcolare l'accelerazione del sensore, data una differenza di capacità, sia di capire come sia possibile ottenere una misurazione della forza dinamica a partire dagli spostamenti di condensatori.

Le strutture base che compongono un micro-accelerometro capacitivo sono delle capacità differenziali.
In una capacità differenziale, il movimento di una parte movibile fa diminuire una capacità e aumentare l'altra. Come descritto precedentemente, questa struttura viene realizzata mantenendo fisse le armature esterne e muovendo il corpo centrale.

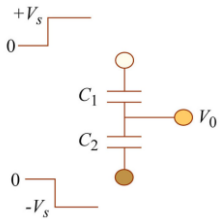


Fig.1.3: Capacità differenziale. Le due tensioni V_S applicate ai capi generano la tensione di uscita V_0.
Immagine tratta da: Stephen Centuria: "Microsystem Design"

Una proprietà fondamentale delle capacità differenziali è che consente di eliminare gli effetti indesiderati del primo ordine, linearizzando il punto di stabilità.
Supponiamo di avere il sistema descritto in precedenza, con le capacità C_1 e C_2 e di applicare alle armature esterne una tensione -V_S e +V_S, come in figura 1.3.
Applicando la sovrapposizione degli effetti, cioè facendo agire le tensioni singolarmente, e applicando un partitore di tensione capacitivo è possibile trovare la tensione in uscita, il cui valore è:

$$V_O = \frac{-C_2}{C_1+C_2}(V_S) + \frac{C_1}{C_1+C_2}(V_S) = \frac{C_1-C_2}{C_1+C_2} V_S \tag{1.10}$$

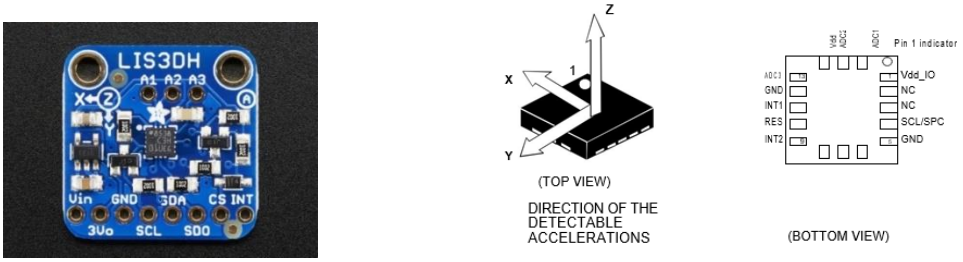
Indicando con G_1 e G_2 le differenze delle due capacità dai valori nominali si ottiene:

$$V_O = \frac{G_2-G_1}{G_1+G_2} V_S \tag{1.11}$$

Come scrive Stephen Centuria[3], nel caso in cui si abbia una variazione di capacità, l'uscita rimane una funzione lineare della tensione in ingresso.

[3] Autore del libro "Microsystem Design"

1.5 LIS3DH
1.5.1 STRUTTURA DELL'ACCELEROMETRO



(a) (b)

Fig.1.4: (a) immagine accelerometro. (b) disposizione dei pin / orientazione dell'accelerometro.
Fonti: (a) https://www.adafruit.com/product/2809,
(b) https://www.st.com/resource/en/datasheet/lis3dh.pdf, p.8

Lis3dh (Fig. 1.4 (a)) è un accelerometro a tre assi lineare di bassa potenza ad alte prestazioni[4].
Si connette al "mondo esterno" attraverso due possibili interfacce I^2c[5] e Spi[6] (protocolli il cui funzionamento sarà descritto nei paragrafi successivi).
Come riporta il datasheet:
" L'accelerometro è stato realizzato per mezzo di strutture di silicio sospese, che sono state attaccate al substrato in alcuni punti chiamati ancore e sono state lasciate libere di muoversi per poter misurare la forza dinamica. Quando viene applicata una forza esterna al sensore, la massa di prova si sposta dalla sua posizione, causando uno squilibrio al semi-ponte capacitivo".
Si tratta della struttura descritta nel paragrafo precedente, per i micro-accelerometri capacitivi: l'analisi svolta può essere collegata a questo sensore.

1.5.2 CARATTERISTICHE DELL'ACCELEROMETRO

L'alimentazione dell'accelerometro, variabile tra i 3.3V e 5V, permette di adattarsi a tutte le piattaforme disponibili (è consigliato utilizzare la logica a 5V, poiché l'altra causa un calo delle prestazioni).
Esso è composto da diversi pin che consentono la connessione con diversi controllori, come si può notare dalla figura 1.4(b). Vi sono l'ingresso V_{in}, per alimentare il dispositivo, i canali SDA, SCL e SDO, per la ricezione e invio dei dati, e il pin CS, per la selezione dell'accelerometro.
Ci sono due ulteriori pin, INT1 e INT2, che prendono il nome di pin di interrupt; essi permettono di inviare dei segnali di interruzione che servono ad indicare l'avvenimento di determinati "eventi".
La loro funzione può essere spiegata dal seguente esempio[7]. Supponiamo di avere una serie di sensori collegati ad un controllore e che quest'ultimo li analizzi in maniera sequenziale: i pin di interrupt entrano in gioco quando è necessario comunicare un evento asincrono, al fine di

[4] Informazione tratta dal data-sheet dello strumento
[5] I^2c: Inter-Integrated circuit
[6] Spi: Serial peripheral interface
[7] Esempio tratto da https://www.maffucci.it/2012/06/11/appunti-su-arduino-interrupts/

dirigere l'attenzione del controllore sul sensore in cui si è verificato un cambiamento delle condizioni.
I pin A1, A2 e A3 in fig.1.4(a) costituiscono un convertitore analogico/digitale ausiliario del Lis3dh; se non sono necessari si può lasciarli liberi, a massa oppure alimentati a 3,3V.
Infine, l'accelerometro è composto da innumerevoli registri interni, ognuno con il proprio indirizzo.
Essi svolgono un numero elevato di operazioni come, ad esempio, la selezione del range di ampiezza di accelerazione, variabile tra $\pm 2g$ e $\pm\ 16g$, l'impostazione della velocità di campionamento dei dati ad un valore compreso tra 1 Hz e 5.3 kHz (i registri saranno descritti in maniera dettagliata nei capitoli successivi).

1.5.3 DIAGRAMMA A BLOCCHI

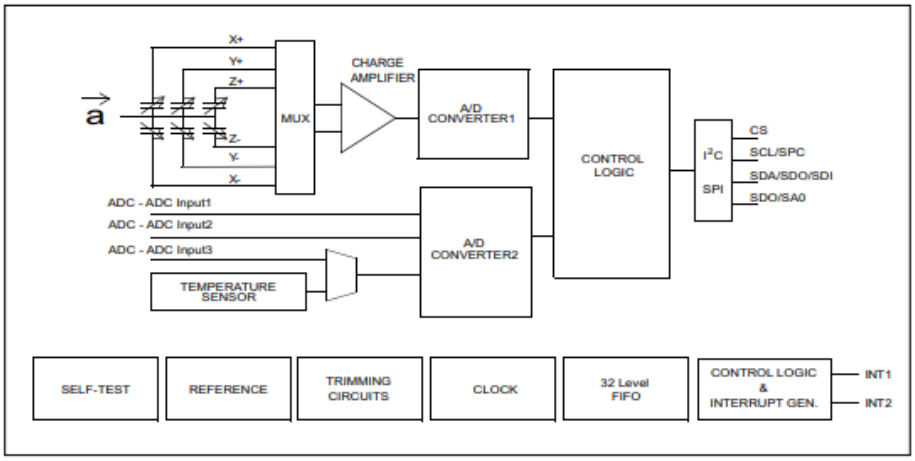


Fig.1.5: Diagramma a blocchi dell'accelerometro LIS3DH.
Fonte: https://www.st.com/resource/en/datasheet/lis3dh.pdf, p.8

Nello schema a blocchi riportato in figura (1.5) è possibile osservare i seguenti passaggi fatti dal "segnale" accelerazione:

- La forza applicata all'accelerometro viene rilevata da una serie di condensatori differenziali, i quali la convertono in una tensione di uscita;
- La tensione di uscita passa attraverso un amplificatore capacitivo a basso rumore, che la converte in una tensione analogica;
- Un convertitore analogico/digitale opera ad un'ulteriore conversione in digitale;
- I dati risultano presentati all'utente tramite le periferiche Spi oppure I^2c.

1.5.4 BUFFER FIFO

Il Lis3dh dispone di un buffer interno a 10-bit FIFO[8] integrato a 32 livelli, esso permette di immagazzinare i dati, provenienti dai tre assi x, y, z, con la tecnica First In First Out.
Esistono quattro modalità differenti: FIFO MODE, STREAM MODE, STREAM-TO-FIFO-MODE e BYPASS MODE. Per ognuna di esse il data-sheet riporta le seguenti descrizioni:

[8] FIFO: First In First Out

- FIFO MODE: il buffer continua a ricevere dati da x ,y ,z. Una volta pieno il contenuto rimane invariato.
- STREAM MODE: al termine del riempimento del buffer, avviene una sovrascrizione dei vecchi valori; finché l'accelerometro non viene letto tale sovrascrittura continua.
- STREAM-TO-FIFO-MODE: situazione intermedia tra il FIFO MODE e lo STREAM MODE.
- BYPASS MODE: i dati non vengono salvati, viene utilizzato solo il primo indirizzo mentre tutti gli altri rimangono vuoti.

Vi sono opportuni registri che attivano queste opzioni.

1.6 PROTOCOLLI DI COMUNICAZIONE DEI SENSORI

1.6.1 INTRODUZIONE

I sensori possono essere collegati ad ulteriori dispositivi, che consentono di gestire e controllare i trasferimenti di informazioni.
In questa struttura è possibile operare la seguente distinzione:

- Master è il dispositivo a capo della comunicazione; il suo scopo è quello di gestire una trasmissione;

- Lo Slave è un dispositivo che può solo ricevere e inviare dati dal/al Master; non è in grado di inizializzare o controllare una trasmissione. Il clock, con cui si invia o riceve i dati, è fornito dal Master e allo Slave non è consentito modificarlo.

Il protocollo I^2c oppure il protocollo Spi garantiscono lo scambio di informazioni tra il Master e lo Slave.

1.6.2 PROTOCOLLO I^2C

1.6.2.a DESCRIZIONE

I^2c è un protocollo che permette di gestire la comunicazione tra più Master e Slave; esso si basa su tre canali chiamati Serial Data (SDA), Serial Clock (SCL) e Ground (GND).
I dati che vengono inviati sul layer fisico sono costituiti dai seguenti valori:

- Indirizzo Slave 7-bit: ogni dispositivo ha un proprio indirizzo; ad esempio quello del Lis3dh è 0x18 (0x19 se si connette il pin SDO a 3v3);
- 1 bit dedicato alla tipologia di operazione che si vuole effettuare: lettura oppure scrittura.
- 8 bit assegnati ai dati;
- Bit singoli assegnati alle condizioni di START, STOP e ACK[9].

[9] ACK: Acknowledgement

Start	Slave Address	Rd/nWr	Ack	Data	Ack	Data	Ack	Stop
1 bit	7 bits	1bit	1bit	8 bits	1 bit	8 bits	1 bit	1 bit

Tab.1.1: esempio di pacchetto dati per il protocollo I^2c;
in tabella è riportata una trasmissione multipla di dati.

1.6.2.b FUNZIONAMENTO

La comunicazione inizia quando il Master invia una condizione di START sul bus fisico. Tutti i sistemi connessi, che siano altri Master o che siano altri Slave, leggono l'informazione ed iniziano ad ascoltare il layer in attesa di nuovi dati.
Il Master comunica l'indirizzo del dispositivo, insieme al bit corrispondente alla operazione che vuole effettuare, e possono verificarsi due situazioni differenti:

- Se non risulta alcuna corrispondenza, il dispositivo rimane in ascolto del bus, in attesa della condizione di STOP;
- Se vi è corrispondenza, il dispositivo produce un ACK.

Ricevuto il messaggio di conferma, inizia lo scambio delle informazioni in pacchetti da 8 bit. Per terminare la comunicazione il Master invia la condizione di Stop. Questo segnale indica che il bus è stato liberato e quindi un nuovo IC[10] può impegnare la linea.

Quando un Master vuole ricevere o trasmettere dati ad uno Slave pone il bit RD/nWr (come riportato in Tab. 1.1), rispettivamente, al valore logico di 0 oppure 1.
In Tab 1.2 sono rappresentati due esempi di trasferimento dati: uno in cui è il Master ad inviare i dati, un altro dove è lo Slave a trasmettere.[11]

Start	Slave Address	0	0	Data	0	Data	0	Stop
1 bit	7 bits	1bit	1bit	8 bits	1 bit	8 bits	1 bit	1 bit

(a)

Start	Slave Address	1	0	Data	0	Data	1	Stop
1 bit	7 bits	1bit	1bit	8 bits	1 bit	8 bits	1 bit	1 bit

(b)

Tab 1.2 (a) esempio di trasmissione di 2 byte in cui si sono colorati i dati inviati dal master;
(b) esempio di ricezione di 2 byte in cui si sono colorati i dati inviati dal master.

[10] IC: Integrated Circuit
[11] La struttura dati è tratta da *An Introduction to I^2c and SPI protocols*

1.6.3 LS3DH I²C

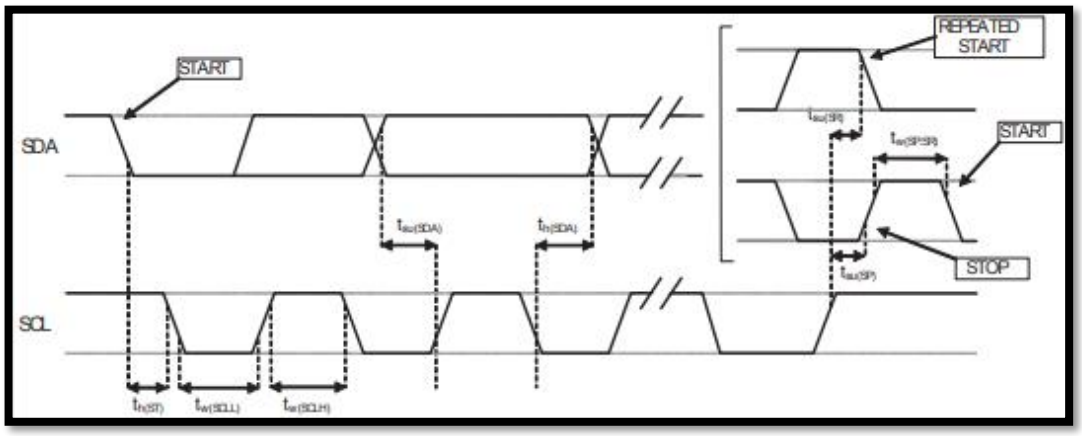


Fig.1.6: Andamento del segnale dati e del segnale di clock.
Fonte: https://www.st.com/resource/en/datasheet/lis3dh.pdf, p.14

Nel Lis3dh la procedura risulta essere simile a quella descritta nel caso generale con le seguenti particolarità.
Come si può notare dal grafico (figura 1.6), la condizione di START è definita come una transizione ALTO-BASSO sulla linea SDA, mentre il clock SCL è tenuto ad un valore logico alto.
Dopo lo START, il Master deve trasmettere l'indirizzo del sensore con cui vuole comunicare; svolge tale compito inviando sul layer un pacchetto dati formato da 8 bit : 7 bit contenenti l'indirizzo e 1 bit per l'indicazione di scrittura o lettura.
Si definisce SAD[12] l'indirizzo dell'accelerometro: nel caso del Lis3dh il SAD è 00110xb[13].
Mettendo il pin SDO in float[14] o a 3,3V , è possibile cambiare l'indirizzo rispettivamente in 001100b oppure in 001101b .
Ad ogni informazione consegnata sul bus, il protocollo impone la conferma attraverso un messaggio di acknowledgement.
Come riporta il data-sheet, il clock-pulse acknowledge è realizzato ponendo la linea dati bassa per tutto il periodo in cui il clock si mantiene alto.
Come fatto con il SAD, il Master deve inviare l'indirizzo di uno dei registri del sensore; questo messaggio, definito SUB[15] , è composto da 8bit: 7 bit meno significativi dedicati all'indirizzo di uno dei registri che compongono l'accelerometro e il bit rimanente per confermare o meno la possibilità di ricevere/trasmettere più dati sul bus fisico.
Successivamente, vengono inviati o ricevuti dal Lis3dh, le informazioni presenti sulla linea SDA.
Nel caso in cui il Master voglia leggere i dati dallo Slave, è necessario che, una volta ricevuto l'acknowledgement al SUB, venga inviata nuovamente una condizione di START con annesso SAD e il bit indicante l'operazione. Per ogni pacchetto dati ricevuto il Master deve sempre inviare un messaggio di conferma tranne per i dati che precedono la condizione di STOP.

[12] SAD: Slave Address (riferimento a data-sheet)
[13] Il data-sheet tratta l'indirizzo 001100xb come indirizzo fisico (0x18/0x19)
[14] Sinonimo di libero, non connesso a niente
[15] SUB: Sub Address (riferimento a data-sheet)

Master	Start	Sad+W		Sub		Data		Sp
Slave			Sak		Sak		Sak	

(a)

Master	Start	Sad+W		Sub		Data		Data		Sp
Slave			Sak		Sak		Sak		Sak	

(b)

Master	Start	Sad+W		Sub		Start	Sad+R			Nmak	Sp
Slave			Sak		Sak			Sak	Data		

(c)

Master	Start	Sad+W		Sub		Start	Sad+R			Mak		Nmak	Sp
Slave			Sak		Sak			Sak	Data		Data		

(d)

Tab. 1.2: (a) pacchetto dati protocollo I^2c per l'accelerometro LIS3DH, in cui il Master sta scrivendo allo Slave. (b) Il Master invia pacchetti multipli di dati
(c) pacchetto dati protocollo I^2c, in cui il Master sta ricevendo dati dallo Slave. (d) Il Master riceve pacchetti multipli di dati

1.6.4 PROTOCOLLO SPI

Spi, allo stesso modo dell'I^2c , è un protocollo che permette la comunicazione tra più dispositivi digitali. A differenza di quello precedente questo consente l'interfacciamento di un solo Master. Vi è un solo sistema, garante della trasmissione, connesso per mezzo di un bus, ad un certo numero di Slave.
Sul mezzo trasmissivo viaggiano i quattro segnali utili alla comunicazione:

- Un segnale di clock (SCLK,SCL o CK) imposto dal Master, a cui tutti gli altri Slave si devono adeguare.
- Un segnale di selezione (SS o CS) inviato dal Master per selezionare uno Slave sul layer fisico.
- Un segnale (MOSI, SDO) per inviare dati dal Master allo Slave, chiamato Master Out Slave In;
- Un segnale (MISO, SDI) per inviare dati dallo Slave al Master, chiamato Master In Slave Out.

