POLITECNICO DI TORINO

**Corso di Laurea Magistrale**
**in Ingegneria Informatica**

Tesi di Laurea Magistrale

# Realizzazione di un Controller MIDI con Interfaccia Accessibile

**Relatore**
Prof. Antonio Servetti

**Candidato**
Antonio Di Martino

Dicembre 2017

*Alla mia famiglia,*

*a Chiara,*

*a tutti gli amici del progetto ADO.*

# Sommario

# Indice delle figure

# Capitolo 1
# INTRODUZIONE

## 1.1 Come tutto è iniziato

Il Wireless Midi Controller è il risultato finale del progetto di tesi ideato, studiato e realizzato con l'unico intento di fornire uno strumento musicale "alternativo" a persone che, per i più svariati motivi, non hanno la possibilità di utilizzarne di comuni.

Lo scopo sociale del progetto, in verità, non era esattamente quello che ricercavo nel momento della scelta dell'argomento per la mia tesi. Quello che mi premeva maggiormente era di trovare e di lavorare ad un qualcosa di pratico, che coniugasse l'informatica con altre branche dell'ingegneria e che mi desse la possibilità di partire da un'idea iniziale, svilupparla nella teoria e realizzarla nella pratica, con tutte le difficoltà che questo processo comporta.

Il progetto di tesi messo a disposizione dal prof. Servetti e che prevedeva la realizzazione di interfacce musicali accessibili ha coniugato tutto questo e molto di più.

L'idea da cui si parte è molto semplice. Realizzare veri e propri strumenti musicali, caratterizzati da un hardware più o meno complesso, con la sola peculiarità di avere delle interfacce che si prestino ad essere utilizzate da persone con disabilità.

Il lavoro di tesi, in realtà, si inserisce all'interno del più ampio progetto condotto dal Politecnico di Torino chiamato "Arduino Disability Orchestra" (ADO).


**Figura 1.1 – Arduino Disability Orchestra**

Lo scopo del progetto è, per l'appunto, l'ideazione e la realizzazione di strumenti musicali elettronici per persone con disabilità.
ADO non è "solo" questo, in realtà. L'opera forse più importante messa in campo dall'orchestra è l'effettivo utilizzo di questi strumenti, insieme ad altri convenzionali, in una vera e propria orchestra interamente composta da persone con disabilità fisica o mentale, dando loro la possibilità di cimentarsi e di inserirsi in un contesto per loro difficile o del tutto precluso.

## 1.2 **Musicoterapia**

In linea con quanto detto, è palese che il progetto ADO si inserisce a pieno titolo tra quelli che contemplano la musicoterapia come un utilissimo strumento di supporto e cura per persone affette da disabilità.
Gli effetti benefici della musica sono oramai una certezza scientifica. L'utilizzo della musica come strumento terapico apporta migliorie tanto a livello comportamentale che fisico (nel caso di disabilità cognitive più gravi).
Una delle ragioni per cui la musica è diventata velocemente uno strumento di cura nelle più disparate terapie è la sua capacità di stimolare contemporaneamente entrambi gli emisferi del nostro cervello, a dispetto di molte altre attività capaci di attivarne uno solo per volta.
Questo significa che un terapista, attraverso l'utilizzo di canzoni, suoni e strumenti musicali, può supportare e stimolare l'attività cognitiva della persona, migliorandone l'autoconsapevolezza e incoraggiando le relazioni con gli altri.
Quest'ultimo aspetto è quello che più palesemente viene riscontrato nelle sessioni musicoterapiche. La musica, infatti, stimola comportamenti che tendono alla comunicazione con gli altri piuttosto che l'isolamento (uno dei problemi maggiormente sentiti dalle persone affette da disabilità).

**Figura 1.2 – Sessione musicoterapia**

Avendo la possibilità di prendere parte ad una qualsiasi sessione di musicoterapia, quello che facilmente si riscontra è che, in linea con quanto detto, la musica spinge i musicisti all'interazione con le altre persone che stanno suonando in quel momento, spinta che non viene avvertita come un qualcosa di "faticoso" ma piuttosto qualcosa di piacevole.

Ecco che la comunicazione, l'interazione con gli altri che prima appariva un ostacolo insormontabile (soprattutto in persone affette da particolari patologie come l'autismo) diventa uno stimolo naturale, uno sforzo piacevole guidato dalla volontà al fine di raggiungere un'armonia tra suoni, strumenti e musicisti.

La prima forma di interazione a cui ogni musicista deve sottoporsi non è tanto quella con le altre persone della band, quanto quella con lo strumento stesso che ha tra le mani.

Quello che agli occhi di tutti potrebbe sembrare ovvio o banale, per una persona affetta da disabilità cognitiva è una grande scoperta. L'idea che dall'interazione possa nascere qualcosa di piacevole e divertente è l'elemento chiave della terapia e costituisce uno dei grandi meriti attribuiti alla musicoterapia.

L'interazione, quindi, viene introdotta gradualmente. Ad ogni passo corrisponde una scoperta diversa, un piacere nuovo volto a stimolare il musicista alle più normali forme comunicative.

Molte volte, però, questo processo potrebbe interrompersi nei primi passi, quando cioè il musicista dovrebbe approcciarsi allo strumento musicale vero e proprio.

Quello che non raramente accade è che non si riesca a suonare correttamente gli strumenti musicali convenzionali, semplicemente perché le loro interfacce non sono pensate per essere utilizzate da tutti.

È esattamente in questo frangente che si inserisce il lavoro di tesi realizzato in questi mesi, un lavoro volto a conciliare interfacce accessibili e musica, fornendo tutti gli elementi di base necessari a questa attività.

## 1.3 **Uno sguardo generale**

Il Wireless Midi Controller è uno strumento musicale a tutti gli effetti ma non convenzionale. Combina infatti la produzione di messaggi Midi (strumento musicale elettronico) con un'interfaccia utente particolare, che dà la possibilità di essere usato anche a persone con disabilità.

La sintesi dei messaggi Midi prodotti dal controller non è effettuata dallo strumento stesso ma è affidata a software di sintesi musicale esterni come per esempio Ableton Live.


**Figura 1.3 - Interfaccia Ableton Live 9**

Questa caratteristica lo rende estremamente versatile, dal momento che può essere utilizzato come un vero e proprio strumento musicale (il movimento delle mani sul sensore produce note musicali), oppure come controller midi ovvero come strumento di controllo delle più svariate variabili di qualsiasi traccia musicale.

Questa versatilità è fornita tanto dalla caratteristica del protocollo midi che esso utilizza quanto dal fatto che la sintesi e la gestione degli stessi è affidata ad un software esterno con il quale i messaggi midi in input possono essere utilizzati nei più svariati modi.

A livello hardware, il controller utilizza due sensori tattili sovrapposti, uno sensibile alla posizione in cui viene toccato, l'altro alla pressione che viene esercitata su di esso.


**Figura 1.4 - Position Sensor**


**Figura 1.5 - Pressure Sensor**

La combinazione di queste due informazioni permette un'infinita possibilità di scelta per quanto riguarda i messaggi midi da inviare all'esterno.

L'output analogico dei due sensori è gestito da un microcontrollore Arduino Mega, il quale interpreta l'input, fornito su una scala da 0 a 1023, mappandolo su una scala più consona al protocollo midi, e costruisce i diversi messaggi midi da inviare all'esterno.

Al microcontrollore è anche collegato un LCD 4x20 con il quale vengono mostrati diversi messaggi durante il funzionamento del controller e che è necessario per l'interazione dall'esterno con il controller stesso nel momento in cui si voglia modificare alcuni parametri di funzionamento, settaggio reso possibile tramite l'utilizzo di 4 pulsanti collegati all' LCD.

Il collegamento tra il controller e il sintetizzatore software è stato implementato in due modalità: wired, tramite l'utilizzo di un connettore female DIN5, e wireless tramite l'utilizzo di una particolare libreria, AppleMidi.h, e l'utilizzo di un software, rtp-midi, che ha lo scopo di creare una sessione Midi virtuale e effettuare il peering con il controller in attesa di connessione.

Completano il controller due strisce di 12 led ciascuna per il controllo visivo del pitch e del volume, oltre ad altri 7 disposti sulla parte superiore del controller che visualizzano lo stato corrente del sistema.

Nei prossimi capitoli verrà fatta una breve introduzione ai protocolli utilizzati per la comunicazione, al microcontrollore utilizzato e alla shield per la comunicazione wireless per poi analizzare in dettaglio il codice sviluppato che implementa il progetto.

# Capitolo 2
# PROTOCOLLO MIDI

## 2.1 **Cenni generali**

Il MIDI (Musical Instrument Digital Interface) è un protocollo che fornisce un mezzo efficiente e standardizzato per la conversione di informazioni derivanti da strumenti musicali in dati elettronici.
La trasmissione di queste informazioni avviene sotto forma di messaggi MIDI che, ricevuti da un sintetizzatore, vengono interpretati come istruzioni necessarie per generare specifici suoni che abbiano determinate caratteristiche
Il motivo della sua nascita è da ricercare agli inizi degli anni ’80, quando si cominciò ad avvertire l’esigenza di semplificare le modalità di connessione e comunicazione tra sintetizzatori connessi tra loro.
Da allora, il protocollo MIDI è diventato un vero e proprio standard indiscusso del campo e, nonostante i numerosi tentativi di modificarlo o sostituirlo, è rimasto pressoché immutato negli anni rispetto alla prima versione rilasciata.
L’avvento del MIDI è stato una vera e propria svolta sia in termini di connessioni tra apparati differenti che in termini di storage e gestione dei file audio che diventano sempre più piccoli e gestibili paragonati ai vecchi file di audio campionato.

## 2.2 **Struttura e connessioni**

Il componente principale del protocollo è il messaggio MIDI che trasferisce le istruzioni, una o più parole di 1 byte, da un dispositivo detto Master ad uno chiamato Slave, che ha il compito di eseguirle trasformandole opportunamente in suoni.
L’innovazione che porta il protocollo sta appunto in questi messaggi: non si trasmette più l’informazione musicale così com’è, ma istruzioni utili al sintetizzatore per poterla replicare.
Uno strumento che voglia interagire con un altro tramite protocollo MIDI è necessario che

possegga specifiche interfacce, tramite le quali vengono inviati e ricevuti i messaggi MIDI. Esistono tre tipologie di porte MIDI:

- MIDI IN: utilizzata per la ricezione dei messaggi in arrivo da un altro strumento;
- MIDI OUT: utilizzata per l'invio dei messaggi ad altre macchine;
- MIDI THRU: utilizzata per reindirizzare un messaggio proveniente dalla porta MIDI IN verso un'altra macchina.

Uno schema classico di interconnessione master-slave potrebbe essere:

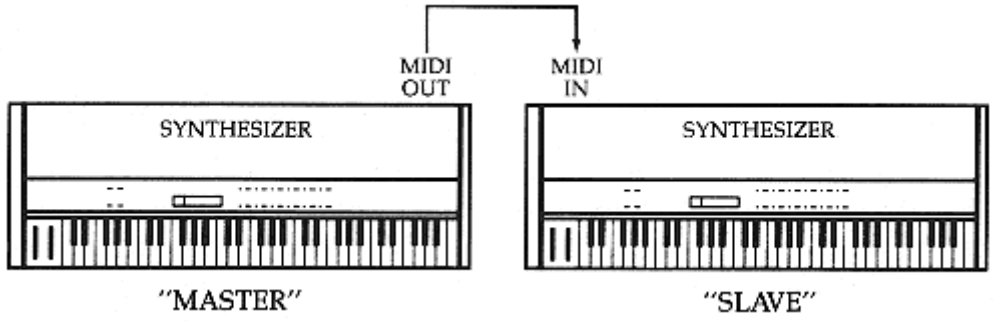


**Figura 2.1 – Connessione MIDI Master-Slave**

Se occorre utilizzare due o più slave, invece, si può ricorrere alla porta MIDI THRU e collegare in cascata le macchine:

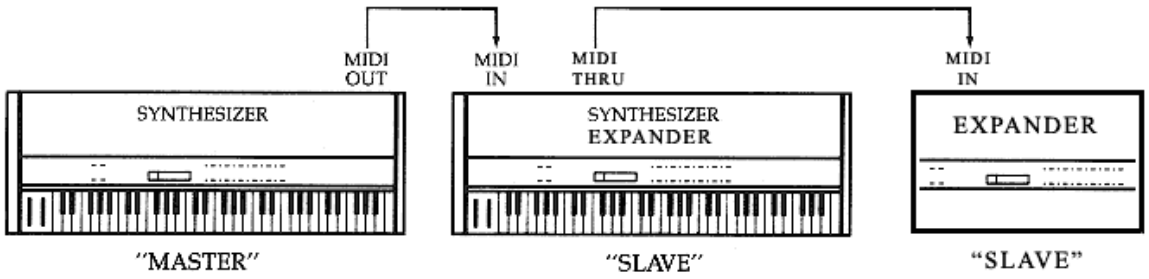


**Figura 2.2 – Connessione MIDI Master-Multi slave**

In questo ultimo caso però, si pone il problema di quali messaggi far interpretare ad uno slave piuttosto che all'altro. A risolvere questo problema ci pensa una proprietà del protocollo: il canale MIDI.

Esistono infatti 16 canali indipendenti su cui far viaggiare i messaggi. Ogni strumento musicale con un'interfaccia midi è abilitato alla ricezione dei messaggi o alla loro trasmissione su specifici canali e il settaggio di questi ultimi è ad opera del musicista. È dunque possibile, lato

master, indirizzare i messaggi midi su specifici canali di modo che gli slave in ascolto sugli stessi eseguano solo e soltanto i messaggi in arrivo sul canale sul quale sono in ascolto.
Nel caso invece si parli di MIDI THRU, i messaggi in arrivo non devono essere interpretati ma solamente trasferiti dalla porta di ingresso allo strumento collegato alla THRU stessa. In questo caso l'unica criticità è capire quando un messaggio midi termina e ne inizia un altro.

## 2.3 **Struttura dei messaggi MIDI**

I messaggi MIDI sono strutturati gerarchicamente in funzione delle operazioni che svolgono nel sistema. In generale si dividono in:

- Channel Message: sono messaggi che vengono indirizzati su specifici canali e servono alla produzione vera e propria delle note lato sintetizzatore;
- System Message: sono messaggi che non sono diretti a specifici strumenti connessi ma forniscono impostazioni generali al sistema intero.

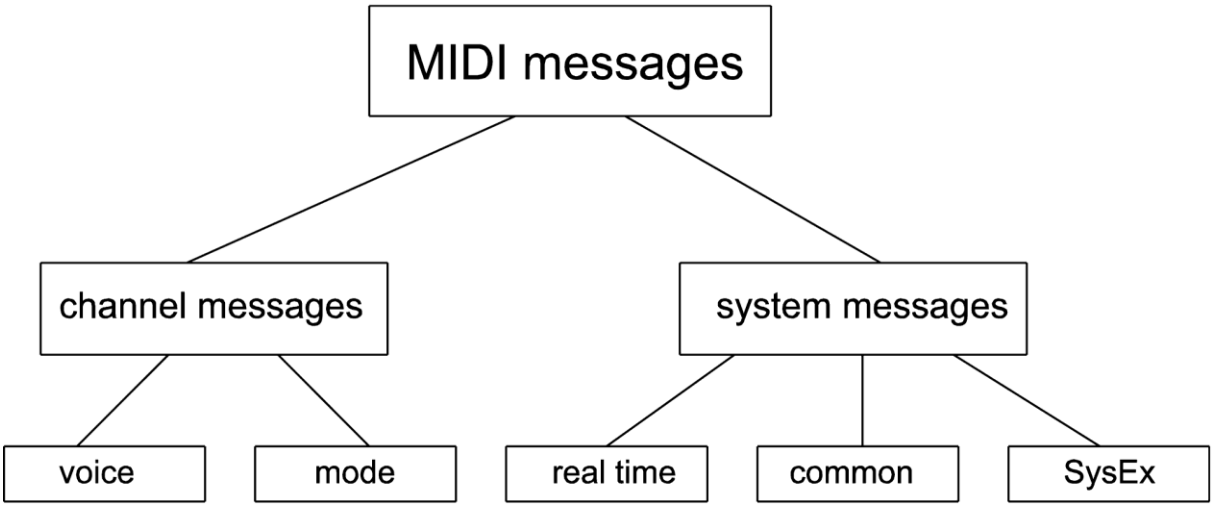


**Figura 2.3 – Gerarchia messaggi MIDI**

Un messaggio MIDI è costituito sempre da un primo byte chiamato Status Byte ed è generalmente seguito da uno o più Data Byte. Ogni byte a sua volta è diviso in due Nibble: la parte più significativa viene chiamata MSB mentre quella meno significativa LSB.
Lo status byte porta diverse informazioni al suo interno:

- Comando MIDI trasportato;

- Tipo di informazione (nota suonata, volume, pitch bend…) (bit 4, 5 e 6);
- Numero di canale su cui viaggia il messaggio (bit 0, 1, 2, 3).

Il Data Byte, invece, porta in sé il valore numerico del parametro anticipato nello Status che lo precede.

Status byte e data byte si distinguono grazie al loro bit più significativo: lo status byte presenta 1 come suo bit più significativo, mentre data byte 0.

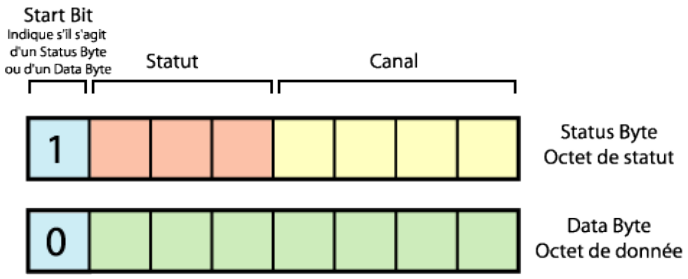


**Figura 2.4 – MIDI Status e Data Byte**

### 2.3.1 Status Byte

Nel caso dello status byte, i 3 bit seguenti il MSB servono ad identificare uno degli otto messaggi possibili derivanti dalla combinazione dei tre bit:

- Note ON,
- Note OFF,
- Polyphonic Key Pressure,
- Controll Change,
- Programme Change,
- Channel Pressure,
- Pitch Bend Change,
- System Message.

Gli ultimi 4 bit dello status byte, il secondo nibble, possono avere un duplice significato:

- Nel caso si tratti di un Channel Voice Message, i 4 bit rappresentano il canale su cui il messaggio viaggia. Avendo a disposizione 4 bit il numero massimo di canali è 16.

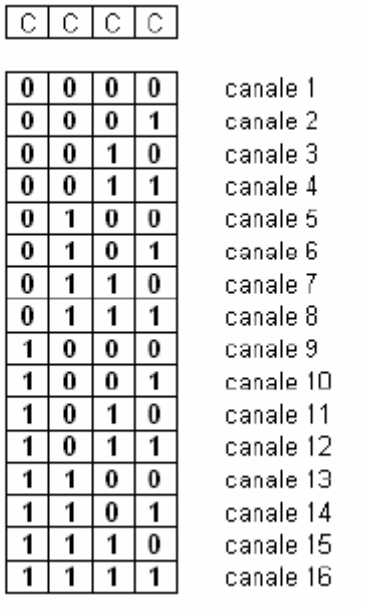


**Figura 2.5 – Canali MIDI**

- Nel caso invece si parla di System Message, i bit rappresentano la tipologia di messaggio inviato. In questo caso non tutte le 16 combinazioni sono sfruttate ma soltanto 12.

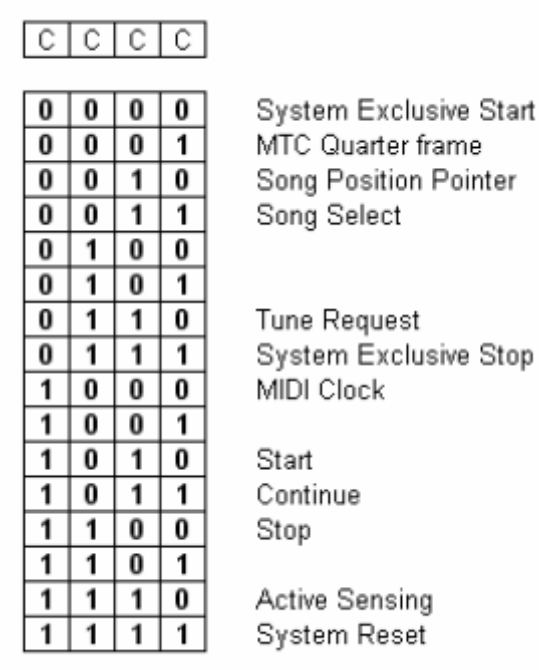


**Figura 2.6 – MIDI System Messages**

### 2.3.2 Data Byte

La struttura del data byte, invece, prevede, come detto, il MSB a 0 mentre i restanti 7 bit sono dedicati alla definizione del messaggio. Essendo 7 bit, sono esprimibili 128 valori massimi che vanno da 0 a 127.

## 2.4 **Messaggi Midi**

Come accennato precedentemente, esistono due categorie di messaggi midi:

- Channel Message
- System Message

### 2.4.1 Channel Message

I Channel Message sono messaggi di canale che vengono scambiati tra master e slave e rappresentano i messaggi principali del protocollo.
A loro volto sono divisi in:

- Voice Message
- Mode Message.

#### 2.4.1.1 Voice Message

I Voice Message sono 7, sono definiti dai tre bit successivi al MSB dello status byte e servono per mandare le vere e proprie informazioni musicali al sintetizzatore.
Si distinguono in:

- NOTE ON: è il messaggio midi che viene inviato dal master allo slave nel momento in cui viene suonata una nota sullo strumento. Si compone di Status Byte e 2 Data Byte di cui il primo rappresenta il valore dell'altezza (nota) e il secondo la velocity ovvero la forza di pressione del tasto. Al sintetizzatore, il primo data byte viene utilizzato per capire quale nota suonare mentre il secondo per gestirne l'ampiezza.

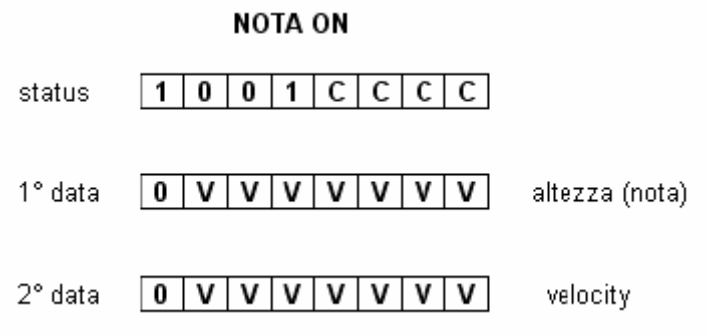


**Figura 2.7 – MIDI NoteOn**

- NOTE OFF: messaggio inviato dal master allo slave nel momento del rilascio del tasto premuto. Anche in questo caso troviamo status byte e due data byte. Il primo specifica la nota che viene rilasciata, il secondo la velocity del rilascio, di norma a zero.

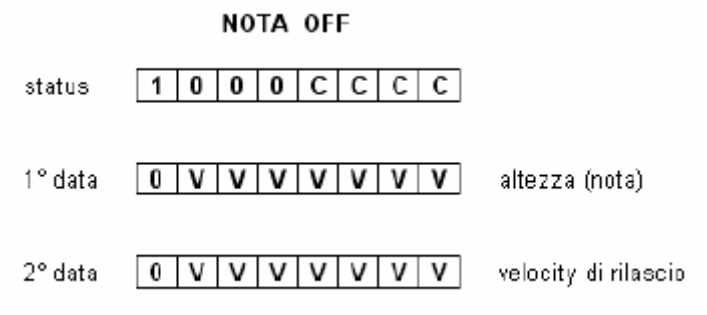


**Figura 2.8 – MIDI NoteOff**

- CHANNEL PRESSURE: messaggio inviato per comunicare una variazione di pressione associata alla nota suonata. Alcuni strumenti, infatti, hanno la possibilità di variare la pressione applicata ai tasti mentre è in esecuzione la nota ad essi associata. Questa informazione di pressione, chiamata "aftertouch" può essere utilizzata dal sintetizzatore per controllare alcuni aspetti del suono prodotto (per esempio vibrato). Il valore associato al Data Byte riporta esattamente il valore di pressione di aftertouch.

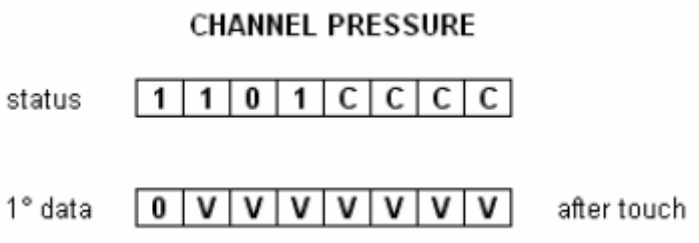


**Figura 2.9 – MIDI Channel Pressure**

- POLYPHONIC KEY PRESSURE: messaggio inviato al sintetizzatore per comunicare una variazione di aftertouch relativa ad un particolare tasto. In particolari strumenti musicali, infatti, sensori di pressione presenti su tutti i tasti danno la possibilità di modificare il valore di aftertouch associato ad ogni singolo tasto. In questo caso due Data Byte si rendono necessari: uno per specificare il tasto mentre l'altro il valore di afertouch associato.

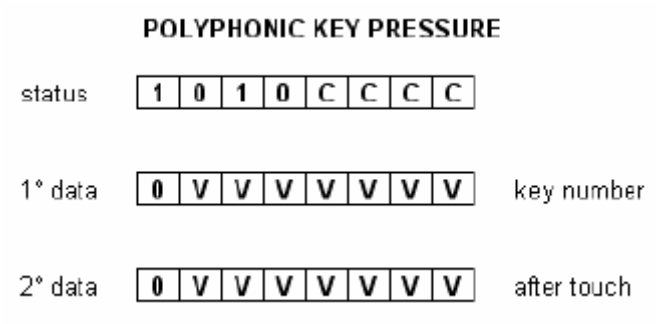


**Figura 2.10 – MIDI Polyphonic Key Pressure**

- PROGRAMME CHANGE: messaggio inviato per segnalare un cambio di strumento con il quale suonare le note in arrivo su uno specifico canale. Questo rappresenta la modifica del timbro associate alle note. Solo un Data Byte si rende necessario e contiene il nuovo programme number.

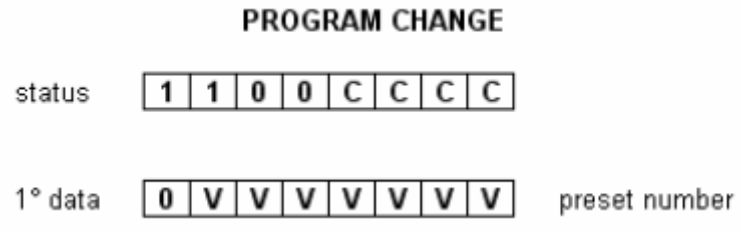


**Figura 2.11 – MIDI Program Change**

- PITCH BENDER: messaggio inviato per variare il pitch della nota in esecuzione su un determinato canale. Sono necessari due Data Byte per portare un'informazione di pitch ad una risoluzione sufficientemente alta da far apparire il cambiamento come continuo.

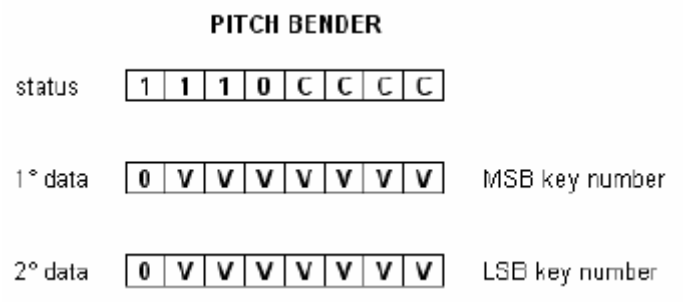


**Figura 2.12 – MIDI Pitch Bender**

- CONTROLL CHANGE: messaggio che viene utilizzato per controllare una vasta gamma di funzioni del sintetizzatore ed elencate in una tabella da 128 entry (il massimo numero di combinazioni possibili utilizzando i 7 bit del primo Data Byte). Il secondo Data Byte associato al messaggio specifica il valore del parametro indicato dal primo.

