UNIVERSITÀ DEGLI STUDI DI MILANO

FACOLTÀ DI SCIENZE E TECNOLOGIE

DIPARTIMENTO DI INFORMATICA
Corso di Laurea in: INFORMATICA MUSICALE

Generazione di messaggi MIDI per batteria tramite Arduino

Relatore:
Prof. Luca A. Ludovico

Correlatore:
Prof. Adriano Baratè

Studente:
Gioele Cortese
774065

Anno Accademico 2012/2013

Sommario

INTRODUZIONE

L’informatica è uno degli ambiti in cui l’innovazione è così repentina che a pochi anni di distanza una tecnologia considerata innovativa diventa subito obsoleta [b9]. Nel caso dell’informatica applicata alla musica, vi è però un’eccezione significativa: lo standard MIDI. Nato negli anni ottanta del Novecento in seguito ad un’iniziativa comune dei maggiori produttori di hardware a scopo musicale dell’epoca, tuttora, nonostante l’avanzamento tecnologico sia stato enorme, è uno dei pochi sistemi di comunicazione tra apparecchiature di diversa fabbricazione pienamente funzionante. La sua “imposizione” da parte di coloro i quali ne erano, e sono ancora, i principali fruitori ne ha decretato la longevità e la sua centralità nel campo della comunicazione tra le apparecchiature informatiche per la musica.

Per dimostrare l’attualità dello standard MIDI in questo elaborato si mostreranno la sua compatibilità e il suo utilizzo mediante la piattaforma di sviluppo hardware Arduino. Essa è nata dalla precisa intenzione di voler fornire uno strumento facilmente utilizzabile a coloro i quali avessero bisogno di uno strumento semplice per implementare un qualche tipo di apparecchiatura elettronica, senza per forza avere le conoscenze per costruirla e programmarla completamente. La fortuna di Arduino è stata la sua natura completamente votata all’Open Source, grazie alla quale il progetto iniziale si è diffuso e migliorato, grazie ad una folta schiera di appassionati, esperti o semplici hobbisti, che hanno permesso al team di sviluppo di portare la loro creatura a un livello internazionale di popolarità.

Dopo aver analizzato le caratteristiche di entrambe le tecnologie, sarà mostrerà come queste si integrino per realizzare uno strumento hardware in grado di generare messaggi MIDI per batteria, nonostante la differente epoca in cui sono nate e nonostante gli ambienti che le hanno generate siano quasi diametralmente opposti.

CAPITOLO I
Lo Standard MIDI

1.1 Introduzione

Il MIDI (Musical Instruments Digital Interface) è uno standard che definisce il protocollo, l'interfaccia grafica e i connettori per la comunicazione tra strumenti musicali elettronici e digitali. La stesura delle specifiche venne realizzata dagli stessi produttori di strumenti riuniti nella MIDI Manufacturers Association (MMA) a Los Angeles e parallelamente dalla sezione specifica della Association of Musical Electronics Industry (AMEI) a Tokio. Queste due realtà lavorano tuttora per espandere le possibilità di connessione e le potenzialità del protocollo.

1.2 Brevi cenni storici

L'evoluzione della strumentazione musicale elettronica conobbe un intenso sviluppo nella seconda metà degli anni Settanta. I musicisti che però possedevano più unità erano alle prese con una serie di problematiche di comunicazione, in quanto questa avveniva attraverso connessioni elettriche e ogni casa di produzione tarava i propri strumenti su differenti valori. Inoltre i parametri delle singole unità erano comandati anch'essi da circuiteria analogica e a risentirne era soprattutto l'intonazione, resa instabile dal riscaldamento delle parti elettriche. Questa situazione rendeva, di fatto, l'uso della strumentazione particolarmente difficoltosa nelle esibizioni dal vivo [b1].