Supponendo che un Master voglia inizializzare una comunicazione con uno Slave, per inviare o richiedere dati, esso compie fondamentalmente due azioni: fa tendere la linea di selezione ad un valore logico basso e attiva il segnale di clock ad una frequenza usabile da entrambi[16].

[16] Descrizione tratta da dalla rivista *An Introduction to I^2c and SPI protocols* di Frederic Leens

Successivamente può iniziare a trasmettere i dati. La caratteristica principale dei canali MISO e MOSI è la monodirezionalità: come è affermato in http://www.microcontroller.it/Tutorials/Elettronica/SPI.html, questo permette di stabilire una comunicazione full-duplex, garantendo la co-presenza sia di dati in arrivo dal Master sullo SDI, sia di dati inviati dallo Slave sullo SDO. Vi sono degli ulteriori parametri chiamati polarità clock CPOL[17] o CKP e la fase del clock CPHA[18] o CKE.
Le combinazioni possibili risultano 4, date dall'abbinamento dei due parametri.
Quando CPOL vale zero la comunicazione è inattiva quando il clock è a zero: in queste condizioni, se il parametro CPHA risulta pari a zero, i dati vengono ricevuti sul fronte di salita del clock e trasmessi sul fronte di discesa; se CPHA risulta essere pari uno avviene la situazione opposta.
Simmetricamente, quando CPOL vale uno, la comunicazione rimane inattiva quando il clock è uno. In questo caso, quando CPHA vale zero, i dati vengono ricevuti sul fronte di discesa del clock e sono inviati sul fronte di salita; se CPHA vale uno avviene la situazione opposta.

La comunicazione tra Master e Slave è possibile solo se si dispongono degli stessi parametri di frequenza SCLK, CPOL e CPHA. Collegando più Slave ad uno stesso Master, è necessario che il Master riconfiguri le variabili appena trattate al termine di ogni trasmissione.

1.6.5 LIS3DH SPI

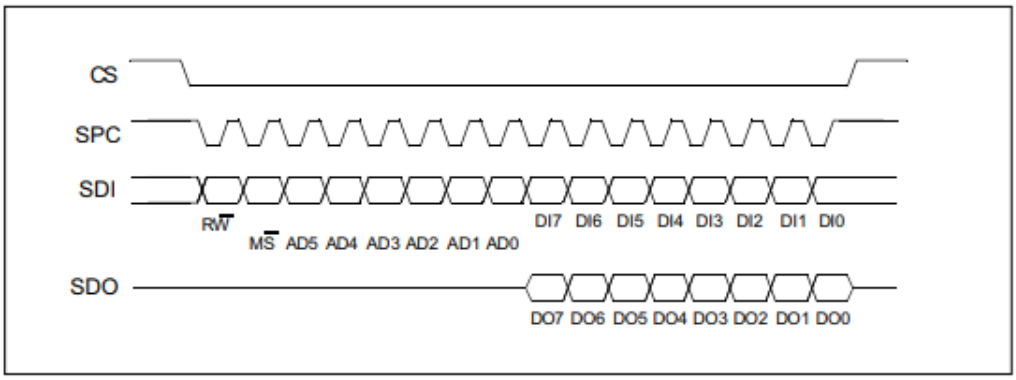


Fig. 1.7: Andamento dei segnali MISO , MOSI, del clock e del pin CS.
Fonte: https://www.st.com/resource/en/datasheet/lis3dh.pdf, p.27

Nel Lis3dh le letture e le scritture vengono realizzate in 16 colpi di clock, oppure in multipli di 8 nel caso di operazioni multiple[19]; questo perché tra i primi 8 bit sono contenuti i valori degli indirizzi dell'accelerometro, mentre negli altri sono contenuti i dati.
Prima di iniziare una trasmissione sulla linea, il Master porta il valore del Selection Pin a 0: questo indica al sensore collegato che sta per ricevere o che deve inviare delle informazioni.
Come riportato dal data-sheet dell'accelerometro i 16 bit inviati sul bus sono i seguenti:

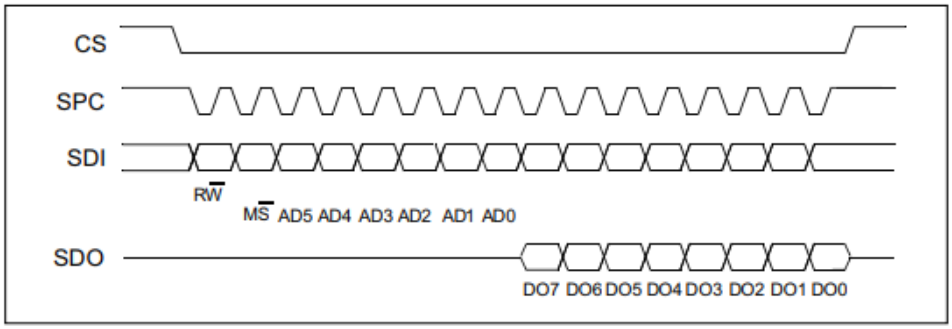
- Bit 0: RW bit. Quando risulta essere a zero, le informazioni contenute in DI(7:0) vengono scritte nel dispositivo. Quando è a uno i dati D0(7:0) vengono letti dal dispositivo.

[17] CPOL : Clock Polarity
[18] CPHA: Clock Phase
[19] Informazione tratta dal data-sheet

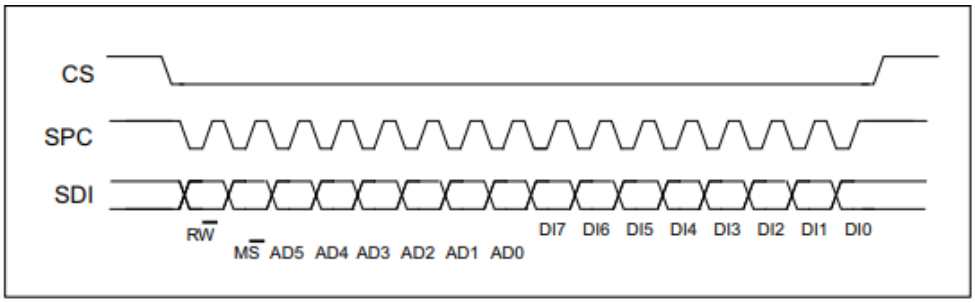
- Bit 1 : MS bit. Quando è 0 l'indirizzo rimane invariato per letture e scritture singole. Altrimenti viene auto-incrementato.
- Bit 2-7 AD(5:0): contiene l'indirizzo di uno dei registri dell'accelerometro.
- Bit 8-15 DO(7:0): dati che devono essere letti o scritti nel dispositivo.

Lo Spi è un protocollo molto semplice; a suo svantaggio, come si può notare dalla tipologia di bit inviati, non presenta né un sistema di indirizzi, né richiede la conferma della ricezione delle informazioni.

L'accelerometro si connette con 4 segnali: MISO, MOSI, SCL e CS; si dice che è un dispositivo 4-wire; esiste anche la variante a 3 fili in cui vi è un solo canale di trasmissione e ricezione delle informazioni.



(a)



(b)

Fig.1.9: (a) Lettura da parte del protocollo SPI. (b) Scrittura da parte del protocollo SPI.
Fonte: https://www.st.com/resource/en/datasheet/lis3dh.pdf, p.28-29

CAPITOLO 2

2.ORANGE-PI

2.1 INTRODUZIONE ALLE SINGLE BOARD COMPUTER

Da circa un decennio si sono sviluppate piccole piattaforme, chiamate single-board computer, che hanno permesso di racchiudere le funzionalità e le caratteristiche di un computer in piccole dimensioni. Da ciò deriva anche una riduzione dei costi e dei consumi.
Questa tipologia di piattaforme è utilizzata per sistemi di sviluppo, a scopi educativi, oppure come controller incorporati in computer (quelli che si chiamano embedded system).
Esempi di queste piattaforme sono l'Arduino, il Rasberry Pi, l'Orange Pi …
L'orangePiPc2 è un minicomputer single-board open-source utile per chi vuole iniziare a progettare sistemi.
I costi di questo dispositivo risultano essere ridotti rispetto a quelli di Rasberry PI, di Arduino oppure di una qualsiasi altra piattaforma: essendo entrata nel mercato da poco non si sono ancora sviluppate applicazioni che permettono di gestire in maniera rapida e senza errori le risorse disponibili.

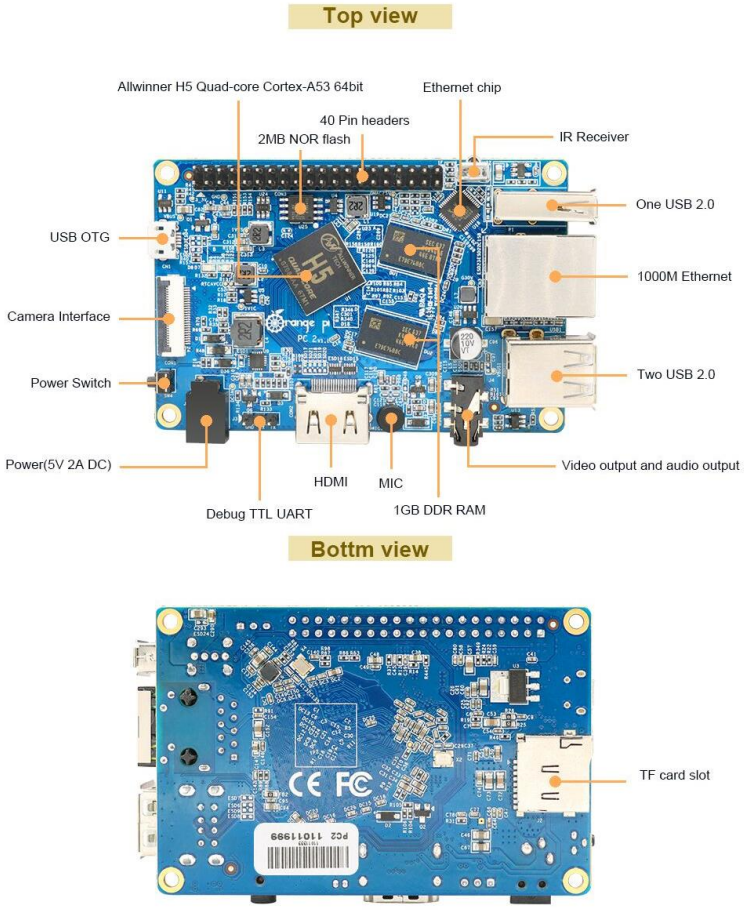


Fig.2.1: Piattaforma Orange-Pi Pc 2: struttura e componenti principali.
Fonte: "http://www.orangepi.org/orangepipc2/"

2.2 HARDWARE

2.2.1 DESCRIZIONE GENERALE

L'Orangepi, ugualmente al Raspberrypi[20], è composto da una piattaforma di 8,5 x 5,5 centimetri e si sviluppa su un'altezza pari a circa 1,5 centimetri, con un peso di appena 70 grammi.
È dotata di una serie di connettori, i quali ci consentono di collegarla con il "mondo esterno": la funzione di queste periferiche risulta essere uguale a quella di personal computer, PC fissi…
Vi sono:

- Tre porte USB 2.0;
- Un'uscita video HDMI per poter visualizzare il dispositivo su uno schermo;
- Jack audio;
- Punto di alimentazione;
- Slot per scheda SD card (fino a 64 GB) , che sostituirà il disco fisso dei computer;
- Serie di 40 pin (GPIO) che permettono il collegamento con altri dispostivi e sensori.

Esistono diversi tipi di Orangepi: la struttura risulta essere simile per tutte le tipologie di modello tranne per il fatto che per alcuni vi è anche la possibilità di creare un collegamento wireless con la rete Wi-Fi.
Di seguito vengono analizzati i principali componenti dell'Orangepi secondo la struttura riportata da Valter Minute in *"Raspberry Pi: Guida all'uso"*.

2.2.2 IL PROCESSORE

Sul dispositivo vi è un chip di forma quadrata che racchiude il processore Quad-Core Allwinner H5 (Arm Cortex-A53) con grafica Mali450.

2.2.3 MEMORIA

In questa versione di Orangepi (Orangepi pc 2) la RAM è pari a 1 Gb.
Se si guarda al dispositivo si può leggere DDR RAM: tale dicitura sta per double data rate access memory e garantisce una velocità di lettura/scrittura doppia rispetto ad una RAM normale.

2.2.4 ALIMENTAZIONE

L'alimentatore consigliato per l'Orangepi supporta al massimo 2A per 5V.
Nell'utilizzare dispositivi che erogano minore tensione e corrente si potrebbe andare incontro ad un degrado delle prestazioni e a improvvisi spegnimenti della piattaforma.

[20] Caratteristiche confrontate con *Raspberry PI: Guida all'uso*

2.2.5 USB

Le porte USB permettono il collegamento a tastiere e mouse per il controllo diretto dell'Orangepi.
Ciò si dimostra utile nelle fasi di inizializzazione della piattaforma: prima di passare ad un controllo remoto del dispositivo è necessario installare il sistema operativo.
È necessario controllare che le USB non siano dotate di alimentazione.

2.2.6 STORAGE DATI

L'Orangepi consente di caricare il sistema operativo su una scheda SD. Quest'ultima deve avere una dimensione compresa tra gli 8 Gb e 16 Gb. Lo spazio dedicato all'archiviazione dati risulta essere ridotto qualsiasi scheda prendiamo: quando viene installato il sistema operativo viene destinata una parte prefissata alla memoria, come quando si cerca di masterizzare un DVD.
Lo storage dei dati, oltre alla TF Card, è garantito anche da una piccola memoria NOR flash (2Mb), non volatile, presente sul dispositivo.

2.2.7 COLLEGAMENTO INTERNET

Per il collegamento alla rete locale esiste una porta ethernet. La versione utilizzata per questa tesi (Orangepi pc 2) è sprovvista di scheda wireless; in commercio esistono due varianti in cui risulta essere garantito il collegamento Wi-Fi.

2.2.8 AUDIO/VIDEO

L'Orangepi ha una sola uscita HDMI per poter essere interfacciato attraverso uno schermo.
Se il monitor non supporta una porta HDMI, ma dispone di un ingresso digitale DVI[21] o analogico VGA[22] è necessario utilizzare rispettivamente un convertitore HDMI-DVI o HDMI-VGA.
Per non incorrere in problemi di visualizzazione, è necessario collegare prima il cavo video-audio (HDMI) e successivamente alimentare la piattaforma.
Vi sono ulteriori tre pin che permettono il collegamento all'Orangepi: l'accesso sarà garantito solo tramite terminale.

Per informazioni più dettagliate riguardo le componenti hardware è possibile consultare la tabella al sito http://www.orangepi.org/orangepipc2/ .

[21] DVI: Digital Visual Interface
[22] VGA: Video Graphics Array

2.3 GPIO

La gpio (general purpose input/output) è un'interfaccia composta da 40 pin a cui è possibile collegare l'Orangepi a diversi sensori. Essi permettono la comunicazione in entrambi i sensi.

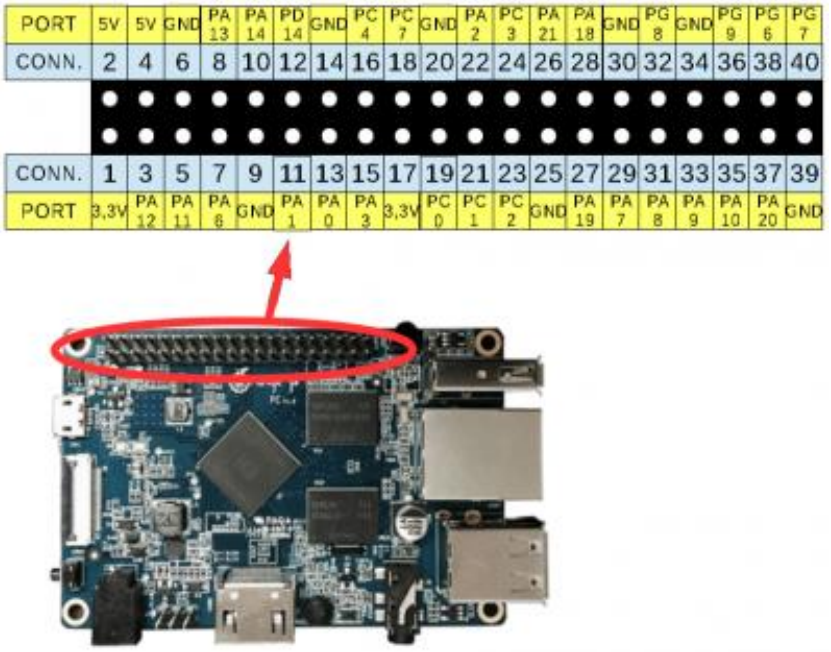


Fig. 2.2: diposizione dei pin nel dispositivo.
Fonte: "https://alpha6.ru/blog/2016/10/08/orange-pi-power-connector/"

Comprendere le funzionalità offerte dai pin è fondamentale per poter collegare successivamente i sensori in maniera corretta. Un errore potrebbe causare danni ai dispositivi interessati.
La Gpio offre 8 collegamenti a massa ai pin 6,9,14,20,25,30,34,39 (come è possibile vedere in figura 2.2).
Inoltre, vi sono 4 pin dedicati all'alimentazione: i numeri 1,17 erogano 3,3V, mentre i numeri 2,4 erogano 5V. Alcuni dispositivi supportano logica a 3,3V, mentre altri, come il LIS3DH, lavorano a 5V: è bene controllare queste specifiche nel data-sheet del sensore al fine di evitare spiacevoli inconvenienti come riavvio della piattaforma, eventuali surriscaldamenti ….
Come visto nel primo capitolo, la comunicazione sensore-Orangepi è gestita dai protocolli I^2c e Spi che usano diversi segnali base.
Partendo dall'I^2c, i segnali di trasmissione dati SDA e di clock SCL sono gestiti dai pin 3,5. Dalla figura è possibile notare che vengono indicati come TWI0_SDA e TWI0_SCL, per poterli distinguere da TWI1_SDA e TWI1_SCL: ciò permette di collegare almeno due dispositivi.
Per quanto riguarda il protocollo SPI i pin 19,21,23 e 24 garantiscono rispettivamente la trasmissione dati, ricezione dati, clock e selezione del dispositivo. Analogamente al caso precedente, è possibile collegare più sensori : il Clock (23) e il Selection Chip (24) sono duplicati da SPI1_CS(10) e SPI1_CLK(12).
UART è l'abbreviazione di universal asyncronous receiver-transmitter ed è un protocollo utilizzato nel collegamento di controller differenti.
Quando si collegano piattaforme diverse tra di loro, se le logiche sono discordi, l'UART garantisce comunque la comunicazione adeguando i segnali di tensioni e corrente.