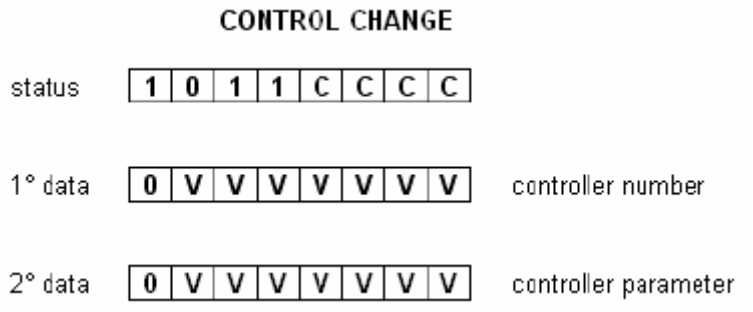


**Figura 2.13 – MIDI Control Change**

Non tutti i 128 valori di control change sono definiti, ma alcuni rimangono a disposizione per eventuali definizioni future. Particolare importanza rivestono gli ultimi 4 che costituiscono i Mode Message di cui si parlerà nel prossimo paragrafo.

| | | | |
|---|---|---|---|
| 0 | Bank Select | 45 | Effect control 2 |
| 1 | Modulation wheel | 46...47 | Non definito |
| 2 | Breath control | 48...51 | General #1,2,3,4 |
| 3 | Non definito | 52...63 | Non definito |
| 4 | Foot control | 64 | Dumper pedal |
| 5 | Portamento time | 65 | Portamento on/off |
| 6 | Data Entry | 66 | Sostenuto on/off |
| 7 | Channel Volume | 67 | Soft pedal on /off |
| 8 | Balance | 68 | Legato Footswitch |
| 9 | Non definito | 69 | Hold 2 |
| 10 | Pan | 70...79 | Sound Controller 1-10 |
| 11 | Expression Controller | 80...83 | General Purpose 5,6,7,8 |
| 12 | Effect control 1 | 84 | Portamento control |
| 13 | Effect control 2 | 85...90 | Non definito |
| 14...15 | Non definito | 91...95 | Effects 1,2,3,4,5 Depth |
| 16...19 | General #1,2,3,4 | 96 | Data entry +1 |
| 20..31 | Non definito | 97 | Data entry -1 |
| 32 | Bank Select | 98 | NRPN LSB |
| 33 | Modulation wheel | 99 | NRPN MSB |
| 34 | Breath control | 1000 | RPN LSB |
| 35 | Non definito | 101 | RPN MSB |
| 36 | Foot control | 102...119 | Non definito |
| 37 | Portamento time | 120 | All Sound Off |
| 38 | Data Entry | 121 | Reset All Controllers |
| 39 | Channel Volume | 122 | Local control on/off |
| 40 | Balance | 123 | All note off |
| 41 | Non definito | 124 | Omni mode off |
| 42 | Pan | 125 | Omni mode on |
| 43 | Expression | 126 | Poly mode on/off |
| 44 | Effect control 1 | 127 | Poly mode on |

**Figura 2.14 – MIDI Mode Messages**

2.4.1.2 Mode Message

I Mode Messages, che corrispondono ai MIDI Mode Messages che vanno dal 121 al 127, come suggerito dal nome, descrivono il modo in cui il sintetizzatore risponde ai dati in arrivo sui canali:

- 121: resetta tutti i controller;
- 122: abilita/disabilita il controllo locale;
- 123: disabilita riproduzione;
- 124-125: disabilita/abilita la modalità OMNI, il che permette al sintetizzatore di rispondere ai messaggi in arrivo rispettivamente su un singolo canale/tutti i canali.
- 126-127: disabilita/abilita la modalità POLY, il che permette al sintetizzatore di suonare rispettivamente una sola nota alla volta/più note contemporaneamente.

OMNI ON:

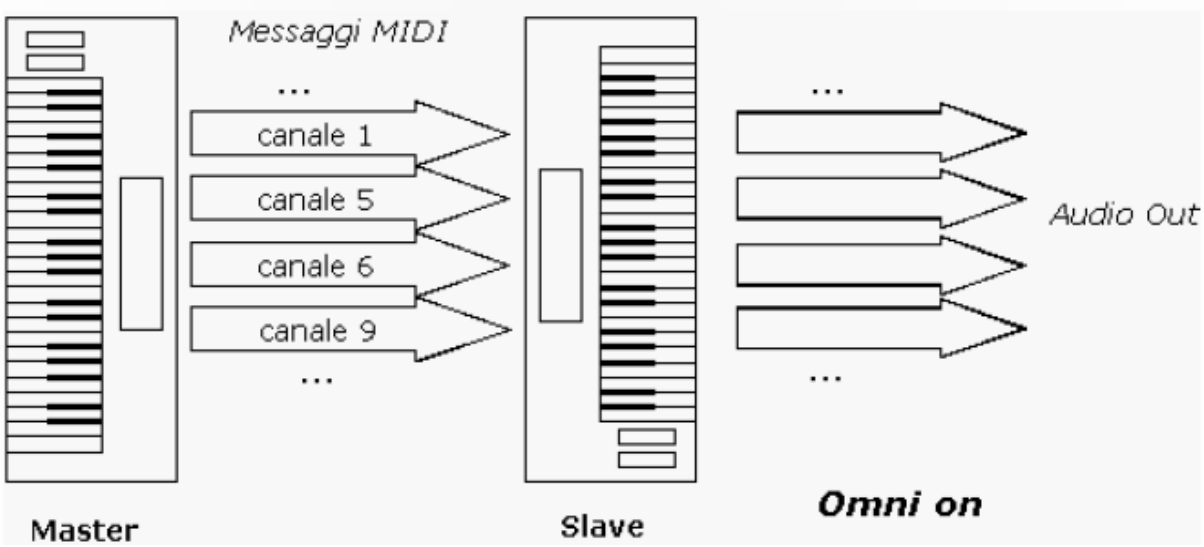


**Figura 2.15 – MIDI Omni On**

OMNI OFF:

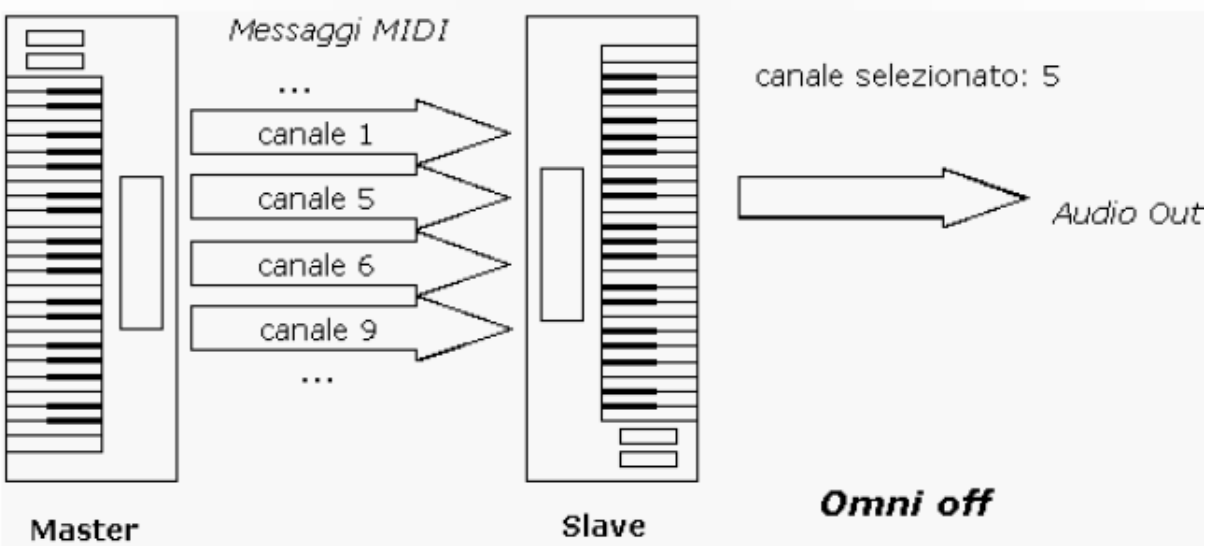


**Figura 2.16 – MIDI Omni Off**

Dati i tre stadi, è possibile ottenere 4 modi operativi:

- Omni On/Poly: il sintetizzatore suonerà polifonicamente tutti i messaggi ricevuti su tutti e 16 i canali;
- Omni On/Mono: il sintetizzatore, in ascolto su tutti i canali, suonerà una sola nota per volta.

- Omni Off/Poly: il sintetizzatore suonerà in modo polifonico tutte le note in arrivo su uno specifico canale;
- Omni Off/Mono: il sintetizzatore, in ascolto su un unico canale, suonerà una sola nota.

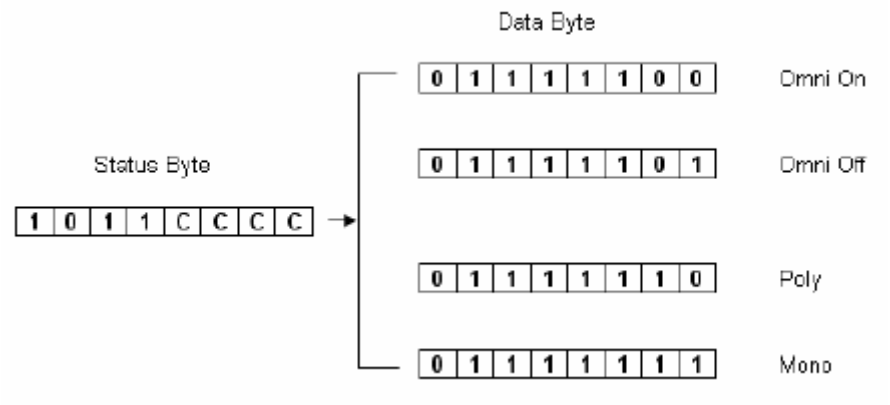


**Figura 2.17 – MIDI Modes**

## 2.4.2 System Message

System Message, come detto, non sono messaggi di canale con le funzionalità simili a quelle precedentemente descritte, ma sono utilizzati per la gestione del sistema MIDI.
Si dividono in 3 categorie: Common Message, Real Time Message e System Exclusive Message (SysEx).

I Common Message danno istruzioni generali al sistema:

- MIDI Time Code Quarter Frame: per la sincronizzazione degli strumenti;
- Song Position Pointer: permette la riproduzione di una sequenza audio da una posizione diversa da quella di partenza. Il puntatore, costantemente incrementato durante la riproduzione della sequenza, è sempre multiplo di un beat (6 MIDI clocks), mantiene il riferimento alla posizione di riproduzione corrente all'interno della sequenza;
- Song Select: specifica quale canzone o sequenza di un sequencer deve essere suonata al ricevimento del messaggio di avvio;
- Tune Request: utilizzato con i sintetizzatori analogici, per richiedere l'accordo di tutti gli oscillatori.

I Real Time Message sono messaggi utilizzati per sincronizzare tutti gli strumenti che utilizzano il clock MIDI. Questi messaggi posso essere mandati in qualsiasi momento per permettere al sistema di rimanere sincronizzato anche durante l'esecuzione. Sono 6:

- MIDI Clock: messaggio utilizzato per sincronizzare gli strumenti MIDI clock-based. Viene inviato 24 volte per ogni nota del valore di un quarto.

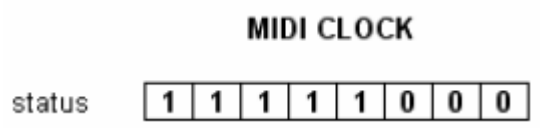


**Figura 2.18 – MIDI Clock**

- Start: messaggio inviato quando viene premuto il pulsante start sul master e ha lo scopo di avvertire tutti gli slave di partire all'inizio della canzone o sequenza.


**Figura 2.19 – MIDI Start**

- Continue: messaggio che fa continuare la sequenza dal punto in cui si era interrotta.

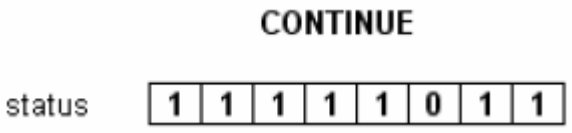


**Figura 2.20 – MIDI Continue**

- Stop: arresta la riproduzione di tutti gli slave


**Figura 2.21 – MIDI Stop**

- Active Sensing: mantiene la connessione attiva tra master e slave. Viene inviato ogni 300 ms se nessun altro messaggio MIDI è stato scambiato in quel lasso di tempo.


**Figura 2.22 – MIDI Active Sensing**

- System Reset: riporta tutti gli slave nelle loro impostazioni di default.

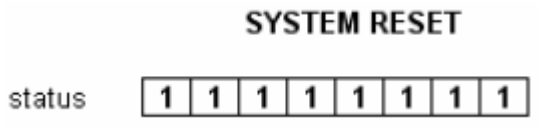


**Figura 2.23 – MIDI System Reset**

I System Exclusive Message (SysEx) sono particolari messaggi che non fanno riferimento a nessun canale e hanno lo scopo di inviare particolari parametri, dati o istruzioni ad un determinato strumento/gruppo di strumenti tra gli slave.
Questi messaggi hanno una loro particolare forma che dipende dalle aziende produttrici degli strumenti. Ciò vuol dire che non tutti gli strumenti riescono ad interpretare il contenuto di questi messaggi, nel qual caso rispondono con opportuni messaggi di errore.
Tutti i parametri e le funzionalità di uno strumento gestibili tramite l'utilizzo di questi particolari messaggi sono, di solito, indicati nei vari manuali rilasciati dalle case produttrici dello strumento.
Per evitare conflitti con messaggi esclusivi non compatibili, ogni strumento è fornito di un particolare ID con il quale è possibile verificarne la compatibilità a priori.

# Capitolo 3
# PROTOCOLLO RTP-MIDI

## 3.1 Cenni generali

Il protocollo RTP-MIDI (Real Time Protocol MIDI) è un protocollo utilizzato per il trasporto di messaggi MIDI utilizzando pacchetti RTP in qualsiasi rete Ethernet o Wi-Fi.
Anche conosciuto come AppleMIDI, il protocollo offre notevoli servizi aggiuntivi rispetto alla versione MIDI 1.0 come, per esempio, gestione della sessione MIDI, sincronizzazione degli strumenti, controllo sulla perdita di pacchetti e rigenerazione dei dati andati perduti.
Introdotto per la prima volta nel 2005 dalla compagnia Apple come parte del suo sistema operativo, oggi RTP-MIDI è utilizzabile sulla quasi totalità delle piattaforme esistenti grazie ai driver dedicati che rendono compatibile il protocollo con i sistemi sottostanti. In questo ultimo caso, a differenza dei sistemi Mac OS per i quali un'apposita classe (Bonjour) dedicata venne creata per la gestione dei dispositivi compatibili con il protocollo, la connessione con i dispositivi che intendono prendere parte alla sessione avviene manualmente, immettendo indirizzo IP e porte.
Il driver RTP-MIDI, nel momento della creazione di una nuova sessione virtuale, crea delle porte MIDI, chiamate "Sessioni", che appaiono come vere e proprie porte MIDI IN / MIDI OUT, come avviene per il protocollo MIDI 1.0.

Per il progetto, utilizzando Windows OS, è stato necessario installare un'implementazione Windows del driver RTP-MIDI di Apple rilasciata da Tobias Erichsen. Il driver utilizza un pannello di connessione per connettere il PC al controller e crea delle porte virtuali che sono visibili a qualsiasi applicazione MIDI in esecuzione sul PC.

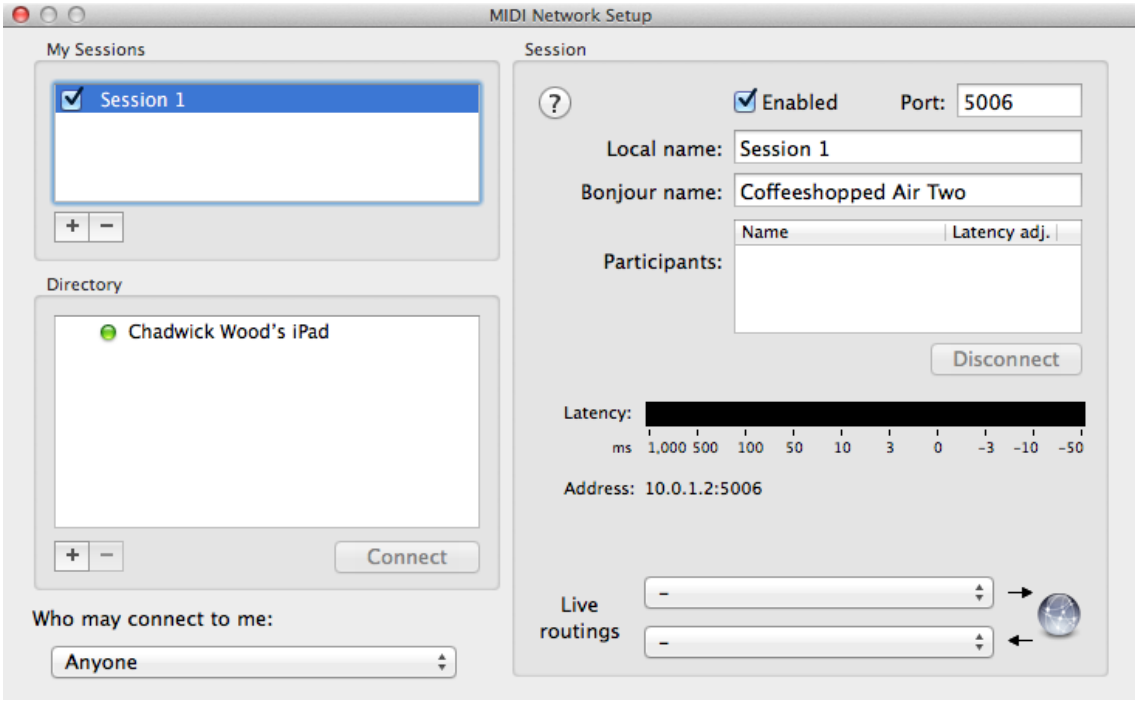


**Figura 3.1 - Interfaccia connessione sessione MIDI**

Per permettere al controller di essere individuabile dal PC e supportare il protocollo viene sfruttata una libreria appositamente creata per la piattaforma Arduino, chiamata AppleMidi.h, che appunto permette alla scheda, provvista di adattatore Ethernet o Wi.Fi shield, di sfruttare il protocollo RTP-MIDI per scambiare messaggi MIDI con sintetizzatori esterni.

## 3.2 **Sessioni RTP-MIDI**

Nell'instaurazione di una connessione tra due devices che supportino il protocollo RTP-MIDI, è necessario l'apertura di due porte, chiamate sessioni virtuali, che costituiscono un percorso di comunicazione tra i due dispositivi.

La creazione di una sessione presuppone sempre e solo la presenza di due dispositivi (controllore e ascoltatore) anche se un controller può creare una moltitudine di sessioni parallele tra dispositivi diversi che non hanno consapevolezza dell'esistenza delle altre sessioni e non possono interferire con queste ultime. In questo ultimo caso il controllore espone la stessa interfaccia duplicando di fatto i flussi MIDI verso dispositivi diversi in ascolto ma che condividono le stesse porte con il controllore.

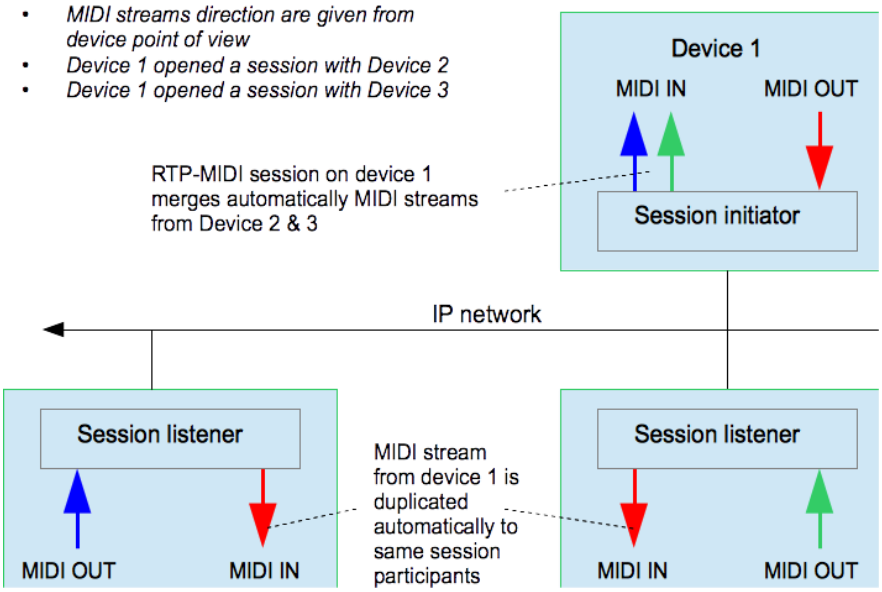


**Figura 3.2 - RTP-MIDI Session**

Solitamente un controllore ha il compito di inizializzare la sessione e gestire la sincronizzazione tra i due dispositivi. Un controllore ha la possibilità di essere, a sua volta, un ascoltatore ma non è sempre vero il contrario.

La definizione di una sessione stabilisce la connessione tra gli endpoint del controllore e dell'ascoltatore (MIDI IN controllore con MIDI OUT ascoltatore e viceversa) così come avviene "fisicamente" nel caso della connessione wired del MIDI 1.0.

Un endpoint può accettare più sessioni. In questo caso, tutti i flussi MIDI in arrivo sono uniti insieme prima di spedire i dati all'applicazione che li gestisce. Al contrario, i dati in uscita dal sistema e diretti verso un endpoint connesso a più sessioni, sono duplicati e inviati a tutti i sistemi connessi.

## 3.3 **Protocollo di sessione**

Un progetto iniziale, descritto anche nell'RFC6295, prevedeva l'uso del protocollo SDP (Session Description Protocol) e SIP (Session Iniziator Protocol) per l'instaurazione e la gestione delle sessioni MIDI. Questi due protocolli, però, non sono mai stati utilizzati in questo ambito perché ritenuti troppo complessi e pesanti, soprattutto se si considera che molti dispositivi che prendono tranquillamente parte ad una sessione MIDI hanno ridotte capacità di

calcolo e non sarebbero quindi compatibili con i protocolli SDP e SIP oppure, se compatibili, sarebbero affetti da latenze così importanti da diventare incompatibili all'intero protocollo RTP-MIDI.

Per questo motivo è stato creato, e si utilizza tutt'ora, un protocollo dedicato per la creazione, gestione e chiusura della sessione MIDI virtuale. Tale protocollo è appunto conosciuto come AppleMidi.

Il protocollo richiede la creazione di due porte UDP:

- Control Port: utilizzata per l'invio di comandi;
- Data Port: utilizzata per l'invio di dati.

Le due porte sono consecutive, la prima precedente alla seconda e possono essere due qualsiasi tra e 65536 disponibili purché non impegnate da altri processi applicativi.

In tutte le fasi previste dal protocollo, la gestione è affidata a particolari messaggi formati da 32 bit di cui i primi due impostati a 255 e gli ultimi due impostati ad un valore che descrive il significato del messaggio.

| Description | Wireshark header definition | Field value (hex) | Field value (chars) |
|---|---|---|---|
| Invitation | `APPLEMIDI_COMMAND_INVITATION` | `0x494e` | `IN` |
| Invitation accepted | `APPLEMIDI_COMMAND_INVITATION_ACCEPTED` | `0x4f4b` | `OK` |
| Invitation refused | `APPLEMIDI_COMMAND_INVITATION_REJECTED` | `0x4e4f` | `NO` |
| Closing session | `APPLEMIDI_COMMAND_ENDSESSION` | `0x4259` | `BY` |
| Clock synchronization | `APPLEMIDI_COMMAND_SYNCHRONIZATION` | `0x434b` | `CK` |
| Journalling synchronization | `APPLEMIDI_COMMAND_RECEIVER_FEEDBACK` | `0x5253` | `RS` |
| Bitrate | `APPLEMIDI_COMMAND_BITRATE_RECEIVE_LIMIT` | `0x524c` | `RL` |

**Figura 3.3 - AppleMIDI Session Messages**

L'utilizzo di tali messaggi è legata a specifiche fasi della sessione MIDI:

- Invitation Phase: si instaura la connessione tra controllore e ascoltatore;
- Synchronization Phase: i dispositivi appena connessi cominciano lo scambio di informazioni volte alla loro sincronizzazione, minata dalla latenza della rete. Questa fase è ripetuta diverse volte all'inizio della sessione per migliorare l'accuratezza dei valori calcolati e si ripete ciclicamente da 2 a 6 volte al minuto per mantenere aggiornati i valori di sincronizzazione e per verificare un'eventuale disconnessione dell'ascoltatore;
- Journal Update: permette, in caso di perdita di messaggi MIDI in rete, di recuperare le informazioni perse direttamente lato ricevente, evitando così di inviarli di nuovo.

Questo è possibile mantenendo in memoria "immagini MIDI" riferite al dispositivo remoto connesso e relative a diversi istanti ciclici di corretto funzionamento della sessione.

- Disconnection: avviene la disconnessione dei partecipanti alla sessione.

I messaggi utilizzati nelle varie fasi hanno il seguente utilizzo:

- Invitation: messaggio inviato dal controllore della sessione ad un ipotetico ascoltatore pronto per la connessione sulla porta di controllo durante la fase di instaurazione della connessione;
- Invitation Accepted/refused: messaggio inviato dall'ascoltatore al controllore sulla porta di controllo dopo la ricezione del messaggio di invitation per accettare/rifiutare la connessione.
- Clock Synchronization: messaggio utilizzato per scambiarsi informazioni per la sincronizzazione temporale dei dispositivi connessi nella stessa sessione. In particolare, all'inizio della sessione, il controllore invia un messaggio di clock synchronization (CK0) all'ascoltatore. L'informazione contenuta in questo primo messaggio rappresenta il valore di local time, espresso in 64 bit, del controllore. Se l'ascoltatore non risponde a questo messaggio vengono effettuati altri tentativi e, in caso di reiterata mancata risposta, l'ascoltatore viene giudicato disconnesso e la sessione cade. In caso contrario, a sua volta, l'ascoltatore risponderà con un messaggio CK1 che recapita al controllore il suo local time. Con questi due messaggi, entrambi i dispositivi sono a conoscenza di entrambi i local time e possono calcolare un valore di offset da applicare alle informazioni temporali del protocollo (Timestamp e Deltatime) per la sincronizzazione della sessione. Dopo tale calcolo, il controllore della sessione invia un ultimo messaggio all'ascoltatore, CK2, e che contiene i 64 bit del suo local time rilevato nel momento esatto della ricezione del messaggio CK1. In questo modo è possibile calcolare un valore di latenza media della rete, necessario per la corretta gestione della comunicazione tra i due dispositivi.
- Journalling Synchronization: messaggio inviato ciclicamente da ciascun partecipante alla sessione virtuale e che indica l'ultimo numero di sequenza ricevuto correttamente. In caso non ci siano errori di trasmissione, il partecipante può cancellare, se necessario, la memoria contenente i vecchi dati di journalling;

- Closing session: messaggio inviato da un partecipante (sia controllore che ascoltatore) alla sessione per lasciarla, il che causa anche il rilascio di tutte le risorse allocate per la sessione e la chiusura della stessa.

## 3.4 **Pacchetto RTP-MIDI**

L'unica parte obbligatoria specificata negli RFC 4695 e 6295, nei quali viene definito il protocollo RTP-MIDI, e che deve essere assolutamente rispettata per la corrispondenza al protocollo in essi descritto è il formato del payload dei pacchetti RTP che trasportano informazioni MIDI.
Un generico pacchetto RTP-MIDI assume il seguente formato:

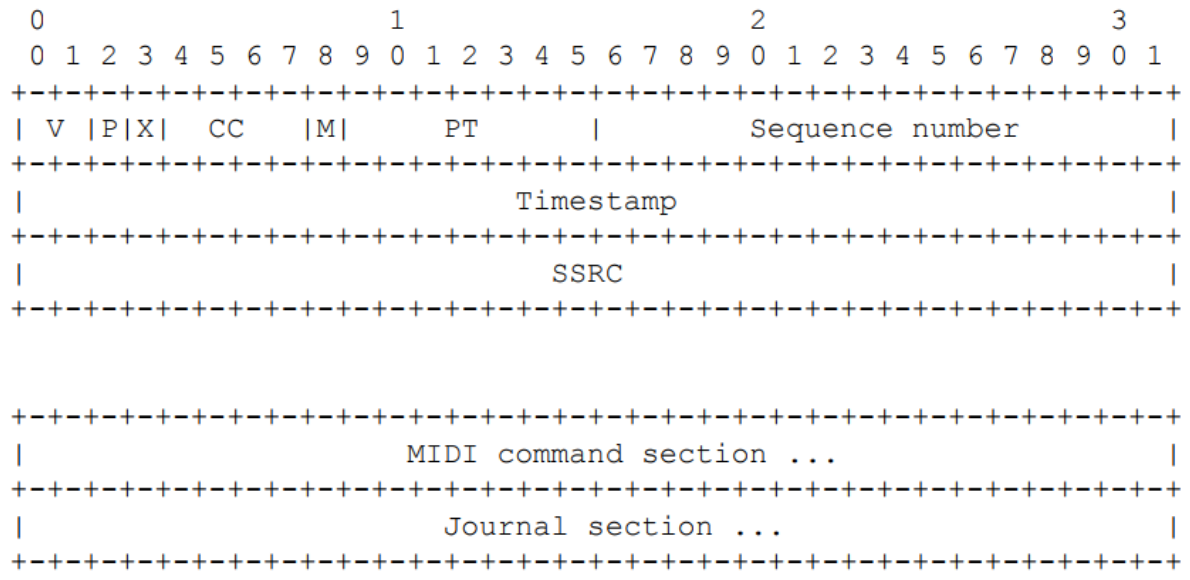


**Figura 3.4 - RTP-MIDI Packet**

Nella parte di header del pacchetto le informazioni più importati, in questo caso sono:

- Bit M: media type dello stream. Nel caso in cui la sezione dedicata ai comandi MIDI non sia nullo il bit deve essere necessariamente settato a 1 mentre, in caso contrario, a 0;
- Sequence Number: valore a 16 bit, inizialmente impostato ad un valore random, che si incrementa ad ogni pacchetto inviato e serve per la detection di pacchetti persi o duplicati;

- Timestamp: valore a 32 bit utilizzato per settare il valore di base del timestamp per il pacchetto, caratterizzato dal parametro di clock rate che per le sessioni MIDI deve assumere un valore consono alle informazioni che il pacchetto trasporta (32, 44,1, 48, 64, 88,2, 96, 176,4, 192 KHz);
- MIDI Command Section: sezione di payload relativa ai comandi midi che il pacchetto trasporta;
- Journal section: sezione, opzionale, che porta le informazioni per la funzionalità di journalling.

### 3.4.1 MIDI Command Section

La sezione di payload di un pacchetto RTP-MIDI ha una particolare struttura e porta al suo interno la lista di uno o più comandi MIDI associati a determinati parametri di utilizzo.