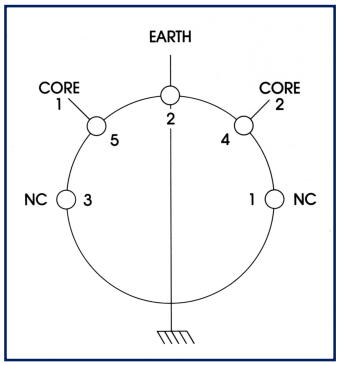
Verso la fine del decennio il mercato di tali apparecchiature si era consolidato e le dimensioni dei singoli strumenti divenivano sempre più ridotte, grazie soprattutto alle nuove tecnologie digitali, le quali risolvevano molte, ma non tutte, le questioni derivanti dall'uso di tecnologie analogiche. Fu così che gli ingegneri delle aziende produttrici di strumenti digitali presero in considerazione l'idea di creare uno standard che permettesse a varie unità, anche profondamente diverse, di comunicare tra loro. Nel 1983 vennero rese pubbliche le specifiche MIDI 1.0 e in seguito venne istituito il MIDI International User Group, diventato poi l'International MIDI Association (IMA), tutt'oggi esistente, il cui compito principale era di fornire ai musicisti e i produttori i migliori strumenti per lo sfruttamento del nuovo standard.

L'introduzione del MIDI rappresentò una svolta nel modo di suonare e successivamente anche nel modo di comporre musica. Infatti, l'uso del protocollo MIDI da parte di sequencer e campionatori divenne prassi negli studi di registrazione, soprattutto nella fase di pre-produzione, rendendo così non necessario l'utilizzo dei turnisti fino alle registrazioni vere e proprie. Inoltre le ridotte dimensioni degli Standard MIDI File, introdotti nel 1991, i file che contengono le informazioni relative a un brano, consentivano una maggiore condivisione dei contenuti stessi, soprattutto in relazione alle tecnologie esistenti all'epoca della sua

introduzione. La nascita e la diffusione dei personal computer aprì una nuova frontiera al MIDI, che si impose come strumento di comunicazione tra i software musicali, Digital Audio Workstation, software di partitura, Virtual Studio Technology, e gli hardware esterni, ma anche tra i software stessi. Al giorno d'oggi la nuova frontiera dello standard sono i dispositivi portable e touchscreen, sempre più diffusi e con potenzialità considerevoli in ambito di trasportabilità e interconnessione tra di essi e le tradizionali tecnologie MIDI[s1?].

1.3 Caratteristiche hardware dello standard [b3]

La trasmissione di dati tramite messaggi MIDI avviene in modo seriale e alla frequenza di 31,25 Kbaud. I connettori sono DIN a cinque poli, dei quali però vengono usati solamente i tre contatti centrali. In tempi meno recenti era possibile collegare un dispositivo MIDI a un Personal Computer attraverso una porta Joystick a quindici pin, chiamata anche Game port, mentre attualmente è ormai comune trovare dispositivi che comunicano tra loro attraverso porte USB, FireWire o addirittura su canali senza fili. Una porta MIDI è comunemente composta di tre prese, chiamate MIDI IN, MIDI OUT e MIDI THRU. La prima è la connessione da cui vengono ricevuti i dati, la seconda è destinata alla trasmissione in uscita mentre la terza serve per replicare in uscita i dati in entrata senza elaborarli. Esistono tuttavia porte composte soltanto dai primi due jack, in cui il jack OUT funge anche da THRU, oppure con una sola connessione, a seconda della funzione specifica della macchina (per esempio: una Master Keyboard può svolgere le funzioni per le quali è stata costruita presentando una porta MIDI composta da un solo jack OUT) o ancora con un numero di porte variabili.



Connettore DIN a 5 Poli

1.4 I messaggi MIDI [b3]

La comunicazione a livello grammaticale consiste in una serie di messaggi, composti da un numero variabile di byte. Il primo byte di ogni messaggio è denominato Status byte, o byte di stato, ed è caratterizzato dal bit più significativo posto a 1. I byte seguenti, che possono essere non presenti o in numero massimo di due, sono chiamati Data byte, o byte di dati. Il loro bit più significativo è posto a 0 .