Questi protocolli non lavorano con i livelli logici di tensione e di corrente del dispositivo: vi sono opportune interfacce che consentono, se programmate, di standardizzare i segnali e collegare piattaforme diverso.
Il protocollo è erogato, nella gpio dell'Orangepi, dai pin 10,12,13,15,17,22,32,36,38,40.

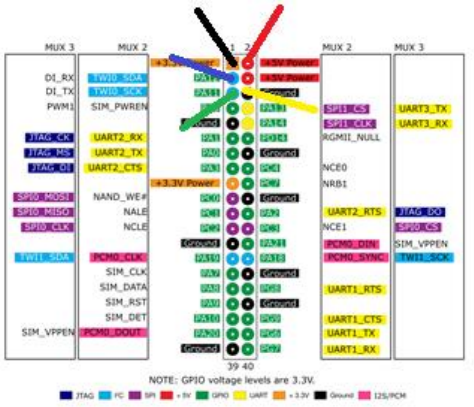
2.4 COLLEGAMENTO PIN ORANGEPI-LIS3DH

Fig.2.3: Riferimento colori per collegamento orangepi-lis3dh. L'unico filo rosso utilizzato nelle immagini successive è quello relativo alla tensione di ingresso (nell'immagine è quello posto più in alto).

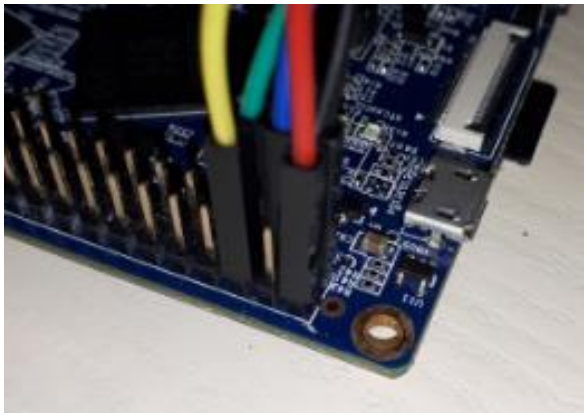
Nella figura 2.3 è riportato l'accelerometro LIS3DH con i relativi fili collegati ai pin del dispositivo.
I colori utilizzati sono: rosso per V_{in}, arancione per il pin a 3,3V, giallo per la massa GND , verde per SCL, blu per SDA, nero per SDO, marrone per il pin CS e rosso scuro per il pin INT. Per non cadere in possibili fraintendimenti nella distinzione dei colori tra il primo e ultimo pin , nelle quattro immagini successive sarà utilizzato solo quello relativo alla tensione V_{in}.

2.4.1 PERIFERICA I^2c

Conoscendo le funzionalità dei pin dell'Orangepi e i pin del sensore LIS3DH è possibile fare i seguenti collegamenti per l'interfaccia I^2c. (Per la periferica I^2c si è scelto di rappresentare il collegamento dell'accelerometro LIS3DH con il pin SDO a 3,3V).



(a)

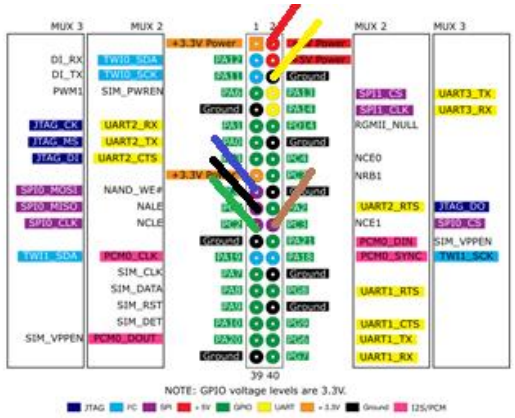


(b)

Fig.2.4: (a) Rappresentazione del collegamento per l'interfaccia I^2C; (b) Rappresentazione reale; SDO a 3,3 V

2.4.2 PERIFERICA SPI

Conoscendo le funzionalità dei pin dell'Orangepi e i pin del sensore LIS3DH è possibile fare i seguenti collegamenti per l'interfaccia Spi.



(a)

(b)

Fig.2.5: (a) Rappresentazione del collegamento per l'interfaccia Spi; (b) Rappresentazione reale

2.5 SOTWARE E INSTALLAZIONE

Le immagini presenti su www.orangepi.com alla voce downloads, risultano essere mancanti di molte informazioni base utili alla progettazione di programmi per il controllo di sensori.
Ad esempio, l'immagine desktop di Ubuntu risulta non avere alcuni moduli kernel.
Per questo motivo è utile scaricare il software Armbian da https://www.armbian.com/orange-pi-pc2/ (versione desktop).
Armbian è un sistema operativo Linux per computer basato su Debian e Ubuntu per schede ARM[23].
Nel manuale dell'Orangepi viene riportata una descrizione dettagliata di installazione del sistema e la sua inizializzazione.
Il primo passo è dato dal "caricamento" di Armbian sulla scheda SD. Per fare ciò è necessario disporsi di due applicazioni: SD Card Formatter e Win32DiskImager.
Il primo programma serve a formattare la scheda, mentre il secondo è utile per l'installazione.
Svolte le due operazioni a questo punto è possibile inserire la scheda SD nello slot dedicato della piattaforma e accendere il dispositivo.
L'Armbian, a differenza degli altri software, richiede, già dalla schermata iniziale, la generazione di un user: una volta inserite le credenziali di root e la password 1234, il sistema impone la compilazione di un nuovo utente e un nuovo codice di sicurezza.
Nei sistemi Unix, il termine root si riferisce al nome predefinito di amministratore di sistema, mentre con user viene indicato l'utente generico.
Il sistema fa sì che già dal primo accesso non sia attiva la modalità root, riducendo a zero il rischio di modifica dei file system, specialmente per chi si sta approcciando per la prima volta a tale ambiente.

[23] Definizione tratta da https://en.wikipedia.org/wiki/Armbian

2.6 CONNESSIONE DEL DISPOSITIVO ALLA RETE

L'Orangepi può stabilire una connessione alla rete solamente con un cavo ethernet.
Si possono verificare solo due possibili scenari: collegamento del cavo direttamente al router oppure attraverso il pc.
Nel primo caso la connessione viene stabilita direttamente; per quanto riguarda la seconda modalità bisogna operare sul computer interessato nella connessione.
Se ci si trova in ambiente Windows[24] si apre la cartella impostazione; successivamente accedendo alla rete Internet, si visualizza la voce modifica opzioni scheda.
A questo punto sono visibili varie connessioni: Wi-Fi, connessione di rete Bluetooth…
Selezionando sia la rete Wi-Fi che ethernet, si preme il tasto destro abilitando la connessione con bridging.
Viene stabilito un "ponte" che permette la connessione tra Orangepi e la rete.

[24] Unico caso trattato, risultano mancanti gli altri sistemi operativi

CAPITOLO 3

3. INSTALLAZIONE E INTRODUZIONE AGLI STRUMENTI BASE DELL'ORANGE-PI

3.1 INTRODUZIONE AL SISTEMA LINUX-ORANGEPI

Tramite il terminale è possibile svolgere innumerevoli operazioni come spostarsi tra le cartelle, modificare i file system, aumentare momentaneamente i propri privilegi….
Come spiegato nel secondo capitolo, quando si accede al sistema Armbian lo si fa in qualità di user. L'utente ha delle limitazioni, ad esempio non può installare programmi, bloccare processi….
Per fare questo è necessario passare alla modalità di root.
Tale funzione è svolta attraverso sudo. Sudo è l'abbreviazione di Super User Do e serve a eseguire i comandi come Super User[25].
La cartella /etc/sudoers contiene le regole di configurazione per l'utilizzo di sudo: essa definisce quali utenti hanno il permesso di usare tale comando, l'insieme di codici vincolati al suo utilizzo….
Il file è modificale digitando nel terminale la stringa visudo.
Mentre sudo esegue i comandi direttamente come root, su (Super User) permette di cambiare utente; si può dedurre tale diversità a partire dalla loro sintassi: nel primo caso sudo è seguito dal comando da eseguire, mentre su deve essere seguito dal nome dello user, con cui ci si vuole "scambiare".
Tuttavia, anche con su è possibile diventare amministratore del sistema: a livello esecutivo, risultano essere equivalenti le diciture sudo su e su root. Tra le due scritture vi è solo una piccola differenza data dal fatto che nel primo caso viene richiesta la password dell'utente, mentre nel secondo caso il codice di sicurezza del root.

Compresa l'importanza di sudo, è possibile introdurre i comandi principali utilizzati per svolgere questa tesi:

- Sudo apt: i comandi apt (advanced package tool) permettono di installare i pacchetti dati, aggiornare il sistema….
  Nella maggior parte dei casi è necessario disporre di privilegi di amministratore, per questo apt deve essere preceduto da sudo.

  Da esso dipendono[26]:
  1. Sudo apt install package-name: installa un pacchetto dati;
  2. Sudo apt remove package-name: elimina un pacchetto dati dal sistema senza rimuovere i file di configurazione;
  3. Sudo apt purge package-name: rimuove sia il pacchetto dati che tutti i file di configurazione;
  4. Sudo apt update: aggiorna la lista di pacchetti disponibili nei repository

[25] Sinonimo di root
[26] La seguente lista è presente in https://wiki.ubuntu-it.org/AmministrazioneSistema/InstallareProgrammi/Apt

(archivi web, contenenti pacchetti sotware del sistema operativo[27]);

5. Sudo apt upgrade: scarica e installa gli aggiornamenti per tutti i pacchetti installati. Bisogna prestare attenzione al fatto che, essendo la memoria dell'Orangepi limitata, tale comando causa una occupazione quasi completa.

Il comando apt può essere accompagnato , a volte, anche dalla dicitura apt-get: le due scritture risultano equivalenti ai fini dell'esecuzione, tuttavia, bisogna tener conto che si tratta di programmi differenti.

- Sudo nano file-name: è un editor di testo per terminale. Una volta digitato il comando compariranno, nel limite inferiore dell'interfaccia, le principali azioni consentite:

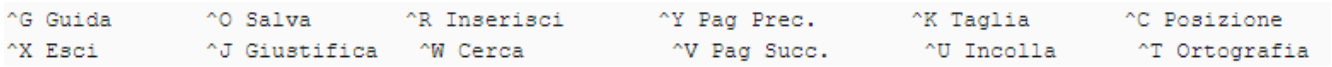


Figura 3.1 Comandi principali presenti nella parte inferiore della facciata quando si digita sudo-nano nel terminale

Risulta essere comodo quando si devono modificare dei file-system, oppure file in cui sono richiesti privilegi da amministratore.

- Chmod: viene utilizzato quando si vuole modificare i permessi di alcuni file.
Il comando può essere accompagnato da dei numeri oppure da delle lettere: le due rappresentazioni sono equivalenti.
Prima di analizzarlo nel dettaglio è necessario considerare il fatto che esistono tre categorie di user che andranno ad utilizzare i file (il proprietario, il gruppo e tutti gli altri utenti), e tre azioni diverse eseguibili su file (lettura, scrittura, esecuzione) .

1. Rappresentazione numerica: vengono assegnati dei numeri diversi a seconda che si debba svolgere una esecuzione, una scrittura o una lettura del file; rispettivamente vengono attribuiti i valori 1, 2, 4.
Se voglio leggere e scrivere un file si dovrà scrivere chmod 6 dove 6 rappresenta la somma di 2 e 4.
Tenendo presente che le categorie di user sono 3, la dicitura completa del comando è:

$ chmod UGO file-name[28]

dove le lettere sono in relazione, riferite a utente, gruppo e altri utenti.
Nella seguente tabella vengono riportate le relative combinazioni delle azioni:

Valori numerici	Permessi
7	Lettura, scrittura ed esecuzione
6	Lettura e scrittura
5	Lettura ed esecuzione
4	Solo lettura
3	Scrittura ed esecuzione

[27] Definizione di repository tratta da https://wiki.ubuntu-it.org/Repository
[28] Dicitura tratta da https://it.wikipedia.org/wiki/Chmod

2	Solo scrittura
1	Solo esecuzione
0	Nessuno

Tab. 3.1: Rappresentazione numerica dei permessi di lettura, scrittura ed esecuzione

Es: chmod 654 file-name permette a utente sia lettura che scrittura, al gruppo lettura ed esecuzione, mentre agli altri utenti solo lettura.

2. Rappresentazione letterale: per le azioni vengono utilizzate r, w, x per lettura, scrittura ed esecuzione; per i fruitori del file invece u, g, o, a seconda che si tratti del proprietario, del gruppo o degli altri utenti.
   In aggiunta a queste notazioni vengono utilizzati anche dei segni operazionali:
   - +: somma le azioni indicate a quelle già presenti;
   - - : rimuove le azioni indicate;
   - =: viene utilizzato sempre prima di dichiarare quali azioni andranno aggiunte o tolte alla classe di utenza.

   Es: $ chmod: " u =r " "g=+rw", "o=-x" file-name : questa scrittura garantisce lettura al proprietario, lettura e scrittura al gruppo, mentre toglie la possibilità di esecuzione a tutti gli altri utenti.

I comandi sopra elencati sono stati utilizzati per scaricare la libreria wiringpi e per installare diversi programmi utili per lo svolgimento della tesi.

3.2 INSTALLAZIONE LIBRERIA WIRINGPI

La libreria Wiringpi costituisce uno strumento veramente importante nella realizzazione di progetti per l'Orangepi: permette di operare e controllare i pin della piattaforma, a cui andranno collegati i vari sensori.
Scaricamento ed installazione della Wiringpi avvengono tramite terminale digitando le seguenti istruzioni :

```
git clone https://github.com/kazukioishi/WiringOP.git -b h5

cd WiringOP
chmod +x ./build
sudo ./build
```

Fig.3.2: codice per il download della libreria wiringpi

Il comando git clone copia dal sito indicato il programma Wiringpi e lo mette nella cartella WiringOp; dopo essersi spostati in tale cartella ed aver aumentato i propri privilegi, si esegue il file con la dicitura ./.
(il processo sopra elencato è descritto nel manuale utente scaricabile da http://www.orangepi.org/downloadresources/) :

Tra le sottocartelle presenti in WiringOp è utile menzionare quelle denominate gpio e wiringpi: esse contengono degli algoritmi, programmi che permettono di visualizzare la configurazione dei pin, scambiare dati con il sensore collegato all'Orangepi e altre operazioni.

3.3 GPIO SU TERMINALE

Per mezzo dei file contenuti in questa cartella, successivamente alla compilazione della libreria Wiringpi, sono resi disponibili delle funzionalità visualizzabili dalla linea del terminale digitando $sudo gpio.

```
Usage: gpio -v
       gpio -h
       gpio [-g|-1] [-x extension:params] ...
       gpio <read/write/aread/awritewb/pwm/mode> ...
       gpio readall/reset
       gpio unexportall/exports
       gpio export/edge/unexport ...
       gpio pwm-bal/pwm-ms
       gpio pwmr <range>
       gpio pwmc <divider>
       gpio load spi/i2c
       gpio i2cd/i2cdetect
       gpio usbp high/low
```

Fig.3.3: Output del comando $ sudo gpio

Tra le funzionalità elencate in figura 3.3 le più importanti sono:

- Gpio read-all: permette la visualizzazione completa dei pin con valori e nomi ad essi associati; nel terminale verrà stampata la seguente tabella:

```
+-----+-----+---------+------+---+---Pi 3---+---+------+---------+-----+-----+
| BCM | wPi |   Name  | Mode | V | Physical | V | Mode | Name    | wPi | BCM |
+-----+-----+---------+------+---+----++----+---+------+---------+-----+-----+
|     |     |    3.3v |      |   |  1 || 2  |   |      | 5v      |     |     |
|   2 |   8 |   SDA.1 |   IN | 1 |  3 || 4  |   |      | 5v      |     |     |
|   3 |   9 |   SCL.1 |   IN | 1 |  5 || 6  |   |      | 0v      |     |     |
|   4 |   7 | GPIO. 7 |   IN | 1 |  7 || 8  | 0 | IN   | TxD     | 15  | 14  |
|     |     |      0v |      |   |  9 || 10 | 1 | IN   | RxD     | 16  | 15  |
|  17 |   0 | GPIO. 0 |  OUT | 0 | 11 || 12 | 0 | IN   | GPIO. 1 | 1   | 18  |
|  27 |   2 | GPIO. 2 |   IN | 0 | 13 || 14 |   |      | 0v      |     |     |
|  22 |   3 | GPIO. 3 |   IN | 0 | 15 || 16 | 0 | IN   | GPIO. 4 | 4   | 23  |
|     |     |    3.3v |      |   | 17 || 18 | 0 | IN   | GPIO. 5 | 5   | 24  |
|  10 |  12 |    MOSI |   IN | 0 | 19 || 20 |   |      | 0v      |     |     |
|   9 |  13 |    MISO |   IN | 0 | 21 || 22 | 0 | IN   | GPIO. 6 | 6   | 25  |
|  11 |  14 |    SCLK |   IN | 0 | 23 || 24 | 1 | IN   | CE0     | 10  | 8   |
|     |     |      0v |      |   | 25 || 26 | 1 | IN   | CE1     | 11  | 7   |
|   0 |  30 |   SDA.0 |   IN | 1 | 27 || 28 | 1 | IN   | SCL.0   | 31  | 1   |
|   5 |  21 | GPIO.21 |   IN | 1 | 29 || 30 |   |      | 0v      |     |     |
|   6 |  22 | GPIO.22 |   IN | 1 | 31 || 32 | 0 | IN   | GPIO.26 | 26  | 12  |
|  13 |  23 | GPIO.23 |   IN | 0 | 33 || 34 |   |      | 0v      |     |     |
|  19 |  24 | GPIO.24 |   IN | 0 | 35 || 36 | 0 | IN   | GPIO.27 | 27  | 16  |
|  26 |  25 | GPIO.25 |   IN | 0 | 37 || 38 | 0 | IN   | GPIO.28 | 28  | 20  |
|     |     |      0v |      |   | 39 || 40 | 0 | IN   | GPIO.29 | 29  | 21  |
+-----+-----+---------+------+---+----++----+---+------+---------+-----+-----+
| BCM | wPi |   Name  | Mode | V | Physical | V | Mode | Name    | wPi | BCM |
+-----+-----+---------+------+---+---Pi 3---+---+------+---------+-----+-----+
```

Fig.3.4: Tabella stampata su terminale al comando gpio read-all

Sotto la voce wPi sono contenuti i numeri da utilizzare con alcune funzioni della libreria Wiringpi.