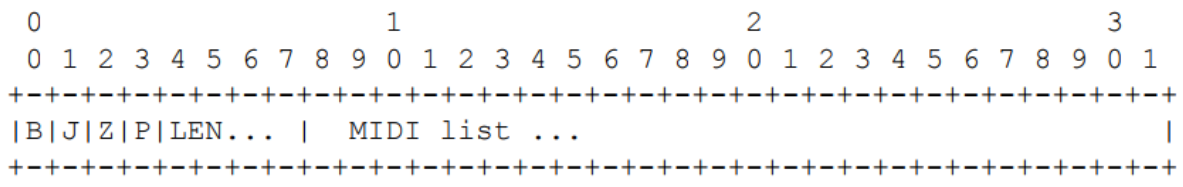


**Figura 3.5 - MIDI Command Section**

Una serie di parametri danno inizio alla sezione dedicata ai comandi MIDI:

- B: indica la lunghezza dell'header. Se 0, l'header ha lunghezza 1 byte di cui il parametro LEN ne occupa la metà, mentre se 1 la lunghezza dell'header raddoppia e il paramentro LEN passa da 4 a 12 bit di lunghezza;
- J: indica la presenza o meno della sezione relativa al journalling;
- Z: da indicazioni sulla struttura della MIDI list che potrebbe cominciare con un valore di Delta Time diverso da 0 e seguito da un comando MIDI (Z=1) oppure direttamente con un comando MIDI e Delta Time nullo (Z=0);
- LEN: indica la lunghezza in byte della MIDI list in cui sono contenuti i comandi MIDI;
- MIDI list: sezione dedicata alla lista di comandi MIDI trasportata dal pacchetto.

La struttura di una MIDI list è molto semplice e appare come una vera e propria lista composta da coppie di Delta Time, necessario per la sincronizzazione dei dispositivi, e comandi MIDI. La lista termina sempre con un comando MIDI vuoto.

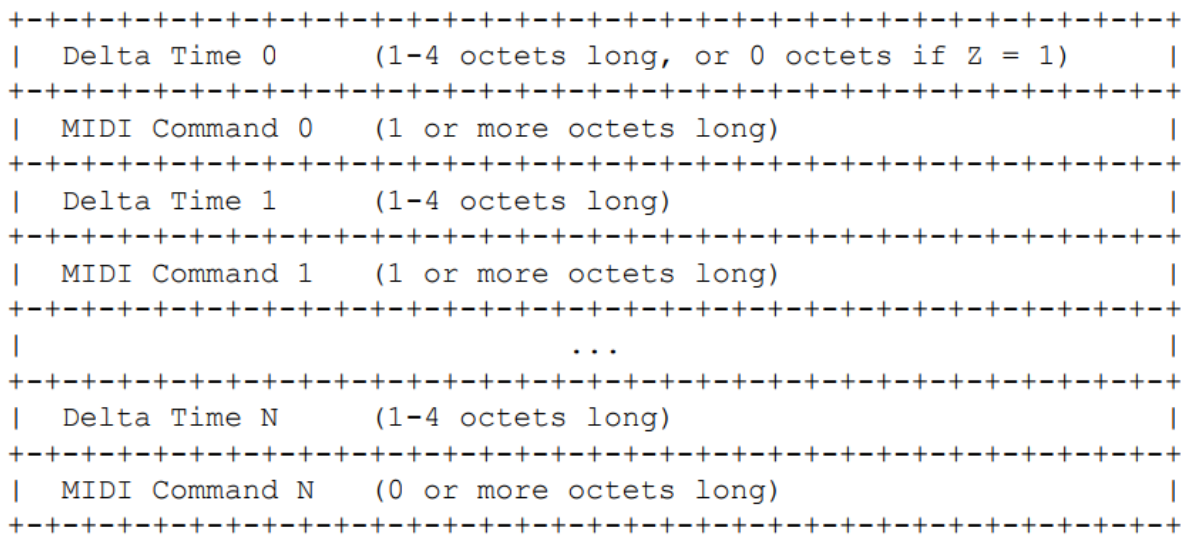


**Figura 3.6 - MIDI List**

### 3.4.2 Journal section

La sezione del pacchetto RTP-MIDI dedicata alla funzionalità di journalling è l'ultima della struttura e del tutto opzionale, anche se tale sezione è quasi sempre implementata e usata per la sua utilità. Tramite questa sezione infatti viene implementato l'unica tecnica di ripristino delle informazioni perse nel tragitto dei messaggi in rete. Utilizzando il protocollo UDP, infatti, non esiste una tecnica nativa per il recupero dei pacchetti persi, a differenza del TCP.
La perdita di un pacchetto MIDI può causare degli artefatti molto importanti nella riproduzione del suono ricostruito, motivo per cui quasi in nessun caso la sezione di Journal viene omessa.
Tale tecnica non utilizza la ritrasmissione dei pacchetti persi come avviene per il TCP ma si basa sulla trasmissione, insieme ai comandi MIDI da eseguire, delle informazioni passate fino ad un certo pacchetto che sarà chiamato pacchetto di checkpoint.
Tale "storia" viene inviata per ogni pacchetto spedito di modo che quando c'è perdita di un pacchetto è possibile ricostruire il flusso corretto facendo affidamento esclusivamente sulle informazioni del ricevente, facendo una semplice differenza tra l'ultima history ricevuta precedente al packet loss e quella appena inviata dal mittente.
Il formato sella sezione di journal è strutturato in tre livelli:

- Top level header;
- Channel and system journal headers;
- Chapters.

Nella figura è mostrato l'header di livello maggiore del formato:

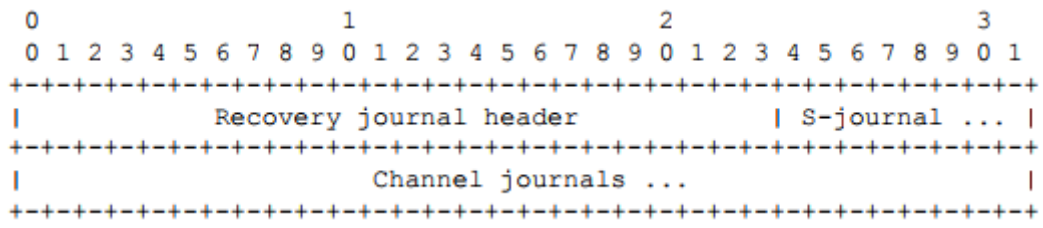


**Figura 3.7 - Top Level Header**

Mentre il System journal e Channel journal sono campi opzionali del primo header, il Recovery journal header ne costituisce la parte essenziale ed assume la seguente struttura:

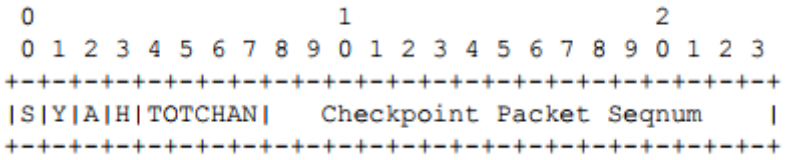


**Figura 3.8 - Recovery journal header**

In cui:

- S: bit impostato per il ricevente del pacchetto affinché classifichi o meno il recovery journal come necessario per il recupero di un pacchetto perso;
- Y: bit utilizzato per informare della presenza o meno del valore di System journal nella sezione;
- A: bit utilizzato per informare della presenza o meno di TOTCHAN + 1 valori di Channel journals (Channel MIDI) all'interno della sezione;
- H: bit utilizzato per indicare se i canali MIDI sono configurati per utilizzare una codifica migliorata per il MIDI control change;
- Checkpoint Packet Seqnum: valore del numero di sequenza del pacchetto di checkpoint per il journal che definisce di quanti pacchetti passati è composta la storia di checkpoint.

Il Channel journal contiene tutte le informazioni necessarie al ripristino di un singolo voice channel e presenta la seguente struttura:

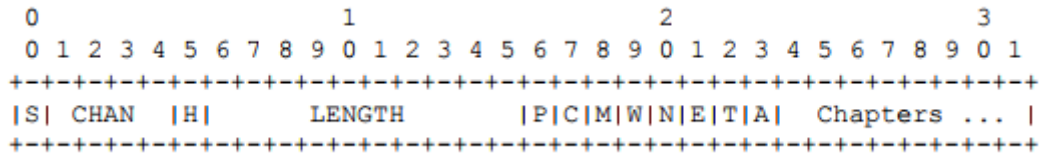


**Figura 3.9 - Channel journal format**

Dove:

- H e S conservano lo stesso significato descritto in precedenza;
- CHAN: valore del canale MIDI;
- LENGHT: 10 bit utilizzati per indicare la grandezza del channel journal:

La parte finale del channel journal è occupata dai channel Chapters che vengono utilizzati, all'interno delle informazioni del channel journal, per la recovery di comandi MIDI.
In particolare:

- P: MIDI Programme Change;
- C: MIDI Control Change;
- M: MIDI Parameter System;
- W: MIDI Pitch Wheel;
- N: MIDI Note Off;
- E: MIDI Note Command Extras;
- T: MIDI Channel Aftertouch;
- A: MIDI Poly Aftertouch;

Il System journal, invece, viene utilizzato per codificare le informazioni di ripristino di tutti i comandi di sistema e assume la seguente struttura:

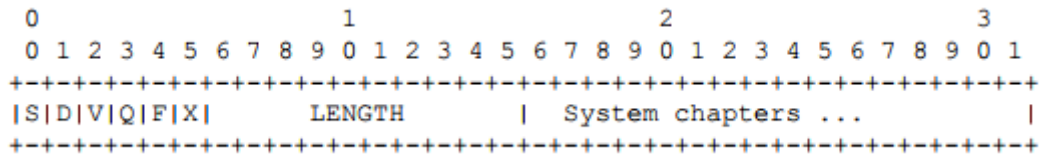


**Figura 3.10 - System journal**

In cui:

- S e LENGHT: mantengono lo stesso significato visto in precedenza;
- D: può assumere diversi significati a seconda del valore assegnato. In particolare può essere utilizzato per indicare comandi di: Song Select, Tune Request, Reset, comando di sistema sconosciuto;
- V: utilizzato per il comando di Active Sense;
- Q: Sequencer Stete;
- F: MTC Tape Position;
- X: utilizzato per indicare i messaggi di System Exclusive.

La parte finale del System journal è occupata dai System Chapters che servono per codificare le informazioni associate ad una o più classi di comandi MIDI di sistema che sono specificati nella prima parte della struttura.

# Capitolo 4
# L'HARDWARE

La realizzazione fisica del controller ha richiesto l'utilizzo di un discreto numero di componenti elettronici al fine di eseguire tutte le funzionalità che erano state pensate in fase progettuale.
Nello specifico, per il progetto è stato utilizzato:

- Microcontrollore Arduino Mega R3 2560,
- Arduino Wi-Fi Shield R3,
- Strip position sensor,
- Strip pressure sensor,
- LCD 20x4,
- Adattatore I2C,
- Modulo bluetooth HC-06,
- Connettore MIDI DIN a 5 poli,
- 4 bottoni,
- On/Off switch,
- 31 led
- 37 resistenze 10 KΩ, 1 da 220 Ω, 1 da 1 MΩ,
- Breadbord,
- Batteria 9V,
- Cavi per i collegamenti.

L'intero controller, infine, è stato adattato ad un opportuno case realizzato per lo scopo.
Con l'ausilio del software "Fritzing" è stato realizzato uno schema circuitale che raccoglie con buona approssimazione tutti i componenti sopra elencati e le connessioni che sono state operate nella realizzazione del controller. Sebbene alcuni componenti non rappresentano esattamente quelli utilizzati nel progetto reale, questo ausilio è stato molto utile in fase di progettazione, soprattutto per i sui aspetti legati alla simulazione poiché mette a riparo da possibili danni

irreversibili ai componenti in caso di connessioni errate o qualsiasi altro problema possa scaturire da un non corretto utilizzo degli stessi.
Di seguito è mostrato lo schema generale del controller.

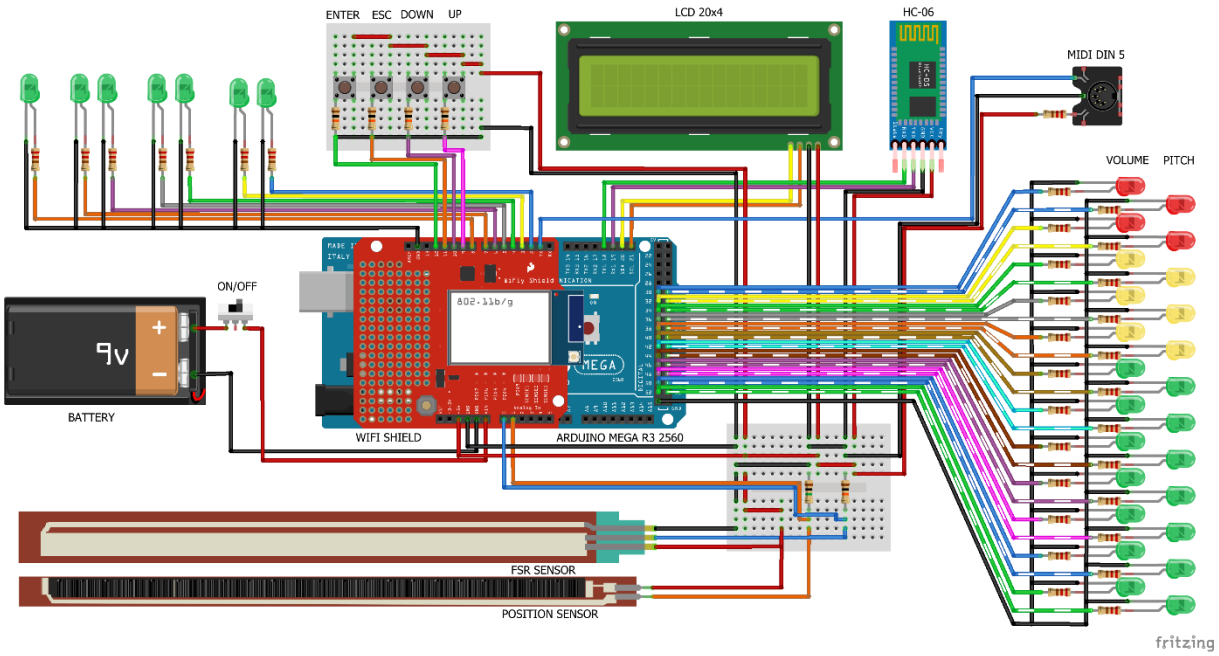


**Figura 4.1 – Schema moduli MIDI Controller**

Il centro dello schema è occupato dal microcontrollore Arduino Mega e dalla Wi-Fi shield che è connessa su di esso.
Tutto il sistema è alimentato da una batteria di 9V connessa all'microcontrollore tramite gli appositi pin di alimentazione esterna "Vin" e "GRD". Un semplice switch On/Off, inserito sul cavo positivo di alimentazione, regola l'accessione e lo spegnimento del controller.
Nella parte alta, un LCD si occupa di mostrare all'utente lo stato di funzionamento del controller e di permettere la modifica dei parametri di funzionamento del sistema. L'LCD è connesso ad un modulo I2C (Inter Integrated Circuit) per abilitare la schermo ad un uso di tipo Master-Slave con il microcontrollore e per evitare un cablaggio molto più complesso. Con questo modulo aggiuntivo, le connessioni tra l'LCD e l'Arduino passano da 16 (numero di connessioni tra l'LCD e il modulo I2C) a 4. In particolare le uniche connessione di cui si ha bisogno sono:

- +5V,
- GRD,
- SDA: (Serial Data) per lo scambio di dati,
- SCL: (Serial Clock) per la sincronizzazione del Master con lo Slave (segnale di clock).

Nella figura viene mostrato l'utilizzo del modulo I2C per la connessione dell'LCD al microcontrollore.

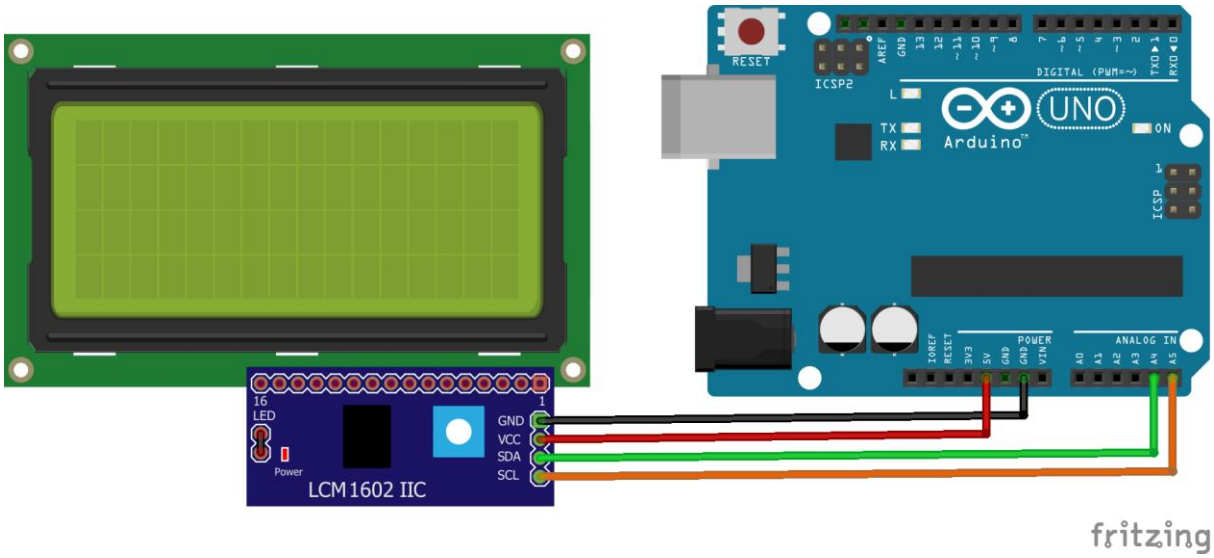


**Figura 4.2 – I²C Module Connections**

Per permettere all'utente la gestione dei parametri di funzionamento del controller, è stata inserita una piccola tastiera formata da 4 pulsanti.

La tastiera è formata da semplici bottoni collegati a 4 distinti pin digitali del controllore:

- Pin 8: bottone UP,
- Pin 9: bottone DOWN,
- Pin 10: bottone ESC,
- Pin 11: bottone ENTER.

Per permettere un corretto passaggio tra i due stati possibili associati ai bottoni (LOW-HIGH) ed evitare qualsiasi problema di falsa pressione dovuto a correnti parassite, è stata inserita una resistenza di 10KΩ, come da specifiche del produttore, sul percorso verso massa di modo da aumentare la resistenza complessiva e fermare qualsiasi corrente parassita.

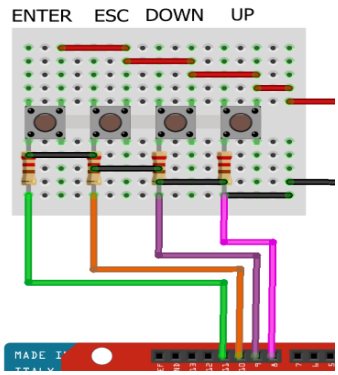


**Figura 4.3 – Button Connections**

Nella parte sinistra dello schema circuitale è presente il connettore DIN 5 utilizzato per la connessione wired del controller al dispositivo di sintesi.
Il connettore è collegato alla porta seriale 0 del microcontrollore e in particolare sul pin TX, ovvero quello abilitato alla scrittura dei dati. In questo caso nessun collegamento è stato effettuato tra il connettore DIN e il pin RX della porta seriale poiché il controller non è abilitato alla ricezione dei dati MIDI ma solo alla scrittura degli stessi.
Gli altri due collegamenti presenti tra il connettore e l'Arduino riguardano la massa e la VCC per la quale un'opportuna resistenza regola la corrente in ingresso.
Nell'immagine seguente sono mostrati i collegamenti effettuati secondo le specifiche del connettore.

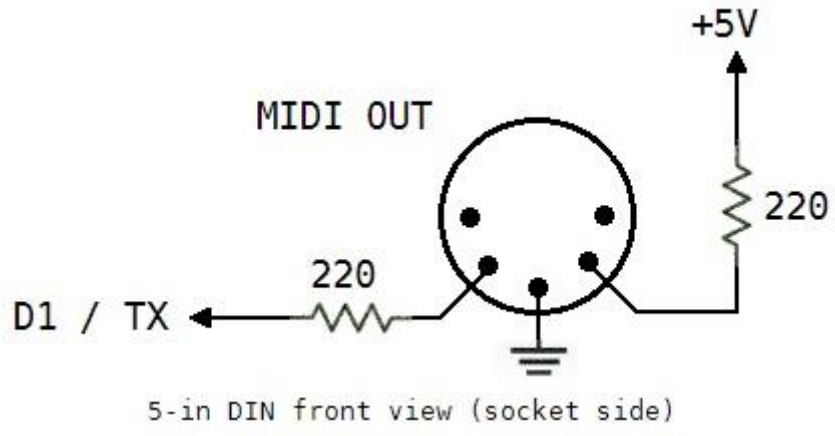


**Figura 4.4 – Schema elettrico DIN 5 poli**

Nella parte in altro a destra dello schema invece è presente un modulo bluetooth HC-06 inserito nel sistema con il solo scopo di fornire all'utente la possibilità di modifica dei valori di SSID e PASSWORD per la connessione alla rete. Attuare questa operazione tramite l'ausilio dell'LCD e dei 4 pulsanti, infatti, sarebbe stato una procedura lunga e piena di insidie. L'utilizzo del modulo bluetooth invece offre la possibilità all'utente di connettere al controller un qualsiasi terminale bluetooth tramite il quale poter "dialogare" con il controller stesso durante la procedura di modifica delle credenziali di connessione alla rete LAN.
Il modulo HC-06 presenta 6 pin dei quali solo 4 sono utilizzati per lo scopo:

- VCC: collegato ai 5V del microcontrollore,
- GRD: collegato alla massa del microcontrollore,
- RX: collegato al pin TX1 della porta seriale 1 del microcontrollore,
- TX: collegato al pin RX1 dalla porta seriale 1 del microcontrollore.

Il modulo, quindi, è collegato ad una delle porte seriale dell'Arduino ed è attivato solo e soltanto quando l'utente invoca la procedura di modifica delle credenziali tramite l'apposita voce di menù. In modalità di funzionamento normale del controller, quindi, il dispositivo bluetooth HC-06 non è visibile ai dispositivi esterni e un'eventuale interazione con lo stesso non produce alcun seguito nel controller.

Nell'immagine sono mostrati i collegamenti appena descritti.

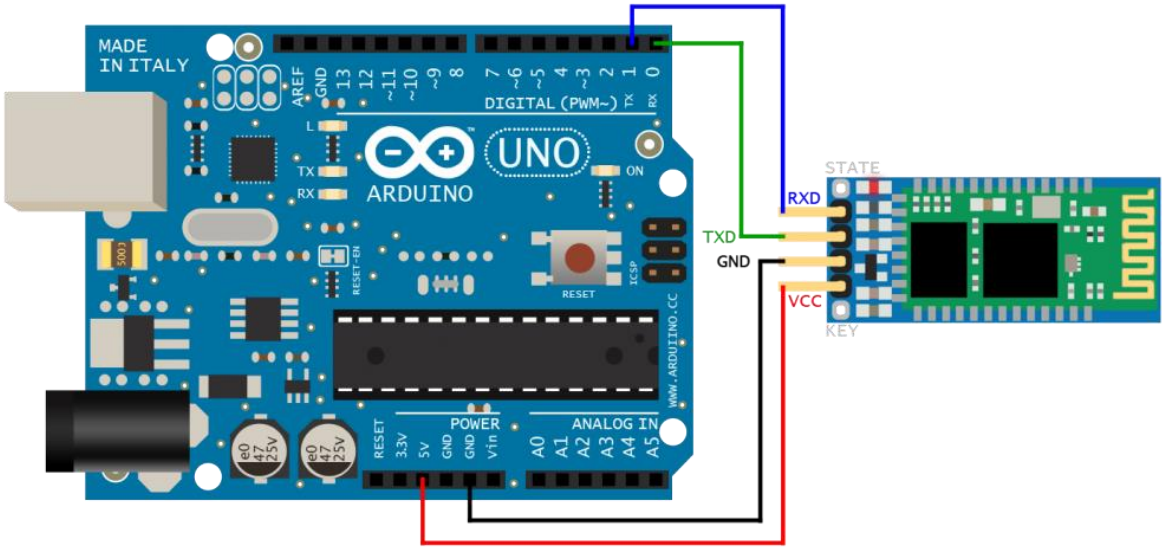


**Figura 4.5 – HC-06 Module Connections**

Gli ultimi due elementi che compongono il controller sono i due sensori che rappresentano la vera e propria interfaccia musicale con l'utente.


**Figura 4.6 - Sensore di posizione e pressione**

Il primo è quello di pressione il quale fornisce un valore sulla scala 0-1023 in relazione alla pressione esercitata su di esso.

Il secondo invece reagisce in base alla posizione in cui viene toccato. Entrambi hanno una lunghezza di 50 cm e sono sovrapposti: il primo sotto il secondo.

In questo modo l'utente con un unico tocco può controllare due valori contemporaneamente:

quello di posizione, quasi sempre associato ai messaggi di NoteOn (ma non sempre come vedremo più avanti), e quello di pressione che può essere utilizzato come valore di controllo dei più svariati parametri MIDI (volume, pitch, effetti…).
Il sensore di pressione presenta soltanto due pin: il primo è collegato a VCC mentre il secondo, dopo essere passato per una resistenza da 10KΩ per ridurre la corrente che circola, si collega alla massa e al pin A1 del microcontrollore, dal quale si leggeranno i valori di tensione generati in seguito a pressione.
Il sensore di posizione invece presenta tre pin: VCC, GRD e collector che, dopo aver regolato la corrente con una resistenza da 1MΩ si connette al pin A0 del microcontrollore.

Una serie di led (in alto a sinistra nello schema) offrono una visuale sullo stato corrente del sistema. In particolare, controllano lo stato di:

- On/Off state;
- Playing/Setting Mode;
- Mode1/Mode2;
- Wired/Wireless mode.

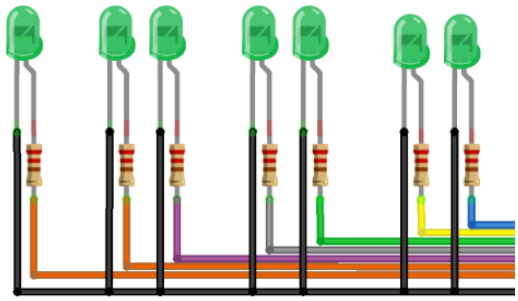

**Figura 4.7 - Led di stato**

Una seconda serie di 24 led di differenti colori controllano lo stato del volume e del pitch durante l'utilizzo del controller, dandone un feedback visivo.

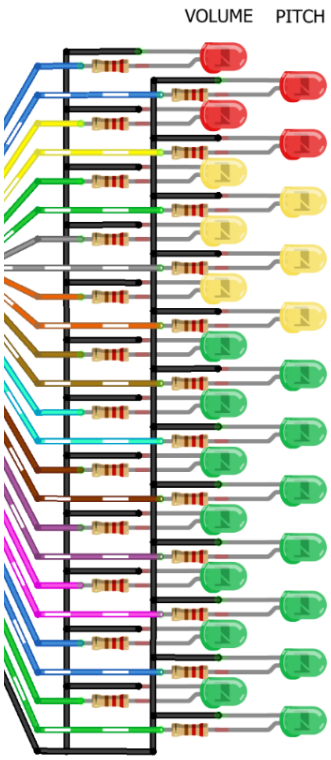


**Figura 4.8 - Led di controllo del volume e pitch**

Saranno ora trattati i componenti più importanti del controller, evidenziandone le specifiche e il loro utilizzo all'interno del progetto.

## 4.1 **Arduino Mega**

L'Arduino Mega 2560 è una scheda a microcontrollore basata sull'ATmega2560. Della famiglia, l'Arduino Mega è quello maggiormente fornito sia da un punto di vista di memoria SRAM utilizzabile per il codice eseguibile, sia del numero maggiore di connessioni digitali e analogiche che mette a disposizione rispetto alle altre versioni della stessa famiglia.

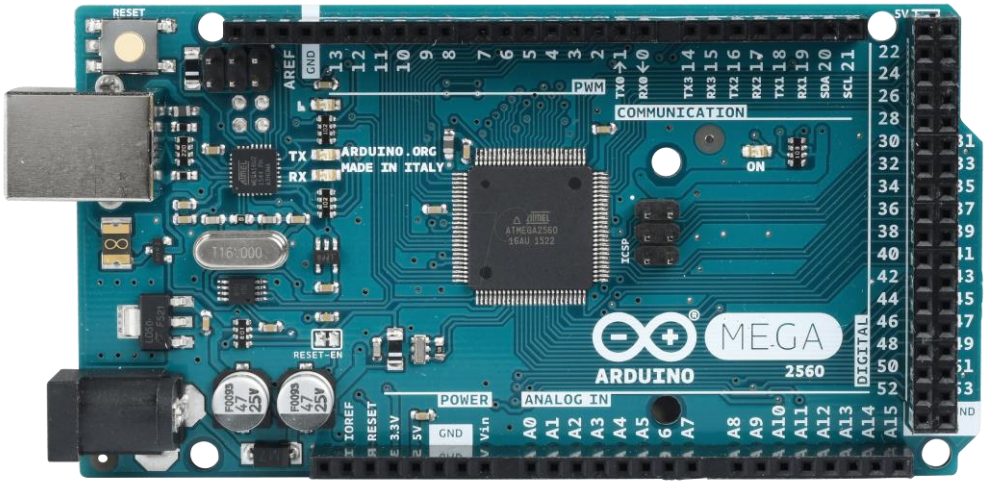


**Figura 4.9 – Arduino Mega R3 2560**

Come è possibile notare dall'immagine, ben 54 pin I/O digitali e 16 analogici sono messi a disposizione dell'utente per le più svariate connessioni di hardware esterno.

La memoria utilizzabile per i progetti passa dai soliti 2KB a 8KB, ampliando le possibilità progettuali dell'utente.