I messaggi possono essere classificati in due categorie: Channel Message e System Message.

1.4.1 Channel Message

I Channel Message sono indirizzati verso uno specifico canale tra i sedici messi a disposizione dal canale di comunicazione MIDI. Il loro byte di stato è caratterizzato, oltre dal valore 1 come bit più significativo, da tre bit che ne descrivono la funzione e da quattro bit dedicati al canale. Questa categoria di messaggi può essere divisa a sua volta in due sotto categorie: i messaggi Channel Voice e i messaggi Channel Mode. I messaggi Channel Voice [1]sono dedicati alla descrizione della performance, quindi all'esecuzione di note e al loro spegnimento, del cambio del programma sonoro, dell'uso del comando per il pitch bend oppure del cambio di uno dei parametri gestiti dal protocollo e dallo strumento collegato. Il loro nome richiama che essi descrivono, in qualche modo, la "voce" del canale cui fanno riferimento. La struttura di questi messaggi è composta dallo Status byte, che specifica il tipo di messaggio e il canale al quale è diretto, seguito da due Data byte. Per esempio per un messaggio di Note On, il primo Data Byte rappresenta il pitch, mentre il secondo descrive la velocity, ovvero l'intensità della nota. Per un messaggio di Pitch Bend invece i due Data byte servono a tracciare l'andamento della variazione dell'intonazione con un metodo di combinazione tra MSB (Most Significant Byte) e LSB (Least Significant Byte), funzione implementata anche per i messaggi di Program Change. Un caso a specifico è costituito da alcuni messaggi tra i Control Change. Questi messaggi presentano uno Status byte comune, mentre il primo Data byte specifica quale parametro deve essere cambiato e il secondo a quale valore impostarlo[2]. Alcuni di questi parametri però richiedono una maggiore granularità per descrivere il valore assegnatogli con maggiore accuratezza. La specifica MIDI ricorre dunque a coppie di messaggi Control Change, i secondi Data byte dei quali seguono la logica del MSB+LSB impiegata per i tipi di messaggi descritti in precedenza. Si può prendere come esempio il comando di Bank Select. Con un singolo messaggio si potrebbero selezionare soltanto centoventotto banchi di memoria, limitando perciò la dimensione della memoria stessa degli strumenti. Utilizzando due messaggi consecutivi si riesce a espandere la capacità di selezione a più di sedicimila valori.

I messaggi Channel Mode[3] Control Change sono stati pensati per descrivere e modificare le modalità con cui il canale a cui fanno riferimento ricevono e trasmettono messaggi. Come struttura questi messaggi sono identici ai Control Change, ma sono definiti da una particolare

[1] La lista completa dei messaggi Channel Voice è presente in appendice, nella tabella 1.
[2] La lista completa dei valori dei messaggi Control Change è presente in appendice, nella tabella 2.1 e 2.2.
[3] Per la lista completa dei messaggi Channel Mode Control Change consultare la tabella 3 presente in appendice.

numerazione del primo Data Byte, compreso tra i valori 120 e 127. Tra i messaggi presenti in questa categoria ci sono i controlli di All Notes OFF e di All Sound OFF. La differenza tra i due è che il primo genera un Note Off per tutte le note attive, mentre il secondo "muta" completamente i suoni, tagliando le eventuali code. Fondamentali per il corretto funzionamento di una catena MIDI è la conoscenza dei modi MIDI e i quindi i messaggi usati per controllare questi ultimi. Tra i messaggi di tipo Channel Mode sono presenti, infatti, quattro messaggi, chiamati rispettivamente Omni Mode Off, Omni Mode On, Mono Mode On (o Poly Off) e Poly Mode On (o Mono Off). I primi due regolano i canali dai quali i dispositivi ricevono messaggi: il primo abilita il solo canale base, il secondo invece abilita la ricezione su tutti i canali. I secondi invece impostano la capacità dello strumento di agire in modo monofonico o in modo polifonico. Quindi nel caso si presentino due Note On "simultanei" (ovvero che avvengono molto vicini nel tempo, ma comunque seriali), se il dispositivo è impostato come monofonico spegnerà la prima nota con l'avvento della seconda, mentre nel caso opposto verrà suonato un accordo. Combinando le quattro modalità offerte dallo standard si possono ottenere gli altrettanti modi MIDI, che servono a coprire le varie esigenze di performance. I quattro modi possono essere così riassunti[b2]:

- Modo 1 o Omni mode (Omni On + Poly) :Il ricevitore risponde su tutti i canali in modo polifonico. Usato per testare i componenti di una catena, o per catene molto semplici, in quanto mescola i messaggi dei vari canali perdendone i riferimenti.
- Modo 2 (Omni On + Mono): Il ricevitore risponde su tutti i canali in modo monofonico. Raramente implementato, non gestisce la polifonia nonostante riceva da tutti i canali.
- Modo 3 o Poly Mode (Omni Off + Poly): Il ricevitore risponde solamente al suo canale base in modo polifonico. È il modo più implementato, poiché può essere usato nelle catene più complesse. Il dispositivo è in ascolto su un solo canale e gestisce la polifonia su di esso.
- Modo 4 (Omni Off + Mono): Il ricevitore risponde solamente al suo canale base in modo monofonico. È un modo usato soprattutto per l'emulazione di strumenti monofonici, come i fiati.

Al di fuori dello standard ma largamente impiegati sono i due modi evoluzione del Modo 3 e del Modo 4. Il primo è conosciuto come Multi-timbral Mode o Multi Mode, nel quale ogni singolo dispositivo è visto come la combinazione di n strumenti polifonici indipendenti. Il

secondo invece, noto come Mono Mode, permette di sfruttare n canali monofonici singolarmente, come succede per esempio nelle chitarre MIDI, dove ogni corda viene trattata indipendentemente.

1.4.2 System Message

I messaggi di tipo System sono caratterizzati dall'assenza di riferimento al canale, poiché la loro funzione è di agire su parametri, come dice il nome stesso, del sistema. Il loro Status Byte è formato dai quattro bit più significativi posti a 1, mentre i successivi quattro indicano la funzione che essi svolgono.

I messaggi di System Common[4] trasmettono informazioni comuni a tutti i canali e in particolare si occupano di alcune funzioni relative all'esecuzione dei brani e allo spostamento al loro interno. Il MIDI Time Code è un sistema di sincronizzazione basato sulla suddivisione in ore, minuti, secondi, frame secondo lo standard SMPTE. Per rappresentare tutte le informazioni della codifica temporale si ricorre a una catena di otto messaggi, in cui l'unico Data byte contiene l'identificatore del dato nei primi quattro bit e il valore stesso nei secondi quattro, come rappresentato nella tabella 1.

ID del Data byte	Funzione
0000	Primo nibble del numero di frame
0001	Secondo nibble del numero di frame
0010	Primo nibble del numero di secondi
0011	Secondo nibble del numero di secondi
0100	Primo nibble del numero di minuti
0101	Secondo nibble del numero di minuti
0110	Primo nibble del numero di ore
0111	Secondo nibble del numero di ore

Tabella 1

È fatto uso comune utilizzare il messaggio che dovrebbe contenere il secondo nibble delle ore per indicare il frame rate, utilizzando il secondo e il terzo bit secondo la codifica illustrata nella tabella 2.

00	24 fps
01	25 fps
10	30 fps Drop Frame
11	30 fps

Tabella 2

I messaggi di MTC vengono detti anche quarter frame, in quanto vengono inviati quattro messaggi per ogni frame, con una frequenza media di che varia dai centoventi ai settantadue messaggi al secondo, tenendo presente che il traffico dei messaggi influisce sulla stessa.

[4] Per la lista completa dei messaggi System Common consultare la tabella 4 presente in appendice.