- Gpio i2cdetect: stampa sul terminale l'indirizzo del sensore collegato all'Orangepi.
  Se la revisione dell'Orangepi è di tipo 1 è necessario digitare i2cdetect -y 0 , altrimenti se è di tipo 1 è necessario digitare i2cdetect -y 1**.**

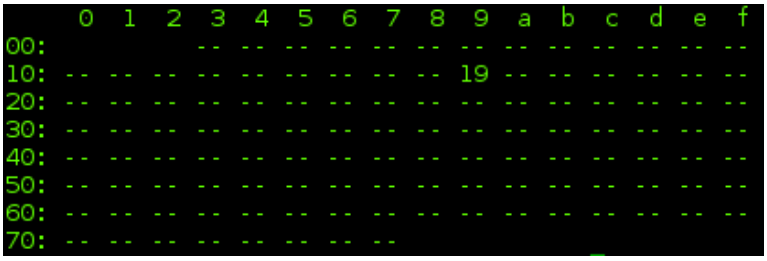


Fig.3.5: output del comando

Nel caso di errori, ad esempio sensore collegato male o non funzionante, il sistema riporterà nessun valore oppure codici esadecimali casuali.

- Gpio spiload : questo comando è utile sia per caricare il modulo kernel dello spi sia per avere maggiore memoria per il buffer interno; come riporta http://wiringpi.com/the-gpio-utility/ il buffer ha le dimensioni espresse in Kb, con un valore di default di 4kb.

- Gpio i2cload: viene usato per caricare il modulo kernel dell'I^2c e per fissare una velocità di trasmissione espressa in kb/s. Anche in questo caso il sito http://wiringpi.com/the-gpio-utility/ afferma che il valore di default è 100 kb/s.

I comandi sopra-elencati devono essere eseguiti in modalità root dal momento che si vanno a compiere azioni che possono compromettere l'integrità del sistema.

3.4 MODIFICA DELLA LIBRERIA WIRINGPI

3.4.1 INTRODUZIONE

Queste librerie sono state scritte per operare su un dispositivo simile all'Orangepi: il Raspberrypi.
Questo fa sì che quando si proverà ad eseguire i comandi sopracitati, oppure ad eseguire programmi per il controllo di sensori, che si basano sulla libreria Wiringpi, l'azione si concluderà con un fallimento.
Per questo motivo è necessario compiere delle modifiche su alcuni file affinché tali librerie risultino compatibili con il sistema.

3.4.2 MODIFICA FILE

Il primo problema risulta essere dato dal fatto che il sistema operativo installato non presenta nella destinazione /proc/ dei file-system il file cpuinfo con i relativi valori utilizzati dalla libreria Wiringpi.

Per risolvere tale evenienza è essenziale aggiungere un file di testo nella pagina principale con le informazioni necessarie, oppure nel desktop, recante i seguenti dati:

```
Processor       : AArch64 Processor rev 1 (aarch64)
processor : 0
processor : 1
processor : 2
processor : 3
BogoMIPS  : 48.00
Features  : fp asimd evtstrm aes pmull sha1 sha2 crc32 cpuid
CPU implementer   : 0x41
CPU architecture: 8
CPU variant   : 0x0
CPU part  : 0xd03
CPU revision  : 4

Hardware        : sun50iw2
Serial          : 3400503640144c2a058d
```

Fig.3.6: Formato dei dettagli hardware dell' Orangepi PC 2

La destinazione di questo documento è particolarmente importante dal momento che è necessario sostituirla a quella già presente nel file wiringpi.c sotto la dicitura /proc/cpuinfo. Ad esempio, si dovrà cambiare la seguente stringa (in verde la vecchia dicitura):

```
/* if ((cpuFd = fopen ("/proc/cpuinfo", "r")) == NULL)   */
    if ((cpuFd = fopen ("/home/nome_utente/cpuinfohw", "r")) == NULL)
   piBoardRevOops ("Unable to open /home/nome_utente/cpuinfohw") ;
  while (fgets (line, 120, cpuFd) != NULL)
    {
      if (strncmp (line, "Hardware", 8) == 0)
      break ;
    }
```

Fig.3.7: codice da sostituire; al posto di nome_utente
indicare la cartella in cui è stato posto il file con i dettagli hardware

Il processo di sostituzione può essere accelerato se da tastiera si digita contemporaneamente ctrl e f: si aprirà una barra di controllo in cui si potrà scrivere /proc/cpuinfo; fatto questo il sistema metterà in evidenza tutte le "voci" all'interno del file.

3.5 FILE BOOT

Un'altra problematica dell'Orangepi è legata alle interfacce I^2c e Spi. Queste periferiche risultano essere utilizzate da altri componenti del dispositivo, come la memoria FLASH del processore: quando l'accelerometro viene collegato, in entrambe le modalità, esso non viene rilevato e sul terminale sarà stampato l'indirizzo del componente della piattaforma.
Per ovviare a questo è indispensabile modificare dei parametri contenuti nel file boot del file system.
Digitando nella linea di comando sudo nano /boot/armbianEnv.txt si accede al corpo del file armbianEnv.txt, contenuto nella cartella boot, e successivamente si aggiunge la riga overlays i2c0 i2c1 i2c2 spidev e si modifica lo spiparameter a 1 anziché a zero.

La stringa overlays riporta tutti i moduli poiché se utilizzo due linee separate , uno per i parametri I^2c e una per quelli Spi, otterrei un annullamento da parte del sistema delle prime condizioni.
Qui di seguito il risultato di tale modifica.

```
verbosity=1
console=both
overlay_prefix=sun50i-h5
overlays=i2c0 i2c1 i2c2 spi-spidev
rootdev=UUID=6342048c-723e-476d-a785-c345ff4c7099
rootfstype=ext4
#overlays=spi-spidev
param_spidev_spi_bus=1
usbstoragequirks=0x2537:0x1066:u,0x2537:0x1068:u
```

Fig.3.8: contenuto del file armbianEnv.txt: togliendo l'asterisco verrebbe eseguito solo il secondo comando.

Una volta operate le opportune modifiche, i comandi della gpio visti in precedenza risulteranno funzionare correttamente.

3.5.1 GPIO.C

In relazione al modulo spidev, caricato con overlays, vi sono alcune voci che devono essere modificate nella cartella gpio al file gpio.c.
Il modulo indicato nel file è spidev0.1 anziché spidev1.0; per trovare tutte le occorrenze, come per il caso wiringpi, basta digitare ctrl+f.

```
module1 = "spidev" ;
module2 = "spi-sun7i" ;
file1  = "/dev/spidev1.0" ;
// file2  = "/dev/spidev0.1" ;  è presente solo il modulo spidev1.0
```

Fig.3.9: esempio di sostituzione del parametro nel file gpio.c

3.6 INSTALLAZIONE DI SPIDEV TEST

A differenza della periferica I^2c, per cui c'è il comando gpio i2cdetect, per quella Spi non esiste una reale verifica di connessione, dal momento che , come riferito al capitolo 1, non funziona attraverso indirizzi.
L'unica azione che è possibile svolgere è volta al solo controllo dei pin della gpio, creando una linea di loop test che metta in comunicazione il canale MISO con il MOSI; questi ultimi devono essere collegati fisicamente per mezzo di un cavo, come riportato nella seguente figura:

Fig. 3.10: il cavo nero è collegato in modo da formare
un anello chiuso sui pin MISO e MOSI

A questo proposito esistono vari programmi atti a questa verifica.
Di seguito viene riportato l'output rilasciato dal programma scaricabile da terminale con il comando wget https://raw.githubusercontent.com/raspberrypi/linux/rpi-3.10.y/Documentation/spi/spidev-test.c ed eseguibile tramite ./spidev_test -D/dev/spidev 1.0.

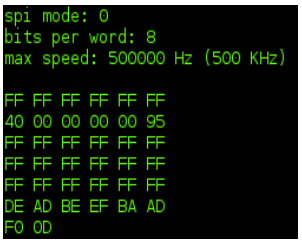


Fig. 3.11: output del programma spidev test

Tramite il canale MISO viene inviato un insieme di bit scritti in esadecimale che vengono rilevati dal canale MOSI e scritti sul terminale.
Se si verifica qualche malfunzionamento l'output sarà rappresentato da una stringa di zeri, mentre se tutto procede correttamente in uscita si avranno gli stessi valori inviati (fig. (3.11))
Questo programma stampa anche la massima velocità di clock, espressa in Hz, dell'Orangepi.

3.7 INSTALLAZIONE PROGRAMMI BASE TESI

3.7.1 PROGRAMMA DI SCRITTURA GEDIT

Il software Armbian è sprovvisto di un editor di testo per la scrittura e lettura dei file. Per realizzare il testo relativo al codice in "C", esposto nel quarto capitolo di questo elaborato, è stato utilizzato il programma gedit.
Esso è compatibile con lo standard Unix e include una sintassi suddivisa in base al colore e le schede per la modifica di più file contemporaneamente[29].
L'installazione viene fatta attraverso il terminale digitando nella linea di terminale il codice:

$ sudo apt install gedit

[29] Descrizione tratta da https://it.wikipedia.org/wiki/Gedit

3.7.2 I²C TOOLS

Attraverso il comando i2c-detect è possibile la lettura del dispositivo collegato alla linea 0, cioè a sda.0 e scl.0: se si connette un ulteriore dispositivo ai pin sda.1 e scl.1, che richiedono il modulo i2c1, non è possibile rilevare il suo indirizzo.
È consigliato l'installazione di un ulteriore strumento per la rilevazione di indirizzi dei sensori, con output uguali a quelli proposti dalla libreria wiringpi.
Per l'installazione si procede sempre attraverso il comando apt install

$ sudo apt install i2ctools

3.7.3 COMUNICAZIONE IPV4

Nel capitolo successivo sarà proposta una comunicazione Client-Server per lo scambio dei dati dell'accelerometro. La compilazione del programma risulterà impossibilitata dal fatto che l'orangepi è mancante di una struttura che crei una connessione TCP[30], fondamentale per il trasferimento dati.
Il sistema indicherà direttamente i file necessari da installare sempre attraverso apt install.

$ sudo apt install ucspi-tcp

3.8 ANALISI DELLA LIBRERIA WIRINGPI

3.8.1 FUNZIONI DELLA WIRINGPI

La libreria wiringpi offre innumerevoli funzioni, che permettono di operare tra il livello fisico della gpio e il sensore.

Prima di tutto, se si vuole utilizzare tale libreria in un programma "C", basta includere nelle sorgenti la seguente espressione : #include <wiringPi.h>; nella compilazione manuale del codice, tramite terminale, invece, si deve aggiungere, al termine della stringa, la dicitura -lwiringPi -lpthread, mentre nel makefile le librerie possono essere linkate con -l/usr/local/include -l/usr/local/lib -lwiringPi -lpthread.
#include <wiringpiI2C.h>,#include <wiringSPI.h> permettono di accedere alle funzioni per la gestione dei protocolli rispettivamente I²c e Spi.

Come appreso nel primo capitolo, per inizializzare il bus di comunicazione Orangepi-sensore devo fornire dei dati al sistema: nel caso I²c deve essere inserito l'indirizzo fisico dello Slave, nel caso dello Spi bisogna immettere sia la frequenza di clock che il valore di canale (0 oppure 1) .

Questo compito viene svolto dalle funzioni wiringPiI2cSetup e wiringPiSpiSetup.

[30] Riferimento al quarto Capitolo, comunicazione Client-Server

- int wiringPiI2cSetup(int devid) : inizializza il sistema I^2c con l'indirizzo fisico del sensore; attraverso ulteriori funzioni riceverà il numero di revisione dell'orangepi, così da poter aprire il modulo I^2c corretto contenuto in /dev/ (sottocartella del filesystem). L'accesso alle informazioni hardware è possibile grazie alla correzione descritta in questo capitolo.
  Il valore restituito all'utente sarà un descrittore di file standard di Linux, oppure -1 nel caso in cui si sia verificato un errore.

- int wiringPiSpiSetup(int channel, int clock) : per mezzo dei due argomenti, permette l'inizializzazione di un canale (o lo 0 oppure l'1), e di impostare la velocità di clock, espressa in Hz. Anche in questo caso viene restituito -1 se si è verificato un errore.

La sostanziale differenza dei bus sta nella scrittura e nella lettura dei dati; per il canale I^2c vi sono numerose funzioni, mentre per il canale Spi c'è ne solo una.

- int wiringPiI2CWriteReg8(int fd, int reg, int dati) : permette la scrittura dei registri inviando pacchetti da 8 bit. Esiste la versione anche da 16bit: basta operare una semplice sostituzione tra 8 e 16, cioè scrivere int wiringPiI2CWriteReg16(int fd, int reg, int dati).
  Sia per la lettura che per la scrittura, vi è una struttura dati, struct in "C", denominata i2c_smbus_data che lavora a livello fisico per il trasferimento dei dati.

- int wiringPiI2CReadReg8(int fd, int reg): consente la lettura dei registri ricevendo dati in pacchetti da 8 bit. Anche in questo caso esiste la versione da 16bit (int wiringPiI2CReadReg16(int fd, int reg)).

- Nel caso in cui il sensore non presenti la necessità di accedere ai registri interni si possono utilizzare int wiringPiI2cRead(int fd) per lettura oppure int wiringPiI2cWrite(int fd, int dati) per la scrittura.

Per il bus Spi la situazione risulta essere più complessa poiché , avendo a disposizione una sola funzione per scrivere e leggere da accelerometro, si introduce la sovrascrittura.
Per spiegare int wiringPiSPIDataRW (int channel, unsigned char *data, int len) è necessario semplificare il problema suddividendo le operazioni svolte in due parti distinte.

```c
uint8_t leggiBytebuffer( uint8_t reg )
{
    uint8_t tx [2];
    uint8_t rx [2];

    struct spi_ioc_transfer spi;

   tx[0] = reg;
   tx[1] = 0  ;


   spi.tx_buf = (unsigned long) tx;
   spi.rx_buf = (unsigned long) rx;

   spi.len = 2;

   spi.delay_usecs = spiDelay;
   spi.speed_hz = spiSpeed;
   spi.bits_per_word = spiBPW;

   if( ioctl(spi_fd, SPI_IOC_MESSAGE(1),&spi) == -1 )
   {
     printf(" errore nella trasmissione dei dati\n");
     exit(1);
   }

   return rx[1];
}
```

(a)

```c
void scriviBytebuffer( uint8_t reg, uint8_t data )
{
   uint8_t spiBufTx[2];
   uint8_t spiBufRx[2];

   struct spi_ioc_transfer spi;

   spiBufTx[0] = reg;
   spiBufTx[1] = data;

   spi.tx_buf = (unsigned long ) spiBufTx;
   spi.rx_buf = (unsigned long ) spiBufRx;

   spi.len = 2;

   spi.delay_usecs = spiDelay;
   spi.speed_hz = spiSpeed;
   spi.bits_per_word = spiBPW;

   if( ioctl(spi_fd, SPI_IOC_MESSAGE(1),&spi) == -1 )
   {
     printf(" errore nella trasmissione dei dati\n");
     exit(1);
   }
}
```

(b)

Fig.3.12: (a) programma “C” che simula la lettura della funzione wiringPiSPIDataRW. (b) programma “C” che simula la scrittura della funzione wiringPiSPIDataRW.

Questi codici rappresentano in maniera estesa ciò che svolge wiringPiSPIDataRW; in entrambi i casi si utilizza una struct della libreria <linux/spi/spidev.h> formata in questo modo:

```c
struct spi_ioc_transfer
  {
    __u64 tx_buf;
    __u64 rx_buf;
    __u32 len;
    __u32 speedhz;
    __u16 delay_usecs;
    __u8  sbits_per_word;
    __u8  cs_change;
    __u32 pad;
  }
```

Fig.:3.13 Struttura dati su cui si basa wiringPiSPIDataRW

Vi sono due buffer, uno adibito alla trasmissione, ed un altro alla ricezione. Una volta salvati i dati nel buffer di trasmissione, il contenuto viene inviato a tx_buf della struct, che spedisce sul canale ricevente l’informazione.
Passando alla versione ridotta della funzione della Wiringpi, viene usato un solo vettore. È richiesto che l’array data contenga due elementi: il registro e il dato da inviare; questo implica che il secondo elemento, sia nella scrittura, ma soprattutto nella lettura, sarà sovrascritto. Quindi, in ricezione, data conterrà nuovamente il registro, ma anche la nuova informazione.

```
int wiringPiSPIDataRW (int channel, unsigned char *data, int len)
{
  struct spi_ioc_transfer spi ;

  channel &= 1 ;

  spi.tx_buf        = (unsigned long)data ;
  spi.rx_buf        = (unsigned long)data ;
  spi.len           = len ;
  spi.delay_usecs   = spiDelay ;
  spi.speed_hz      = spiSpeeds [channel] ;
  spi.bits_per_word = spiBPW ;

  return ioctl (spiFds [channel], SPI_IOC_MESSAGE(1), &spi) ;
}
```

Fig.:3.14 Funzione wiringPiSPIDataRW tratta dalla libreria Wiringpi

La wiringPiSPIDataRW assicura un controllo di eventuali errori: restituisce -1 se l'operazione non è andata a buon fine, oppure la grandezza del vettore inviato.

La libreria possiede delle funzioni correlate alla sincronizzazione.
Unsigned int Milis(void) e Unsigned int micros(void) restituiscono rispettivamente il numero in millisecondi o microsecondi da quando il programma fa una chiamata ad una delle funzioni della WiringPiSetup.
Dall'altro lato, se si vuole bloccare l'esecuzione del programma per un certo istante di tempo vi sono void delay(unsigned int HowLong) e void delayMicroseconds(unsigned int HowLong): nel primo caso il ritardo è espresso in millisecondi mentre nel secondo in microsendi.
Come riportato nel sito http://wiringpi.com/reference/ il tempo massimo di ritardo possibile è dato da 49 giorni nel primo caso, mentre 71 minuti del secondo caso.

Per quanto riguarda il controllo dei pin della gpio la libreria offre le seguenti funzioni:

- void pinMode (int pin, int mode) : imposta la modalità del pin secondo il parametro mode che può assumere i valori INPUT , OUTPUT[31].
- void digitalWrite (int pin , int value): Imposta il pin, selezionato come uscita, al valore indicato da value, che può assume i valori HIGH o LOW (1 oppure 0). Il sistema interpreterà value come HIGH per qualunque numero diverso da zero, non soltanto l'uno.
- int digitalRead (int pin): Questa funzione restituisce il valore letto su pin. Sarà HIGH o LOW a seconda del livello logico al pin.

[31] Alla pagina http://wiringpi.com/reference/raspberry-pi-specifics/ risultano essere descritte ulteriori funzioni non utilizzate in questo elaborato.