Altra peculiarità dell'Arduino Mega è la presenza di ben 4 porte seriali (3 in più rispetto all'Arduino Uno) grazie alle quali è possibili avere contemporaneamente più comunicazioni seriali da/verso dispositivi esterni basandosi esclusivamente su porte seriali native della scheda, superando la necessità di utilizzare librerie esterne che ne simulassero il funzionamento.

La scelta di utilizzare come microcontrollore l'Arduino nella sua versione Mega è stata dettata esattamente da problemi legati alle tre peculiarità appena descritte di questa scheda.

Il progetto, infatti, aveva la necessità di utilizzare più porte seriali per le comunicazioni con moduli esterni (DIN 5 e HC-06), numerosi pin I/O e, soprattutto, una quantità di memoria SRAM per lo sketch che superava abbondantemente quella messa a disposizione dall'Arduino UNO utilizzato in principio.

Come evidenziato nello schema, il microcontrollore è alimentato da una batteria da 9V collegata direttamente sul pin "vin" e "grd" della scheda che costituiscono, insieme, uno dei tre metodi utilizzabili per alimentare l'Arduino. È possibile utilizzare anche gli altri due metodi ma, volendo rendere il controller libero da alimentazione esterna, l'utilizzo dei pin risulta essere quello più versatile e pratico.

Da un punto di vista dell'impiego dei pin della scheda:

- Vin: alimentazione esterna della scheda,

- GRD1: alimentazione esterna della scheda,
- 5V: alimentazione dispositivi connessi alla breadbord,
- GRD2: alimentazione dispositivi connessi alla breadbord,
- A0 (analogico): sensore di posizione,
- A1 (analogico): sensore di pressione,
- SDA: serial data LCD,
- SCL: serial clock LCD,
- RX1: TX bluetooth HC-06,
- TX1: RX bluetooth HC-06,
- TX: DIN 5 poli,
- 8 (digitale): UP button,
- 9 (digitale): DOWN button,
- 10 (digitale): ESC button,
- 11 (digitale): ENTER button.

## 4.2 **Wi-Fi shield**

Per il progetto di rendere il controller un dispositivo wireless, si è reso necessario l'utilizzo di una shield che fornisse alla scheda Arduino tale funzionalità, dal momento che nessuna tra le schede della famiglia Arduino supporta la comunicazione wireless nativa.
La shield utilizzata per lo scopo è quella fornita da Arduino stesso nella sua versione 3.
Da un punto di vista funzionale, la shield è semplicissima da montare e utilizzare poiché, grazie ad una serie di connettori che ricalcano alla perfezione quelli della scheda Arduino sulla quale si connette, non ha bisogno di collegamenti tramite fili o jack particolari come avviene invece per altre shield che forniscono la stessa funzionalità.


**Figura 4.10 – Arduino Wi-Fi Shield**

Nel caso specifico del progetto, la shield viene inserita nella parte iniziale della scheda Arduino Mega (essendo quest'ultima più lunga della UNO) e comunica con la scheda tramite i sei pin (Reset, SCK, MISO, GRD, MOSI, +5V) posti nella parte centrale della shield e che costituiscono insieme uno dei metodi di programmazione della scheda Arduino, ovvero ICSP che sta per In Circuit Serial Programming.

In questa configurazione (shield montata sulla scheda) il pin digitale n.7 viene utilizzato come handshake tra la scheda stessa e la shield e dunque non può essere utilizzato.

Fatta eccezione per quest'ultimo, tutti gli altri pin presenti sulla shield e che, come detto, riproducono esattamente quelli della scheda sottostante nella quale si inseriscono, possono essere utilizzati normalmente.

La shield permette al controller di collegarsi alla rete LAN desiderata sfruttando la libreria nativa Wifi.h che fornisce un supporto semplice e molto intuitivo alla programmazione della sezione di codice che si occupa della connessione e gestione della comunicazione wireless.

Nonostante le ottime premesse, un problema di compatibilità tra la libreria esistente e l'ultima versione della shield rilasciata rende problematica la comunicazione wireless nella sua parte conclusiva ovvero di dialogo con altri dispositivi all'interno di una sessione MIDI virtuale. Sebbene, dunque, l'intero codice che gestisce tale comunicazione sia stato scritto e verificato in termini di funzionalità sulla vecchia versione dalla shield non più in commercio, il controller rimane, per ora, sprovvisto della possibilità di comunicazione wireless, in attesa di una patch software della libreria o del firmware della shiled.

## 4.3 **Position Sensor**

Uno degli elementi più importanti del controller è il primo dei due sensori che mettono a punto l'interfaccia musicale del controller.
Per il progetto è stato utilizzato un sensore di posizione della Spectra Symbol appartenente alla famiglia dei Linear SoftPot, ovvero sensori lineari che vanno da pochi centimetri fino a 2 metri. Per il controller è stato scelto quello di lunghezza consona tra i disponibili ovvero 50cm.

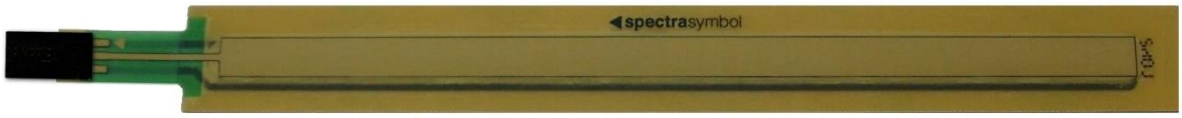


**Figura 4.11 – Spectra Symbol Position Sensor**

A livello elettrico, il sensore presenta tre pin, come descritto in precedenza, e una resistenza interna che, data la sua lunghezza, è attestabile sui 20KΩ.
In figura è mostrato uno schema elettrico dello stesso.

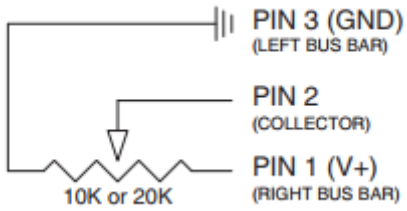


**Figura 4.12 – Schema elettrico pin Position Sensor**

Come si evince dallo schema, il suo funzionamento è del tutto simile a quello di un potenziometro alla cui famiglia il Linear Position Sensor appartiene, tralasciando la forma e il modo di interazione.
Il suo funzionamento infatti è del tutto riconducibile a quello di un normale potenziometro, in cui l'output è funzione di un qualche parametro "regolabile".
In questo caso specifico, l'output elettrico del sensore è funzione della distanza dall'origine in cui l'utente tocca il sensore stesso.
Per capire come questo possa avvenire occorre spiegare come il sensore è costruito.
Il SoftPot è un elemento resistivo e nasce dalla sovrapposizione di 4 layers. A partire dall'alto troviamo:

- Collector: layer sul quale viene esercitata la pressione e costituisce il primo dei due circuiti elettrici che devono venire a contatto per generare l'output.
- Spacer: layer molto sottile e cavo nel mezzo, utilizzato per mantenere separati, in assenza di pressione sul primo circuito, i due circuiti stessi.
- Resistor: secondo dei due circuiti elettrici,
- Adhesive: ultimo layer utilizzato per fissare il sensore su superfici fisse.


**Figura 4.13 – Struttura Position Sensor**

Vista la struttura, capirne il funzionamento è molto semplice. Applicando una forza sul circuito superiore con una mano o qualsiasi altro oggetto anche non conduttivo, si genera una depressione dello stesso che, attraverso la superficie cava del distanziatore (2° layer) viene in contatto con il circuito inferiore generando uno specifico output potenziometrico in funzione del punto in cui è avvenuto il contatto.

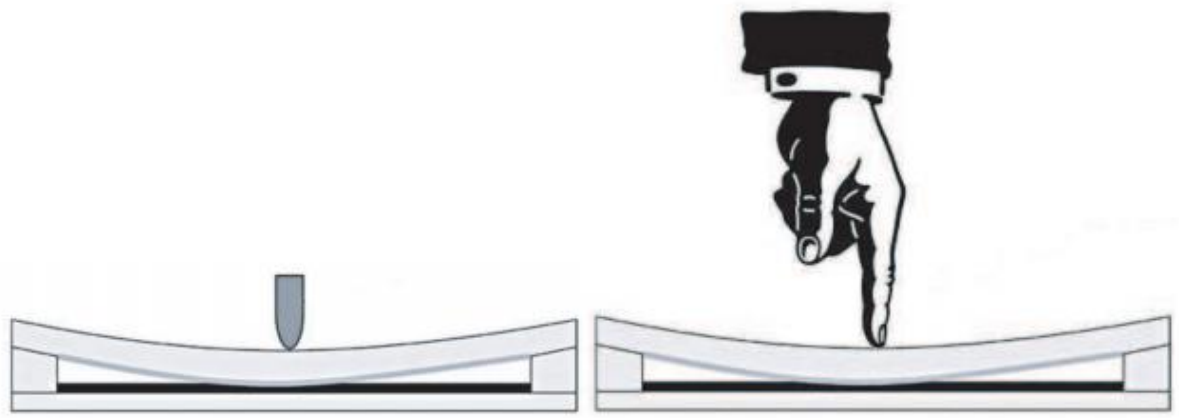

**Figura 4.14 – Funzionamento Position Sensor**

Dal momento che il collector e il resistor sono separati da soli 0.15 mm, anche una lieve pressione risulta sufficiente a chiudere il circuito.

In figura viene mostrato un esempio di interazione con un Softpot di 300mm, mettendo in relazione distanza del tocco dall'origine e la resistenza elettrica relativa, causa dello specifico output.

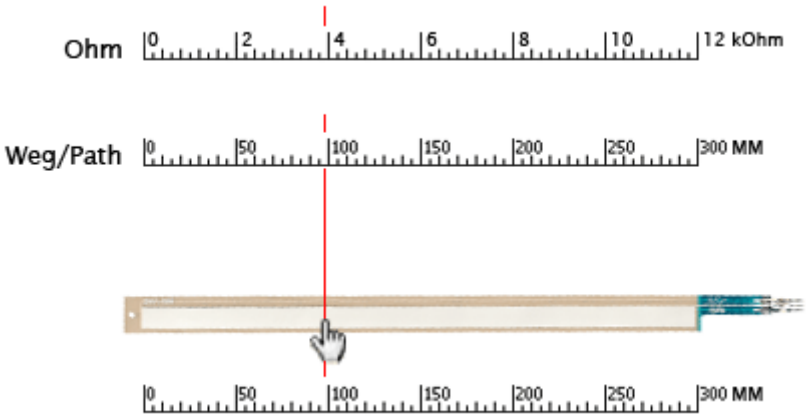


**Figura 4.15 – Relazione posizione-resistenza Position Sensor**

## 4.4 **Pressure Sensor**

Il secondo sensore che completa l'interfaccia è rappresentato da un sensore FSR (Force Sensing Resistor) che, per l'appunto, è sensibile alla pressione che viene esercitata su di esso. Per il progetto è stato scelto modello 408 appartenente alla famiglia dei sensori FSR Strip dell'Interlink Electronics di lunghezza pari a 60cm.

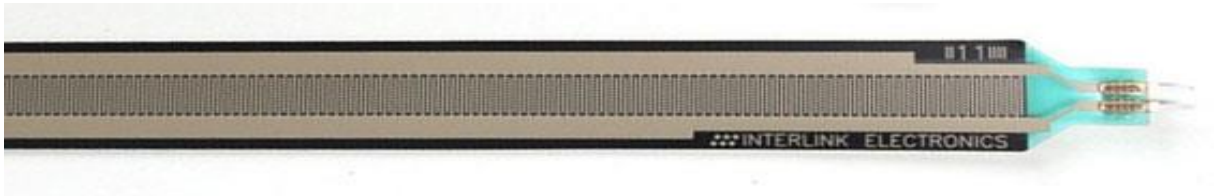


**Figura 4.16 – FSR Sensor**

L'idea di utilizzare un simile sensore nesce dall'esigenza di rilevare in qualsiasi momento dell'iterazione dell'utente con il sensore di posizione anche il valore di pressione associato all'interazione stessa per utilizzarlo come valore di control change. Per far questo era necessario

avere, per tutta la lunghezza del sensore di posizione, tanti sensori di pressione, il che avrebbe reso il sistema contorto e inefficiente.

Utilizzando questa soluzione invece, la superficie sensibile alla pressione è distribuita lungo l'intero, unico sensore letto attraverso un unico pin analogico.

L'interazione dell'utente in un punto del sensore produrrà lo stesso identico risultato della stessa pressione applicata in un qualsiasi altro punto.

Il sensore di pressione, definito come Polymer thick film (PTF), è un dispositivo che, all'aumentare della pressione sulla sua superficie, diminuisce il valore di resistenza fornita in output. Il range di pressione va dai 0.2 N fino ai 20 N.

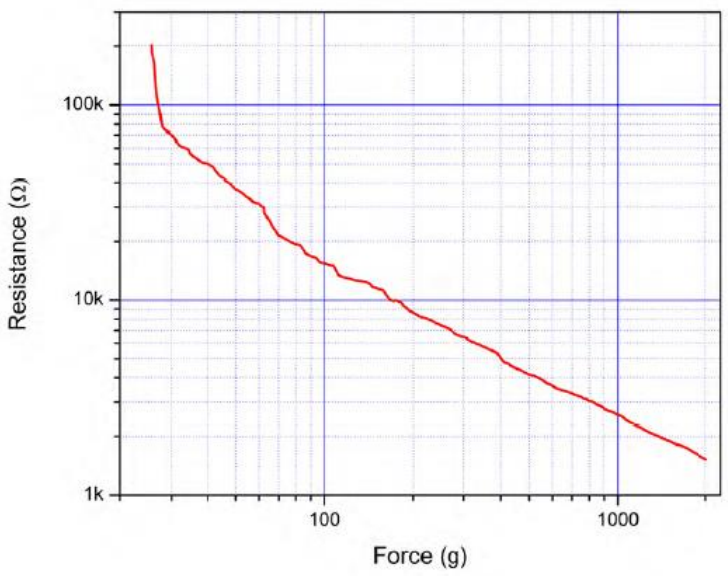


**Figura 4.17 – Relazione Pressione-Resistenza FSR Sensor**

Trattandosi di una vera e propria resistenza variabile, il sensore avrà due soli pin: il primo viene utilizzato per applicare una tensione di ingresso e il secondo, collegato a massa, per leggere il valore di output in uscita.

Il valore di output, però, va interpretato come valore di tensione e non di resistenza. Per una semplice conversione forza applicata – valore tensione di uscita, il sensore di pressione può essere inserito in un circuito composto da partitore di tensione e un buffer, la cui tensione d'uscita è descritta dalla seguente equazione:

$$V_{out} = \frac{R_M}{R_M + R_{FSR}} V^+$$

Dove:

$R_M$ è la resistenza di misura scelta per massimizzare il range di sensibilità della forza e per limitare la corrente;

$R_{FSR}$ è la resistenza variabile del sensore di pressione;

$V^+$ è la tensione fornita al circuito.

In figura è mostrato il circuito descritto.

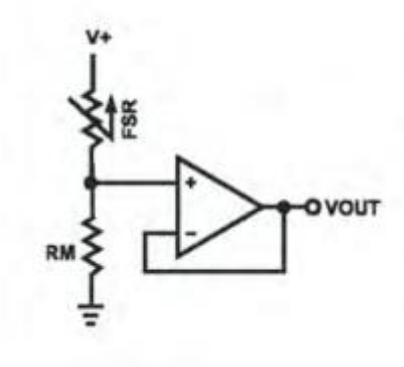


**Figura 4.18 – Circuito FSR-$V_{OUT}$**

In tal configurazione, la tensione in uscita aumenta all'aumentare della forza mentre se le due resistenze vengono invertite si avrà il risultato opposto.

Utilizzando differenti valori di $R_M$ applicati allo stesso partitore di tensione con un valore di tensione di ingresso pari a 5V, avremo una famiglia di curve forza-$V_{out}$ come mostrato in figura.

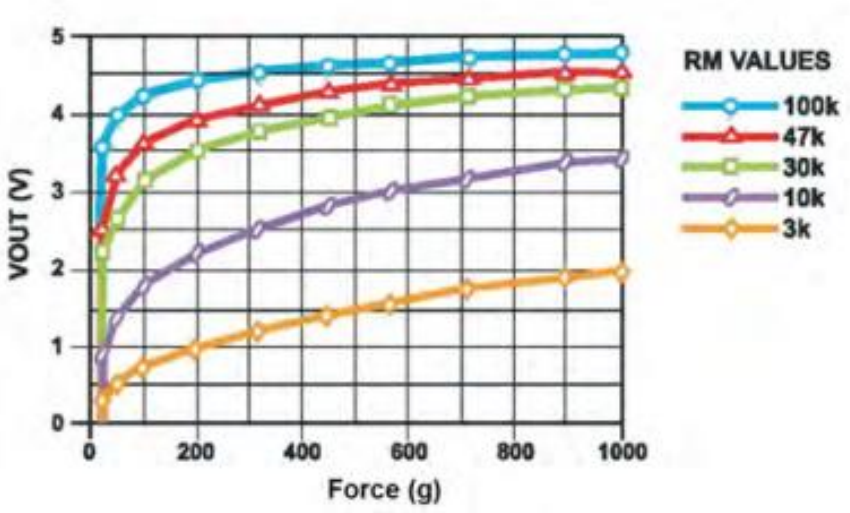


**Figura 4.19 – Relazione Pressione-$V_{OUT}$ FSR Sensor**

Da un punto di vista costruttivo l'FSR è costituito da due membrane separate da un sottile gap. Le due membrane sono tenute insieme e allo stesso tempo distanziate da un sottilissimo layer adesivo con una cavità centrale.
La membrana inferiore del sensore è rivestita da due piste separate e molto vicine tra loro ma elettricamente separate che formano una sorta di serpentina lungo tutta la superficie della membrana. L'altra membrana invece è ricoperta di inchiostro conduttivo.
In figura è mostrata la struttura del sensore.


**Figura 4.20 – Layers FSR Sensor**

**Figura 4.21 – Struttura FSR Sensor**

Quando viene esercitata una forza sulla parte superiore del sensore, si genera una depressione e la membrana superiore ricoperta di inchiostro conduttivo collassa su quella inferiore con un'estensione proporzionale alla pressione esercitata sulla membrana stessa. Collassando, l'inchiostro conduttivo va a connettere le piste presenti sulla membrana sottostante chiudendo, di fatti, il circuito.

# Capitolo 5
# IL SOFTWARE

## 5.1 La struttura

Il codice scritto per la gestione del controller utilizza i linguaggi C/C++ dal momento che sono quelli comunemente utilizzati per la programmazione del microcontrollore utilizzato nel progetto.
Volendo fornire una visione generale del codice che prescinde dall'ordine di scrittura delle funzioni ma tiene conto soltanto delle funzionalità fornite, è possibile suddividere lo sketch in 5 parti fondamentali:

- Librerie e variabili,
- Setup function,
- Loop function,
- LCD controller functions,
- Support & Setting functions.

Ognuna delle 5 parti menzionate offre funzionalità specifiche al sistema.

La prima parte del codice, come consueto, si occupa della definizione e inizializzazione delle numerose variabili necessarie alla gestione dei sensori, creazione dei messaggi MIDI e interazione con il sistema di riproduzione. Le prime righe di questa parte si riferiscono all'inclusione delle librerie esterne utilizzate nel codice.

Alla parte di definizione delle variabili segue la fase di Setup del codice. Quest'ultima fa parte delle funzioni fondamentali degli sketch operanti su microcontrollori che fanno parte della famiglia Arduino.
Questa funzione, che viene eseguita una volta soltanto all'avvio del microcontrollore, ha il compito di istanziare e settare tutte le strutture, variabili, gli indirizzi hardware del microcontrollore, che saranno poi utilizzati in seguito, nonché lanciare alcune funzioni di setting necessarie, per esempio, alla calibrazione dei sensori esterni. In questo caso, inoltre, in

questa fase viene effettuato anche il recupero dei dati dalla memoria EEPROM del microcontrollore al fine di utilizzarli per settare le variabili di controllo del sistema.

Dopo la fase di Setup, segue quella di Loop. Come la precedente, anche quest'ultima fa parte delle funzioni fondamentali e necessarie per un corretto funzionamento dello sketch sul microcontrollore. In particolare questa funzione è quella che, come suggerisce il nome, viene lanciata iterativamente dopo la fase di Setup e per tutto il tempo in cui il microcontrollore è in funzione.
In questo progetto, la funzione di Loop è quella che si occupa delle operazioni fondamentali di lettura dei dati dai sensori esterni, interpretazione e gestione degli stessi, comunicazione dei messaggi MIDI all'esterno.
Oltre a ciò, nel Loop sono anche contenute alcune funzioni che servono per settare a run time alcune variabili fondamentali di funzionamento del sistema.

La parte di LCD controller function, come suggerisce il nome, è un insieme di funzioni che si occupano della gestione visiva e funzionale dell'LCD collegato al sistema. Tramite queste funzioni, infatti, è possibile cambiare il testo visualizzato sull'LDC nonché definire i criteri di gestione del menù interno che viene utilizzato per mostrare all'esterno la possibilità di interazione e setting del sistema.

La possibilità di gestire e settare al meglio le variabili di funzionamento del sistema è affidata a quella che è stata identificata come Setting function. In realtà si tratta di una serie di funzioni, localizzate alla fine del codice e che vengono richiamate ogni qualvolta che l'utente, tramite menù, accede alla sezione di setting delle variabili.
Diverse opzioni sono possibili, dal settaggio delle variabili che più strettamente riguardano la produzione del MIDI, alla comunicazione dello stesso con il sistema esterno, al setting dell'hardware.

Nei paragrafi seguenti saranno trattati nello specifico tutte le fasi, analizzando anche il codice che le implementa.

## 5.2 **Librerie e Variabili**

### 5.2.1 Le librerie esterne

L'effettiva realizzazione del codice che implementa il progetto del controller si appoggia sulle funzionalità di alcune librerie esterne open source a cui si attinge lasciandole nella loro forma originaria oppure rivisitata nei casi in cui si aveva la necessità di qualcosa di molto simile ma non strettamente uguale a ciò che l'autore della libreria aveva concepito.
Le librerie esterne che sono state utilizzate nel codice sono 10:

```
1.  #include <WiFiUdp.h>
2.
3.  #include <WiFiServer.h>
4.
5.  #include <WiFiClient.h>
6.
7.  #include <WiFi.h>
8.
9.  #include <SoftwareSerial.h>
10.
11. #include <LiquidCrystal_I2C.h>
12.
13. #include <AppleMidi.h>
14.
15. #include <MIDI.h>
16.
17. #include <EEPROM.h>
18.
19. #include <MenuBackend.h>
```

Le librerie verranno trattate per sommi capi, riportandone solo aspetti e funzionalità considerate di rilievo.

#### 5.2.1.1 WiFi Libraries

Le prime 4 librerie incluse nel codice sopra riportato fanno parte della classe che si occupa della comunicazione wireless tra il microcontrollore e il dispositivo deputato alla sintesi e riproduzione del suono.
Come detto in precedenza, infatti, il controller ha la possibilità di lavorare sia in modalità wired tramite cavo e interfaccia MIDI OUT oppure wireless.
Si noti che queste librerie offrono solo supporto per istanziare la comunicazione wireless ma non offrono alcun supporto ai messaggi MIDI, i quali sono gestiti da una libreria diversa.

Andando un po' più nello specifico, le prime 3 (WiFiUdp.h, WiFiServer.h, WiFiClient.h) offrono funzionalità di supporto alla stessa libreria WiFi.h e contengono il codice necessario per l'uso rispettivamente di:

- Udp, il quale viene utilizzato come protocollo di trasporto nel momento in cui si iniziano le operazioni di instaurazione della comunicazione tra il microcontrollore e altri dispositivi di rete.
- Comunicazione Server-Client e viceversa a seconda se il microcontrollore sia predisposto a fornire contenuti (server) o a richiederli (client).

Diverse sono le funzionalità messe a disposizione dalla libreria WiFi.h. Degna di nota è sicuramente:

```
int begin(char* ssid, const char *passphrase);
```

La funzione begin prende come parametri SSID e PASSPHRASE della rete alla quale si ci vuole connettere e lancia un tentativo di connessione alla rete designata, ritornando lo stato della connessione.

```
IPAddress localIP();
```

restituisce l'indirizzo IP del microcontrollore se quest'ultimo si sia correttamente registrato in rete tramite la funzione begin.

#### 5.2.1.2 SoftwareSerial.h

Questa libreria nasce con l'intento di sopperire alla scarsezza di porte per la comunicazione seriale implementate sul microcontrollore Arduino.

La comunicazione seriale, infatti, nativamente è implementata solo sui pin 0 e 1 di Arduino e, tra l'altro, è la stessa che viene utilizzata per la comunicazione tra l'Arduino stesso e il PC con il quale viene programmato. Nel caso in cui si abbia bisogno di più porte seriali per la comunicazione con più dispositivi esterni si può adoperare un microcontrollore provvisto di più UART, oppure rifarsi a questa libreria.

SoftwareSerial è stata sviluppata per consentire la comunicazione seriale su altri pin digitali dell'Arduino, utilizzando software per replicarne le funzionalità. È possibile avere più porte seriali software con velocità fino a 115200 bps.

Nel nostro caso specifico, oltre alla porta seriale nativa utilizzata per la comunicazione wired con il PC, occorreva una seconda porta per il collegamento e la comunicazione del dispositivo bluetooth HC-06 che viene utilizzato dal metodo "SetCredential" e serve per collegare al

controller un terminale bluetooth attraverso il quale settare nuovi SSID e PASSWORD nel caso di cambio rete LAN.

Tra le funzioni principali si trovano:

```
SoftwareSerial(uint8_t receivePin, uint8_t transmitPin, bool inverse_logic = false);
```

prende come parametri i pin digitali designati come ricevente e trasmettitore della nuova porta seriale e istanzia parametri e strutture necessari (costruttore);

```
void begin(long speed);
```

prende come parametro la velocità in bps con cui avverrà la comunicazione sulla porta e inizializza parametri e strutture per il suo funzionamento.

```
int SoftwareSerial::read();
```

legge da seriale e ritorna il numero dei caratteri letti;

```
size_t SoftwareSerial::write(uint8_t b)
```

scrive su seriale e ritorna la dimensione del buffer scritto.

#### 5.2.1.3 LiquidCrystal_I2C.h

Questa libreria, come suggerisce il nome, offre supporto per la gestione di un LCD collegato al microcontrollore come dispositivo I2C.

Tale protocollo di comunicazione si contrappone per concetti, collegamenti e funzionalità a quello seriale ed offre notevoli vantaggi in tutti i casi in cui si abbia la necessità di comunicare con più dispositivi connessi sulla stessa linea di comunicazione introducendo delle regole di sincronizzazione tra gli stessi.

Quello che, infatti, implementa questo protocollo è un tipo di comunicazione più comunemente conosciuto come Master-Slave, in cui il microcontrollore (in questo caso Master) dialoga con uno dei dispositivi connessi alla linea I2C utilizzando uno specifico indirizzo (LCD Slave), dandogli determinati comandi e attendendo (o meno) riscontro.

Il protocollo I2C semplifica enormemente le cose anche nel caso in cui si necessiti di sincronizzazione tra le operazioni compiute dal Master e quelle ordinate ed eseguite dallo Slave (anche se non è questo il caso).

Le principali funzionalità messe a disposizione dalla libreria LiquidCrystal_I2C.h sono:

```
LiquidCrystal_I2C(uint8_t lcd_Addr,uint8_t lcd_cols,uint8_t lcd_rows);
```

costituisce il costruttore dell'istanza della libreria e prende come parametri di ingresso l'indirizzo I2C del dispositivo (LCD) che verrà utilizzato per ridirigere la conversazione solo

ed esclusivamente all'LCD, il numero di colonne e quello di righe di cui è composta la matrice dell'LCD.

```
void LiquidCrystal_I2C::begin(uint8_t cols, uint8_t lines, uint8_t dotsize);
```

inizializza l'istanza appena creata prendendo come parametri colonne e righe della matrice LCD nonché la grandezza del punto.

```
void LiquidCrystal_I2C::clear()
```

ripulisce l'LCD da qualsiasi scrittura precedente.

```
void LiquidCrystal_I2C::setCursor(uint8_t col, uint8_t row)
```

porta il cursore nella posizione desiderata alla colonna e riga specificati dai parametri.

```
void LiquidCrystal_I2C::backlight(void)
```

abilita la retroilluminazione dello schermo.

```
void LiquidCrystal_I2C::print(const char c[])
```

scrive la stringa presa come parametro a partire dalla posizione del cursore.

#### 5.2.1.4 AppleMidi.h

Come visto nel capitolo dedicato alla descrizione dei protocolli, RTP-MIDI (noto anche come AppleMIDI) è un protocollo per il trasporto dei messaggi MIDI in pacchetti RTP (Real-Time Protocol) su reti Ethernet e WiFi.

L'omonima libreria utilizzata in questo progetto offre supporto esattamente per questo scopo, mettendo a disposizione funzionalità che permettono sia di inviare i più classici messaggi MIDI su rete LAN come i Channel Message, ma anche offre la possibilità di controllare la sessione MIDI con messaggi di tipo Mode Message e System Message.