Tra i messaggi System Common è presente il comando di Song Position Pointer, in altre parole il messaggio che viene utilizzato per spostarsi all'interno di una canzone. Tale messaggio è composto, oltre al byte di stato, da due Data byte, che, con la logica della combinazione MSB + LSB, permettono di puntare ad un punto preciso di ogni canzone sfruttando i messaggi di Timing Clock. Ogni sei pulsazioni viene conteggiato uno step, permettendo una precisione al sedicesimo. Tramite il messaggio di Song Select è invece possibile scegliere una delle canzoni di un set MIDI, in quale può essere composto di, al massimo, centoventotto brani, il valore più grande che può essere indicato dall'unico Data byte che compone il messaggio stesso. Gli altri messaggi di questo tipo sono il messaggio di Tune Request, che avviava un processo di intonazione per i sintetizzatori analogici, e i messaggi di SysEx e End of SysEx, di cui si spiegherà la funzione nella sezione dedicata ai messaggi System Exclusive.

I messaggi Real Time fanno parte della categoria System Common ma, come suggerisce il nome stesso, sono stati creati per interagire in tempo reale con le altre apparecchiature MIDI. I messaggi di questa categoria hanno una priorità più alta rispetto ai messaggi e possono anche interpolarsi tra di essi, per cercare di limitare i ritardi dovuti al traffico seriale di messaggi. Tra di essi i messaggi di MIDI Timing Clock costituiscono un vero e proprio "metronomo" per la sincronizzazione di vari strumenti, come sequencer o drum machine. A differenza dei messaggi MIDI Time Code, il MIDI Timing Clock non aderisce allo standard SMPTE, in non quanto si tratta di un riferimento temporale ma di una scansione relativa alla frequenza dei battiti al minuto del brano in esecuzione. In particolare per ogni beat vengono generati ventiquattro messaggi di clock, costituiti dal solo byte di stato. Con l'ausilio di questi messaggi e dei messaggi di Song Position Pointer i messaggi di Start, Stop e Continue possono controllare l'esecuzione del brano senza utilizzare i complessi messaggi MIDI Time Code. Tra i sistemi implementati dal protocollo, ma non sempre realizzata, per il controllo delle connessioni è presente l'Active Sensing: questo sistema forza i moduli componenti la catena MIDI a trasmettere o a ricevere un messaggio MIDI ogni 300 millisecondi. Questa funzione viene attivata dopo il primo invio di un messaggio di Active Sensing. Se il trasmettitore non invia altri messaggi, viene mandato un nuovo messaggio di questo tipo, altrimenti il ricevente considera chiuso la connessione e comanda un All Notes Off sui moduli sonori. È possibile inoltre, tramite il messaggio di System Reset, riportare le apparecchiature alle impostazioni predefinite.

Lo standard MIDI, essendo stato concepito e sviluppato in collaborazione con i maggiori produttori di dispositivi musicali digitali, comprende una categoria di messaggi che sono

specifici a seconda del produttore e altri messaggi che potenzialmente potrebbero essere interpretati da tutti i dispositivi aderenti lo standard stesso, pur non rientrando nelle categorie. Questi messaggi sono noti come System Exclusive[5] o, in forma contratta, SysEx. Questi messaggi hanno la caratteristica di avere una lunghezza variabile e di non avere una composizione standard in termini di numero di byte, l'unico modo per identificarli è includerli tra due messaggi di SysEx e End of SysEx, facenti parte della categoria dei System Common. I System Exclusive dei produttori hanno un numero variabile di byte, tipicamente otto ma ad esempio Roland presenta nove byte, dei quali il secondo, dopo il messaggio di inizio di un messaggio System Exclusive, è l'identificatore del produttore che ha implementato quel tipo di messaggio. Questo ID è reso disponibile dalla MMA in modo da prevenire ambiguità tra diversi marchi[6]. È possibile anche che i byte di identificazione siano estesi a tre, siccome il numero inizialmente ridotto di case di produzione si è ampliato notevolmente. È di conseguenza ampio il numero di variabili all'interno della catena dei byte di System Exclusive. La MMA ha inoltre riservato tre ID speciali, il primo dedicato ai progetti di ricerca, quindi esclusi dalle pubblicazioni, mentre i secondi due sono indicati per i messaggi System Exclusive universali, divisi in Non-Real Time (0x7E) e Real Time (0x7F). Questi tipi di messaggi sono stati pensati con lo scopo di allargare i parametri gestibili attraverso lo standard, senza però dover cambiare la struttura dei messaggi già esistenti. Questi messaggi specificano a quale dispositivo è rivolto il SysEx, specificando il suo indirizzo di porta o il canale, oppure usando il codice 0x7F, trasmettendo ovvero su tutti i canali, e successivamente specificano tramite due byte quale è la funzione dei dati in contenuti nel messaggio. Per esempio è possibile attivare o disattivare la compatibilità con lo standard General MIDI.