CAPITOLO 4

4.PROGRAMMA “C” PER L’ACQUISIZIONE DATI DA ACCELEROMETRO

4.1.1 VISUALIZZAZIONE DATI DA ACCELEROMETRO

4.1.1 INTRODUZIONE

Per realizzare il programma relativo all’acquisizione dati da accelerometro, è stato utilizzato il materiale presente sul data-sheet[32] dello strumento, da cui sono state ricavate le informazioni riguardanti i registri.
Il sito della adafruit fornisce delle librerie base per accelerometro, scaricabili da https://learn.adafruit.com/adafruit-lis3dh-triple-axis-accelerometer-breakout/arduin; questi programmi, essendo usati per operare sulla piattaforma Iot Arduino, sono stati elaborati in modo da lavorare su Orangepi ed essere compatibili con la Wiringpi.
Dal momento che esistono due periferiche diverse, la I^2c e la Spi, sono stati prodotti due programmi differenti. In entrambi è possibile distinguere tre sezioni specifiche che sono: “inizializzazione del dispositivo”, “inizializzazione dei registri” e “ lettura dati da accelerometro”.

4.1.2 INIZIALIZZAZIONE DEL DISPOSITIVO

In questa prima parte vengono verificati i parametri hardware della piattaforma contenuti nel file /proc/cpuinfo, il quale è stato prontamente modificato come indicato nel capitolo precedente.
L’analisi viene effettuata dalla funzione di Setup delle due periferiche.
Per quanto riguarda la I^2c viene scritto:

```
fd = wiringPiI2CSetup(LIS3DH_DEFAULT_ADDRESS);
```

Fig.4.1: funzione wiringPiI2CSetup che prende in esame l’indirizzo dell’accelerometro

Alla funzione wiringPiI2CSetup viene passato il parametro LI2S3DH_DEFAULT_ADDRESS, che contiene l’indirizzo del dispositivo.
Alla fine del controllo viene restituito un valore numerico; salvandolo nella variabile fd è possibile verificare con un blocco if se si sono verificati degli errori nella lettura dei dati dell’orangepi: se tutto procede correttamente sarà restituito un numero intero maggiore di -1.
Questo procedimento viene seguito anche nel caso dello Spi: la differenza sta nella diversa tipologia di funzione di Setup.

```
fd = wiringPiSPISetup(channel, clock);
```

Fig.4.2: funzione wiringPiSpiSetup che prende in esame il canale e la frequenza di clock

[32] https://www.st.com/resource/en/datasheet/lis3dh.pdf

Come descritto nel terzo capitolo, vengono richiesti due parametri: il canale del collegamento sensore-Orangepi, il quale può assumere valore 0 oppure 1, e la frequenza del clock dell'Orangepi, che deve rientrare nei limiti di quella dell'accelerometro[33].
Per questa tesi sono stati assunti i seguenti valori:

- Valore Channel 0 definito dal pin CE0[34] (a cui è collegato il Selection Pin dell'accelerometro);

- Valore Clock 500000 Hz ottenuto dallo speed-test, come mostrato nel capitolo precedente.

4.1.3 REGISTRI DELL'ACCELEROMETRO LIS3DH

L'accelerometro LIS3DH è composto da un insieme di registri, i quali devono essere scritti oppure letti al fine di un corretto utilizzo del dispositivo.
Vi sono i registri che garantiscono l'abilitazione delle funzionalità base: CTRL_REG1, CTRL_REG4, CTRL_REG3, REG_TEMP_CFG, REG_WHO_AM_I.
Di seguito viene riportata una descrizione sommaria delle loro caratteristiche principali.

- REG_CTRL1: è un registro legato al consumo di potenza, alla risoluzione dati e abilitazione assi.
  I primi 4 bit più significativi, denominati ODR[3:0], stabiliscono la modalità di consumo che può essere bassa, media o alta. Il caso particolare è dato dalla combinazione 0000 che corrisponde ad una sua disabilitazione.
  Gli altri 4 bit servono, rispettivamente, a regolare la modalità di risoluzione e attivare gli assi x, y, z.

ODR3	ODR2	ODR1	ODR0	LPen	Zen	Yen	Xen
Consumo di potenza dato dai valori: • 0000: modalità di spegnimento; • Da 0001 a 0001: modalità a scelta tra alto, medio e basso consumo a diverse frequenze; • 1000: basso consumo; • 1001: Alto e medio consumo a frequenze differenti da basso consumo				Risoluzione	Abilitazione Asse z	Abilitazione Asse y	Abilitazione Asse x

Tab.4.1 Valori principali del registro REG_CTRL1

[33] Tale informazione è reperibile sul data-sheet dell'accelerometro
[34] Nome del pin 24 dell'Orange-pi, valore assegnato nel "codice C" 10 (vedi fig. 3.3 cap. 3)

- REG_CTRL4: consente di determinare la modalità di lettura dati dell'accelerometro. Ad esempio, ponendo ad 1 o a 0 la sezione BLE, si imposta la disposizione dei byte secondo il Big Endian e il Little Endian: nel primo caso si parte dal bit più significativo per arrivare quello meno significativo, mentre nel secondo caso succede l'inverso.
  Si evidenzia, inoltre, la funzione del bit SIM che per il valore basso seleziona per il protocollo Spi una lettura a 4 fili, mentre per il valore altro 3 fili: l'accelerometro LIS3DH comunica con l'Orangepi utilizzando i canali MISO per la trasmissione, MOSI per la ricezione, CS per la selezione del dispositivo e CLK per il clock.
  Il registro garantisce la scelta del massimo range di accelerazione, da un minimo di $\pm 2g$ a un massimo $\pm 16g$, come indicato in tab.4.2.

BDU	BLE	FS1	FS0	HR	ST1	ST0	SIM
Aggiornamento dati	Disposizione bit all'interno dei byte	Selezione del range di accelerazione. • 00: $\pm 2g$ • 01: $\pm 4g$ • 10: $\pm 8g$ • 11: $\pm 16g$		Abilitazione risoluzione output	Self Test		Spi 3-Wire 4-Wire

Tab.4.2 Valori principali del registro REG_CTRL4

- REG_CTRL3: questo registro permette di abilitare Int1 come pin di interrupt per segnalare diversi eventi.
  Come descritto nel primo capitolo, i pin di interrupt servono a comunicare un cambiamento di alcune condizioni del dispositivo al master.
  Il registro REG_CTRL6 svolge lo stesso compito, a differenza del fatto che viene attivato il pin Int2.

I1_CLICK	I1_IA1	I1_IA2	I1_ZYXDA	I1_321DA	I1_WTM	I1_OVERRUN	--
Funzione click	Pin IA1	Pin IA2	Variazione sui tre assi x, y , z	Variazione su tre assi 1, 2 ,3	FIFO programmabile watermark level	FIFO overrun	

Tab.4.3 Valori principali del registro REG_CTRL3

- TEMP_CFG_REG: registro che consente di abilitare la rilevazione di temperatura da parte dell'accelerometro e i pin A1, A2 e A3 (ADC) per il secondo convertitore analogico-digitale.

ADC_EN	TEMP_EN	0	0	0	0	0	0
Abilita ADC	Abilita il sensore di temperatura						

Tab.4.4 Valori principali del registro TEMP_CFG_REG

- LIS3DH_WHO_AM_I: registro che serve a verificare che l'accelerometro sia collegato correttamente alla piattaforma Iot. Esso contiene l'indirizzo di una cella fantoccio "dummy" che restituisce il valore esadecimale 0x33 se il controllo risulta essere andato a buon fine.

0	0	1	1	0	0	1	1

Tab.4.4: Valori del registro WHO_AM_I

I registri costituiscono le basi per il controllo diretto dell'accelerometro e devono essere scritti opportunamente al fine di abilitare le funzioni volute.
REG_CTRL3 e TEMP_CFG_REG consentono di attivare le funzionalità legate ai pin di interrupt e ai pin A1, A2 e A3. Da essi dipendono tutta una serie di ulteriori registri, la cui funzione, però, non sarà trattata in questo elaborato.

Ve ne sono altri da cui è possibile effettuare solo la lettura e da cui vengono ottenuti i valori di accelerazione lungo i tre assi.
Questi registri sono: LIS3DH_REG_OUT_X_L, LIS3DH_REG_OUT_X_H, LIS3DH_REG_OUT_Y_L, LIS3DH_REG_OUT_Y_H, LIS3DH_REG_OUT_Z_L, LIS3DH_REG_OUT_Z_H.
Per ogni asse i dati sono restituiti come valore alto e come valore basso: combinandoli assieme in maniera opportuna è possibile ottenere l'accelerazione misurata. Il datasheet inoltre riporta che i registri restituiscono i risultati in complemento a 2, con i registri H contenenti i bit più significativi.

Di seguito vengono riportati le caratteristiche di tutti i registri che compongono l'accelerometro:

Name	Type	Register address		Default	Comment
		Hex	Binary		
Reserved (do not modify)		00 - 06			Reserved
STATUS_REG_AUX	r	07	000 0111	Output	
OUT_ADC1_L	r	08	000 1000	Output	
OUT_ADC1_H	r	09	000 1001	Output	
OUT_ADC2_L	r	0A	000 1010	Output	
OUT_ADC2_H	r	0B	000 1011	Output	
OUT_ADC3_L	r	0C	000 1100	Output	
OUT_ADC3_H	r	0D	000 1101	Output	
Reserved (do not modify)		0E			Reserved
WHO_AM_I	r	0F	000 1111	00110011	Dummy register
Reserved (do not modify)		10 - 1D			Reserved
CTRL_REG0	rw	1E	001 1110	00010000	
TEMP_CFG_REG	rw	1F	001 1111	00000000	
CTRL_REG1	rw	20	010 0000	00000111	
CTRL_REG2	rw	21	010 0001	00000000	
CTRL_REG3	rw	22	010 0010	00000000	
CTRL_REG4	rw	23	010 0011	00000000	
CTRL_REG5	rw	24	010 0100	00000000	
CTRL_REG6	rw	25	010 0101	00000000	
REFERENCE	rw	26	010 0110	00000000	
STATUS_REG	r	27	010 0111	Output	
OUT_X_L	r	28	010 1000	Output	
OUT_X_H	r	29	010 1001	Output	
OUT_Y_L	r	2A	010 1010	Output	
OUT_Y_H	r	2B	010 1011	Output	
OUT_Z_L	r	2C	010 1100	Output	
OUT_Z_H	r	2D	010 1101	Output	
FIFO_CTRL_REG	rw	2E	010 1110	00000000	
FIFO_SRC_REG	r	2F	010 1111	Output	

Name	Type	Register address		Default	Comment
		Hex	Binary		
INT1_CFG	rw	30	011 0000	00000000	
INT1_SRC	r	31	011 0001	Output	
INT1_THS	rw	32	011 0010	00000000	
INT1_DURATION	rw	33	011 0011	00000000	
INT2_CFG	rw	34	011 0100	00000000	
INT2_SRC	r	35	011 0101	Output	
INT2_THS	rw	36	011 0110	00000000	
INT2_DURATION	rw	37	011 0111	00000000	
CLICK_CFG	rw	38	011 1000	00000000	
CLICK_SRC	r	39	011 1001	Output	
CLICK_THS	rw	3A	011 1010	00000000	
TIME_LIMIT	rw	3B	011 1011	00000000	
TIME_LATENCY	rw	3C	011 1100	00000000	
TIME_WINDOW	rw	3D	011 1101	00000000	
ACT_THS	rw	3E	011 1110	00000000	
ACT_DUR	rw	3F	011 1111	00000000	

Fig.: 4.3 Tabella tratta da https://www.st.com/resource/en/datasheet/lis3dh.pdf , p.31-32, raffigurante le caratteristiche principali dei registri dell'accelerometro

4.1.4 INIZIALIZZAZIONE DEI REGISTRI

In questa sezione bisogna operare ad una distinzione tra le due periferiche I^2c e Spi: anche se i parametri sono gli stessi, nel secondo caso ho necessità di creare un ulteriore funzione per soddisfare i requisiti richiesti da questo protocollo.

4.1.4.a PROTOCOLLO I^2C

La scrittura dei registri avviene utilizzando la funzione descritta nel capitolo precedente, messa a disposizione dalla Wiringpi, la wiringPiI2CWriteReg8.
Per l'inizializzazione, viene passato il valore in esadecimale dell'indirizzo del registro e il dato; dal valore di quest'ultimo dipendono le funzionalità che si vogliono abilitare all'accelerometro. Di seguito viene riportata una breve descrizione dei valori passati alla funzione:

```
wiringPiI2CWriteReg8(fd,LIS3DH_REG_CTRL1, 0x07);
```

(a)

```
wiringPiI2CWriteReg8(fd,LIS3DH_REG_CTRL4, 0x88);
```

(b)

```
wiringPiI2CWriteReg8(fd,LIS3DH_REG_CTRL3, 0x10);
```
(c)
```
wiringPiI2CWriteReg8(fd,LIS3DH_REG_TEMPCFG, 0x80);
```
(d)

Fig. 4.4:(a) scrittura del registro REG_CTRL1 con il dato 0000 0100; (b) scrittura del registro REG_CTRL4 con il dato 1111111; (c) scrittura del registro REG_CTRL3 con il dato 00010000; (d) scrittura del registro REG_TEMP_CFG con il dato 10000000

LIS3DH_REG_CTRL1, LIS3DH_REG_CTRL4, LIS3DH_REG_CTRL3, LIS3DH_REG_TEMPCFG rappresentano le variabili contenenti gli indirizzi dei registri presentati in fig. 4.3.
Riprendendo l'analisi dei registri del precedente paragrafo, vengono attivate le seguenti funzioni:

- Attraverso REG_CTRL1 sono abilitati gli assi x,y,z;
- Per il REG_CTRL4 sono abilitate tutte le funzionalità;
- Attraverso REG_CTRL3 si seleziona l'interrupt per le variazioni sugli assi x,y,z;
- Attraverso REG_TEMP_CFG viene attivato l'ulteriore convertitore analogico/digitale;

REG_WHO_AM_I richiede una lettura; si utilizza la funzione WiringPiReadReg8, che restituisce un valore in esadecimale lungo 8 bit, dopo aver letto l'indirizzo di tale registro.
Il valore può essere salvato in una variabile e, come fatto per la funzione di Setup, inserire la condizione in un blocco if, per verificare la correttezza del dato.

```
void checkconnection(int fd)
{
    uint8_t deviceid = wiringPiI2CReadReg8(fd, LIS3DH_REG_WHOAMI);

   if(deviceid != 0x33)
   {
      printf(" nessuna connessione con il dispositivo\n " );
      exit(1);
   }

}
```

Fig. 4.5: Struttura per verificare che la variabile letta sia 0x33; se il sensore non è collegato correttamente viene restituito un valore diverso.

Infine, sono state costruite ulteriori funzioni per fissare la velocità di trasmissione dei dati e la massima escursione del segnale.

Il registro da modificare per variare il data-rate è il REG_CTRL1. La seguente funzione imposta la velocità di campionamento dell'accelerometro ad un valore prefissato.
Come riportato nell'appendice B, nella libreria è presente una struttura dati, denominata lis3dh_dataRate_t, che contiene diverse velocità: a seconda del parametro passato si possono scegliere velocità comprese tra 1 Hz e 400 Hz.

```
void setDataRate(int fd, lis3dh_dataRate_t dataRate )
{
   uint8_t ctl1 = wiringPiI2CReadReg8(fd, LIS3DH_REG_CTRL1);
   ctl1 &= ~(0xF0);
   ctl1 |= ( dataRate << 4 );
   wiringPiI2CWriteReg8(fd,LIS3DH_REG_CTRL1, ctl1);
}
```

(a)

```
typedef enum
{
  LIS3DH_DATARATE_400_HZ     = 0b0111, //  400Hz
  LIS3DH_DATARATE_200_HZ     = 0b0110, //  200Hz
  LIS3DH_DATARATE_100_HZ     = 0b0101, //  100Hz
  LIS3DH_DATARATE_50_HZ      = 0b0100, //   50Hz
  LIS3DH_DATARATE_25_HZ      = 0b0011, //   25Hz
  LIS3DH_DATARATE_10_HZ      = 0b0010, // 10 Hz
  LIS3DH_DATARATE_1_HZ       = 0b0001, // 1 Hz
  LIS3DH_DATARATE_POWERDOWN  = 0,
  LIS3DH_DATARATE_LOWPOWER_1K6HZ  = 0b1000,
  LIS3DH_DATARATE_LOWPOWER_5KHZ  =  0b1001,

} lis3dh_dataRate_t;
```

(b)

Fig.4.6: (a) funzione per la modifica della velocità di trasmissione;
(b) struttura dati denominata lis3dh_dataRate_t

Nella funzione setDataRate, viene letto dal registro il valore preimpostato di velocità di trasmissione; successivamente, viene effettuato un AND tra tale valore ed il complemento a 1 del codice esadecimale 0xF0, così da ottenere una variabile composta da 0000dddd (dove d è un valore); lo shift del valore dataRate di 4 posizioni porta a forme del tipo dddb00: applicando un OR tra essa e il parametro precedentemente calcolato si ottiene la nuova velocità di trasmissione.

È possibile modificare anche la massima escursione del segnale, cioè l'intervallo entro il quale le informazioni di accelerazione vengono presentati. Il registro interessato è il REG_CTRL4: i valori vanno da un'ampiezza di $\pm 2g$ a $\pm 16g$. Anche in questo caso vi è una struttura dati, chiamata lis3dh_range_t, che riporta tutti i range di accelerazione.

```
void setRange(int fd, lis3dh_range_t range )
{
    uint8_t r = wiringPiI2CReadReg8(fd,LIS3DH_REG_CTRL4);
    r &= ~(0x30);
    r |= range << 4;
    wiringPiI2CWriteReg8(fd,LIS3DH_REG_CTRL4,r);
}
```

(a)

```
typedef enum
{
  LIS3DH_RANGE_16_G         = 0b11,   // +/- 16g
  LIS3DH_RANGE_8_G           = 0b10,   // +/- 8g
  LIS3DH_RANGE_4_G           = 0b01,   // +/- 4g
  LIS3DH_RANGE_2_G           = 0b00    // +/- 2g (default value)
} lis3dh_range_t;
```

(b)

Fig.4.7: (a) funzione per la modifica del range di accelerazione;
(b) struttura dati denominata lis3dh_range_t.

Analogamente al caso precedente, anche in questo vengono effettuate operazioni simili per ottenere un nuovo valore di range.

Una volta impostati i valori è possibile leggerli mediante le seguenti funzioni:

```
lis3dh_dataRate_t getDataRate(int fd)
{
    return (lis3dh_dataRate_t)((wiringPiI2CReadReg8(fd, LIS3DH_REG_CTRL1) >> 4 )& 0x0F);
}
```

(a)

```
lis3dh_range_t getRange(int fd)
{
    return (uint8_t)((wiringPiI2CReadReg8(fd,LIS3DH_REG_CTRL4) >> 4) & 0x03);
}
```

(b)

Fig.4.8: (a) funzione per ottenere la velocità di trasmissione;
(b) funzione per ottenere il range di accelerazione.