Le più importati funzionalità offerte dalla libreria, nonché quelle utilizzate nel progetto sono:

```
APPLEMIDI_CREATE_INSTANCE (WiFiUDP, AppleMIDI);
```

crea un'istanza AppleMidi passando come parametri un'istanza della classe che gestisce la comunicazione (WiFiUDP) e il nome dell'istanza di classe.

```
inline bool AppleMidi_Class<UdpClass>::begin(const char* sessionName, uint16_t
port)
```

inizializza la sessione MIDI e prende come parametro il nome della sessione (arbitrario) e la porta sulla quale la sessione deve essere avviata. Solitamente la porta che si sceglie è la 5004. Dal momento che Apple MIDI Protocol utilizza due porte consecutive per l'invio e la ricezione di dati e comandi, se definiamo come valore di porta 5004 sarà utilizzata anche la 5005.

```
inline void AppleMidi_Class<UdpClass>::run();
```

avvia la sessione AppleMidi.

```
inline void AppleMidi_Class<UdpClass>::invite(IPAddress ip, uint16_t port);
```

prende come parametri un IP e una porta e invita il dispositivo con quell'indirizzo di rete a partecipare alla sessione MIDi sulla porta specificata.

```
inline void noteOn(DataByte inNoteNumber, DataByte inVelocity, Channel
inChannel);
```

invia un messaggio di NoteOn con valori di Nota, Velocity, Numero di canale specificati come parametri.

```
inline void noteOff(DataByte inNoteNumber, DataByte inVelocity, Channel
inChannel);
```

invia un messaggio di NoteOff con valori di Nota, Velocity, Numero di canale specificati come parametri.

```
inline void controlChange(DataByte inControlNumber, DataByte inControlValue,
Channel inChannel);
```

invia un messaggio di Control Change con valori di Control Number , Valore , Numero di canale specificati come parametri.

```
inline void start();
```

invia messaggio di inizio.

```
inline void stop();
```

invia messaggio di fine.

```
inline void OnConnected(void(*fptr)(uint32_t, char*));
```

specifica la routine da eseguire quando un dispositivo si connette alla sessione MIDI.

```
inline void OnDisconnected(void(*fptr)(uint32_t));
```

specifica la routine da eseguire alla disconnessione del dispositivo connesso.

#### 5.2.1.5 MIDI.h

La libreria MIDI.h ha le stesse funzionalità della libreria AppleMidi.h con la differenza che il canale di comunicazione dei messaggi midi passa dal wireless al wired.

Tale libreria infatti implementa la gestione e l'invio dei messaggi MIDI tramite interfaccia seriale.

Le più importati funzionalità offerte dalla libreria, nonché quelle utilizzate nel progetto sono:

```
MIDI_CREATE_INSTANCE(HardwareSerial, Serial, MIDI);
```

crea un'istanza Midi passando come parametri il tipo di istanza (è possibile anche una soluzione con seriale software), la porta seriale che gestisce la comunicazione (Serial) e il nome dell'istanza di classe (MIDI).

```
void begin(Channel inChannel = 1);
```

inizializza la sessione MIDI e prende come parametro il numero di canale su cui avverrà la comunicazione.

```
inline void sendNoteOn(DataByte inNoteNumber, DataByte inVelocity, Channel
inChannel);
```

invia un messaggio di NoteOn con valori di Nota, Velocity, Numero di canale specificati come parametri.

```
inline void sendNoteOff(DataByte inNoteNumber, DataByte inVelocity, Channel
inChannel);
```

invia un messaggio di NoteOff con valori di Nota, Velocity, Numero di canale specificati come parametri.

```
inline void sendControlChange(DataByte inControlNumber, DataByte
inControlValue, Channel inChannel);
```

invia un messaggio di Control Change con valori di Control Number , Valore , Numero di canale specificati come parametri.

#### 5.2.1.6 EEPROM.h

EEPROM.h, come suggerisce il nome, offre supporto per l'utilizzo della memoria EEPROM che i microcontrollori della famiglia Arduino montano per consentire all'utente di lavorare con dati che necessitano di essere memorizzati e mantenuti in memoria anche al momento dello spegnimento del microcontrollore.

Nel progetto, tale possibilità viene sfruttata per memorizzare in specifiche celle EEPROM (cap. 4.3, 12-19) valori validi delle variabili MIDI che il controller usa nel suo funzionamento. Nel momento dell'accensione del controller, infatti, l'utente può scegliere se effettuare o meno la calibrazione dei sensori. Nel primo caso i parametri di funzionamento del controller sono settati durante la procedura di calibrazione stessa. Nel secondo caso invece occorre assegnare ai parametri un set di valori che abbiano senso per l'utente e che non siano casuali. In quest'ultima circostanza i dati memorizzati nelle celle EEPROM durante sessioni di calibrazione e funzionamento del controller precedenti vengono prelevati e assegnati alle variabili opportune (cap. 4.3, 36-44), riportando il sistema nell'identico stato precedente allo spegnimento del controller.

I microcontrollori supportati sulle varie schede Arduino e Genuino hanno diverse quantità di EEPROM: 1024 byte sul Atmega328, 512 byte sul ATmega168 e ATmega8, 4 KB (4096 byte) sulla ATmega1280 e ATmega2560.
Avendo utilizzato per questo progetto un Arduino Mega con microcontrollore ATmrga2560, ben 4096 celle di memoria sono a disposizione per la memorizzazione di dati non volatili, anche se, in realtà, molte meno ne sono state utilizzate.
Le celle di memoria sono indirizzate come quelle di una comune matrice (buffer bidimensionale nelle versioni più articolate) o, più semplicemente come un buffer lineare di 4096 posizioni (0-4095). Particolare attenzione va posta nell'utilizzo delle celle di memoria in scrittura. Dal momento che una cella equivale ad un byte, la scrittura di dati superiori alla dimensione della cella potrebbe comportare due differenti errori a seconda della libreria utilizzata e del microcontrollore che monta la EEPROM: troncamento del dato da scrivere al byte oppure sovrascrittura della cella/e seguenti con i dati che eccedono il byte indirizzato.
Le funzionalità utilizzate per questo progetto sono:

```
uint8_t read( int idx );
```

legge la cella di memoria indirizzata dal parametro richiesto e ne restituisce il valore.

```
void write( int idx, uint8_t val );
```

scrive nella cella di memoria indirizzata dal primo parametro richiesto il valore del secondo parametro.

```
void update( int idx, uint8_t val );
```

sovrascrive il valore della cella indirizzata dal primo parametro con il valore passato con il secondo parametro.

```
EEPtr begin();
```

ritorna l'indirizzo della prima cella della memoria EEPROM (nella maggioranza dei casi 0x00).

```
EEPtr end();
```

ritorna l'indirizzo dell'ultima cella EEPROM disponibile alla scrittura.

```
uint16_t length();
```

ritorna la lunghezza dell'EEPROM utilizzata.

#### 5.2.1.7 MenuBackend.h

L'utilizzo di MenuBackend.h nasce dall'esigenza di utilizzare una struttura più o meno complessa che desse la possibilità di organizzare la logica del sistema in una struttura ordinata di menù di modo da poterla esporre all'utente attraverso l'utilizzo dell'LCD e con la quale

l'utente potesse interagire attraverso l'utilizzo di appositi pulsanti.

Tale struttura poteva essere creata anche senza l'utilizzo di alcuna libreria esterna ma, implementando tutto nel codice principale e soprattutto nella funzione Loop (sarebbe stato necessario dal momento che l'interazione deve avere le caratteristiche proprie del real time), si andava incontro a problemi di "intasamento" della funzione stessa la quale avrebbe dovuto occuparsi dell'interazione esterna dovuta ai pulsanti e nello stesso tempo quella propria dei sensori, il che sarebbe possibile ma sconsigliato a causa degli eccessivi ritardi che si genererebbero.

Per questi motivi si sceglie di appoggiarsi alle funzionalità di questa libreria che ottimizza l'utilizzo di queste strutture e rende più veloce e ordinata qualsiasi operazione di interazione, riducendo notevolmente il rischio di ritardi.

Le funzionalità della libreria che sono state utilizzate sono:

```
MenuItem(const char* itemName, char shortKey='\0' );
```

inizializza con il nome passato come parametro un nuovo item del menù.

```
inline const char* getName();
```

ritorna il nome dell'item del menù (utile in caso di confronti).

```
MenuItem &add(MenuItem &mi);
```

aggiunge l'item di menù passato come parametro alla struttura del menù. Questa è la scelta di default nel caso di menù "verticali".

```
MenuItem &addBefore(MenuItem &mi);
MenuItem &addRight(MenuItem &mi);
MenuItem &addAfter(MenuItem &mi);
MenuItem &addLeft(MenuItem &mi);
```

aggiungono l'item di menù passato come parametro rispettivamente prima, a destra, dopo, a sinistra rispetto alla radice del menù o, in generale, all'ultimo item della struttura.

```
bool menuTestStrings(const char *a, const char *b);
```

effettua un controllo di uguaglianza su tutti i caratteri passati come parametri. Ritorna `true` solo se le due stringhe sono identiche.

```
bool operator==(MenuItem &lhs, char* test);
```

ridefinisce l'operatore di uguaglianza utilizzando la funzione precedente per effettuare facilmente il confronto tra l'item di menù e la stringa passati come parametri.

```
    struct MenuChangeEvent {
    const MenuItem &from;
    const MenuItem &to;
};

struct MenuUseEvent {
    const MenuItem &item;
};
```

strutture di supporto per gestire rispettivamente il cambiamento di ciò che viene mostrato dal display e le azioni associate al cambiamento.

```
MenuBackend(cb_use menuUse, cb_change menuChange = 0)
```

istanzia un nuovo oggetto menù prendendo come parametri le due strutture precedenti.

```
MenuItem &getRoot();
```

restituisce la root del menù.

```
void moveBack();
void moveUp();
void moveDown();
void moveLeft();
void moveRight();
```

effettua un movimento rispettivamente all'indietro, su, giù, sinistra, destra rispetto alla posizione corrente nel menù.

```
void toRoot();
```

riporta alla root del menù.

### 5.2.2 Le variabili

La parte iniziale dello sketch è caratterizzata dalla definizione di numerose variabili necessarie ad un corretto funzionamento del sistema a run time e per un corretto ripristino delle variabili salvate dopo lo spegnimento del sistema.
È possibile dividere concettualmente le variabili in base alla loro funzione all'interno del codice.

- Gestione della rete

```
1.  char ssid[50] = "ssid rete";
2.  char password[50] = "password";
3.  bool isConnected = false;
```

(1): SSID della rete LAN alla quale si vuole connettere il microcontrollore,

(2): PASSWORD della LAN,

(3): booleano per il controllo dello stato di connessione.

- Assegnamento pin fisici del microcontrollore

```
4.  const int sensorPosition = A0;
5.  const int sensorForce = A1;
6.  const int buttonPinUp = 8;
7.  const int buttonPinDown = 9;
8.  const int buttonPinEsc = 10;
9.  const int buttonPinEnter = 11;
10. const int rxPin = 6;
11. const int txPin = 5;
```

(4-5): assegnamento pin analogici A0 e A1 rispettivamente al sensore di posizione e pressione.

(6-9): assegnamento pin digitali 8,9,10,11 ai bottoni Su, Giù, Esc, Enter.

(10-11): assegnamento pin digitali 5 e 6 alle variabili per la gestione della seriale software.

- Assegnamento indirizzo EEPROM

```
12. const int addrSensorForceMin = 1;
13. const int addrSensorForceMax = 2;
14. const int addrSensorPosCC = 3;
15. const int addrSensorForceCC = 4;
16. const int addrMinNote = 5;
17. const int addrMaxNote = 6;
18. const int addrMidiChannel = 7;
19. const int addrPlayMode = 8;
```

(12-19): Assegnamento indirizzo celle di memoria EEPROM per la memorizzazione del valore di pressione minima, pressione massima, control change sensore posizione, control change sensore di pressione, nota minima, nota massima, MIDI channel e play mode.

- Memorizzazione dello stato di esecuzione

```
20. const int statusPlayMode1 = 1;
21. const int statusPlayMode2 = 2;
22. const int statusSetCCPosition = 5;
23. const int statusSetCCForce = 6;
24. const int statusSetMinNote = 7;
25. const int statusSetMaxNote = 8;
26. const int statusMidiTeach = 9;
27. const int statusSetMidiChannel = 10;
28. const int statusSendWired = 11;
29. const int statusSendWireless = 12;
```

(20-29): variabili utilizzate per separare e discernere tra i differenti stati di esecuzione del

sistema. In particolare:

20-21: modalità di produzione dei messaggi MIDI,

22-27: modalità di setting parametri del sistema (CCPosition, CCForce, MinNote, MaxNote, MidiTeach, MidiChannel),

28-29: modalità di comunicazione (wired o wireless).

- Gestione messaggi del menù per l'LCD

```
30. char strMenu[21] = "        MENU        ";
31. char strMenuMidi[21] = "   MIDI SETUP MENU  ";
32. char strMenuModes[21] = "  MODES SETUP MENU  ";
33. char strSetSendMode[21] = "  SENDING MODE MENU ";
34. char strSettingWifi[21] = "   WIFI SETUP MENU  ";
35. char strSetMinimumNote[21] = "Set minimum note    ";
36. char strSetMaximumNote[21] = "Set maximum note    ";
37. char strSetPositionCC[21] = "Set position CC";
38. char strSetForceCC[21] = "Set force CC        ";
39. char strSetMidiChannel[21] = "Set MIDI channel    ";
40. char str11[6] = "Wired";
41. char str12[6] = "Wi-Fi";
```

(30-41): stringhe per la gestione del menù e dei messaggi che sono visualizzati sull'LCD durante il funzionamento del sistema.

- Gestione dei bottoni

```
42. int lastButtonPushed = 0;
43. int lastButtonEnterState = LOW;
44. int lastButtonEscState = LOW;
45. int lastButtonUpState = LOW;
46. int lastButtonDownState = LOW;
```

(42): ultimo bottone premuto tra i 4 disponibili,

(43-46): stato di ognuno dei 4 bottoni, inizialmente impostato a "LOW" ovvero rilasciato.

- Gestione dell'iterazione

```
47. const int PressuresecurityRange = 5;
48. long lastEnterDebounceTime = 0;
49. long lastEscDebounceTime = 0;
50. long lastUpDebounceTime = 0;
51. long lastDownDebounceTime = 0;
52. long debounceDelay = 500;
```

(47): il range minimo (sensibilità) entro il quale l'iterazione con il sensore di pressione può avvenire. Se si omette un range di sicurezza, il sensore rileverebbe troppi tocchi mandando il sistema in crash.
(48-51): ultima pressione (temporale) dei bottoni Enter, Esc, Up, Down.
(52): debounce time. Necessario per evitare fenomeni di flickering dell'output.

- Gestione modalità di esecuzione

```
53. int currentSendingMode = 11;
54. int currentStatusMode;
55. int lastPlayingMode;
56. int id;
57. int currentUsingMode;
```

(53): modalità di comunicazione corrente,
(54): stato di esecuzione corrente tra quelli elencati sopra,
(55): modalità di produzione messaggi MIDI precedente,
(56): variabile di switch per gestione del valore di SSID o PASSWORD,
(57): modalità di utilizzo del controller (dita/mano).

- Variabili Midi

```
58. int minNote = 80;
59. int maxNote = 100;
60. int lastNote = 0;
61. int noteVelocity = 100;
62. int ccPositionSetting;
63. int ccForceSetting;
64. int minNoteSetting;
65. int maxNoteSetting;
66. int midiChannelSetting;
67. int midiChannel;
68. int ccPosition;
69. int ccForce;
70. int finger = 5;
71. int hand = 150;
```

(58-69): variabili utilizzate per salvare informazioni per la costruzione dei messaggi MIDI,
(70-71): variabili di setting del security range per la modalità dita-mano.

- Gestione sensori

```
72. int sensorForceValue = 0;
73. int sensorPositionValue = 0;
```

```
74. int sensorForceMinValue;
75. int sensorForceMaxValue;
76. int sensorPositionMinValue = 0;
77. int sensorPositionMaxValue = 1023;
78. int sensorPositionLastValue = 0;
79. int sensorForceRawValue = 0;
80. int sensorPositionRawValue = 0;
81. int sensorForcePreviousValue = -1;
82. int sensorPositionPreviousValue = -1;
```

(72-82): variabili utilizzate per la gestione dei valori in ingresso al microcontrollore rilevati dai sensori.

```
83. MIDI_CREATE_INSTANCE(HardwareSerial, Serial, MIDI);
84. APPLEMIDI_CREATE_INSTANCE(WiFiUDP, AppleMIDI);
85. SoftwareSerial bluetooth(rxPin, txPin);
86. LiquidCrystal_I2C lcd(0x27, 20, 4);
87. static void menuUsed(MenuUseEvent used);
88. static void menuChanged(MenuChangeEvent changed);
89. MenuBackend menu = MenuBackend(menuUsed, menuChanged);
```

(83-84): creazione di una nuova istanza della classe MIDI ed AppleMidi

(85-86): creazione di una nuova istanza della classe SoftwareSerial e LiquidiCrystal

(89): nuova istanza di menù

- Inizializzazione item di menù

```
90.         MenuItem menuModesSetup = MenuItem("menuModesSetup");
91.         MenuItem menuItemMode1 = MenuItem("menuItemMode1");
92.         MenuItem menuItemMode2 = MenuItem("menuItemMode2");
93.         MenuItem menuItemMidiSetup = MenuItem("menuItemMidiSetup");
94.         MenuItem menuItemCCPosition = MenuItem("menuItemCCPosition");
95.         MenuItem menuItemCCForce = MenuItem("menuItemCCForce");
96.         MenuItem menuItemMinNote = MenuItem("menuItemMinNote");
97.         MenuItem menuItemMaxNote = MenuItem("menuItemMaxNote");
98.         MenuItem menuItemMidiChannel = MenuItem("menuItemMidiChannel");
99.         MenuItem menuItemMidiTeach = MenuItem("menuItemMidiTeach");
100.        MenuItem menuSendMode = MenuItem("menuSendMode");
101.        MenuItem menuItemSend1 = MenuItem("menuItemSend1");
102.        MenuItem menuItemSend2 = MenuItem("menuItemSend2");
103.        MenuItem menuItemWifiSetup = MenuItem("menuItemWifiSetup");
104.        MenuItem menuItemWifi1 = MenuItem("menuItemWifi1");
105.        MenuItem menuItemWifi2 = MenuItem("menuItemWifi2");
106.        MenuItem menuItemWifi3 = MenuItem("menuItemWifi3");
107.        MenuItem menuCalibration = MenuItem("menuCalibration");
```

(91-104): vengono istanziati gli item del menù che andranno a comporre la sua struttura.

- Gestione led di controllo

```
1.  const int ledOnOff = 2;
2.  const int ledPlayng = 3;
3.  const int ledSetting = 4;
4.  const int ledMode1 = 5;
5.  const int ledMode2 = 6;
6.  const int ledWired = 7;
7.  const int ledWifi = 8;
8.
9.  int LedVolumeValue;
10. int OldLedVolumeValue;
11.
12. int LedPitchValue;
13. int OldLedPitchValue;
```

(1-7): pin digitali per i led di controllo dello stato del controller
(9-13): variabili di controllo per accensione e spegnimento dei led.

## 5.3 **Setup Function**

La funzione di Setup è una delle due funzioni principali di qualsiasi sketch Arduino. La sua funzione è quella di inizializzazione e avvio di variabili e processi che verranno poi gestiti dalla funzione di Loop o da altre. La funzione di Setup viene eseguita una volta sola all'avvio del sistema.

```
1.  Serial.begin(9600);
2.  bluetooth.begin(9600);
3.  lcd.init();
4.  lcd.backlight();
5.  MIDI.begin(midiChannel);
```

Tra le prime operazioni troviamo quelle di setting della comunicazione seriale (1) sui pin nativi di Arduino e di quella creata ad hoc (2) con la libreria SoftwareSerial. Viene anche settato l'LCD (3) con abilitazione della retroilluminazione (4). Viene infine avviata la sessione midi su canale 1 (5).

```
6.      pinMode(sensorForce, INPUT);
7.      pinMode(sensorPosition, INPUT);
8.      pinMode(buttonPinUp, INPUT);
9.      pinMode(buttonPinDown, INPUT);
10.     pinMode(buttonPinEnter, INPUT);
11.     pinMode(buttonPinEsc, INPUT);
```

I pin di Arduino collegati ai due sensori e ai 4 bottoni devono essere settati in modalità input (6-11).

```
1.  //led controllo
2.  pinMode(ledOnOff, OUTPUT);
```

```
3.  pinMode(ledPlayng, OUTPUT);
4.  pinMode(ledSetting, OUTPUT);
5.  pinMode(ledMode1, OUTPUT);
6.  pinMode(ledMode2, OUTPUT);
7.  pinMode(ledWired, OUTPUT);
8.  pinMode(ledWifi, OUTPUT);
9.
10. //led volume
11. pinMode(30, OUTPUT);
12. pinMode(32, OUTPUT);
13. pinMode(34, OUTPUT);
14. pinMode(36, OUTPUT);
15. pinMode(38, OUTPUT);
16. pinMode(40, OUTPUT);
17. pinMode(42, OUTPUT);
18. pinMode(44, OUTPUT);
19. pinMode(46, OUTPUT);
20. pinMode(48, OUTPUT);
21. pinMode(50, OUTPUT);
22. pinMode(52, OUTPUT);
23.
24. //led pitch
25. pinMode(31, OUTPUT);
26. pinMode(33, OUTPUT);
27. pinMode(35, OUTPUT);
28. pinMode(37, OUTPUT);
29. pinMode(39, OUTPUT);
30. pinMode(41, OUTPUT);
31. pinMode(43, OUTPUT);
32. pinMode(45, OUTPUT);
33. pinMode(47, OUTPUT);
34. pinMode(49, OUTPUT);
35. pinMode(51, OUTPUT);
36. pinMode(53, OUTPUT);
```

Vengono settati in modalità “OUTPUT” tutti i pin digitali di Arduino collegati ai led di controllo dello stato del controller (13-19), di controllo del volume (22-33) e del pitch (36-47).

```
37. for (int i = 2; i < 9; i++) {
38.     digitalWrite(i, LOW);
39. }
40.
41. for (int i = 30; i < 54; i++) {
42.     digitalWrite(i, LOW);
43. }
44.
45. for (int i = 31; i < 55; i++) {
46.     digitalWrite(i, LOW);
47. }
48.
49. digitalWrite(ledOnOff, HIGH);
```

Vengono inizializzati a “LOW” tutti i led: controllo di stato (37-39), volume (41-43), pitch (45-47) e viene acceso il led di “POWER ON” (49).

```
50.     while (millis()<5000 && pressed == LOW) {
51.         lcd.setCursor(0, 0);
52.         lcd.print("Push enter to");
53.         lcd.setCursor(0, 1);
54.         lcd.print("reset controller");
```

```
55.         pressed = digitalRead(buttonPinEnter);
56.
57.     }
```

All'avvio del sistema, occorre attendere 5 sec per permettere all'utente di resettare i valori dei sensori, riportandoli al loro stato precedente lo spegnimento del controller, o effettuare una calibrazione. Per 5 secondi viene mostrato a display la procedura per calibrare il controller (51-54) e viene letto il valore del bottone Enter (55). La sua pressione infatti è il trigger per la calibrazione.

```
57. if (sensorForceMinValue == 255 || pressed == HIGH) {
58.         calibration();
59.         ccPosition = 20;
60.         ccForce = 21;
61.         minNote = 80;
62.         maxNote = 100;
63.         midiChannel = 1;
64.         currentStatusMode = statusPlayMode1;
65.         currentSendingMode = statusSendWired;
66.         currentUsingMode = statusUseFinger;
67.         EEPROM.write(addrSensorPosCC, ccPosition);
68.         EEPROM.write(addrSensorForceCC, ccForce);
69.         EEPROM.write(addrMinNote, minNote);
70.         EEPROM.write(addrMaxNote, maxNote);
71.         EEPROM.write(addrMidiChannel, midiChannel);
72.         EEPROM.write(addrPlayMode, currentStatusMode);
73.         EEPROM.write(addrSecurityRange, finger);
74.
75.         digitalWrite(ledPlayng, HIGH);
76.         digitalWrite(ledSetting, LOW);
77.         digitalWrite(ledMode1, HIGH);
78.         digitalWrite(ledMode2, LOW);
79.         digitalWrite(ledWired, HIGH);
80.         digitalWrite(ledWifi, LOW);
81.         digitalWrite(ledOnOff, HIGH);
82.
83.     }
```

Se l'utente preme sul pulsante Enter durante i 5 secondi previsti per effettuare la calibrazione oppure non sono memorizzati dati nella EEPROM (57), viene lanciato un metodo per la calibrazione che verrà trattato in seguito (cap. 4.6.3). Vengono inizializzati i valori delle restanti variabili MIDI con quelle di default (59-63), viene impostato la prima modalità di funzionamento (64), la comunicazione wired (65) e modalità dita (66), infine vengono memorizzati nella EEPROM i dati appena utilizzati per un loro utilizzo anche dopo uno spegnimento del sistema (67-73). Al termine di tali procedure vengono opportunamente settati i led di controllo di stato del controller (75-81)

```
84.     else {
85.         sensorForceMinValue = sensorForceMinValue * 4;
86.         sensorForceMaxValue = (EEPROM.read(addrSensorForceMax)) * 4;
87.         ccPosition = EEPROM.read(addrSensorPosCC);
88.         ccForce = EEPROM.read(addrSensorForceCC);
```

```
89.         minNote = EEPROM.read(addrMinNote);
90.         maxNote = EEPROM.read(addrMaxNote);
91.         midiChannel = EEPROM.read(addrMidiChannel);
92.         currentStatusMode = EEPROM.read(addrPlayMode);
93.         securityRange = EEPROM.read(addrSecurityRange);
94.
95.         digitalWrite(ledPlayng, HIGH);
96.         digitalWrite(ledSetting, LOW);
97.
98.         if (currentStatusMode == 1) {
99.              digitalWrite(ledMode1, HIGH);
100.             digitalWrite(ledMode2, LOW);
101.        }
102.        else {
103.             digitalWrite(ledMode1, LOW);
104.             digitalWrite(ledMode2, HIGH);
105.        }
106.
107.        if (securityRange == hand) {
108.             currentUsingMode = statusUseHand;
109.        }
110.        else {
111.             currentUsingMode = statusUseFinger;
112.        }
113.
114.        digitalWrite(ledWired, HIGH);
115.        digitalWrite(ledWifi, LOW);
116.
117.       }
```

In caso contrario, vengono utilizzati i dati precedentemente memorizzati nella EEPROM leggendoli direttamente da essa (85-93) e vengono settati opportunamente i led di stato del controller a seconda dei valori appena letti (95-115).

```
118.           menu.getRoot().addRight(menuModesSetup).add(menuItemMidiSetup).add(menuSe
    ndMode).add(menuItemWifiSetup).add(menuCalibration);
119.           menuModesSetup.addRight(menuItemMode1).add(menuItemMode2);
120.           menuItemMidiSetup.addRight(menuItemCCPosition).add(menuItemCCForce).add(m
    enuItemMinNote).add(menuItemMaxNote).add(menuItemMidiChannel).add(menuItemMidiTeach)
    ;
121.           menuSendMode.addRight(menuItemSend1).add(menuItemSend2);
122.           menuItemWifiSetup.addRight(menuItemWifi1).add(menuItemWifi2).add(menuItem
    Wifi3);
123.           menu.toRoot();
```

Viene infine costruita la struttura di menù (118-123) utilizzando gli item istanziati in precedenza e disponendoli secondo la struttura ad albero che andranno a ricoprire nel menù.

## 5.4 **Loop Function**

La funzione di Loop è l'altra funzione necessaria in uno sketch Arduino affinché tutto funzioni correttamente. A differenza di quella di Setup, la funzione Loop viene eseguita iterativamente dal sistema fino al suo spegnimento.
In questo caso alla funzione di Loop sono assegnate le operazioni di lettura e processamento dei dati in arrivo dai sensori, la lettura dei bottoni e la parte di setting delle variabili di funzionamento del sistema.

```
1.      readButtons();
```

La prima operazione che viene eseguita dal Loop è quella della lettura dello stato dei pulsanti (1) in caso l'utente voglia accedere alle impostazioni di setting.

```
2.      if (currentSendingMode == 12) {
3.          AppleMIDI.run();
4.      }
```

Viene effettuato un controllo sulla modalità di comunicazione impostata. Se quest'ultima risulta essere wireless (2), viene avviato il protocollo AppleMidi (3) tramite la sua funzione run().

```
5.      switch (currentStatusMode)
```

Viene effettuato un controllo sulla modalità di funzionamento del sistema (5) che comprende due modalità di creazioni dei messaggi MIDI ed altre che servono per il setting dei parametri dei primi due.

### 5.4.1 PlayMode1

La modalità di funzionamento 1 del sistema è quella che più si avvicina all'idea classica di controller MIDI poiché associa al cambiamento della posizione sul sensore di posizione una variazione di note, e ad una variazione di pressione sul sensore di pressione un control change che può essere associato a qualunque variazione MIDI grazie al settaggio del software che si occupa della sintesi.

```
1.      case statusPlayMode1:
2.
3.          navigateMenus();
4.          sensorPositionRawValue = analogRead(sensorPosition);
5.
6.          if (sensorPositionRawValue>(sensorPositionMinValue + 10)) {
7.
8.              delay(5);
9.
10.             sensorPositionRawValue = analogRead(sensorPosition);
11.
12.             sensorPositionValue = map(sensorPositionRawValue, sensorPositionMinValue
    , sensorPositionMaxValue, minNote, maxNote);
13.
14.             sensorPositionValue = constrain(sensorPositionValue, minNote, maxNote);
15.
16.             sensorForceRawValue = analogRead(sensorForce);
17.
18.             sensorForceValue = map(sensorForceRawValue, sensorForceMinValue + 200, s
    ensorForceMaxValue, 0, 127);
19.
20.             sensorForceValue = constrain(sensorForceValue, 0, 127);
21.
22.             Serial.print(sensorPositionValue);
23.
24.             if (currentSendingMode == 12 && isConnected == true) {
25.                 AppleMIDI.noteOn(sensorPositionValue, noteVelocity, midiChannel);
26.                 AppleMIDI.controlChange(ccForce, sensorForceValue, midiChannel);
27.             }
28.             else {
29.                 MIDI.sendNoteOn(sensorPositionValue, noteVelocity, midiChannel);
30.                 MIDI.sendControlChange(ccForce, sensorForceValue, midiChannel);
31.             }
32.
33.             lastNote = sensorPositionValue;
34.
35.             lcd.setCursor(0, 1);
36.             lcd.print("   Sending Data...  ");
```

La modalità 1 di funzionamento è composta da una prima parte che si occupa della prima interazione tra l'utente e i sensori e di una seconda parte che esegue pressappoco le stesse operazioni fatte in precedenza fin tanto che c'è interazione, ma con la sostanziale differenza che in questa fase occorre capire quando è avvenuto un cambiamento nell'interazione che occorre gestire oppure no.