1.4.3 MIDI Running Status [b5]

L'intenso traffico che può presentarsi sui canali di trasmissione MIDI ha portato alla scelta di alcuni adattamenti per cercare di ridurre dove possibile, senza compromettere la comprensibilità dei dati, tale flusso di messaggi. Il MIDI Running Status permette di mandare una serie di messaggi caratterizzati da uno Status Byte identico senza bisogno di ripeterlo. Ad esempio, una performance prevede una serie di note sullo stesso canale, grazie alla tecnica del Running Status verrà inviato un solo Status Byte di Note On, seguito dalle coppie di Data byte relative alle note suonate, finché non occorrerà un messaggio dallo Status Byte differente.

[5] Per la lista completa dei messaggi System Exclusive universali consultare la tabella 5 presente in appendice.

[6] La lista degli identificatori stabiliti dalla MMA è disponibile all'indirizzo http://www.midi.org/techspecs/manid.php.

Prendendo in considerazione la serialità dei messaggi MIDI e la loro velocità di trasmissione, i miglioramenti sono considerevoli.

1.5 General MIDI [b3]

In seguito al rilascio delle specifiche MIDI, la connessione tra diversi dispositivi digitali era possibile ma persistevano ancora alcune problematiche relative, soprattutto, alla gestione dei timbri e di altri parametri. Di fatto, lo standard MIDI al suo lancio era un protocollo software e hardware di comunicazione, ma non imponeva dei particolari requisiti alle apparecchiature su cui era implementato. Nel 1991 la MMA e la sua controparte nipponica, la JMSC, stilarono una serie di specifiche obbligatorie per poter aumentare la compatibilità soprattutto tra i diversi banchi di timbri dei diversi produttori. Se per esempio un musicista avesse creato un brano basandosi sulla libreria di un dato modulo sonoro, per esempio di marca Yamaha, non avrebbe avuto la certezza che lo stesso venisse riprodotto con gli stessi timbri da un sintetizzatore marcato Roland.

Le caratteristiche che deve supportare un prodotto per essere compatibile con il GM devono essere:

- Polifonia a minimo ventiquattro voci su tutti i canali, di cui almeno sedici melodici e otto ritmici.
- Supporto completo e simultaneo dei sedici canali MIDI, esclusiva del canale dieci per i suoni percussivi.
- Fino a sedici strumenti diversi suonanti contemporaneamente, un minimo di centoventotto strumenti, ordinati secondo le specifiche GM1.
- Supporto dei controlli continui numero 1, 7, 10, 11, 64, 121 e 123, supporto dei Registered Parameters Number (RPN), supporto del comando di Channel Pressure e di Pitch Bend.
- supporto degli RPN per il coarse tuning e il pitch bend range, supporto del messaggi System Exclusive per l'abilitazione GM.