4.1.4.b PROTOCOLLO SPI

Il codice realizzato per questa periferica è simile a quello dell'interfaccia I^2c.
Quello che risulta essere differente sono le funzioni di base della libreria Wiringpi.
In questa parte i registri e i valori assegnati risultano essere gli stessi rispetto al caso precedente: aumenta la complessità del codice dal momento che la funzione wiringPiSPIDataRW richiede un numero maggiore di dati rispetto a wiringpiI2CreadReg8 o wiringpiI2CwriteReg8.
Prima di una lettura e di una scrittura è necessario portare il pin CE0 basso e rialzarlo al termine.

```
void write_register_spi(uint8_t reg, uint8_t value)
{
    uint8_t tx_buff[2];

    tx_buff[0] = reg & ~0x80;
    tx_buff[1] = value;
    digitalWrite(CE0, LOW);

    if(wiringPiSPIDataRW(channel, tx_buff , 2) == -1 )
    {
        printf("errore nella trasmissione dati");
        exit(1);
    }

    digitalWrite(CE0, HIGH);
}
```

(a)

```
uint8_t read_register_spi(uint8_t reg)
{
    uint8_t tx_buff[2];

    tx_buff[0] = reg | 0x80;
    tx_buff[1] = 0x00;

    digitalWrite(CE0, LOW);

    if(wiringPiSPIDataRW(channel, tx_buff , 2) == -1 )
    {
        printf("errore nella trasmissione dati");
        exit(1);
    }

    digitalWrite(CE0, HIGH);

    return tx_buff[1];
}
```

(b)

Fig.4.9: (a) funzione di scrittura per il protocollo Spi; (b) funzione di lettura per il protocollo Spi

Per i motivi elencati, sono state costruite due funzioni apposite write_register_spi e read_register_spi. Le due funzioni sono state opportunamente separate (per spiegare quando detto nel capitolo precedente per quanto riguarda wiringPiSPIDataRW) .
Le operazioni effettuate sul valore esadecimale dei registri in figura 4.9 (a) e (b), sono tali da porre il primo bit a 0 o a 1, a seconda che si voglia scrivere o leggere; i primi sette bit contengono l'indirizzo, il rimanente l'operazione da svolgere (come spiegato nel primo capitolo) .

I registri e i parametri visti per il protocollo I^2c risultano essere gli stessi: a REG_CTRL1, REG_CTRL4, REG_CTRL3 e REG_TEMP_CFG vengono rispettivamente passati i valori esadecimali 0x07, 0x88, 0x10, 0x80; anche in questo caso è necessario controllare il corretto collegamento del sensore tramite il registro REG_WHO_AM_I e modificare REG_CTRL1, e REG_CTRL4 per impostare differenti velocità di trasmissione e differenti range di accelerazione.

4.1.5 LETTURA DATI DA ACCELEROMETRO

Per leggere i dati è necessario conoscere come vengono trasmessi al dispositivo e in quale unità di misura.
Per ogni asse l'informazione contenuta in due variabili costituiti da 8 bit ciascuna: vi è un valore alto che rappresenta i bit più significativi e un valore basso che rappresenta i bit meno significativi.
L'unità dei valori risulta essere g/LSB[35]: devono essere convertiti in int, e successivamente, divisi per un parametro deciso in base al range di accelerazione stabilita.
Di seguito, viene effettuata l'analisi per il protocollo I^2c; per lo Spi avviene la stessa cosa a differenza del fatto che, come visto nel paragrafo precedente, ho la necessità di utilizzare delle ulteriori funzioni intermedie.

```
x0 = wiringPiI2CReadReg8(fd, LIS3DH_REG_OUT_X_L); // valore basso
y0 = wiringPiI2CReadReg8(fd, LIS3DH_REG_OUT_Y_L);
z0 = wiringPiI2CReadReg8(fd, LIS3DH_REG_OUT_Z_L);
x1 = wiringPiI2CReadReg8(fd, LIS3DH_REG_OUT_X_H); // valore alto
y1 = wiringPiI2CReadReg8(fd, LIS3DH_REG_OUT_Y_H);
z1 = wiringPiI2CReadReg8(fd, LIS3DH_REG_OUT_Z_H);
```

Fig.4.10: I dati vengono letti dai registri e salvati in variabili da 8 bit ciascuna

```
*x = (short int)((((uint16_t) x1 ) << 8) | x0);
*y = (short int)((((uint16_t) y1 ) << 8) | y0);
*z = (short int)((((uint16_t) z1 ) << 8) | z0);
```

Fig.4.11: Unione della componente alta e bassa per ogni asse

I valori x1, y1, z1 vengono convertiti in interi da 16 bit e spostati verso destra di 8 posizioni attraverso uno shift; il segmento dati sarà composto da una parte "vuota" da 8 bit e una parte numerica da 8bit e, eseguendo un OR con gli altri dati, contenuti rispettivamente in x0, y0 e z0, si ottiene l'informazione di accelerazione espressa in g/LSB.
È stato utilizzato short int al posto int, poiché il secondo permette la scrittura dei soli valori positivi.

```
uint8_t range = getRange(fd);
float divider = 1;
if ( range == LIS3DH_RANGE_16_G)
    divider = 1365;
if ( range == LIS3DH_RANGE_8_G)
    divider = 4096;
if ( range == LIS3DH_RANGE_4_G)
    divider = 8190;
if ( range == LIS3DH_RANGE_2_G)
    divider = 16380;

buff[0] = ((float)*x) / divider;
buff[1] = ((float)*y) / divider;
buff[2] = ((float)*z) / divider;
```

Fig.4.12: Per ottenere i valori espressi in g devo dividere per un fattore dipendente dal range di accelerazione scelto

A seconda dell'accelerazione selezionata inizialmente, per ottenere una misura in g, cioè scritta in funzione dell'accelerazione di gravità, è necessario dividere i dati per un valore prefissato.

[35] LSB : bit meno significativo (Least significant Bit)

4.2 COMUNICAZIONE CLIENT-SERVER PER LO SCAMBIO DATI ACCELEROMETRO

4.2.1 INTRODUZIONE ALLE RETI DI CALCOLATORI

Per comprendere il funzionamento del programma per lo sviluppo di una comunicazione Client-Server è necessario operare ad una breve introduzione alle reti di calcolatori.
Una rete di calcolatori è formata da un insieme di elaboratori, come computer, router, che costituiscono i suoi nodi e risultano essere connessi tra di loro, scambiandosi informazioni.
Queste reti rappresentano un sistema complesso di comunicazione, per facilitare ciò si è operato ad una sua suddivisione in livelli o strati costruiti uno sull'altro.
Gli elaboratori sfruttano questi livelli per poter inviare dati nella rete. Ogni computer è costituito da un insieme di strati, che elaborano le informazioni.
La comunicazione tra livelli uguali di sistemi diversi è garantita da insiemi di regole chiamate protocolli; il libro reti di calcolatori di Andrew S.Tanenbaum e David J. Wetherall le definisce come " un accordo tra parti che comunicano sul modo in cui deve procedere la comunicazione".
All'utente finale viene presentato solamente l'ultimo livello, mentre gli altri risultano essere "nascosti".

Livelli	ISO/OSI
7	Applicazione
6	Presentazione
5	Sessione
4	Trasporto
3	Rete
2	Data-Link
1	Fisico

(a)

Livelli	TCP/IP
4	Applicazione
3	Trasporto
2	Internet
1	Accesso alla rete

(b)

Tab.4.5: (a) Struttura logica modello ISO/OSI;
(b) Struttura logica modello TCP/IP

La Tab. 4.5 (a) si riferisce alla struttura logica data alla rete dall' ISO/OSI, modello standard sviluppato dall'International Standard Organization.
Vi è un ulteriore modello che è stato sviluppato per Internet: il TCP/IP, riportato in Tab. 4.5(b); le principali differenze, rispetto all' ISO/OSI, sono che presenta solamente
quattro livelli, la sua struttura risulta essere poco utilizzata , ma i suoi protocolli sono largamente utilizzati.[36]
Definita la rete logica, e i protocolli di comunicazione, ognuno di questi livelli può "dialogare" con quelli superiori o inferiori, mediante operazioni elementari e servizi, che vengono descritti da opportune interfacce.
A loro volta i servizi si suddividono in: servizi orientati alla connessione e servizi non orientati alla connessione, affidabili e non affidabili.
Nei servizi orientati alla connessione prima si stabilisce una connessione e successivamente si inviano i dati. Questa tipologia di servizio risulta essere affidabile nel momento in cui viene richiesto un riscontro da parte del ricevitore/trasmettitore della corretta ricezione/trasmissione del pacchetto.

[36] Differenze espresse da Andrew S.Tanenbaum e David J. Wether in reti di calcolatori

Nei servizi non orientati alla connessione i messaggi vengono inviati in maniera indipendente gli uni dagli altri e può succedere che le informazioni arrivino in maniera disordinata: se non viene richiesta conferma dell'avvenuta ricezione di un dato, questa tipologia di servizio risulta essere poco affidabile; sono utilizzati, ad esempio, nelle comunicazioni Real-Time.
Affidabilità e non affidabilità di un servizio discendono dal controllo effettuato sui dati in transito: esistono servizi orientati alla connessione non affidabili, come servizi non orientati alla connessione affidabili.

Ogni servizio può essere interrogato mediante particolari entità chiamate primitive, operazioni che hanno un ruolo fondamentale nello scambio di informazioni tra livelli adiacenti.
Quando due livelli vicini di uno stesso elaboratore interagiscono, usano quattro tipi di primitive:

- Request: il livello superiore chiede determinati servizi a quello inferiore;
- Indicate: risposta del livello inferiore per il servizio richiesto;
- Response: completamento della procedura aperta da indicate;
- Confirm: primitiva inviata dal livello inferiore a quello superiore, la quale implica la fine della trasmissione.

4.2.2 LIVELLO DEL TRASPORTO

I primi tre livelli del modello ISO/OSI svolgono le funzioni di supporto fisico, controllo di flusso dati, instradamento dei pacchetti …,mentre il livello del trasporto permette di gestire l'affidabilità del transito dei messaggi. Esso è stato creato per compensare gli errori che possono accadere a livello della rete e garantire una comunicazione affidabile ed efficiente tra le parti coinvolte.
Il livello del trasporto fa uso di due protocolli che sono il TCP e l'UDP: il primo è un protocollo affidabile orientato alla connessione che garantisce la ricezione delle informazioni, senza errori; il secondo è un protocollo non affidabile, senza controllo degli errori.
Questo livello fornisce servizi efficienti, affidabili e convenienti alle applicazioni, mette a disposizione funzioni di libreria per i programmi applicativi.
Le primitive associate al trasporto sono visibili e quindi manipolabili direttamente dagli utenti.
Si prende in considerazione un sistema costituito da un server e un client remoto; vengono presentate le principali primitive[37] utilizzate nelle comunicazioni:

- LISTEN: il server si blocca in attesa di ricevere connessioni da parte dei client;
- CONNECT: primitiva eseguita dal client per connettersi al server;
- SEND/RECEIVE: permettono di inviare / ricevere dati da entrambi le parti; il sistema di trasferimento risulta essere molto complicato dal momento che ogni messaggio deve essere seguito da un acknowledgement e da possibili ritrasmissioni nel caso in cui i pacchetti vadano persi;
- DISCONNECT: disconnessione da parte del lato che ha usato tale primitiva.

Queste funzioni risultano essere molto importanti per spiegare ulteriori primitive appartenenti ai socket per TCP.

[37] Esempio di primitive tratto dal libro reti di calcolatori di Andrew S.Tanenbaum e David J. Wether

4.2.3 SOCKET: INTRODUZIONE

I socket, in informatica, sono dei programmi che permettono la comunicazione tra sistemi diversi.
Nei socket si possono distinguere due parti: il Server e i Client. L'apertura e la gestione del socket avvengono per mezzo del Server ; i Client, invece, inviano informazioni al Server.
Ad ogni socket può essere associato un solo Server, mentre i Client possono essere multipli.
Esistono tre tipologie di socket, rispettivamente dipendenti da come vengono gestite le comunicazioni:

- socket STREAM: viene fissata una connessione tra il client e il server, e successivamente vengono inviati i dati; si basa sul protocollo connesso TCP;
- socket UDP: i messaggi vengono inviati senza connessione e si basa sul protocollo non connesso UDP;
- socket RAW: il livello applicativo controlla la rete senza passare per il livello del trasporto.

A livello del trasporto i socket dispongo di primitive proprie. Esse saranno analizzate nel paragrafo successivo relativamente al codice "C".

4.2.4 TRASMISSIONE DATI CLIENT-SERVER DEI DATI DELL'ACCELEROMETRO

Per realizzare il programma "C" sono state utilizzate le librerie messe a disposizione dal professor Pier Luca Montessoro alla pagina http://web.diegm.uniud.it/pierluca/public_html/teaching/reti_di_calcolatori/reti_di_calcolatori.html.
Il Socket utilizzato nel progetto è di tipo Stream, cioè si è stabilita una connessione sfruttando il protocollo TCP.

4.2.4.a SOCKET LATO SERVER

Nell'apertura di un di un socket, la prima primitiva, che deve essere usata dal Server, è SOCKET: essa serve a creare il punto in cui deve essere stabilita la comunicazione e ad allocare una struttura dati per la memorizzazione delle informazioni riguardanti il nodo di connessione.
Alla primitiva SOCKET (figura 4.13) viene passato il parametro SOCK_STREAM per generare un socket di tipo STREAM, cioè basato sul protocollo TCP.

```
/* create a socket descriptor */
if ((sk = socket (AF_INET, SOCK_STREAM, 0)) < 0)
{
    error_handler ("socket() [create_tcp_server()]");
    return -1;
}
```

Fig.4.13: primitiva SOCKET con la dicitura socket(dominio, tipologia socket, protocollo).

Generato il socket, il Server può identificarsi sulla rete attraverso una porta ed un indirizzo di rete.

Per porta si intende un insieme di numeri utilizzati a livello del trasporto, i quali possono assumere un valore intero compreso tra 0 e 65535: rendono possibile l'instaurarsi di più connessioni su uno stesso elaboratore a programmi applicativi diversi.
Le porte possono essere distinte in due categorie principali: le prime 1024 porte (0-1023) sono definite well-known ports e risultano essere già allocate a servizi server standard; tutte le altre possono essere utilizzate senza vincoli.
L'indirizzo di rete è un'etichetta "assegnata" al calcolatore quando esso si connette alla rete.

Prima di associare un indirizzo di rete ed una porta al Server è necessario compilare una struttura dati.
L'indirizzo ip da passare come variabile è 0.0.0.0, dal momento che il socket deve essere generato sull'elaboratore preso in considerazione; con questa operazione, qualunque indirizzo di rete associato al computer verrà considerato come punto di accesso al Server.

```
/* fill the (used) fields of the socket address with
   the server information (local socket address) */
bzero ((char *) &srv, sizeof (srv));
srv.sin_family = AF_INET;
srv.sin_addr.s_addr = inet_addr (ip_address);
srv.sin_port = htons (port);
```

Fig. 4.14: Esempio della compilazione della struttura dati

La struct appena creata viene passata alla variabile BIND per l'identificazione del socket.
Porte e indirizzi di rete vengono attribuiti per mezzo di tale primitiva.

```
/* add the local socket address to the socket descriptor */
if (bind (sk, (struct sockaddr *) &srv, sizeof(srv)) < 0)
{
   error_handler ("bind() [create_tcp_server()]");
   return -1;
}
```

Fig.4.15: primitiva BIND con la dicitura bind(descrittore, "etichetta", lunghezza "etichetta").

Il Server usa la chiamata LISTEN per allocare dello spazio per memorizzare più richieste in arrivo.

```
/* get the next connection request from the queue */
if ((csk = accept (sk, (struct sockaddr *) &cln, &dimcln)) < 0)
{
   error_handler ("accept() [create_tcp_server()]");
   return -1;
}
```

(a)

```
/* make the socket a passive socket (enable the socket
   accepting connections) */
if (listen (sk, QUEUELEN) < 0)
{
   error_handler ("listen() [create_tcp_server()]");
   return -1;
}
```

(b)

Fig.4.16: (a) primitiva ACCEPT; (b) primitiva LISTEN con la dicitura listen(descrittore, dimensioni buffer)

Mentre con ACCEPT si mette in ascolto di possibili connessioni. Attraverso le ultime due primitive è possibile generare collegamenti multipli ai Client dal momento che, quando si stabilisce una connessione, l'entità del trasporto genera un nuovo socket con le stesse proprietà dell'originale, restituendo un descrittore di file.

4.2.4.b SOCKET LATO CLIENT

I client, prima di connettersi, devono creare una propria socket; come nel paragrafo precedente viene chiamata la primitiva SOCKET, da cui si ottiene un descrittore di file.
La differenza avviene nella compilazione dei parametri della struttura dati. In questo caso il Client deve inserire indirizzo di rete e porta del Server.
Successivamente, utilizzando i dati appena creati, effettua una chiamata alla primitiva CONNECT.

```
/* open a connection to the server */
if (connect (sk, (struct sockaddr *) &srv, sizeof(srv)) < 0)
{
    error_handler ("connect() [create_tcp_client_connection()]");
    return -1;
}
```

Fig.4.17: primitiva CONNECT per il lato Client con la dicitura connect(descrittore, "etichetta" Server, lunghezza "etichetta")

Al server sarà passato l'indirizzo-ip del Client, a cui verrà assegnata una nuova porta (diversa da quella utilizzata dal server per ascoltare) , e sarà stabilita una connessione.
In questo modo viene garantita la possibilità di connessioni multiple da parte dei Client al Server.

4.2.4.c TRASMISSIONE DATI ACCELEROMETRO

```
int tcp_send (int sk, char *buffer)
{
   if (write (sk, buffer, strlen(buffer)) != strlen(buffer))
   {
      error_handler ("write() [tcp_send()]");
      return 0;
   }

   return 1;
}
```

(a)

```
int tcp_receive (int sk, char *buffer)
{
   int dim, flags;

   if ((dim = read (sk, buffer, BUFSIZ)) < 0)
   {
      flags = fcntl (sk, F_GETFL, 0);
      if ((flags & O_NONBLOCK) == O_NONBLOCK)
      {   /* non-blocking mode */
         if ((errno == EAGAIN) || (errno == EWOULDBLOCK))
            return 0;
         else
         {
            error_handler ("read() [tcp_receive()]");
            return -1;
         }
      }
      else
      {
         error_handler ("read() [tcp_receive()]");
         return -1;
      }
   }

   buffer[dim] = '\0';
   return dim;
}
```

(b)

Fig.4.18: (a) primitiva SEND tramite write(descrittore, vettore dati, lunghezza vettore dati); (b) primitiva RECEIVE tramite read(descrittore, vettore dati, lunghezza vettore dati)

I dati possono essere inviati o ricevuti tramite le primitive SEND e RECEIVE.
Nel caso del socket UDP, in cui non viene stabilita una connessione, vi sono le primitive SENDTO e RECEIVEFROM, che permettono il corretto indirizzamento del messaggio Client-Server.
SEND può essere espresso sia con le funzioni send() che con write(); allo stesso modo RECEIVE si specifica mediante receive() e read(); è più comune l'utilizzo di write() e di read(), poiché molto somigliante alla scrittura su file, dove il file è costituito, in questo caso , dal socket.
Sapendo che i dati dell'accelerometro vengono salvati dal programma, descritto nel precedente capitolo, in variabili float e che quando si scrive sul socket, questo si comporta come un file, accettando solo stringhe, è stata operata una conversione di tali valori.