Si comincia con la chiamata alla funzione che si occupa della gestione (3) del menù che verrà trattata in seguito (cap. 4.6.2). Viene letto il valore del sensore di posizione (4) e se questo è superiore al minimo consentito più un piccolo valore di sicurezza (per evitare errori dovuti alla sensibilità del sensore) (6), si effettua la mappatura del valore letto tra il minimo e massimo valore consentito per il sensore e tra il minimo e massimo valore di nota (12). In questa fase, di vitale importanza per il corretto funzionamento del sistema è l'aggiunta di un piccolo ritardo (8) che da la possibilità al sensore di posizione di settare la propria resistenza e fornire un valore di output che abbia senso. In questo frangente infatti, ovvero nelle fasi di comunicazione tra il

software e l'hardware, si possono apprezzare tutti i problemi che caratterizzano un sistema in cui il software è sempre un passo avanti rispetto l'hardware, motivo per cui spesso occorre rallentarlo per permettere una corretta interazione fra i due.
Per sicurezza, ottenuto questo valore, si effettua un controllo affinché non sia minore o maggiore della minima e massima nota consentita dal sistema (14). Le stesse operazioni di lettura, mappatura e costrain vengono effettuate anche sul valore in arrivo dal sensore di pressione (16-20).
Ottenuti i due valori, a seconda del metodo di comunicazione che si sta utilizzando (wireless o wired) vengono inviati due messaggi (24-31) al sistema di sintesi:

- Messaggio di NoteOn basato sul valore di posizione e velocity fissa,
- Messaggio di ControlChange basato sul valore di pressione e ControlChange fissato.

Viene salvato il valore della nota appena inviata (33) per utilizzarla successivamente nel messaggio di NoteOff.
Un opportuno messaggio a display mostra il corretto invio dei dati al sintetizzatore (35-36).

In seguito alla gestione dell'iterazione con i sensori, una struttura do-while si occupa della gestione dell'iterazione finché quest'ultima non termina con il rilascio dei sensori. La presenza di una seconda iterazione all'interno della Loop function (per natura iterativa a sua volta) è giustificata dal fatto che, sebbene molto simili, molte operazioni fatte nella prima parte della funzione non devono essere ripetute nella seconda. Schematicamente è possibile pensare al Loop come ad una funzione ciclica che si occupa di gestire l'interazione, mentre al do-while come al codice che gestisce la lunghezza temporale dell'interazione. Le soluzioni per gestire questa differenza erano molteplici, compresa quella di effettuare un controllo sull'interazione all'inizio del case1, distinguendo così il caso in cui sono stati appena toccati i sensori da quello di una pressione prolungata.
Tra le soluzioni è stata preferita la prima poiché, trattandosi di una applicazione real time e date le ridotte capacità di calcolo del microcontrollore, è preferibile la ridondanza di poche istruzioni (soprattutto quando queste sono in numero ridotto) piuttosto che nessuna ridondanza ma più espressioni condizionali, le quali potrebbero influire molto sulla latenza.

```
37.             do {
38.                 sensorForceRawValue = analogRead(sensorForce);
39.
40.                 sensorPositionRawValue = analogRead(sensorPosition);
41.
42.                 sensorPositionLastValue = sensorPositionRawValue;
43.
```

```
                sensorForceValue = map(sensorForceRawValue, sensorForceMinValue, sensorForceMaxValue, 0, 127);

                sensorForceValue = constrain(sensorForceValue, 0, 127);

                if (sensorPositionRawValue>(sensorPositionMinValue)) {

                    if ((sensorPositionRawValue<(sensorPositionPreviousValue - securityRange)) || (sensorPositionRawValue>(sensorPositionPreviousValue + securityRange))) {

                        sensorPositionPreviousValue = sensorPositionRawValue;

                        sensorPositionValue = map(sensorPositionRawValue, sensorPositionMinValue, sensorPositionMaxValue, minNote, maxNote);

                        sensorPositionValue = constrain(sensorPositionValue, minNote, maxNote);

                        if (sensorPositionValue != lastNote) {

                            if (currentSendingMode == 12 && isConnected == true) {

                                AppleMIDI.noteOff(lastNote, 0, midiChannel);
                                AppleMIDI.noteOn(sensorPositionValue, 100, midiChannel);
                            }
                            else {
                                MIDI.sendNoteOff(lastNote, 0, midiChannel);
                                MIDI.sendNoteOn(sensorPositionValue, 100, midiChannel);
                            }

                            lastNote = sensorPositionValue;
                        }
                    }

                    if ((sensorForceValue != sensorForcePreviousValue) && (sensorForceRawValue>(sensorForceMinValue + securityRange))) {

                        sensorForcePreviousValue = sensorForceValue;

                        if (currentSendingMode == 12 && isConnected == true) {
                            AppleMIDI.controlChange(ccForce, sensorForceValue, midiChannel);
                        }
                        else {
                            MIDI.sendControlChange(ccForce, sensorForceValue, midiChannel);
                        }
                    }
                }
            } while (sensorPositionLastValue>(sensorPositionMinValue + 10));

            if (currentSendingMode == 12 && isConnected == true) {
                AppleMIDI.noteOff(lastNote, 0, midiChannel);
            }
            else {
                MIDI.sendNoteOff(lastNote, 0, midiChannel);
            }

            lcd.setCursor(0, 1);
            lcd.print("   Waiting Data...  ");
        }
        break;
```

Le prime operazioni che bisogna effettuare in questa fase sono quelle di lettura dei due sensori (40-42) ma, mentre per il sensore di pressione il valore letto può essere processato subito come fatto in precedenza (44-46), per il valore di posizione occorre prima verificare che esso sia maggiore o minore del valore precedentemente letto più un certo valore di sicurezza (50) per evitare variazioni troppo piccole in conseguenza al movimento sul sensore. Tutto ciò equivale a dire che occorre verificare che l'utente abbia spostato le proprie dita sul sensore abbastanza da causare una variazione di nota. La sensibilità del movimento può essere impostata tramite il valore di security range.
Se quest'ultimo controllo ha successo, occorre operare sul valore di posizione allo stesso modo visto in precedenza (52-56).
Ottenuto il valore di nota, se questo è diverso da quello inviato in precedenza (58), si effettuano le stesse operazioni di invio fatte in precedenza (60-68) con la differenza che, prima di inviare il messaggio di NoteOn basato sul nuovo valore di posizione, occorre mandare un messaggio di NoteOff basato sul vecchio valore di posizione (62) di modo che il sintetizzatore possa operare lo switch tra le due note.
Per quanto riguarda il valore di pressione letto e mappato in precedenza, se questo è maggiore o minore di quello processato precedentemente più un certo valore di sicurezza (74) per evitare variazioni troppo fini, viene utilizzato per inviare un nuovo messaggio di ControlChange utilizzando uno dei metodi proposti allo scopo (78-83).
Tutto questo processo va avanti fin tanto che viene rilevata la presenza delle mani dell'utente sui sensori (86). Quando ciò non accade e, quindi, i due sensori vengono rilasciati, un messaggio di NoteOff viene inviato utilizzando l'ultimo valore di nota rilevato e velocity nulla (88-93).
Infine un opportuno messaggio a display mostra la mancanza di interazione tra il sensore e l'utente (95-96).

### 5.4.2 PlayMode2

La modalità di funzionamento numero 2 è decisamente più specifica e particolare rispetto alla prima. In questa modalità infatti le azioni sul sensore di pressione hanno lo stesso effetto della precedente modalità ma l'interazione con il sensore di posizione genera un effetto in parte diverso.

```
1.   case statusPlayMode2:
2.
3.         navigateMenus();
4.
```

```
5.          sensorPositionRawValue = analogRead(sensorPosition);
6.
7.          if (sensorPositionRawValue>(sensorPositionMinValue + 10)) {
8.
9.              delay(5);
10.
11.             sensorPositionRawValue = analogRead(sensorPosition);
12.
13.             sensorPositionValue = map(sensorPositionRawValue, sensorPositionMinValue
    , sensorPositionMaxValue, minNote, maxNote);
14.
15.             sensorPositionValue = constrain(sensorPositionValue, minNote, maxNote);
16.
17.             if (currentSendingMode == 12 && isConnected == true) {
18.                 AppleMIDI.noteOn(sensorPositionValue, noteVelocity, midiChannel);
19.             }
20.             else {
21.                 MIDI.sendNoteOn(sensorPositionValue, noteVelocity, midiChannel);
22.             }
23.
24.             lastNote = sensorPositionValue;
25.
26.             lcd.setCursor(0, 1);
27.             lcd.print("   Sending Data...  ");
```

La prima parte del case, come in precedenza, è occupata dal codice che gestisce la prima interazione dell'utente con i sensori mentre la seconda gestisce tutte le azioni dell'utente che vanno dalla prima interazione al rilascio del sensore.

Nella prima parte, vengono effettuate le stesse identiche operazioni effettuate in precedenza sul sensore di posizione (5-27). La gestione del sensore di pressione invece è tralasciata e ripresa solo nella seconda parte. In questa fase infatti, per il modo con cui è stata concepita la modalità, è importante mandare il messaggio di NoteOn mentre la gestione dei ControlChange può essere affidata anche alla fase successiva.

Con l'eccezione della mancata gestione del valore letto dal sensore di pressione, nessun altro cambiamento di rilievo si trova nel codice rispetto a quello della modalità 1.

```
28.             do {
29.                 sensorForceRawValue = analogRead(sensorForce);
30.
31.                 sensorPositionRawValue = analogRead(sensorPosition);
32.
33.                 sensorPositionLastValue = sensorPositionRawValue;
34.
35.                 sensorForceValue = map(sensorForceRawValue, sensorForceMinValue, sen
    sorForceMaxValue, 0, 127);
36.
37.                 sensorForceValue = constrain(sensorForceValue, 0, 127);
38.
39.                 if (sensorPositionRawValue>(sensorPositionMinValue)) {
40.
41.                     sensorPositionValue = map(sensorPositionRawValue, sensorPosition
    MinValue, sensorPositionMaxValue, minNote, maxNote);
42.
```

```
43.                     sensorPositionValue = constrain(sensorPositionValue, minNote, ma
    xNote);
44.
45.                     if (currentSendingMode == 12 && isConnected == true) {
46.
47.                         AppleMIDI.controlChange(ccPosition, sensorPositionValue, mid
    iChannel);
48.
49.                         AppleMIDI.controlChange(ccForce, sensorForceValue, midiChann
    el);
50.                     }
51.                     else {
52.                         MIDI.sendControlChange(ccPosition, sensorPositionValue, midi
    Channel);
53.
54.                         MIDI.sendControlChange(ccForce, sensorForceValue, midiChanne
    l);
55.                     }
56.                 }
57.             } while (sensorPositionLastValue>(sensorPositionMinValue + 10));
58.
59.             if (currentSendingMode == 12 && isConnected == true) {
60.                 AppleMIDI.noteOff(lastNote, 0, midiChannel);
61.             }
62.
63.             else {
64.                 MIDI.sendNoteOff(lastNote, 0, midiChannel);
65.             }
66.
67.             lcd.setCursor(0, 1);
68.             lcd.print("   Waiting Data...  ");
69.         }
70.         break;
```

Nella seconda parte, fin tanto che l'utente mantiene il contatto (57), entrambi i valori in output dai sensori di posizione e pressione vengono utilizzati per mandare messaggi di ControlChange differenti: quello associato al sensore di pressione è lo stesso utilizzato nella prima modalità mentre quello associato al sensore di posizione è utilizzato per controllare il pitch della nota inviata con la prima interazione con il sensore. In realtà questi messaggi possono essere utilizzati per controllare qualsiasi parametro MIDI. Nel caso specifico, sono stati utilizzati per modulare in Ableton Live i parametri di volume e pitch ma qualsiasi altro parametro può rimpiazzare la scelta fatta in questo caso.

Nella parte iniziale del blocco do-while vengono letti i valori di pressione e posizione (29-31) e avviene il map del valore di pressione letto tra il minimo e massimo valore assumibile dal sensore e 0-127 (35), che rappresentano i valori minimo e massimo che può assumere un messaggio di ControlChange.

Se il valore di posizione risulta valido ai fini del processamento (39), le stesse azioni di map e costrain effettuate sul valore di pressione sono effettuate su quello di posizione (41-43) e attraverso la modalità di comunicazione settata vengono mandati i due massaggi di ControlChange (45-55).

La parte finale del blocco è praticamente la stessa vista in precedenza, con l'invio del massaggio di NoteOff (59-65) a termine dell'interazione con i sensori e la visualizzazione del messaggio di attesa sull'LCD (67-68).

```
1.  LedPitchValue = map(sensorPositionRawValue, sensorPositionMinValue, sensorPositionMa
    xValue, 1, 12);
2.
3.  LedVolumeValue = map(sensorForceRawValue, sensorForceMinValue, sensorForceMaxValue,
    1, 12);
4.
5.  if (LedPitchValue >= OldLedPitchValue) {
6.     for (int i = 31; i < (31 + (LedPitchValue * 2)); i = i + 2) {
7.         digitalWrite(i, HIGH);
8.     }
9.  }
10. else {
11.    int diff = OldLedPitchValue - LedPitchValue;
12.    for (int i = (29 + (OldLedPitchValue * 2)); i >(29 + (LedPitchValue * 2)); i = i
    - 2) {
13.        digitalWrite(i, LOW);
14.    }
15. }
16.
17. OldLedPitchValue = LedPitchValue;
18.
19. if (LedVolumeValue >= OldLedVolumeValue) {
20.    for (int i = 30; i < (30 + (LedVolumeValue * 2)); i = i + 2) {
21.        digitalWrite(i, HIGH);
22.    }
23. }
24. else {
25.    int diff = OldLedVolumeValue - LedVolumeValue;
26.    for (int i = (28 + (OldLedVolumeValue * 2)); i >(28 + (LedVolumeValue * 2)); i =
    i - 2) {
27.        digitalWrite(i, LOW);
28.    }
29. }
30.
31. OldLedVolumeValue = LedVolumeValue;
```

Sia in modalità 1 che 2, la gestione dei led di volume e pitch avviene allo stesso modo. Vengono mappati i valori di pressione e posizione appena letti tra 1 e 12 (1,3) e, a seconda che il valore appena gestito sia maggiore o minore di quello precedentemente letto, si accendono o spengono i led interessati dalla variazione di pitch (5-17) o volume (19-31).

### 5.4.3 SetCCPosition

Il terzo case dello switch sullo stato di funzionamento è occupato dalla parte di codice che si occupa del settaggio del valore di Control Change associato al sensore di posizione, che verrà utilizzato nella modalità 2 per inviare messaggi di ControlChange al sintetizzatore, variando il

pitch della nota in esecuzione. L'utente può, in qualsiasi momento dell'esecuzione delle due modalità precedentemente viste, accedere a queste impostazioni agendo sul pulsante Enter e scorrendo il menù fino alla voce di menù desiderata.

```
1.  case statusSetCCPosition:
2.
3.          switch (lastButtonPushed) {
4.
5.          case buttonPinEnter:
6.              ccPosition = ccPositionSetting;
7.              currentStatusMode = lastPlayingMode;
8.              EEPROM.write(addrSensorPosCC, ccPosition);
9.              menu.toRoot();
10.             break;
11.
12.         case buttonPinEsc:
13.             ccPositionSetting = ccPosition;
14.             currentStatusMode = lastPlayingMode;
15.             menu.toRoot();
16.             break;
17.
18.         case buttonPinUp:
19.             if (ccPositionSetting<127) {
20.                 ccPositionSetting = ccPositionSetting + 1;
21.                 itoa(ccPositionSetting, sett, 10);
22.                 lcd.clear();
23.                 lcd.setCursor(0, 0);
24.                 lcd.print(strSetPositionCC);
25.                 lcd.setCursor(0, 1);
26.                 lcd.print(strcat(strSetting, sett));
27.             }
28.             break;
29.
30.         case buttonPinDown:
31.             if (ccPositionSetting>0) {
32.                 ccPositionSetting = ccPositionSetting - 1;
33.                 itoa(ccPositionSetting, sett, 10);
34.                 lcd.clear();
35.                 lcd.setCursor(0, 0);
36.                 lcd.print(strSetPositionCC);
37.                 lcd.setCursor(0, 1);
38.                 lcd.print(strcat(strSetting, sett));
39.             }
40.             break;
41.
42.         }
43.         break;
```

Il codice prevede un ulteriore switch interno che dipende dal pulsante premuto tra i quattro disponibili:

- ENTER: viene memorizzato nell'apposita variabile il valore di Control Change per il sensore di posizione impostato (6) e visibile sull'LCD, la modalità di funzionamento del controller rimane la stessa che era in esecuzione prima delle operazioni di setting (7) e viene salvata nell'apposita locazione EEPROM il valore appena impostato (8). Al

termine di queste operazioni il controllo viene riassegnato alla modalità di esecuzione precedente il setting e il menù viene riportato alla sua root (9).

- ESC: tutto rimane immutato. Vengono salvate nelle apposite variabili gli stessi valori utilizzati fino a quel punto (13-14) e si esce dal setting restituendo il controllo alla modalità in esecuzione precedente alla fase di setting.
- UP: incrementa il valore di control change associato al sensore di posizione (20) fin tanto che quest'ultimo rientra nel range di valori consentito (0-127) (19) ma non lo conferma (operazione affidata al bottone ENTER). Un'operazione di cast (21) del valore aggiornato di control change si rende necessario per la visualizzazione dello stesso a display (22-26), concatenato a stringhe definite ad hoc per lo scopo all'inizio del codice.
- DOWN: svolge le stesse operazioni del bottone precedente con la sola differenza che invece di incrementare il valore di control change, lo decrementa (32) fin tanto che questo rimanga nei limiti consentiti (31).

### 5.4.4 SetCCForce

Il quarto case dello switch sullo stato di funzionamento è occupato dalla parte di codice che si occupa del settaggio del valore di Control Change associato al sensore di pressione, che verrà utilizzato nella modalità 1 e 2 per inviare messaggi di ControlChange al sintetizzatore.

```
1.      case statusSetCCForce:
2.
3.          switch (lastButtonPushed) {
4.
5.          case buttonPinEnter:
6.              ccForce = ccForceSetting;
7.              currentStatusMode = lastPlayingMode;
8.              EEPROM.write(addrSensorForceCC, ccForce);
9.              menu.toRoot();
10.             break;
11.
12.         case buttonPinEsc:
13.             ccForceSetting = ccForce;
14.             currentStatusMode = lastPlayingMode;
15.             menu.toRoot();
16.             break;
17.
18.         case buttonPinUp:
19.             if (ccForceSetting<127) {
20.                 ccForceSetting = ccForceSetting + 1;
21.                 itoa(ccForceSetting, sett, 10);
22.                 lcd.clear();
23.                 lcd.setCursor(0, 0);
24.                 lcd.print(strSetForceCC);
25.                 lcd.setCursor(0, 1);
```

```
                lcd.print(strcat(strSetting, sett));
            }
            break;

        case buttonPinDown:
            if (ccForceSetting>0) {
                ccForceSetting = ccForceSetting - 1;
                itoa(ccForceSetting, sett, 10);
                lcd.clear();
                lcd.setCursor(0, 0);
                lcd.print(strSetForceCC);
                lcd.setCursor(0, 1);
                lcd.print(strcat(strSetting, sett));
            }
            break;
        }
        break;
```

Anche in questo caso uno switch interno basato sul bottone premuto regola le azioni eseguite sul valore per il quale si effettua il setting.

Tutti i bottoni eseguono le stesse operazioni viste in precedenza con la sola differenza che l'oggetto del cambiamento è il control change associato al sensore di pressione e non più a quello di posizione.

### 5.4.5 SetMinNote

Dopo il setting dei valori di control change associati ai sensori di posizione e pressione, troviamo quelli per il controllo del range che può assumere la nota inviata nei messaggi di NoteOn sia in modalità 1 che 2.

Il funzionamento è pressoché identico a quello visto in precedenza, con lo switch interno che opera sul valore associato al pulsante premuto. All'interno dei singoli case le operazioni sono le stesse con la sola variazione delle variabili associate ai cambiamenti e alle locazioni di memoria nelle quali vengono memorizzati i dati per usi futuri.

```
    case statusSetMinNote:
        switch (lastButtonPushed) {

        case buttonPinEnter:
            minNote = minNoteSetting;
            currentStatusMode = lastPlayingMode;
            EEPROM.write(addrMinNote, minNote);
            menu.toRoot();
            break;

        case buttonPinEsc:
            minNoteSetting = minNote;
            currentStatusMode = lastPlayingMode;
            menu.toRoot();
            break;

        case buttonPinUp:
```

```
            if (minNoteSetting<127) {
                minNoteSetting = minNoteSetting + 1;
                lcd.clear();
                lcd.setCursor(0, 0);
                lcd.print(strSetMinimumNote);
                lcd.setCursor(0, 1);
                lcd.print(minNoteSetting);
            }
            break;

        case buttonPinDown:
            if (minNoteSetting>0) {
                minNoteSetting = minNoteSetting - 1;
                lcd.clear();
                lcd.setCursor(0, 0);
                lcd.print(strSetMinimumNote);
                lcd.setCursor(0, 1);
                lcd.print(minNoteSetting);
            }
            break;
        }
        break;
```

### 5.4.6 SetMaxNote

Con il seguente codice è possibile variare il valore massimo che la nota può assumere.

Il funzionamento è come quello visto in precedenza, cambiando le variabili e locazioni di memoria in gioco.

```
    case statusSetMaxNote:

        switch (lastButtonPushed) {

        case buttonPinEnter:
            maxNote = maxNoteSetting;
            currentStatusMode = lastPlayingMode;
            EEPROM.write(addrMaxNote, maxNote);
            menu.toRoot();
            break;

        case buttonPinEsc:
            maxNoteSetting = maxNote;
            currentStatusMode = lastPlayingMode;
            menu.toRoot();
            break;

        case buttonPinUp:
            if (maxNoteSetting<127) {
                maxNoteSetting = maxNoteSetting + 1;
                lcd.clear();
                lcd.setCursor(0, 0);
                lcd.print(strSetMaximumNote);
                lcd.setCursor(0, 1);
                lcd.print(maxNoteSetting);
            }
            break;

        case buttonPinDown:
            if (maxNoteSetting>0) {
                maxNoteSetting = maxNoteSetting - 1;
```

```
                lcd.clear();
                lcd.setCursor(0, 0);
                lcd.print(strSetMaximumNote);
                lcd.setCursor(0, 1);
                lcd.print(maxNoteSetting);
            }
            break;
        }
        break;
```

### 5.4.7 SetMidiChannel

Il seguente codice si occupa del setting del midi channel che viene utilizzato per l'invio dei dati al sintetizzatore. In questo caso, anche se la struttura rimane fondamentalmente la stessa, cambiano le variabili in gioco ma anche i bound che il canale può assumere. Le routine associate ai bottoni UP e Down infatti lavorano non più tra 0 e 127 ma da 1 a 16 che sono i valori possibili per la variabile che si sta settando.

```
    case statusSetMidiChannel:

        switch (lastButtonPushed) {

        case buttonPinEnter:
            midiChannel = midiChannelSetting;
            currentStatusMode = lastPlayingMode;
            EEPROM.write(addrMidiChannel, midiChannel);
            menu.toRoot();
            break;

        case buttonPinEsc:
            midiChannelSetting = midiChannel;
            currentStatusMode = lastPlayingMode;
            menu.toRoot();
            break;

        case buttonPinUp:
            if (midiChannelSetting<16) {
                midiChannelSetting = midiChannelSetting + 1;
                lcd.clear();
                lcd.setCursor(0, 0);
                lcd.print(strSetMidiChannel);
                lcd.setCursor(0, 1);
                lcd.print(midiChannelSetting);
            }
            break;

        case buttonPinDown:
            if (midiChannelSetting>1) {
                midiChannelSetting = midiChannelSetting - 1;
                lcd.clear();
                lcd.setCursor(0, 0);
                lcd.print(strSetMidiChannel);
                lcd.setCursor(0, 1);
                lcd.print(midiChannelSetting);
            }
            break;
        }
        break;
```

### 5.4.8 MidiTeach

L'ultimo case della loop function è utilizzata per realizzare la funzione di Midi teach che è molto utile nelle fasi di connessione tra il controller e il software di sintesi. Tramite l'interazione con i pulsanti soli, senza interagire con i sensori, è possibile inviare sul canale 1 massaggi di control change caratterizzati da particolari valori che possono essere controllati lato sintetizzatori in fase di connessione.

In particolare:

- ENTER: non abilitato,
- ESC: lascia tutto invariato, compresa la modalità di esecuzione (6), e riporta il menù alla root (7),
- DOWN: manda due messaggi di ControlChange sul canale 1, basati sul valore associato al sensore di pressione con value rispettivamente 64 e 65 (11-12),
- UP: manda due messaggi di ControlChange sul canale 1, basati sul valore associato al sensore di pressione con value rispettivamente 64 e 65 (16-17).

```
1.      case statusMidiTeach:
2.
3.          switch (lastButtonPushed) {
4.
5.          case buttonPinEsc:
6.              currentStatusMode = lastPlayingMode;
7.              menu.toRoot();
8.              break;
9.
10.         case buttonPinDown:
11.             MIDI.sendControlChange(ccForce, 64, 1);
12.             MIDI.sendControlChange(ccForce, 65, 1);
13.             break;
14.
15.         case buttonPinUp:
16.             MIDI.sendControlChange(ccPosition, 64, 1);
17.             MIDI.sendControlChange(ccPosition, 65, 1);
18.             break;
19.         }
20.         break;
```

## 5.5 **LCD & Menù Controller Functions**

Il controllo dell'LCD è affidato fondamentalmente a due grandi funzioni:

- `void menuChanged(MenuChangeEvent changed);`
- `void menuUsed(MenuUseEvent used).`

Tali funzioni sono offerte dalla libreria MenùBackEnd.h e, seppur macchinose, offrono un supporto molto chiaro e di facile utilizzo per il controllo e la gestione dei messaggi da visualizzare sull'LCD.
La maggior parte del lavoro fatto per la gestione dei messaggi per l'LCD rientra nella costruzione del menù che raggruppa 18 item e che organizzano tutte le funzionalità di esecuzione e setting del controller per una più semplice e intuitiva gestione da parte dell'utente. In particolare le due funzioni hanno funzione differente ma complementare: la prima si occupa di cambiare l'output visivo sull'LCD in risposta all'interazione dell'utente con i bottoni, mentre la seconda si occupa di lanciare le varie routine associate agli item di menù quando e se l'utente le seleziona.

### 5.5.1 MenuChanged function

Come detto, questa funzione ha il compito di aggiornare il contenuto mostrato sull'LCD nel momento dell'interazione dell'utente con i bottoni. Nessun'altra azione è delegata a questa funzione. Leggendo il codice è possibile capire abbastanza chiaramente la struttura stessa del menù. Quello che bisogna tener presente per una corretta comprensione del codice è che l'interazione dell'utente non provoca il cambiamento di particolari elementi mostrati a display ma l'intero aggiornamento del contenuto del display con le modifiche necessarie in conseguenza all'interazione. Per fare un esempio molto semplice: se ci troviamo in un sottomenù con più item su cui poter spostare il cursore per poterli selezionare, l'interazione dell'utente con i bottoni Up e Down non provoca una modifica del cursore che si sposterà da una posizione all'altra del menù ma un cambiamento dell'intero contenuto del display che, a questo punto, ridisegnerà probabilmente gli stessi elementi precedenti con la posizione aggiornata del cursore.
Questo approccio, seppur molto ripetitivo, risulta essere molto intuitivo e soprattutto di facile realizzazione specialmente quando si ha a che fare con dispositivi seriali come l'LCD utilizzato per il progetto.

```
1.  void menuChanged(MenuChangeEvent changed) {
2.      MenuItem newMenuItem = changed.to;
3.      lcd.setCursor(0, 0);
4.
5.      if (newMenuItem.getName() == menu.getRoot()) {
6.          char modeNumber[2];
7.          String invio;
8.          String uso;
9.          itoa(currentStatusMode, modeNumber, 10);
10.
```

```
11.         if (currentUsingMode == 13) {
12.             uso = "USING MODE: HAND";
13.         }
14.         else {
15.             uso = "USING MODE: FINGER";
16.         }
17.
18.         if (currentSendingMode == 11) {
19.             invio = "SENDING MODE: WIRED";
20.         }
21.         else {
22.             invio = "SENDING MODE: Wi-Fi";
23.         }
24.
25.         char strPlaying[17] = "PLAYING MODE: ";
26.         lcd.clear();
27.         lcd.setCursor(0, 0);
28.         lcd.print(strcat(strPlaying, modeNumber));
29.         lcd.setCursor(0, 1);
30.         lcd.print(uso);
31.         lcd.setCursor(0, 2);
32.         lcd.print(invio);
33.
34.         digitalWrite(ledPlayng, HIGH);
35.         digitalWrite(ledSetting, LOW);
36.     }
37.
```

la prima parte della funzione ha lo scopo di creare le strutture di supporto alla gestione della stessa. Per prima cosa si recupera dalla struttura “changed” descritta in precedenza il secondo valore (2) che contiene il valore dell’item successivo a quello corrente. Attraverso un controllo sul nome dell’item (5) è possibile capire in che posizione del menù ci troviamo e in base a questa informazione è possibile adattare il contenuto mostrato sull’LCD a quello della parte di menù in cui ci troviamo.