I banchi di suoni sono divisi in sedici famiglie, contenenti ciascuna otto timbri, corrispondenti ad un Program Change[7]. Per esempio la prima famiglia è quella dei pianoforti, e il timbro numero 1 è l'Acoustic Grand Piano, mentre la quarta famiglia è quella delle chitarre e il timbro numero 28 corrisponde a quello di una chitarra elettrica con suono clean. È da notare

[7] La lista completa dei timbri definiti dal General MIDI è consultabile in appendice, nella tabella 6.1 e 6.2.

che lo standard non fissa dei parametri sulla qualità dei timbri: uno strumento che reca il marchio di compatibilità GM assicura che i banchi sonori rispettino la tabella indicata, non che soddisfino particolari criteri sulla bontà dei timbri stessi. Inoltre le specifiche non sono obbligatorie e possono essere disattivate da un apposito messaggio.
Successivamente, nel 1999, gli organismi di controllo e sviluppo dello standard rilasciarono una nuova versione, denominata General MIDI 2 (aggiornata nel 2003 e nel 2007). Questa propone un'ulteriore stretta sull'assegnazione di alcuni parametri controllati da Control Change, così come l'aumento delle voci contemporanee a trentadue e all'introduzione di nuovi messaggi SysEx e RPN. Inoltre è stato sviluppato uno standard "leggero", chiamato appunto General MIDI Lite, il cui scopo è di definire uno standard per quei dispositivi che non possono o che non necessitano di aderire completamente alle specifiche complete, come per esempio i dispositivi mobile.

1.5.1 GM Drum Set

All'interno delle specifiche General MIDI è riservata una menzione particolare ai suoni percussivi. Essendo poco sensato trattarli come un normale banco di timbri, cioè a ogni nota corrisponde un timbro più acuto o più basso secondo il pitch, è stato scelto di assegnare a determinati valori di pitch determinati tipi di percussione, in modo che in un solo banco sia possibile far riferimento ad un gran numero di suoni diversi. Si è scelto inoltre di relegare al canale dieci (0x9, B1001) questo tipo di suoni, specificandone la natura in una tabella riservata. I valori sono compresi tra il pitch 35, Acoustic Bass Drum, e il pitch 81, Open Triangle. Anche per questo particolare caso è valido il riferimento alla qualità dei timbri del GM.

1.6 Standard MIDI File [b4]

Lo Standard MIDI FILE (SMF) è un file di descrizione di una performance, nato nel 1986 e definito dalla MMA nel 1990, il cui scopo è la condivisione di arrangiamenti e canzoni tra diversi sequencer e software di riproduzione multimediale. Il file è in formato binario e l'estensione è .MID. Le affinità tra il linguaggio MIDI e il linguaggio simbolico della partitura rendono la manipolazione degli SMF molto precisa e fruibile. Inoltre la sua compattezza in termini di spazio di memorizzazione lo rende facilmente interscambiabile.
Gli SMF possono essere classificati in tre tipologie: il tipo 0 prevede un'unica traccia in cui sono accorpate tutti canali e gli strumenti, registrando le informazioni di temporizzazione all'interno della traccia stessa; il tipo 1 invece divide in tracce separate ogni canale, memorizzando i bpm e le altre informazioni del brano nella prima traccia, limitando però ogni

file ad una singola canzone; il tipo 2 invece permette di creare sezioni indipendenti, memorizzando anche le diverse informazioni ad esse relative.
La struttura di questi file è composta di chunk, una successione di bit dal significato comune. Questi chunk presentano un blocco di intestazione composto da otto byte, uno che ne specifica il tipo, l'altro che ne codifica la lunghezza. I chunk possono essere quindi divisi in due categorie, i MIDI Track Header Chunk e i MIDI Track Chunk.