Una prima forma del programma prevedeva la trasmissione separata dei dati; questo causava la perdita di informazioni dal momento che il protocollo TCP, anche se affidabile, non era in grado di supportare la velocità di scrittura.
Il problema è stato risolto inglobando i dati in un'unica stringa. Una volta arrivata a destinazione, il Server analizza tale stringa ed estrae i valori lungo i tre assi.
Il relativo codice in "C" è riportato in Appendice A per il Server ed in Appendice B per il Client.

4.2.4.d CHIUSURA COMUNICAZIONE

Vi sono due strade per chiudere una connessione al livello del trasporto: una simmetrica, in cui entrambe le parti devono inviare la primitiva DISCONNECT per rilasciare la connessione, e una asimmetrica, in cui basta che o il Client o il Server invii tale primitiva.
Nel programma "C" è stata utilizzata la funzione close a cui è stato passato il descrittore creato quando è stata generata la connessione: la chiusura socket avviene sempre simmetricamente, quindi è stato fatto in modo che sia Client che Server invochino tale funzione.

4.3 STAMPA DATI

Per scopi puramente illustrativi, sia per il Server (Appendice A), sia per il Client (Appendice B), è stato scritto un codice "C" per visualizzare, in maniera semi-grafica e dinamica, i dati dell'accelerometro.

CONCLUSIONI E POSSIBILI FUTURI APPROFONDIMENTI

Attraverso il progetto “studio e sviluppo di un’applicazione Client-Server basata su accelerometri per dispositivi Iot”, è stato possibile capire il funzionamento dell’accelerometro e la sua possibile applicazione.
La scelta dell’Orangepi, anziché Arduino o Raspberrypi, ha introdotto diverse problematiche, le quali non si sarebbero presentate prendendo in considerazione le altre due piattaforme, dal momento che hanno avuto il tempo di evolversi.

In questa tesi non è stato possibile approfondire alcune tematiche legate alle caratteristiche interne dell’accelerometro come:

- funzionalità del secondo convertitore analogico-digitale e dei pin A1, A2, A3 ad esso collegati;
- funzionalità di temperatura;
- proprietà del buffer FIFO;
- funzionalità dei pin di Interrupt.

Per quanto riguarda l’Orangepi è stato solo trattato superficialmente il ruolo dell’interfaccia UART nella connessione con altre piattaforme.

La struttura del programma “C”, presentata nel quarto capitolo di questo elaborato, è possibile estenderla ad ulteriori strumenti di misurazione MEMS, ad esempio, a sensori di movimento come giroscopi, o ad altri sensori per il rilevamento di parametri ambientali, come pressione, temperatura, suono … ; in ognuno di questi dispositivi è possibile riconoscere una composizione formata da pin e da registri, i quali, come nel caso dell’accelerometro, devono essere opportunamente configurati per il corretto funzionamento del sistema.

APPENDICE A

In questa appendice viene riportato il codice “C” relativo al Server. I file contengono rispettivamente il main, il corpo della funzione, la libreria

SERVER.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>

#include "serverlib.h"

int main (int argc, char *argv[])
{
  if (argc != 3)
  {
    fprintf (stderr, "required arguments: server_ip_address tcp_port\n");
    exit (EXIT_FAILURE);
  }

  if (create_tcp_server (argv[1], atoi (argv[2])) < 0)
  {
    fprintf (stderr, "error creating TCP server\n");
    exit (EXIT_FAILURE);
  }

  return EXIT_SUCCESS;
}}
```

SERVERLIB.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <fcntl.h>
#include <errno.h>
#include <math.h>
#include "serverlib.h"

/* maximum number of simultaneous connections on the
server side */
#define QUEUELEN      8

#define NUMERO_MAX     80

int create_tcp_server (char *ip_address, int port)
{
  int sk, csk;
  struct sockaddr_in srv, cln;
  unsigned int dimcln;
  int keep_server_alive;
  char buffer[10];
  char ch[3];
  int iterazioni,valore;

/* create a socket descriptor */
if ((sk = socket (AF_INET, SOCK_STREAM, 0)) < 0)
{
  error_handler ("socket() [create_tcp_server()]");
  return -1;
}

/* fill the (used) fields of the socket address with
  the server information (local socket address) */
bzero ((char *) &srv, sizeof (srv));
srv.sin_family = AF_INET;
srv.sin_addr.s_addr = inet_addr (ip_address);
srv.sin_port = htons (port);

/* add the local socket address to the socket descriptor */
if (bind (sk, (struct sockaddr *) &srv, sizeof(srv)) < 0)
{
  error_handler ("bind() [create_tcp_server()]");
  return -1;
}

/* make the socket a passive socket (enable the socket
  accepting connections) */
if (listen (sk, QUEUELEN) < 0)
{
  error_handler ("listen() [create_tcp_server()]");
  return -1;
```

```
  }
  printf("in attesa di stabilire una connessione con un
client\n");

  dimcln = sizeof (cln);

  /* get the next connection request from the queue */
  if ((csk = accept (sk, (struct sockaddr *) &cln, &dimcln))
< 0)
  {
     error_handler ("accept() [create_tcp_server()]");
     return -1;
  }

  printf ("connected to %s:%d\n", inet_ntoa (cln.sin_addr),
(int) cln.sin_port);

  tcp_receive(csk,buffer);
  valore = atoi(buffer);

  do                          /* server loop */
  {

     printf("indicare il numero di iterazioni da chiedere al
client, QUIT per terminare attività client:");
     scanf("%s", buffer );
     tcp_send(csk, buffer);

     if(strncmp(buffer,"QUIT",4 ) == 0 )
     {
        printf("per terminare attività server inserire YES ,
altrimenti NO:");
        scanf("%s", ch);

        if( strncmp(ch,"YES",3 ) == 0)
        {
          printf("chiusura server in corso\n");
          keep_server_alive = 0;
        }
        else
        {
          keep_server_alive = 1;

          printf("in attesa di stabilire una connessione con
un client\n");
          /* get the next connection request from the queue
*/
          if ((csk = accept (sk, (struct sockaddr *) &cln,
&dimcln)) < 0)
          {
            error_handler ("accept() [create_tcp_server()]");
            return -1;
          }
          printf ("connected to %s:%d\n", inet_ntoa
(cln.sin_addr), (int) cln.sin_port);

          tcp_receive(csk,buffer);
          valore = atoi(buffer);
        }
     }
     else
     {
         iterazioni = atoi(buffer);
         keep_server_alive = server_handler (csk, inet_ntoa
(cln.sin_addr), (int) cln.sin_port,iterazioni,  valore); // la
funzione che legge da socket
     }

  } while (keep_server_alive);

  printf("attività server terminata con successo \n");
  close_tcp_connection (csk);

  return 1;
}

void error_handler (const char *message)
{
  printf ("fatal error: %s\n", message);
  exit (EXIT_FAILURE);
}

int close_tcp_connection (int sk)
{
  if (close (sk) != 0)
  {
    error_handler ("close() [close_tcp_connection()]");
    return 0;
  }

  return 1;
}

int tcp_send (int sk, char *buffer)
{
  if (write (sk, buffer, strlen(buffer)) != strlen(buffer))
  {
    error_handler ("write() [tcp_send()]");
    return 0;
  }

  return 1;
}

int server_handler (int csk, char *ip_addr, int port,int
iterazioni, int valore)
{
  char buffer [40],axis_x[10],axis_y[10],axis_z[10];
  float axis[3];
  char str[3] = "XYZ";
  int i,x,y,z ;

  int flag = 0;

  for(flag = 0; flag < iterazioni; flag++)
  {

     tcp_receive (csk, buffer);
     i = 0,x=0, y=0,z=0;
     while( buffer[i] != ' ')
     {
        axis_x[x] = buffer[i];
        i++;
        x++;
     }

     i++;

     while( buffer[i] != ' ')
     {
        axis_y[y] = buffer[i];
        i++;
        y++;
     }
```

```
      i++;

      while( buffer[i] != ' ')
      {
         axis_z[z] = buffer[i];
         i++;
         z++;
      }

     axis[0] = atof(axis_x);
     axis[1] = atof(axis_y);
     axis[2] = atof(axis_z);

     stampa_valori(axis, valore);
     printf("\033c");        // pulisce lo schermo

    }

    return 1;
}

void stampa_valori(float buff[] ,int valore )
{
     int numero_spazi;
     int flag[3]; // tiene in memoria il segno dei valori sui
tre assi
                  // 0 per positivo, 1 per negativo
     int diff;
     float buff_scalato[3]; // conterrà i valori di buff
moltiplicati per il fattore di scala
     int buff_int[3];
     int i,j,k= 0;
     char vect[3] = "XYZ";
     float fattore_di_scala;

     numero_spazi = ricerca_numero_spazi();
     valuta_positivo_negativo(buff, flag);
     fattore_di_scala = ricerca_fattore( valore );
     scala_valori( buff_scalato, buff, fattore_di_scala );
     trasforma_in_intero(buff_scalato, buff_int ); //
funzione che moltiplica per 9,8 il valore scalato e
trasforma in intero il valore float
     stampa_intestazione(numero_spazi);

     // stampa dei valori

     for(i = 0; i < 3; i++)
     {
       if( flag[i] == 1 )
       {
          diff = numero_spazi + buff_int[i];
          for( j = 0; j < diff; j++)
              printf(" ");

          j=0;
          for( k = 0; k < -buff_int[i]; k++)
             printf("▪");

          k=0;
          printf("|  %c:%.02f ", vect[i], buff[i]*9.8);
          stampa_spazio(buff[i]*9.8);
          printf("|\n");
      }

     if( flag[i] == 0 )
     {
        for( j = 0; j < numero_spazi; j++)
            printf(" ");

          j=0;
          printf("|  %c: %.02f ",vect[i],buff[i]*9.8);
          stampa_spazio(buff[i]*9.8);
          printf("|");
          for( k = 0; k < buff_int[i]; k++)
             printf("▪");

          k=0;
          printf("\n");
      }

   }
   i = 0;
}

int ricerca_numero_spazi(void)
{
     return NUMERO_MAX; // risulta essere impostato a
80 e serve a fissare il valore massimo
}

void trasforma_in_intero(float buff_scalato[],int buff_int[]
)
{
   int i,j = 0 ;

   for( i = 0; i < 3; i++ )
   {
       buff_scalato[i] = buff_scalato[i]* 9.8;

   }
   i = 0;

  for( j = 0; j < 3; j++ )
  {

      buff_int[j] = arrotonda(buff_scalato[j]);
  }
  j = 0;
}

int arrotonda(float numero)
{
   int numero_intero = (int) numero;
   float differenza = (float)(numero -
(float)numero_intero);

   if( fabs(differenza) >= 0.5 )
   {
     if( numero >= 0)
       return ((int) numero + 1);
     else
       return ((int) numero - 1);
   }
   else
   {
      return ((int)numero);
   }
}

void  valuta_positivo_negativo( float buff[] ,int flag[])
{
   int i = 0;

   for( i = 0; i < 3; i++)
   {
      if(buff[i] < 0)
      {
```

```
        flag[i] = 1;
      }

      if(buff[i] >= 0)
      {
       flag[i] = 0;
      }
  }
  i = 0;
}

float ricerca_fattore(int valore)
{
  if( valore == 2 )
     return (((float)NUMERO_MAX)/20);

  if( valore == 4 )
      return (((float)NUMERO_MAX)/40);

  if( valore == 8 )
     return (((float)NUMERO_MAX)/80);

  if( valore == 16 )
     return (((float)NUMERO_MAX)/160);
}

void scala_valori(float buff_scalato[] ,float buff[] ,float
fattore_di_scala )
{
   int i;

   i = 0;

   for( i = 0; i < 3; i++ )
   {
     buff_scalato[i] = buff[i]*fattore_di_scala;
   }
}

void stampa_intestazione(int numero_spazi)
{
   stampa_inizio_spazi( numero_spazi);
   printf("negative");
   stampa_fine_spazi( numero_spazi);
   printf("|    AXIS    |");  // 3 spazi prima e dopo di AXIS
   stampa_inizio_spazi( numero_spazi);
   printf("positive");
   printf("\n");
   stampa_underscore(numero_spazi);

}

void stampa_inizio_spazi( int numero_spazi)
{
    int spaziatura = numero_spazi/2 - 4;
    int i = 0;

    for (i = 0; i < spaziatura;i ++)
        printf(" ");

    i = 0;
}

void stampa_fine_spazi(int numero_spazi)
{
   int spaziatura = ( numero_spazi - (numero_spazi/2) - 4
);
   int i;

   for( i = 0; i < spaziatura; i++)
     printf(" ");

  i = 0;
}

void stampa_spazio(float valore)
{
    if(abs(valore) < 100 && abs(valore) >= 10 )
      printf(" ");

    if(abs(valore) < 10 )
      printf("  ");
}

void stampa_underscore(int numero_spazi)
{
  int i;

  for(i = 0; i < (((numero_spazi)*2)+14);i++)
    printf("=");

  i = 0;
  printf("\n");
}}
```

SERVERLIB.h

```
 /*********************************************************************

  Copyright 2001   PIER LUCA MONTESSORO

  University of Udine
  ITALY

  montessoro@uniud.it
  www.montessoro.it

  This file is part of a freeware open source software package.
  It can be freely used (as it is or modified) as long as this
  copyright note is not removed.

 *********************************************************************/
// collegamento server
int create_tcp_server (char *ip_address, int port);
int close_tcp_connection (int sk);
int tcp_send (int sk, char *buffer);
int tcp_receive (int sk, char *buffer);
int server_handler (int sk, char *ip_addr, int port, int iterazioni, int valore);
void error_handler (const char *message);

// stampa dei valori su schermo
void stampa_valori(float buff[], int valore);
int ricerca_numero_spazi( void );
void trasforma_in_intero(float buff_scalato[],int buff_int[] );
int arrotonda(float numero);
void  valuta_positivo_negativo( float buff[],int flag[]);
float ricerca_fattore(int valore);
void scala_valori(float buff_scalato[] ,float buff[] ,float fattore_di_scala );

//intestazione
void stampa_intestazione(int numero_spazi);
void stampa_inizio_spazi( int numero_spazi);
void stampa_fine_spazi(int numero_spazi);
void stampa_spazio(float valore);
void stampa_underscore(int numero_spazi);}
```

APPENDICE B

In questa appendice viene riportato il codice “C” relativo al Client. I file contengono rispettivamente il main, il corpo della funzione, la libreria.

ACC_LIS3DH.c

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <ctype.h>
#include "Lis3dh.h"
#include <wiringPiI2C.h>

int main(int argc, char *argv[])
{
   int fd = 0;
   short int ax, ay, az;
   int valore;
   float buff[3];
   char buffer[20];
   int iterazioni = 0;
   int i = 0;

   // variabili per il collegamento TCP
   int sk;

   if(argc != 4 )
   {
      printf(" indicare il valore di accelerazione da
impostare (2,4,8,16)\n" );
      printf(" indicare indirizzo IP e porta per il
collegamento socket\n"    );
      return 1;
   }

   fd =
wiringPiI2CSetup(LIS3DH_DEFAULT_ADDRESS);

   // inizializzazione della comunicazione i2c
   if( fd < 0 )
   {
     printf(" errore nella lettura del dispositivo
all'indirizzo 0x18");
   }

   // stabilire la connessione tcp con il server
   if ((sk = create_tcp_client_connection (argv[2], atoi
(argv[3]))) < 0)
   {
     fprintf (stderr, "cannot open TCP connection\n");
      exit (EXIT_FAILURE);
   }

   lis3dh_init(fd); // scrittura nei registri
   valore = atoi(argv[1]);
   imposta_accelerazione(fd,valore);
   tcp_send(sk, argv[1]);

   do
   {
      printf("in attesa di ricevere istruzioni dal server\n");

      while(tcp_receive(sk,buffer) > 0)
      {
         if(strncmp(buffer,"QUIT",4) == 0)
         {
            iterazioni = -1;
            break;
         }
         else
         {
            iterazioni = atoi(buffer);
            printf("il server ha indicato %d
iterazioni\n",iterazioni);
            break;
         }
      }

      i = 0;

      if( iterazioni != -1 )
      {
        for( i = 0; i < iterazioni ; i++ )
        {
        lis3dh_read_xyz(&ax,&ay,&az, fd, buff);
        scrivi_su_schermo(buff, valore);
        invia_dati_server(sk,buff,valore);
        delay(100);
        printf("\033c");
        }