Nel primo caso, se il nome coincide con la root del menù (5), vengono mostrati sull’LCD le informazioni principali sullo stato del sistema (10-35). Questo stato in realtà è quello che visualizziamo la maggior parte del tempo durante l’esecuzione del controller perché coincide con l’item di menù associato al normale funzionamento del controller sia in modalità 1 che 2.

Le informazioni mostrate in questo frangente sono: stringa che da informazione sulla modalità di funzionamento corrente del controller (25), la modalità corrente di comunicazione del controller con il dispositivo di sintesi (18-23), modalità d’uso del controller (11-16) e una stringa che cambia a seconda che il controller sia in attesa di interazione o stia inviando dati al sintetizzatore durante l’interazione con i sensori. Nella parte finale invece vengono settati i led di controllo di stato per mostrare se il controller si trovi in una modalità di funzionamento o di setting (34,35). Tale procedura sarà effettuata in tutti i blocchi di codice successivi.

La porzione di codice successiva invece mostra il contenuto del primo sottomenù.

```
else if (newMenuItem.getName() == "menuModesSetup") {
        lcd.print(strMenu);
        lcd.setCursor(0, 1);
        lcd.print("Modes Setup      <--");
        lcd.setCursor(0, 2);
        lcd.print("Using Setup         ");
        lcd.setCursor(0, 3);
        lcd.print("MIDI Setup          ");
        digitalWrite(ledPlayng, LOW);
        digitalWrite(ledSetting, HIGH);
    }
    else if (newMenuItem.getName() == "menuUsingSetup") {
        lcd.print(strMenu);
        lcd.setCursor(0, 1);
        lcd.print("Modes Setup         ");
        lcd.setCursor(0, 2);
        lcd.print("Using Setup      <--");
        lcd.setCursor(0, 3);
        lcd.print("MIDI Setup          ");
    }
    else if (newMenuItem.getName() == "menuItemMidiSetup") {
        lcd.print(strMenu);
        lcd.setCursor(0, 1);
        lcd.print("Modes Setup         ");
        lcd.setCursor(0, 2);
        lcd.print("Using Setup         ");
        lcd.setCursor(0, 3);
        lcd.print("MIDI Setup       <--");
    }
    else if (newMenuItem.getName() == "menuSendMode") {
        lcd.print(strMenu);
        lcd.setCursor(0, 1);
        lcd.print("Using Setup         ");
        lcd.setCursor(0, 2);
        lcd.print("MIDI Setup          ");
        lcd.setCursor(0, 3);
        lcd.print("Sending mode     <--");
    }
    else if (newMenuItem.getName() == "menuItemWifiSetup") {
        lcd.print(strMenu);
        lcd.setCursor(0, 1);
        lcd.print("MIDI Setup          ");
        lcd.setCursor(0, 2);
        lcd.print("Sending mode        ");
        lcd.setCursor(0, 3);
        lcd.print("Wi-Fi Setup      <--");
    }
    else if (newMenuItem.getName() == "menuCalibration") {
        lcd.print(strMenu);
        lcd.setCursor(0, 1);
        lcd.print("Sending mode        ");
        lcd.setCursor(0, 2);
        lcd.print("Wi-Fi Setup         ");
        lcd.setCursor(0, 3);
        lcd.print("Calibration      <--");
    }
```

Come è facilmente intuibile dal codice, il più alto sottomenù della gerarchia contiene 6 item:

- Mode Setup: sottomenù per il setting delle modalità di funzionamento del controller,
- Using Setup: sottomenù per il setting della modalità d'uso del controller (mano/dita),
- MIDI Setup: sottomenù per il setting delle variabili midi utilizzate dal controller,

- Sending mode: sottomenù per il setting della modalità di comunicazione,
- Wi-Fi setup: sottomenù per la modifica dei parametri connessione wifi,
- Calibration: item per il lancio della routine di calibrazione dei sensori.

Sebbene si abbia l'impressione che ogni item sia identificato da uno specifico nome, in realtà non è così. I differenti nomi assegnati agli item infatti servono in questo caso solo e soltanto ad identificare una specifica "schermata" che dovrà essere mandata all'LCD. Ricordiamo infatti che lo scopo di questa funzione non è risalire alla funzione degli item ma solo e soltanto quella di "disegnare" la struttura del menù a display e di far coincidere l'interazione dell'utente ad un'effettiva risposta visuale del sistema.

Ad eccezione dell'ultimo item (Calibration) di questo sottomenù (che fa riferimento direttamente ad una routine), tutti gli altri 4 danno accesso ad un ulteriore livello di menù con altri item.

```
1.      else if (newMenuItem.getName() == "menuItemMode1") {
2.          lcd.print(strMenuModes);
3.          lcd.setCursor(0, 1);
4.          lcd.print("                    ");
5.          lcd.setCursor(0, 2);
6.          lcd.print("Mode 1          <--");
7.          lcd.setCursor(0, 3);
8.          lcd.print("Mode 2              ");
9.      }
10.     else if (newMenuItem.getName() == "menuItemMode2") {
11.         lcd.print(strMenuModes);
12.         lcd.setCursor(0, 1);
13.         lcd.print("                    ");
14.         lcd.setCursor(0, 2);
15.         lcd.print("Mode 1              ");
16.         lcd.setCursor(0, 3);
17.         lcd.print("Mode 2          <--");
18.     }
```

Il primo item (Mode setup), a sua volta sottomenù, contiene altri due item che hanno lo scopo di settare il controller in una delle due modalità di funzionamento:

- Mode 1,
- Mode 2.

```
1.  else if (newMenuItem.getName() == "menuItemUse1") {
2.          lcd.print(strSetUsingMode);
3.          lcd.setCursor(0, 1);
4.          lcd.print("                    ");
5.          lcd.setCursor(0, 2);
6.          lcd.print("Hand mode       <--");
7.          lcd.setCursor(0, 3);
8.          lcd.print("Finger mode         ");
9.      }
10.     else if (newMenuItem.getName() == "menuItemUse2") {
```

```
        lcd.print(strSetUsingMode);
        lcd.setCursor(0, 1);
        lcd.print("                    ");
        lcd.setCursor(0, 2);
        lcd.print("Hand mode           ");
        lcd.setCursor(0, 3);
        lcd.print("Finger mode      <--");
    }
```

Il secondo sottomenù contiene due soli item per il setting della modalità d'uso del controller:

- Utilizzo con una mano,
- Utilizzo con dita:

```
    else if (newMenuItem.getName() == "menuItemCCPosition") {
        lcd.print(strMenuMidi);
        lcd.setCursor(0, 1);
        lcd.print("                    ");
        lcd.setCursor(0, 2);
        lcd.print("Set position CC  <--");
        lcd.setCursor(0, 3);
        lcd.print(strSetForceCC);
    }
    else if (newMenuItem.getName() == "menuItemCCForce") {
        lcd.print(strMenuMidi);
        lcd.setCursor(0, 1);
        lcd.print(strSetPositionCC);
        lcd.setCursor(0, 2);
        lcd.print("Set force CC     <--");
        lcd.setCursor(0, 3);
        lcd.print(strSetMinimumNote);
    }
    else if (newMenuItem.getName() == "menuItemMinNote") {
        lcd.print(strMenuMidi);
        lcd.setCursor(0, 1);
        lcd.print(strSetForceCC);
        lcd.setCursor(0, 2);
        lcd.print("Set minimum note <--");
        lcd.setCursor(0, 3);
        lcd.print(strSetMaximumNote);
    }
    else if (newMenuItem.getName() == "menuItemMaxNote") {
        lcd.print(strMenuMidi);
        lcd.setCursor(0, 1);
        lcd.print(strSetMinimumNote);
        lcd.setCursor(0, 2);
        lcd.print("Set maximum note <--");
        lcd.setCursor(0, 3);
        lcd.print(strSetMidiChannel);
    }
    else if (newMenuItem.getName() == "menuItemMidiChannel") {
        lcd.print(strMenuMidi);
        lcd.setCursor(0, 1);
        lcd.print(strSetMaximumNote);
        lcd.setCursor(0, 2);
        lcd.print("Set MIDI channel <--");
        lcd.setCursor(0, 3);
        lcd.print("MIDI teach          ");
    }
    else if (newMenuItem.getName() == "menuItemMidiTeach") {
        lcd.print(strMenuMidi);
        lcd.setCursor(0, 1);
        lcd.print(strSetMidiChannel);
```

```
        lcd.setCursor(0, 2);
        lcd.print("MIDI teach       <--");
        lcd.setCursor(0, 3);
        lcd.print("                    ");
    }
```

Il terzo sottomenù (Midi Setup) contiene altri 6 item ognuno dei quali da la possibilità di cambiare, grazie alle funzioni discusse nel paragrafo precedente, particolari parametri MIDI che il controller utilizza per la creazione dei messaggi MIDI. Questi sono:

- Set Position CC: per la modifica del control change associato al sensore di posizione,
- Set Force CC: per la modifica del control change associato al sensore di pressione,
- Set Minimun Note: per la modifica della nota minima assegnabile al NoteOn,
- Set Maximun Note: per la modifica della nota massima assegnabile al NoteOn,
- Set MIDI Channel: per la modifica del channel Midi,
- Midi Teach: per lanciare la routine di teach MIDI.

Tutti gli item elencati fanno riferimento a routine presenti nella Loop Function e che sono state trattate in dettaglio nel paragrafo dedicato.

```
    else if (newMenuItem.getName() == "menuItemSend1") {
        lcd.print(strSetSendMode);
        lcd.setCursor(0, 1);
        lcd.print("                    ");
        lcd.setCursor(0, 2);
        lcd.print("Wired mode       <--");
        lcd.setCursor(0, 3);
        lcd.print("Wireless mode       ");
    }
    else if (newMenuItem.getName() == "menuItemSend2") {
        lcd.print(strSetSendMode);
        lcd.setCursor(0, 1);
        lcd.print("                    ");
        lcd.setCursor(0, 2);
        lcd.print("Wired mode          ");
        lcd.setCursor(0, 3);
        lcd.print("Wireless mode    <--");
    }
```

Il quarto item (Sending mode) da accesso al menù per la scelta della modalità di comunicazione del controller con il dispositivo preposto alla sintesi. I due item contenuti al suo interno sono:

- Wired Mode,
- Wireless Mode.

Entrambi gli item lanciano una routine di setting che verrà trattata in seguito.

```
    else if (newMenuItem.getName() == "menuItemWifi1") {
        lcd.print(strSettingWifi);
        lcd.setCursor(0, 1);
        lcd.print("Set SSID         <--");
        lcd.setCursor(0, 2);
```

```
        lcd.print("Set Password        ");
        lcd.setCursor(0, 3);
        lcd.print("Set SSID & Password ");
    }
    else if (newMenuItem.getName() == "menuItemWifi2") {
        lcd.print(strSettingWifi);
        lcd.setCursor(0, 1);
        lcd.print("Set SSID            ");
        lcd.setCursor(0, 2);
        lcd.print("Set Password     <--");
        lcd.setCursor(0, 3);
        lcd.print("Set SSID & Password ");
    }
    else if (newMenuItem.getName() == "menuItemWifi3") {
        lcd.print(strSettingWifi);
        lcd.setCursor(0, 1);
        lcd.print("Set SSID            ");
        lcd.setCursor(0, 2);
        lcd.print("Set Password        ");
        lcd.setCursor(0, 3);
        lcd.print("Set SSID & Pass  <--");
    }
```

Il quinto item porta ad un menù per la scelta del parametro di connessione Wi-Fi che si intende modificare. Le possibilità sono:

- Set SSID: per la modifica dell'SSID di rete,
- Set Password: per la modifica della passphrase associata all'SSID,
- Set SSID & Pass: per la modifica di entrambi.

### 5.5.2 MenùUsed function

La funzione MenùUsed lavora in stretto rapporto con quella precedentemente descritta ma il suo compito è molto diverso. Nel momento in cui l'utente seleziona un particolare item dal menù, questa funzione esegue un controllo sul nome dell'item selezionato dall'utente e se quest'ultimo rientra tra quelli che gestisce vengono eseguite semplicemente operazioni di cambio stato o vengono lanciate vere e proprie routine per gestire gli eventi collegati all'item. Le operazioni di cambio stato, apparentemente banali, sono molto importanti perché su di esse si basa il funzionamento intero del controller, per come è stato concepito.

```
void menuUsed(MenuUseEvent used) {

    if (used.item == menuItemMode1) {
        currentStatusMode = statusPlayMode1;
        EEPROM.write(addrPlayMode, statusPlayMode1);
        menu.toRoot();
    }
    else if (used.item == menuItemMode2) {
        currentStatusMode = statusPlayMode2;
        EEPROM.write(addrPlayMode, statusPlayMode2);
        menu.toRoot();
    }
```

Nel caso in cui l'item di menù selezionato dall'utente è quello per il setting della modalità di funzionamento del controller (Mode 1 o Mode 2), le azioni svolte sono molto semplici e riguardano, in entrambi i casi, l'aggiornamento dello stato corrente con quello selezionato dall'utente (4, 9) e che verrà utilizzato dalla funzione Loop per selezionare il "case" con il codice che lo gestisce, l'aggiornamento della locazione EEPROM (5, 10) per il salvataggio della modalità di funzionamento scelta al fine di ripristinarla anche dopo lo spegnimento del controller e infine il ritorno alla root del menù (6, 11) con il ripristino del funzionamento normale del controller nella nuova modalità scelta dall'utente.

```
1.      else if (used.item == menuItemCCPosition) {
2.          ccPositionSetting = ccPosition;
3.          lastPlayingMode = currentStatusMode;
4.          currentStatusMode = statusSetCCPosition;
5.          char ccPos[4];
6.          itoa(ccPositionSetting, ccPos, 10);
7.          char strCcPos[17] = "CC ";
8.          lcd.clear();
9.          lcd.setCursor(0, 0);
10.         lcd.print(strSetPositionCC);
11.         lcd.setCursor(0, 1);
12.         lcd.print(strcat(strCcPos, ccPos));
13.     }
14.     else if (used.item == menuItemCCForce) {
15.         ccForceSetting = ccForce;
16.         lastPlayingMode = currentStatusMode;
17.         currentStatusMode = statusSetCCForce;
18.         char ccPress[4];
19.         itoa(ccForceSetting, ccPress, 10);
20.         char strCcPress[17] = "CC ";
21.         lcd.clear();
22.         lcd.setCursor(0, 0);
23.         lcd.print(strSetForceCC);
24.         lcd.setCursor(0, 1);
25.         lcd.print(strcat(strCcPress, ccPress));
26.     }
```

Nel caso l'utente voglia cambiare i valori associati al control change del sensore di posizione o pressione, la funzione acquisisce il valore di contro change associato al sensore (2, 15), salva il valore dello stato corrente di funzionamento (3, 16) (sempre uno tra Mode 1 e Mode 2) per rispristinarlo al termine della routine di modifica del CC e imposta la modalità corrente come SetCCPosition (4, 17).

All'esterno l'utente visualizza sul display il valore corrente associato al CC del sensore che intende settare (8-12, 21-25). Nella scrittura dei valori sul display un cast (6, 19) si rende necessario per convertire gli integer salvati nelle variabili in un formato accettato dalla funzione che concatena questi ultimi alla stringa creata appositamente per la stampa a display.

```
1.      else if (used.item == menuItemMinNote) {
2.          minNoteSetting = minNote;
```

```
3.          lastPlayingMode = currentStatusMode;
4.          currentStatusMode = statusSetMinNote;
5.          lcd.clear();
6.          lcd.setCursor(0, 0);
7.          lcd.print(strSetMinimumNote);
8.          lcd.setCursor(0, 1);
9.          lcd.print(minNote);
10.     }
11.     else if (used.item == menuItemMaxNote) {
12.         maxNoteSetting = maxNote;
13.         lastPlayingMode = currentStatusMode;
14.         currentStatusMode = statusSetMaxNote;
15.         lcd.clear();
16.         lcd.setCursor(0, 0);
17.         lcd.print(strSetMaximumNote);
18.         lcd.setCursor(0, 1);
19.         lcd.print(maxNote);
20.     }
21.     else if (used.item == menuItemMidiChannel) {
22.         midiChannelSetting = midiChannel;
23.         lastPlayingMode = currentStatusMode;
24.         currentStatusMode = statusSetMidiChannel;
25.         lcd.clear();
26.         lcd.setCursor(0, 0);
27.         lcd.print(strSetMidiChannel);
28.         lcd.setCursor(0, 1);
29.         lcd.print(midiChannelSetting);
30.     }
```

Stesso discorso va fatto nel caso di setting del valore minino e massimo che si vuole assegnare alle note nei messaggi di NoteOn e per il settaggio del channel su cui gli stessi verranno inviati. In questo caso, eccetto che per l'utilizzo di variabili diverse e l'utilizzo di una modalità differente, le operazioni rimangono invariate rispetto al caso precedente. Qui a differenza di prima nessuna operazione di cast è necessaria poiché il valore della nota non viene concatenato a nessun altro carattere per la stampa.

```
1.      else if (used.item == menuItemMidiTeach) {
2.          lastPlayingMode = currentStatusMode;
3.          currentStatusMode = statusMidiTeach;
4.          lcd.setCursor(0, 0);
5.          lcd.print("MIDI TEACH MODE     ");
6.          lcd.setCursor(0, 1);
7.          lcd.print("Up: send position CC");
8.          lcd.setCursor(0, 2);
9.          lcd.print("Down: send force CC ");
10.     }
```

Per l'item che si riferisce all'operazione di MIDI teach, le operazioni effettuate in questo frangente prevedono il solito set delle modalità al fine di cambiare la corrente e ripristinare la precedente a termine della routine (2-3) e di stampare sull'LCD dei messaggi di istruzione per l'utilizzo di questa funzionalità (4-9).

```
1.      else if (used.item == menuCalibration) {
2.          calibration();
```

```
3.          menu.toRoot();
4.      }
```

L'item calibration esegue la funzione omonima che verrà trattata in seguito e ritorna alla root del menù.

```
1.      else if (used.item == menuItemSend1) {
2.          currentSendingMode = statusSendWired;
3.          menu.toRoot();
4.      }
5.      else if (used.item == menuItemSend2) {
6.          currentSendingMode = statusSendWireless;
7.          SetWireless();
8.          menu.toRoot();
9.      }
10.     else if (used.item == menuItemWifi1) {
11.         SetWifiCredential(0);
12.         menu.toRoot();
13.     }
14.     else if (used.item == menuItemWifi2) {
15.         SetWifiCredential(1);
16.         menu.toRoot();
17.     }
18.     else if (used.item == menuItemWifi3) {
19.         SetWifiCredential(2);
20.         menu.toRoot();
21.     }
```

La parte di gestione della comunicazione e delle impostazioni relative al wifi, è molto semplice perché non fa altro che impostare la modalità corrente come quella selezionata dall'utente (2, 6) e lanciare le relative routine (7, 11, 15, 19) per il setting dei parametri che si vogliono modificare.

```
1.  else if (used.item == menuItemUse1) {
2.          currentUsingMode = statusUseHand;
3.          securityRange = hand;
4.          EEPROM.write(addrSecurityRange, hand);
5.          menu.toRoot();
6.      }
7.      else if (used.item == menuItemUse2) {
8.          currentUsingMode = statusUseFinger;
9.          securityRange = finger;
10.         EEPROM.write(addrSecurityRange, finger);
11.         menu.toRoot();
12.     }
```

Come la parte precedente, anche quella di gestione della modalità d'uso del controller è molto semplice. Si imposta la modalità d'uso corrente con quella selezionata (2,8), si imposta il valore di security range corrispondente (3,9), si salva il valore impostato nell'EEPROM (4,10) e si ritorna alla root del menù (5,11).

In tutti i blocchi di codice appena esaminati viene effettuato anche il settaggio dei led di stato del controller a seconda della modifica di funzionamento appena effettuata. Tale controllo e settaggio avviene sempre prima del ritorno del controller al suo stato di funzionamento normale.

## 5.6 **Support & Setting Functions**

Le funzioni di supporto e setting sono tutti quei metodi e funzioni che nel codice del progetto offrono funzionalità di supporto a funzioni più complesse o di setting di parametri che servono a definire il funzionamento dell'intero controller.
Fra tutte, le più importanti sono:

- ReadButtons() {…}: la sua funzione è quella di leggere lo stato dei bottoni e gestire eventuali interazioni dell'utente con questi ultimi;
- navigateMenus() {…}: si occupa di "navigare" all'interno della gerarchia del menù;
- calibration() {…}: ha la funzione di calibrazione dei sensori;
- SetWireless() {…}: viene chiamata quando l'utente imposta il wireless come modalità di comunicazione;
- SetWifiCredential(int value) {…}: si occupa della modifica dei paramentri di connessione alla rete LAN.

Oltre a queste ci sono altre funzioni di minore importanza che sono utilizzate soprattutto per operazioni di debug a run time e gestione dei messaggi su LCD.

### 5.6.1 ReadButtons

Come accennato, le funzionalità offerte da ReadButtons sono quelle di lettura e gestione degli input esterni che arrivano interagendo con i pulsanti che controllano l'LCD. Quello che potrebbe sembrare banale, in realtà, presenta notevoli problemi. Dal punto di vista concettuale, infatti, quello che questa funzione dovrebbe fare è nient'altro che leggere lo stato dei pin su cui sono collegati i bottoni e reagire di conseguenza a seconda che questi siano "LOW" (pulsante non premuto) o "HIGH" (pulsante premuto). Quello che invece si è costretti a fare per evitare di gestire pressioni inesistenti dovute tipicamente a rumore, è di introdurre un lasso temporale più o meno ampio e di andare a controllare nell'intero periodo di tempo se lo stato del pulsante rimane costante (pressione reale) o meno (rumore o pressione troppo breve). Trovare un giusto compromesso per il valore di questo lasso temporale risulta importante perché un valore troppo

piccolo non risolverebbe il problema, mentre uno troppo grande potrebbe causare all'utente, ignaro del tempo necessario affinché la pressione sia effettivamente gestita come tale, l'impressione di non funzionamento del bottone.

Per ognuno dei 4 pulsanti occorre eseguire la stessa procedura di controllo effettuata sui 4 diversi pin sui quali sono registrati i pulsanti.

```
1.  void  readButtons() {
2.
3.      int reading;
4.      int buttonEnterState = LOW;
5.      int buttonEscState = LOW;
6.      int buttonUpState = LOW;
7.      int buttonDownState = LOW;
8.
9.      //ENTER BUTTON
10.     reading = digitalRead(buttonPinEnter);
11.
12.     if (reading != lastButtonEnterState) {
13.         lastEnterDebounceTime = millis();
14.     }
15.
16.     if ((millis() - lastEnterDebounceTime) > debounceDelay) {
17.         buttonEnterState = reading;
18.         lastEnterDebounceTime = millis();
19.     }
20.
21.     lastButtonEnterState = reading;
22.
23.     //ESC BUTTON
24.     reading = digitalRead(buttonPinEsc);
25.
26.     if (reading != lastButtonEscState) {
27.         lastEscDebounceTime = millis();
28.     }
29.
30.     if ((millis() - lastEscDebounceTime) > debounceDelay) {
31.         buttonEscState = reading;
32.         lastEscDebounceTime = millis();
33.     }
34.
35.     lastButtonEscState = reading;
36.
37.     //DOWN BUTTON
38.     reading = digitalRead(buttonPinDown);
39.
40.     if (reading != lastButtonDownState) {
41.         lastDownDebounceTime = millis();
42.     }
43.
44.     if ((millis() - lastDownDebounceTime) > debounceDelay) {
45.         buttonDownState = reading;
46.         lastDownDebounceTime = millis();
47.     }
48.
49.     lastButtonDownState = reading;
50.
51.     //UP BUTTON
52.     reading = digitalRead(buttonPinUp);
53.
54.     if (reading != lastButtonUpState) {
```

```
55.         lastUpDebounceTime = millis();
56.     }
57.
58.     if ((millis() - lastUpDebounceTime) > debounceDelay) {
59.     buttonUpState = reading;
60.         lastUpDebounceTime = millis();;
61.     }
62.
63.     lastButtonUpState = reading;
64.
65.
66.     if (buttonEnterState == HIGH) {
67.         lastButtonPushed = buttonPinEnter;
68.
69.     }
70.     else if (buttonEscState == HIGH) {
71.         lastButtonPushed = buttonPinEsc;
72.
73.     }
74.     else if (buttonDownState == HIGH) {
75.         lastButtonPushed = buttonPinDown;
76.
77.     }
78.     else if (buttonUpState == HIGH) {
79.         lastButtonPushed = buttonPinUp;
80.
81.     }
82.     else {
83.         lastButtonPushed = 0;
84.     }
85. }
```

All'inizio della funzione viene istanziata un variabile per memorizzare il valore dello stato del pulsante premuto (3) e tutti gli stati dei 4 bottoni vengono messi a "LOW" (non premuto) (4-7).

Per ognuno dei quattro bottoni viene letto il valore dello stato del pin a cui sono collegati (10, 24, 38, 52) e se quest'ultimo è diverso da quello letto precedentemente (12, 26, 40, 54) viene effettuato il controllo sul tempo di permanenza del bottone in quello stato. Se lo stato del bottone che si sta gestendo rimane alto o basso per tutto il lasso di tempo impostato (per evitare problemi di rumore) (16, 30, 44, 58) allora si registra un effettivo cambiamento di stato(17, 31, 45, 59). Nel caso in cui il cambiamento di stato coincida con quello HIGH (66, 70, 74, 78), il bottone sarà registrato come ultimo premuto (67, 71, 75, 79) e, a seconda del bottone, la funzione di loop eseguirà particolari routine ad esso collegate. In caso contrario, nessun bottone risulta in pressione (83), il che lascia la funzione di Loop in una delle sue modalità normali (Mode 1 o 2).

### 5.6.2 NavigateMenus

La funzione NavigateMenus permette all'utente di navigare all'interno della gerarchia degli item che compongono il menù.

```
1.  void navigateMenus() {
2.
3.      MenuItem currentMenu = menu.getCurrent();
4.
5.      switch (lastButtonPushed) {
6.
7.      case buttonPinEnter:
8.          if (!(currentMenu.moveRight())) {
9.              menu.use();
10.         }
11.         else {
12.             menu.moveRight();
13.         }
14.         break;
15.
16.     case buttonPinEsc:
17.         menu.toRoot();
18.         break;
19.
20.     case buttonPinDown:
21.         menu.moveDown();
22.         break;
23.
24.     case buttonPinUp:
25.         menu.moveUp();
26.         break;
27.     }
28.
29.     lastButtonPushed = 0;
30. }
```

La prima operazione che deve essere effettuata per capire in che punto del menù ci troviamo è recuperare il nome dell'item corrente (3) che corrisponderà ad una precisa posizione all'interno della gerarchia. Ogni item può avere, a sua volta, un suo sottomenù o essere un elemento finale della gerarchia, una cosiddetta "foglia". A seconda della natura dell'item, l'interazione con i bottoni può avere dei seguiti piuttosto che altri:

- ENTER: se l'item selezionato è una foglia della gerarchia (8), vengono eseguite le operazioni associate ad essa (9), mentre, se l'item ha un sottomenù, quest'ultimo viene visualizzato (12).
- ESC: a prescindere dalla natura dell'item selezionato, riporta il controller alla root del menù (17). Una gestione più complessa del pulsante avrebbe previsto il ritorno al sottomenù di livello immediatamente più alto (in linea con quanto fa il pulsante ENTER al contrario). Questo modello però, seppur più corretto, esponeva al rischio di latenze

troppo alte a causa della poca capacità di calcolo del microcontrollore e della struttura sempre più complessa del menù necessaria per quell'approccio.

- DOWN-UP: permettono di cambiare la selezione dell'item di un sottomenù (qualora fosse presente una lista) portando il cursore di selezione all'item rispettivamente successivo (21) o precedente (22) fin tanto che quest'ultimo non sia l'ultimo della lista o il primo per i quali l'azione del pulsante si interrompe.

### 5.6.3 Calibration

Nel momento in cui viene acceso il controller, se l'utente lo ritiene necessario o se la memoria EEPROM non contiene dati validi, la funzione calibration viene richiamata per effettuare la calibrazione del sensore di pressione.
Essendo quest'ultimo di tipo analogico, l'output verrà letto dal microcontrollore su una scala da 0 a 1023. È necessario che il più piccolo valore letto e il più grande siano sempre contenuti in quel range. Con la calibrazione, andando a premere in qualsiasi posizione del sensore, impostiamo un lower bound che corrisponde alla forza minima con cui il sensore è stato premuto, al di sotto della quale il sensore non rileverà nessun valore valido. Lo stesso procedimento avviene anche per il valore massimo, 1023, che viene assegnato al valore di pressione massimo raggiunto durante il processo di calibrazione.

```
1.  void calibration() {
2.
3.      lcd.clear();
4.      lcd.setCursor(0, 0);
5.      lcd.print("    CALIBRATION     ");
6.      lcd.setCursor(0, 1);
7.      lcd.print("Please, push ribbon  ");
8.      lcd.setCursor(0, 2);
9.      lcd.print("controller with the ");
10.     lcd.setCursor(0, 3);
11.     lcd.print("max useful force!   ");
12.
13.     int startTime;
14.     int whileTime;
15.
16.     startTime = millis();
17.     whileTime = 0;
18.
19.     sensorForceMinValue = 1023;
20.     sensorForceMaxValue = 0;
21.
22.     while (whileTime < 5000) {
23.         sensorForceValue = analogRead(sensorForce);
24.
25.         if (sensorForceValue > sensorForceMaxValue) {
26.             sensorForceMaxValue = sensorForceValue;
27.         }
28.
```

```
29.         if (sensorForceValue < sensorForceMinValue) {
30.             sensorForceMinValue = sensorForceValue;
31.         }
32.
33.         whileTime = millis() - startTime;
34.     }
35.
36.     int sensorForceNewMinValue = sensorForceMinValue / 4;
37.     int sensorForceNewMaxValue = sensorForceMaxValue / 4;
38.
39.     EEPROM.write(addrSensorForceMin, sensorForceNewMinValue);
40.     EEPROM.write(addrSensorForceMax, sensorForceNewMaxValue);
41.
42.     lcd.clear();
43.     lcd.setCursor(0, 0);
44.     lcd.print("    CALIBRATION     ");
45.     lcd.setCursor(0, 1);
46.     lcd.print("Calibration ended!  ");
47.
48.     delay(1000);
49. }
```

La funzione di calibrazione ha inizio mostrando a display informazioni su come effettuarla correttamente (3-11). Il tempo a disposizione per calibrare il sensore di pressione è di 5 secondi, trascorsi i quali il sistema uscirà automaticamente dalla funzione ed eseguirà tutte le altre operazioni preliminari per l'avvio del sistema. Durante questo lasso di tempo, l'utente può effettuare anche più cicli di calibrazione semplicemente premendo più volte sul sensore. I valori rilevati verranno di volta in volta sovrascritti, motivo per cui l'ultima pressione esercitata sul sensore sarà quella ritenuta valida ai fini della calibrazione. Nel caso non si fosse soddisfatti del risultato ottenuto, tale funzione è richiamabile in qualsiasi momento durante l'esecuzione del controller attraverso l'apposita voce di menù.

Per tener traccia del tempo trascorso dall'inizio della fase di calibrazione vengono istanziate due variabili locali (13-14) e per essere certi che l'operazione di calibrazione accetti i nuovi parametri immessi dall'utente, i due parametri iniziali di minimo e massimo valore ammissibili sono invertiti (19-20). Nel corso dei cinque secondi a disposizione, le letture successive di valori di pressione vanno via via definendo il range esatto voluto dall'utente (22-34). Al termine del tempo previsto, i nuovi valori sono divisi per 4 (36-37) (valore sempre definibile con 1 Byte) e salvati in apposite celle della memoria EEPROM (39-40).

Infine, un messaggio mostrato a display per un secondo (48) informa l'utente della corretta riuscita dell'operazione di calibrazione (42-46).

### 5.6.4 SetWireless

La possibilità di impostare la modalità di comunicazione wireless tra il controller e il sistema preposto alla sintesi è affidata alla funzione SetWireless.
Tale funzione, infatti, svolge due attività: nella prima parte cerca di connettere il dispositivo alla rete LAN il cui SSID e Passphase sono memorizzati nelle variabili di sistema, mentre nella seconda, a connessione avvenuta, prende parte ad una sessione midi virtuale fornendo all'utente tutte le informazioni necessarie.

```
1.  void SetWireless() {
2.
3.      lcd.clear();
4.      lcd.setCursor(0, 1);
5.      lcd.print("    Wi-Fi SETTING   ");
6.      lcd.setCursor(0, 3);
7.      lcd.print("   Please, wait...  ");
8.      delay(2000);
9.
10.     lcd.clear();
11.     lcd.setCursor(0, 0);
12.     lcd.print("   Connecting to:   ");
13.     lcd.setCursor(0, 1);
14.     lcd.print(ssid);
15.
16.
17.     WiFi.begin(ssid, password);
18.
19.     int primoLoop = 0;
20.     int indiceColonna = 7;
21.
22.     while (WiFi.status() != WL_CONNECTED) {
23.
24.         delay(500);
25.
26.         if (primoLoop == 0) {
27.             lcd.setCursor(0, 3);
28.             lcd.print("Waiting");
29.             primoLoop = 1;
30.         }
31.         else {
32.
33.             if (indiceColonna == 20) {
34.                 lcd.setCursor(0, 3);
35.                 lcd.print("Waiting             ");
36.                 indiceColonna = 7;
37.             }
38.
39.             else {
40.                 lcd.setCursor(indiceColonna, 3);
41.                 lcd.print(".");
42.                 indiceColonna++;
43.             }
44.         }
45.     }
46.
47.     lcd.clear();
48.     lcd.setCursor(0, 1);
49.     lcd.print("  Wi-Fi Connected!  ");
```

```
50.     lcd.setCursor(0, 3);
51.     lcd.print("IP:");
52.     lcd.setCursor(4, 3);
53.     lcd.print(WiFi.localIP());
54.     delay(3000);
55.
56.     lcd.clear();
57.     lcd.setCursor(0, 0);
58.     lcd.print("OK! Now enable a");
59.     lcd.setCursor(0, 2);
60.     lcd.print("PC rtpMIDI session");
61.     delay(3000);
62.
63.     lcd.clear();
64.     lcd.setCursor(0, 0);
65.     lcd.print("Add device Host/Port");
66.     lcd.setCursor(0, 2);
67.     lcd.print(WiFi.localIP());
68.     lcd.setCursor(14, 2);
69.     lcd.print(":5004");
70.     delay(3000);
71.
72.     lcd.clear();
73.     lcd.setCursor(0, 0);
74.     lcd.print("    Finally    ");
75.     lcd.setCursor(0, 2);
76.     lcd.print("Press Connect button");
77.
78.     AppleMIDI.begin("test");
79.
80.     AppleMIDI.OnConnected(OnAppleMidiConnected);
81.     AppleMIDI.OnDisconnected(OnAppleMidiDisconnected);
82.
83. }
```

Nella prima parte della funzione, opportuni messaggi mostrati sull'LCD informano l'utente che il processo di connessione alla rete è iniziato correttamente insieme a tutte le informazioni relative alla LAN alla quale il dispositivo tenta di connettersi (3-14). Attraverso la libreria WiFi.h viene effettuato un tentativo di connessione alla rete fornendo SSID e PASSWORD (17). Fin tanto che il dispositivo non si connette, vengono stampati messaggi di attesa sull'LCD (22-45). Un messaggio di avvenuta connessione (47-54) precede le informazioni necessarie per avviare una sessione midi virtuale e connettere il controller (56-76) che, a questo punto, sarà caratterizzato da un proprio indirizzo IP assegnato dal DHCP server in fase di connessione. La sessione MIDI alla quale prendono parte sia il controller che il dispositivo di sintesi viene creata di default sulla porta 5004 (78). In realtà il protocollo rtp-midi esige che due porte siano dedicate per la comunicazione: una per l'invio e la ricezione dei dati e un'altra per la gestione dei comandi. Per questo motivo, nel momento in cui si riserva una determinata porta per la sessione midi, in modo celato, una seconda parte, successiva a quella scelta dall'utente, viene designata per il trasferimento dei comandi.

Ad avvenuto avvio della sessione MIDI, due semplici funzioni attendono la connessione e/o disconnessione di dispositivi esterni alla stessa (80-81), stampando opportuni messaggi di informazione.

### 5.6.5 SetWifiCredentials

Potendo connettersi alla rete LAN, il controller necessitava di un modo per poter, in qualsiasi momento e senza doverlo collegare ad un PC per accedere e modificare il codice sorgente, modificare i parametri di connessione alla rete.
Il modo più intuitivo sarebbe quello di utilizzare lo stesso LCD che il controller monta per interfacciarsi con l'utente e permettergli tali modifiche. Sebbene possibile, tale soluzione risultava abbastanza complessa e soggetta a molti problemi.
Uno dei più importanti è legato alla tecnologia dell'LCD e al metodo di immissione dei caratteri. L'LCD infatti non supporta il touch. In quel caso creare una semplice interfaccia con tutti i caratteri ammessi sarebbe stato semplice e intuitivo. Trattandosi però di un dispositivo analogico, necessita di un meccanismo sequenziale per l'immissione dei caratteri che sarebbero stati inseriti uno per volta scorrendo un vettore di tutti i caratteri possibili. Estremamente scomodo e lungo, soprattutto in caso di errore.
Il modo quindi che è stato ideato per la modifica di SSID e Password utilizza l'antenna bluetooth HC-06 che il sensore monta appositamente per lo scopo con la quale è possibile interfacciarsi con qualsiasi terminale che supporti quel protocollo di comunicazione.
Anche se apparentemente la soluzione sembra scomoda e controintuitiva, si è dimostrata molto versatile e di facile realizzazione, sfruttando il fatto che qualsiasi dispositivo portatile (smartphone, tablet…) o fisso supporta la tecnologia bluetooth e il terminale necessario per interfacciarsi con il controller è presente in modo nativo su molti dispositivi o, comunque, facilmente installabile come applicazione free.
La funzione che gestisce la comunicazione accetta un parametro in funzione di cosa l'utente abbia intenzione di modificare: solo SSID, solo PASSWORD o entrambi.
Dal momento che il codice dei primi due casi nasce dalla divisione di quello che gestisce il terzo, si è scelto di trattare solo questo ultimo caso.

```
1.  void SetWifiCredential(int value) {
2.
3.      String message;
4.      int primoLoop = 0;
5.      int indiceColonna = 7;
6.
```

```
7.      switch (value) {
8.
9.      case 2:
10.
11.         lcd.clear();
12.         lcd.setCursor(0, 0);
13.         lcd.print("SSID and Password");
14.         lcd.setCursor(0, 1);
15.         lcd.print("setting procedure");
16.         delay(4000);
17.
18.         lcd.clear();
19.         lcd.setCursor(0, 0);
20.         lcd.print("Connect bluetooth");
21.         lcd.setCursor(0, 1);
22.         lcd.print("editor and follow ");
23.         lcd.setCursor(0, 2);
24.         lcd.print("istructions on lcd");
25.         delay(4000);
26.
27.         lcd.clear();
28.         lcd.setCursor(0, 0);
29.         lcd.print("Send SSID to:");
30.         lcd.setCursor(0, 1);
31.         lcd.print("BT device: HC-06");
32.         lcd.setCursor(0, 2);
33.         lcd.print("Pin: 1234");
34.
35.         while (!bluetooth.available()) {
36.
37.             delay(500);
38.
39.             if (primoLoop == 0) {
40.                 lcd.setCursor(0, 3);
41.                 lcd.print("Waiting");
42.                 primoLoop = 1;
43.             }
44.             else {
45.                 if (indiceColonna == 20) {
46.                     lcd.setCursor(0, 3);
47.                     lcd.print("Waiting             ");
48.                     indiceColonna = 7;
49.                 }
50.                 else {
51.                     lcd.setCursor(indiceColonna, 3);
52.                     lcd.print(".");
53.                     indiceColonna++;
54.                 }
55.             }
56.
57.         }
58.
59.         while (bluetooth.available()) {
60.             message += char(bluetooth.read());
61.         }
62.         id = 0;
63.
64.         SetParameters(id, message);
65.
66.         message = "";
67.
68.         lcd.clear();
69.         lcd.setCursor(0, 0);
70.         lcd.print("SSID setted!");
71.         delay(3000);
```

```
72.
73.         lcd.clear();
74.         lcd.setCursor(0, 0);
75.         lcd.print("Send Password to:");
76.         lcd.setCursor(0, 1);
77.         lcd.print("BT device: HC-06");
78.         lcd.setCursor(0, 2);
79.         lcd.print("Pin: 1234");
80.
81.         primoLoop = 0;
82.         indiceColonna = 7;
83.
84.         while (!bluetooth.available()) {
85.
86.             delay(500);
87.
88.             if (primoLoop == 0) {
89.                 lcd.setCursor(0, 3);
90.                 lcd.print("Waiting");
91.                 primoLoop = 1;
92.             }
93.             else {
94.                 if (indiceColonna == 20) {
95.                     lcd.setCursor(0, 3);
96.                     lcd.print("Waiting             ");
97.                     indiceColonna = 7;
98.                 }
99.                 else {
100.                            lcd.setCursor(indiceColonna, 3);
101.                            lcd.print(".");
102.                            indiceColonna++;
103.                        }
104.                    }
105.
106.                }
107.
108.                while (bluetooth.available()) {
109.                    message += char(bluetooth.read());
110.                }
111.
112.                id = 1;
113.
114.                SetParameters(id, message);
115.
116.                lcd.clear();
117.                lcd.setCursor(0, 0);
118.                lcd.print("Password setted!");
119.
120.                delay(3000);
121.
122.                message = "";
123.
124.                lcd.clear();
125.                lcd.setCursor(0, 0);
126.                lcd.print("All setted!");
127.                lcd.setCursor(0, 1);
128.                lcd.print("Now connect Wi-Fi");
129.                lcd.setCursor(0, 2);
130.                lcd.print("in menu WiFi setting");
131.
132.                delay(3000);
133.
134.                menu.toRoot();
135.
136.                break;
```

```
137.           }
138.       }
```

Nella prima parte della funzione, a display vengono fornite all'utente tutte le informazioni necessarie per effettuare la connessione del terminale al controller, compreso il nome e la password per la connessione (11-33).
Un messaggio successivo informa che il dispositivo è pronto a ricevere la stringa di SSID e in attesa che un terminale si connetta e invii la nuova stringa (35-57).
Non appena l'utente invia il nuovo SSID (59), il controller legge la stringa un carattere per volta dall'interfaccia seriale (60) e tramite un'apposita funzione di supporto la salva nella relativa variabile (64) e mostra a display un messaggio di avvenuta modifica del parametro (68-71).
A questo punto la procedura si ripete allo stesso modo per il setting della nuova password (73-122) e quando anche questa è conclusa come la precedente, un messaggio di conferma informa l'utente dell'avvenuta modifica di entrambi i parametri di connessione (124-130) e il controller viene riportato alla root del menù (134) dalla quale, volendo, l'utente può avviare la procedura di connessione alla rete LAN con i nuovi parametri appena impostati.

# Capitolo 6
# ASSEMBLAGGIO DEL CONTROLLER

La realizzazione finale del controller ha previsto la modifica del progetto iniziale al fine di adattarlo alla struttura appositamente disegnata e realizzata per lo scopo.
Sebbene molti componenti siano rimasti praticamente gli stessi rispetto al modello iniziale, molti altri sono stati sostituiti con altri più adatti.
In particolare, sono stati sostituiti:

- Pulsanti Enter, Down, Up, Esc di controllo del display, dal momento che quelli utilizzati per il modello iniziale non erano adatti per essere applicati ad un case. Sono stati sostituiti con altrettanti bottoni che hanno le stesse caratteristiche elettriche dei primi ma che presentano anche un supporto a vite per il montaggio su case;

**Figura 6.1 - Pulsanti non persistenti**

- Batteria da 9V, utilizzata in fase di test, con un power bank da 15000 mA a doppia uscita USB per l'alimentazione del controller e un ingresso micro USB per la ricarica dello stesso.


**Figura 6.2 - Power bank**

La power bank fornisce in uscita 5V (compatibile quindi con le specifiche di alimentazione di Arduino) e due differenti valori di amperaggio: 2.4A e 1.5A a seconda se vengono utilizzate rispettivamente una sola o entrambe le uscite. Nel controller, quindi, sarebbero state possibili due modalità di alimentazione: la prima prevede l'utilizzo di una sola uscita USB utilizzata per alimentare il solo Arduino, utilizzato a sua volta per alimentare tutti i sensori e attuatori ad esso connessi, mentre la seconda prevede l'utilizzo di entrambe le uscite delle quali una è collegata all'Arduino e la seconda ad alcuni slave del controller (LCD, HC-06), in uno schema a doppia alimentazione e massa comune. Tra le due è stata scelta la seconda ed entrambe le uscite vengono utilizzate per alimentare il sistema. Questa scelta è dettata dal fatto che potrebbe essere controproducente caricare l'Arduino dei pesi dei carichi elettrici di tutti i sensori che esso gestisce. Questo potrebbe causare surriscaldamenti e, quindi, malfunzionamenti o, peggio, la rottura del microcontrollore stesso, nonostante la somma di tutti i carichi sia inferiore al limite degli 800 mA di corrente che l'Arduino può sostenere.

- Cavo di alimentazione Arduino, non più necessario, con un cavo micro USB con interfacce Male/Famale per la ricarica della power bank.

**Figura 6.3 - Cavo Male/Female Micro USB**

Avendo utilizzato un power bank all'interno del controller, si rendeva necessaria la ricarica dello stesso in un modo comodo e che non necessitasse di aprire il controller e accedere alla circuiteria. Per questo motivo è stato creato un "prolungamento" della porta di alimentazione della power bank utilizzando un semplice cavo con interfacce micro USB miste: l'interfaccia di OUT è stata collegata alla power bank stessa mentre quella di IN è stata opportunamente fissata al case mostrandosi sulla faccia laterale esterna dello stesso.

- Switch On/Off, sostituito con uno appropriato all'installazione su case. È stato utilizzano uno switch a tre poli (VCC, COLLECTOR, GRD) e due stati persistenti (ON, OFF).

**Figura 6.4 - Switch On/Off**

In particolare, le connessioni realizzate sono: VCC–5V power bank, COLLECTOR-Arduino (Vin), GRD- GRD power bank e Arduino (GRD).

Tutti i componenti sono stati fissati ad un case appositamente creato per lo scopo. Il cablaggio è stato realizzato con cavo multicolore da 0.35 e tutte le terminazioni, eccetto quelle collegate all'Arduino, sono state saldate con punti di stagno e isolate con guaine termorestringenti.

Per quanto riguarda il case realizzato per contenere ed esporre la sensoristica del controller, è stato pensato e realizzato in AutoCAD, che ha fornito l'ambiente per creare un opportuno file quotato necessario per la sagomatura e il taglio del materiale scelto per il case. Il materiale utilizzato è plexiglass trasparente di 5mm di spessore per tutte le parti del case. Grazie all'utilizzo del taglio laser, sono state realizzati tutti gli alloggi necessari per i componenti del controller esposti.

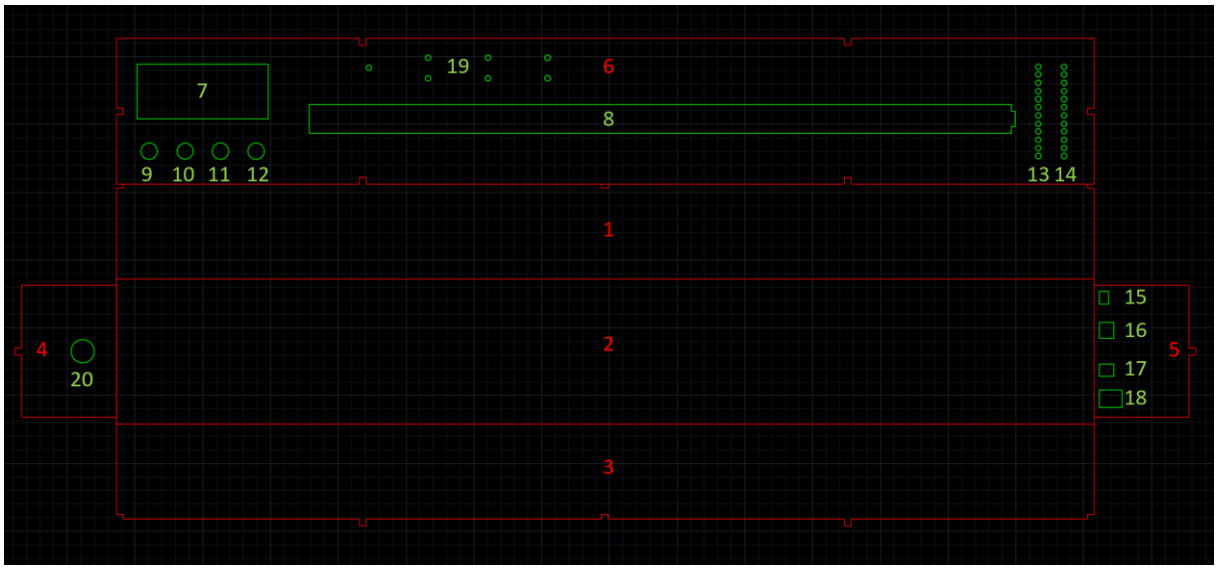


**Figura 6.5 - Schema di taglio del case**

Nello schema sono evidenziate in rosso le linee di taglio esterne mentre in verde quelle interne. In particolare:

- 1-3: lati lunghi del case;
- 4-5: lati corti;
- 2: base
- 6: parte superiore del case;
- 7: alloggiamento LCD;
- 8: alloggiamento sensore di pressione e posizione;
- 9, 10, 11, 12: alloggiamenti pulsanti Enter, Down, Up, Esc;
- 13, 14: alloggiamenti per i led di controllo del volume e pitch;
- 15: alloggiamento per female micro USB;
- 16, 17: alloggiamenti per connettori dell'Arduino mega;
- 18: alloggiamento per switch On/Off;
- 19: alloggiamenti per led di controllo dello stato del sistema;
- 20: alloggiamento per female DIN 5 poli.

Il case è stato assemblato tramite incollaggio nelle parti inferiori mentre la parte superiore è vincolata alla parte inferiore tramite appositi denti di ritenzione e l'utilizzo di magneti dello stesso spessore del plexiglass posizionate sul bordo superiore e inferiore rispettivamente della parte inferiore e superiore del case.

# Capitolo 7
# CONCLUSIONI

Il MIDI Controller nasce dalla volontà di fornire un vero e proprio strumento musicale a persone diversamente abili, in collaborazione con il personale del progetto ADO che si propone di utilizzare tali supporti nell'ambito di sessioni di musicoterapia.
Per far ciò, il controller utilizza un'interfaccia accessibile costituita da due sensori tattili per il rilevamento di pressione e posizione della mano dell'utente. L'utilizzo di software esterni di sintesi permette infine di rimappare i dati forniti dal controller in innumerevoli modi, permettendo all'utente di cimentarsi nel suono dei più comuni strumenti musicali, altrimenti inaccessibili, o di controllarne le caratteristiche sonore nell'ambito di un'orchestra.
Il controller, quindi, si configura già come un valido strumento per essere utilizzato dal gruppo del progetto ADO. Il suo sviluppo ha previsto alcuni cambiamenti fondamentali rispetto al modello originario di base molto più generico, soprattutto per adattarsi a quelle che sono le esigenze dell'orchestra in cui sarebbe stato utilizzato.
In primo luogo è stato introdotta la possibilità di utilizzare il controller non soltanto con l'uso delle dita ma anche con l'utilizzo di qualcosa di più esteso come una mano o un arto in generale. Questo era un requisito fondamentale per il controller per essere ritenuto "accessibile".
È stata inoltre implementata la possibilità di trasferire il MIDI generato dal controller non soltanto via cavo ma anche sfruttando una rete wireless al quale il controller è in grado di collegarsi. Questa caratteristica, sebbene sia da migliorare, viene incontro alle esigenze dell'orchestra di ridurre drasticamente il numero di cavi necessario per collegare tutti gli strumenti dell'orchestra. Insieme alla possibilità di collegare il controller ad una rete Wi-Fi, è stata fornita anche una soluzione pratica alla modifica dei parametri di connessione, per offrire la possibilità di cambio rete anche durante l'esecuzone del controller.

## 7.1 **Sviluppi futuri**

Il controller, sebbene sia già uno stumento funzionante e utilizzabile a tutti gli effetti, offre la possibilità di notevoli sviluppi e miglioramenti.

Uno dei principali spunti di miglioramento del controller è offerto dalla scelta dei sensori utilizzati. Il sensore di pressione e quello di posizione utilizzati per la realizzazione del controller non sono esattamente ciò che ci si aspetterebbe su uno strumento musicale. I sensori infatti vengono più comunemente utilizzati per applicazioni che richiedono una sensibilità e una precisione più grossolana di quelle richieste da un vero controller MIDI. Per questo motivo, il controller non risulta sempre infallibile nella detection dei tocchi e risulta particolarmente instabile quando il tocco è molto lieve. Una soluzione parziale a questo problema viene offerta andando a modificare le resistenze di controllo del circuito elettrico dei sensori e il software che li gestisce. In questo modo è possibile "stabilizzare" l'output dei due sensori, rendendoli adatti all'applicazione di gestione del MIDI. Il risultato ottenuto però può essere notevolmente migliorato o del tutto eliminato utilizzando sensori più affini a questo genere di applicazioni, supportati, anche, dal software sviluppato per la loro gestione.

Altro problema che merita attenzione è quello dell'utilizzo della rete per l'invio dei messaggi MIDI in luogo del più comune cavo MIDI nell'ambito di una connessione wired. Il controller, infatti, presenta dei problemi di comunicazione con il sistema preposto alla sintesi software, nonostante la connessione vada a buon fine. Tale problema è dovuto ad un bug software introdotto con il rilascio della versione R3 della shield Wi-Fi che il controller monta per la gestione della comunicazione wireless. Tale problema, infatti, non si riscontra con l'utilizzo della versione R2 della stessa shield, con la quale il controller funzioni come previsto.

Notevole attenzione, sempre nell'ambito della comunicazione wireless, va posta nella latenza della comunicazione. Ad ora il controller riesce senza problemi ad essere sotto la soglia dei 150 ms di latenza previsti in una comunicazione real time che si rispetti, ma in qualche occasione tale limite viene superato con evidenti ripercussioni sul risultato della sintesi. Un obiettivo potrebbe essere quello di migliorare la gestione e l'invio dei messaggi MIDI per abbassare la latenza generata, così che, anche in situazioni di problemi sulla rete, il suo valore non superi mai i 150 ms.

Per quanto riguarda la parte di controllo, un notevole sviluppo del controller potrebbe prevedere l'introduzione di un display a colori touch così da permettere l'immissione dei dati desidarati direttamente da interfaccia, eliminando così la necessità di pulsanti fisici e del terminale bluetooth come interfaccia di inserimento.

Ciò detto, il MIDI controller si configura già come uno strumento musicale finito in grado di comparire all'interno dell'orchestra del progetto ADO.

# *BIBLIOGRAFIA E SITOGRAFIA*


The MIDI Manufacturers Association, "*The Complete MIDI 1.0. Detailed Specification. Incorporating all Recommended Practices*", document version 96.1, third edition. https://www.midi.org/specifications

V. Lombardo, "*Il MIDI: Musical Instrument Digital Interface*"
http://www.di.unito.it/~vincenzo/TechInfoSuoMus/SlidePDF/midi.pdf

S. Relandini, "*Il Protocollo MIDI*".
http://www.istitutobellini.cl.it/file.php/1/Master/Informatica_Relandini_2012/protocollo_MIDI.pdf

*MIDI Network Driver Protocol, Dicember* 2016.
https://developer.apple.com/library/content/documentation/Audio/Conceptual/MIDINetworkDriverProtocol/MIDI/MIDI.html

L.A. Ludovico, "*Network Musical Performance: RTP-MIDI*".
http://www.ludovico.net/download/materiale_didattico/midi/15_RTP_MIDI_1.ppt
http://www.ludovico.net/download/materiale_didattico/midi/16_RTP_MIDI_2.ppt

J. Lazzaro, J. Wawrzynek, Network Working Group, "*RFC 4695 & 6295: RTP Payload Format for MIDI*", November 2006.
https://tools.ietf.org/html/rfc4695

J. Lazzaro, J. Wawrzynek, Network Working Group, "*RFC 4696: An Implementation Guide for RTP MIDI*", November 2006.
https://datatracker.ietf.org/doc/rfc4696

S. Cheshire (Apple Computer), B. Aboba (Microsoft Corporation), E. Guttman (Sun Microsystem), Network Working Group, "*RFC 3927: Dynamic Configuration of IPv4 Link-Local Addresses*", May 2005.
https://tools.ietf.org/html/rfc3927

T. Erichsen, "*rtpMIDI Tutorial*"
http://www.tobias-erichsen.de/software/rtpmidi/rtpmidi-tutorial.html

Giuseppe Di Cillo, *"Gestire Menu e Display LCD"* e "*Costruire un MIDI Ribbon Controller"*
http://www.coagula.org/albedo/category/maker/

*Arduino Language Reference*, 2017
https://www.arduino.cc/reference/en/

*Interlink Eletronic FSR ® 400 Series Data Sheet – Specifiche elettriche e costruttive*
http://interlinkelectronics.com/datasheets/Datasheet_FSR.pdf
http://interlinkelectronics.com/FSR408.php

*Spectra Symbol SoftPot Sensor Data Sheet – Specifiche elettriche e costruttive*
http://www.spectrasymbol.com/wp-content/uploads/2016/12/SOFTPOT-DATA-SHEET-Rev-F3.pdf
http://www.spectrasymbol.com/product/softpot/

# *RINGRAZIAMENTI*

*Al termine di questo lavoro di tesi e del percorso universitario di laurea magistrale è doveroso il ringraziamento a diverse persone che mi hanno accompagnato, sostenuto e incitato in questo cammino e hanno contribuito a questo risultato.*

*In primis, i miei genitori. A loro devo molto. Nonostante le difficoltà e la distanza che quasi sempre ci separava, sono stati sempre per me un esempio, un modello da imitare e hanno saputo darmi in ogni occasione il giusto stimolo per andare avanti.*

*Alla mia ragazza Chiara. A lei va il merito di essermi stata vicina sempre, sostenendomi e incoraggiandomi in ogni situazione, mettendo sempre me prima di tutto.*

*Un sentito ringraziamento va al mio relatore, Prof. Antonio Servetti, che mi ha dato la possibilità di prendere parte a questo lavoro e mi ha aiutato a svilupparlo, venendo sempre incontro a tutte le mie esigenze e difficoltà.*

*Un grazie a tutti i miei colleghi di corso, per essere stati una delle poche fonti di "distrazione" tra i tanti impegni universitari e militari. Con loro tutto sembra meno oneroso.*

*Infine, ma non ultimo, volevo ringraziare tutti gli amici del progetto ADO perché mi hanno dato la possibilità di conoscere un mondo che prima ignoravo, di capire quanto poco ci vuole per donare un sorriso a persone alle quali sembra mancare molto, a volte tutto, ma che in realtà hanno molto da insegnare e dalle quali tutti possiamo imparare. Lavorare con loro e per loro mi ha dato la possibilità di apprezzare davvero quello che ho, evidenziando quanti pochi siano per me i motivi per lamentarmi o essere arrabbiato.*
*Da loro ho imparato davvero molto ed è a loro che voglio dedicare questo lavoro.*