      }

   }while(iterazioni > 0 );

   printf("il server ha inviato %s\n",buffer);
   printf("client terminato con successo\n");

   close_tcp_connection (sk);

   return EXIT_SUCCESS;
}
```

LIS3DH.c

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <ctype.h>
#include "Lis3dh.h"
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <fcntl.h>
#include <errno.h>
#include <wiringPiI2C.h>

#define NUMERO_MAX 80

void lis3dh_init(int fd)
{
  wiringPiI2CWriteReg8(fd,LIS3DH_REG_CTRL1,
0x07);
  checkconnection(fd);
  setDataRate(fd, LIS3DH_DATARATE_400_HZ);
  wiringPiI2CWriteReg8(fd,LIS3DH_REG_CTRL4,
0x88);
  wiringPiI2CWriteReg8(fd,LIS3DH_REG_CTRL3,
0x10);
  wiringPiI2CWriteReg8(fd,LIS3DH_REG_TEMPCFG,
0x80);

}

void checkconnection(int fd)
{
   uint8_t deviceid = wiringPiI2CReadReg8(fd,
LIS3DH_REG_WHOAMI);

  if(deviceid != 0x33)
  {
    printf(" nessuna connessione con il dispositivo\n " );
    exit(1);
  }

}

void setDataRate(int fd, lis3dh_dataRate_t dataRate )
{
  uint8_t ctl1 = wiringPiI2CReadReg8(fd,
LIS3DH_REG_CTRL1);
  ctl1 &= ~(0xF0);
  ctl1 |= ( dataRate << 4 );
  wiringPiI2CWriteReg8(fd,LIS3DH_REG_CTRL1,
ctl1);
}

lis3dh_dataRate_t getDataRate(int fd)
{
   return (lis3dh_dataRate_t)((wiringPiI2CReadReg8(fd,
LIS3DH_REG_CTRL1) >> 4 )& 0x0F);
}

void setRange(int fd, lis3dh_range_t range )
{
   uint8_t r =
wiringPiI2CReadReg8(fd,LIS3DH_REG_CTRL4);
   r &= ~(0x30);
   r |= range << 4;
   wiringPiI2CWriteReg8(fd,LIS3DH_REG_CTRL4,r);
}
lis3dh_range_t getRange(int fd)
{
   //return
(lis3dh_range_t)((wiringPiI2CReadReg8(fd,LIS3DH_RE
G_CTRL4) >> 4) & 0x03);
   return
(uint8_t)((wiringPiI2CReadReg8(fd,LIS3DH_REG_CTR
L4) >> 4) & 0x03);
}

void lis3dh_read_xyz(short int*x, short int *y,  short int
*z, int fd, float buff[])
{
   uint8_t x0, y0, z0, x1, y1, z1;

   x0 = 0xff - wiringPiI2CReadReg8(fd,
LIS3DH_REG_OUT_X_L); // valore basso
   y0 = 0xff - wiringPiI2CReadReg8(fd,
LIS3DH_REG_OUT_Y_L);
   z0 = 0xff - wiringPiI2CReadReg8(fd,
LIS3DH_REG_OUT_Z_L);
   x1 = 0xff - wiringPiI2CReadReg8(fd,
LIS3DH_REG_OUT_X_H); // valore alto
   y1 = 0xff - wiringPiI2CReadReg8(fd,
LIS3DH_REG_OUT_Y_H);
   z1 = 0xff - wiringPiI2CReadReg8(fd,
LIS3DH_REG_OUT_Z_H);

   /**x = (int)(x1 << 8) + (int)x0;
   *y = (int)(y1 << 8) + (int)y0;
   *z = (int)(z1 << 8) + (int)z0;
   */
   *x = (short int)((((uint16_t) x1 ) << 8) | x0);
   *y = (short int)((((uint16_t) y1 ) << 8) | y0);
   *z = (short int)((((uint16_t) z1 ) << 8) | z0);

   uint8_t range = getRange(fd);
   float divider = 1;
   if ( range == LIS3DH_RANGE_16_G)
      divider = 1365;
   if ( range == LIS3DH_RANGE_8_G)
      divider = 4096;
   if ( range == LIS3DH_RANGE_4_G)
      divider = 8190;
   if ( range == LIS3DH_RANGE_2_G)
      divider = 16380;

  buff[0] = ((float)*x) / divider;
  buff[1] = ((float)*y) / divider;
  buff[2] = ((float)*z) / divider;

}

void imposta_accelerazione(int fd,int valore)
```

```
{
    if( valore == 2 )
       setRange(fd,LIS3DH_RANGE_2_G);

   if( valore == 4 )
       setRange(fd,LIS3DH_RANGE_4_G);

  if( valore == 8 )
       setRange(fd,LIS3DH_RANGE_8_G);

  if( valore == 16 )
       setRange(fd,LIS3DH_RANGE_16_G);
}

void scrivi_su_schermo(float buff[] , int valore)
{
    int numero_spazi;
    int flag[3]; // tiene in memoria il segno dei valori sui
tre assi
                // 0 per positivo, 1 per negativo
    int diff;
    float buff_scalato[3]; // conterrà i valori di buff
moltiplicati per il fattore di scala
    int buff_int[3];
    int i,j,k= 0;
    char vect[3] = "XYZ";
    float fattore_di_scala;

    numero_spazi = ricerca_numero_spazi();
    valuta_positivo_negativo(buff, flag);
    fattore_di_scala = ricerca_fattore( valore );
    scala_valori( buff_scalato, buff, fattore_di_scala );
    trasforma_in_intero(buff_scalato, buff_int ); //
funzione che moltiplica per 9,8 il valore scalato e
trasforma in intero il valore float
    stampa_intestazione(numero_spazi);

    // stampa dei valori

    for(i = 0; i < 3; i++)
    {
      if( flag[i] == 1 )
      {
         diff = numero_spazi + buff_int[i];
         for( j = 0; j < diff; j++)
             printf(" ");

         j=0;
        for( k = 0; k < -buff_int[i]; k++)
           printf("▪");

        k=0;
        printf("|  %c:%.02f ", vect[i], buff[i]*9.8);
        stampa_spazio(buff[i]*9.8);
        printf("|\n");
     }

    if( flag[i] == 0 )
    {
       for( j = 0; j < numero_spazi; j++)
           printf(" ");
       j=0;
       printf("|  %c: %.02f ",vect[i],buff[i]*9.8);
       stampa_spazio(buff[i]*9.8);
        printf("|");
       for( k = 0; k < buff_int[i]; k++)
          printf("▪");

       k=0;
       printf("\n");
     }

  }
  i = 0;
}

int ricerca_numero_spazi(void)
{
     return NUMERO_MAX; // risulta essere impostato a
80 e serve a fissare il valore massimo
}

void trasforma_in_intero(float buff_scalato[],int
buff_int[] )
{
   int i,j = 0 ;

   for( i = 0; i < 3; i++ )
   {
        buff_scalato[i] = buff_scalato[i]* 9.8;

   }
   i = 0;

  for( j = 0; j < 3; j++ )
  {

     buff_int[j] = arrotonda(buff_scalato[j]);
  }
  j = 0;
}

int arrotonda(float numero)
{
   int numero_intero = (int) numero;
   float differenza = (float)(numero -
(float)numero_intero);

   if( fabs(differenza) >= 0.5 )
   {
     if( numero >= 0)
       return ((int) numero + 1);
     else
       return ((int) numero - 1);
   }
   else
   {
      return ((int)numero);
   }
}

void  valuta_positivo_negativo( float buff[] ,int flag[])
{
   int i = 0;

   for( i = 0; i < 3; i++)
   {
      if(buff[i] < 0)
```

```
      {
        flag[i] = 1;
      }

      if(buff[i] >= 0)
      {
       flag[i] = 0;
      }
  }
  i = 0;
}

float ricerca_fattore(int valore)
{
  if( valore == 2 )
     return (((float)NUMERO_MAX)/20);

  if( valore == 4 )
      return (((float)NUMERO_MAX)/40);

  if( valore == 8 )
     return (((float)NUMERO_MAX)/80);

  if( valore == 16 )
     return (((float)NUMERO_MAX)/160);
}

void scala_valori(float buff_scalato[] ,float buff[] ,float
fattore_di_scala )
{
   int i;

   i = 0;

   for( i = 0; i < 3; i++ )
  {
    buff_scalato[i] = buff[i]*fattore_di_scala;
  }
}

void stampa_intestazione(int numero_spazi)
{
   stampa_inizio_spazi( numero_spazi);
   printf("negative");
   stampa_fine_spazi( numero_spazi);
   printf("|    AXIS    |");  // 3 spazi prima e dopo di AXIS
   stampa_inizio_spazi( numero_spazi);
   printf("positive");
   printf("\n");
   stampa_underscore(numero_spazi);

}

void stampa_inizio_spazi( int numero_spazi)
{
    int spaziatura = numero_spazi/2 - 4;
    int i = 0;

    for (i = 0; i < spaziatura;i ++)
        printf(" ");

    i = 0;
}

void stampa_fine_spazi(int numero_spazi)
{
   int spaziatura = ( numero_spazi - (numero_spazi/2) - 4
);
   int i;

   for( i = 0; i < spaziatura; i++)
     printf(" ");

  i = 0;
}

void stampa_spazio(float valore)
{
    if(abs(valore) < 100 && abs(valore) >= 10 )
      printf(" ");

    if(abs(valore) < 10 )
      printf("  ");
}

void stampa_underscore(int numero_spazi)
{
  int i;

  for(i = 0; i < (((numero_spazi)*2)+14);i++)
    printf("=");

  i = 0;
  printf("\n");
}


// invio dati attraverso connesione tcp

void invia_dati_server(int sk, float buff[], int valore)
{
  char str[40]; // contiene 10 valori per riga
  int i;

  i = 0;

   // trasforma in char il valore del float
   sprintf(str,"%0.3f %0.3f %0.3f ",
buff[0],buff[1],buff[2] );


   // invia i valori con tcp send
   // tcp_send prende un messaggio e lo stampa sul socket

          tcp_send( sk, str ); //scrive i valori sul socket
                  //che saranno letti dal server

}

int create_tcp_client_connection (char *ip_address, int
port)
{
  int sk;
  struct sockaddr_in srv;

  /* create a socket descriptor */
  if ((sk = socket (AF_INET, SOCK_STREAM, 0)) < 0)
  {
```

```
    error_handler ("socket()
[create_tcp_client_connection()]");
    return -1;
  }

  /* fill the (used) fields of the socket address with
    the server information (remote socket address) */
  bzero ((char *) &srv, sizeof (srv));
  srv.sin_family = AF_INET;
  srv.sin_addr.s_addr = inet_addr (ip_address);
  srv.sin_port = htons (port);

  /* open a connection to the server */
  if (connect (sk, (struct sockaddr *) &srv, sizeof(srv)) <
0)
  {
    error_handler ("connect()
[create_tcp_client_connection()]");
    return -1;
  }

  return sk;
}

int close_tcp_connection (int sk)
{
  if (close (sk) != 0)
  {
    error_handler ("close() [close_tcp_connection()]");
    return 0;
  }

  return 1;
}

int tcp_send (int sk, char *buffer)
{
  if (write (sk, buffer, strlen(buffer)) != strlen(buffer))
  {
    error_handler ("write() [tcp_send()]");
    return 0;
  }

  return 1;
}

int tcp_receive (int sk, char *buffer)
{
  int dim, flags;

  if ((dim = read (sk, buffer, BUFSIZ)) < 0)
  {
    flags = fcntl (sk, F_GETFL, 0);
    if ((flags & O_NONBLOCK) == O_NONBLOCK)
    {  /* non-blocking mode */
      if ((errno == EAGAIN) || (errno ==
EWOULDBLOCK))
        return 0;
      else
      {
        error_handler ("read() [tcp_receive()]");
        return -1;
      }
    }
    else
    {
      error_handler ("read() [tcp_receive()]");
      return -1;
    }
  }

  buffer[dim] = '\0';
  return dim;
}

void error_handler (const char *message)
{
  printf ("fatal error: %s\n", message);
  exit (EXIT_FAILURE);
}}
```

LIS3DH.h

```
#ifndef __LIS3DH_H__
#define __LIS3DH_H__

#include <wiringPi.h>
#include <wiringPiI2C.h>
#include <stdint.h>

/*=========================================
==================================
   I2C ADDRESS/BITS
   ------------------------------------------------------------------
-----*/
   #define LIS3DH_DEFAULT_ADDRESS  (0x18)    //
if SDO/SA0 is 3V, its 0x19
/*=========================================
==================================*/

#define LIS3DH_REG_STATUS1       0x07
#define LIS3DH_REG_OUTADC1_L     0x08
#define LIS3DH_REG_OUTADC1_H     0x09
#define LIS3DH_REG_OUTADC2_L     0x0A
#define LIS3DH_REG_OUTADC2_H     0x0B
#define LIS3DH_REG_OUTADC3_L     0x0C
#define LIS3DH_REG_OUTADC3_H     0x0D
#define LIS3DH_REG_INTCOUNT      0x0E
#define LIS3DH_REG_WHOAMI        0x0F
#define LIS3DH_REG_TEMPCFG       0x1F
#define LIS3DH_REG_CTRL1         0x20
#define LIS3DH_REG_CTRL2         0x21
#define LIS3DH_REG_CTRL3         0x22
#define LIS3DH_REG_CTRL4         0x23
#define LIS3DH_REG_CTRL5         0x24
#define LIS3DH_REG_CTRL6         0x25
#define LIS3DH_REG_REFERENCE     0x26
#define LIS3DH_REG_STATUS2       0x27
#define LIS3DH_REG_OUT_X_L       0x28
```

```
#define LIS3DH_REG_OUT_X_H      0x29
#define LIS3DH_REG_OUT_Y_L      0x2A
#define LIS3DH_REG_OUT_Y_H      0x2B
#define LIS3DH_REG_OUT_Z_L      0x2C
#define LIS3DH_REG_OUT_Z_H      0x2D
#define LIS3DH_REG_FIFOCTRL     0x2E
#define LIS3DH_REG_FIFOSRC      0x2F
#define LIS3DH_REG_INT1CFG      0x30
#define LIS3DH_REG_INT1SRC      0x31
#define LIS3DH_REG_INT1THS      0x32
#define LIS3DH_REG_INT1DUR      0x33
#define LIS3DH_REG_CLICKCFG     0x38
#define LIS3DH_REG_CLICKSRC     0x39
#define LIS3DH_REG_CLICKTHS     0x3A
#define LIS3DH_REG_TIMELIMIT    0x3B
#define LIS3DH_REG_TIMELATENCY  0x3C
#define LIS3DH_REG_TIMEWINDOW   0x3D
#define LIS3DH_REG_ACTTHS       0x3E
#define LIS3DH_REG_ACTDUR       0x3F

typedef enum
{
  LIS3DH_RANGE_16_G       = 0b11,   // +/- 16g
  LIS3DH_RANGE_8_G        = 0b10,   // +/- 8g
  LIS3DH_RANGE_4_G        = 0b01,   // +/- 4g
  LIS3DH_RANGE_2_G        = 0b00    // +/- 2g
(default value)
} lis3dh_range_t;

typedef enum
{
  LIS3DH_AXIS_X       = 0x0,
  LIS3DH_AXIS_Y       = 0x1,
  LIS3DH_AXIS_Z       = 0x2,
} lis3dh_axis_t;

typedef enum
{
  LIS3DH_DATARATE_400_HZ     = 0b0111, //  400Hz
  LIS3DH_DATARATE_200_HZ     = 0b0110, //  200Hz
  LIS3DH_DATARATE_100_HZ     = 0b0101, //  100Hz
  LIS3DH_DATARATE_50_HZ      = 0b0100, //   50Hz
  LIS3DH_DATARATE_25_HZ      = 0b0011, //   25Hz
  LIS3DH_DATARATE_10_HZ      = 0b0010, // 10 Hz
  LIS3DH_DATARATE_1_HZ       = 0b0001, // 1 Hz
  LIS3DH_DATARATE_POWERDOWN  = 0,
  LIS3DH_DATARATE_LOWPOWER_1K6HZ  =
0b1000,
  LIS3DH_DATARATE_LOWPOWER_5KHZ  =
0b1001,

} lis3dh_dataRate_t;

//lettura accelerazione
void lis3dh_init(int fd);
void checkconnection(int fd);
void setDataRate(int fd, lis3dh_dataRate_t dataRate );
lis3dh_dataRate_t getDataRate(int fd);
void setRange(int fd,lis3dh_range_t range );
lis3dh_range_t getRange(int fd);
void lis3dh_read_xyz(short int*x, short int *y,  short int
*z, int fd, float buff[]);
void imposta_accelerazione(int fd,int valore);

// stampa dei valori su schermo
void scrivi_su_schermo(float buff[], int valore);
int ricerca_numero_spazi( void );
void trasforma_in_intero(float buff_scalato[],int
buff_int[] );
int arrotonda(float numero);
void  valuta_positivo_negativo( float buff[],int flag[]);
float ricerca_fattore(int valore);
void scala_valori(float buff_scalato[] ,float buff[] ,float
fattore_di_scala );

//intestazione
void stampa_intestazione(int numero_spazi);
void stampa_inizio_spazi( int numero_spazi);
void stampa_fine_spazi(int numero_spazi);
void stampa_spazio(float valore);
void stampa_underscore(int numero_spazi);


/*********************************************
*********************

 Copyright 2001   PIER LUCA MONTESSORO

 University of Udine
 ITALY

 montessoro@uniud.it
 www.montessoro.it

 This file is part of a freeware open source software
package.
 It can be freely used (as it is or modified) as long as this
 copyright note is not removed.

*********************************************
*********************/
//collegamento server
void invia_dati_server(int sk, float buff[], int valore );
int create_tcp_client_connection (char *ip_address, int
port);
int close_tcp_connection (int sk);
int tcp_send (int sk, char *buffer);
int tcp_receive (int sk, char *buffer);
void error_handler (const char *message);

#endif}
```

Bibliografia

1) Tai-Ran Hsu *"Mems And Microsistem Design, Manufacture, And Nanoscale Engeneering"* Second Edition (2008) John Wiley & Sons, Inc.

2) Sthephen D. Senturia *" Microsystem design*" (2002) Kluwer Academic Publisher

3) Julian W. Gardner, Vijay K. Varadan, Osama O. Awadelkarim *" Microsensors Mems and Smart devices"* (2001) John Wiley & Sons LTD.

4) Sergey Edward Lyshevsky *"Nano and micro electromechanical system"* (2001) CRC Press LLC

5) Valter Minute *"Raspberry Pi: Guida all'uso"* , (2013) Edizione Fag Milano

6) Andrew S.Tanenbaum David J. Wetherall *" Reti di calcolatori"* ,quinta edizione , (2011) Pearson

7) Frederic Leens "*An introduction to I^2c and SPI protocols*", (2009) IEEE Instruments & Measurement Magazine, pages 8-13

8) Yu-Sian Liu, Kuei-Ann Wen "*Implementation of a COMS/MEMS Accelerometer with ASIC Processes*", (2019)

Sitografia

1) Stmicroelectronics "https://www.st.com/resource/en/datasheet/cd00274221.pdf", (2016)
2) Michele Maffucci "https://www.maffucci.it/2012/06/11/appunti-su-arduino-interrupts/"
3) Afg "http://www.microcontroller.it/Tutorials/Elettronica/SPI.htm" (ultima modifica 24/12/11)
4) Wikipedia "https://en.wikipedia.org/wiki/Single-board_computer"
5) Orangepi "http://www.orangepi.org/orangepipc2/"
6) Wikipedia "https://en.wikipedia.org/wiki/Universal_asynchronous_receiver-transmitter"
7) "http://nicolasansone.altervista.org/scuola/wiringPi.pdf"
8) Canonical Ltd "https://wiki.ubuntu-it.org/Ufficio/EditorDiTesto/Nano"
9) Canonical Ltd "https://wiki.ubuntu-it.org/AmministrazioneSistema/InstallareProgrammi/Apt"
10) Canonical Ltd "https://wiki.ubuntu-it.org/Repository"
11) "https://help.ubuntu.com/lts/serverguide/apt.html.en"
12) Wikipedia "https://it.wikipedia.org/wiki/Chmod"

13) Wiringpi “http://wiringpi.com/the-gpio-utility/” (2019)
14) Wiringpi “http://wiringpi.com/reference/” (2019)