1.6.1 MIDI Track Header Chunk (MThd)

I MIDI Track Header Chunk sono le strutture che descrivono le tracce MIDI all'interno di un file. L'intestazione presenta un codice ASCII di quattro byte che la identifica come tale, seguito da altri quattro byte che indicano la lunghezza totale del chunk, esclusa l'intestazione, che per questa categoria è fissata a sei byte. Questi byte costituiscono i dati veri e propri del MThd e contengono le informazioni fondamentali della traccia, ovvero a quale tipo di SMF appartiene, quante tracce sono presenti in totale e quale tipo di risoluzione temporale è stata usata. Se il bit più significativo della coppia di byte destinata ad essa è posto a 1, la risoluzione è basata sulla suddivisione del secondo in frame, come accade nella codifica SMTPE e MIDI Time Code. I bit dal quattordicesimo all'ottavo descrivono in complemento a due il tipo di suddivisione del secondo, indicando con -29 il formato Drop Frame. I restanti bit indicano invece quale risoluzione viene adottata all'interno di ogni frame (ad esempio, la cifra 4 rappresenta il MIDI Time Code). Se invece il bit più significativo è posto a 0, la suddivisione si basa sul tempo metronomico. I bit successivi indicano in quante frazioni è possibile dividere la nota da un quarto (Pulse Per Quarter Note, PPQN).
Le differenti codifiche servono allo scopo di rappresentare il Delta Time, il differenziale di tempo tra due eventi rappresentati all'interno di una traccia MIDI.

1.6.2 MIDI Track Chunk (MTrk)

I MTrk, come gli Header Chunk, presentano un codice identificativo e quattro byte dedicati alla descrizione della lunghezza della sezione dati, che in questo caso può essere variabile. I dati rappresentano coppie Delta Time (Δt) – Evento, in cui il Δt indica la differenza tra l'evento a lui accoppiato e quello precedente, secondo la codifica temporale descritta nel Mthd della traccia. La rappresentazione di questo tempo avviene con un numero variabile di byte, ma utilizzando solamente sette bit di questi. Il bit più significativo di ogni byte viene utilizzato di fatto come un flag: il valore 0 indica l'ultimo byte, tutti gli altri hanno questo bit posto a 1. La ricostruzione del valore definitivo avviene secondo una logica MSB+LSB.
Gli eventi MIDI sono raggruppati in tre categorie:

- MIDI Event, ossia un qualsiasi evento MIDI.

- SysEx Event, caratterizzati dal primo byte 0xF0 che li identifica.
- Meta Event, informazioni del brano non identificabili con messaggi MIDI, caratterizzati dal primo byte identificativo 0xFF.

Tra i Meta Event sono presenti l'armatura di chiave, il tempo metronomico, il metro della misura e l'armatura di chiave[8]. Molto importante tra questi tipi di dato è il MIDI Track End, il descrittore della fine della traccia. Esso è codificato come 0xFF2F00 ed è precedente alla fine di ogni traccia, accoppiato al proprio Δt.

[8] La lista completa dei codici relativi ai Meta Event è consultabile in appendice, nella tabella 7.

CAPITOLO II
Arduino

2.1 Introduzione

Arduino è una piattaforma hardware di sviluppo Open Source progetta e realizzata da un team capitanato da Matteo Banzi. Lo scopo di tale lavoro era offrire agli studenti dell'Interactive Design Institute di Ivrea una soluzione semplice e comprensibile per realizzare progetti interattivi elettronici senza scomodare linguaggi di programmazione complessi e conoscenze avanzate [b6]. Ben presto però il progetto si perfezionò fino a renderlo competitivo a livello commerciale e imponendosi come uno delle schede di prototipazione più conosciute e sfruttate in molti campi. Il successo di Arduino è da ricercare, oltre che nella sua architettura hardware e software, nel contributo attivo fornito dalla vasta community dei "Makers" e alla sua nativa "openess"[1].

2.2 Il Maker Movement

Il movimento dei Makers (o Maker Culture) è un movimento che nasce negli Stati Uniti nel secondo dopoguerra per il congiungimento di una serie di fattori tecnologici e sociali. La spinta tecnologica dell'industria bellica aveva portato alla disponibilità sul mercato di dispositivi ad alta tecnologia in modo massiccio e fiorirono una serie di riviste dedicate a questi "hobbisti della tecnologia". Col tempo prese forma quella che viene definita la "DIY Ethic"[2], ispirata all'etica della "craftmanship" mutuata dall'"Arts & Crafts Movement", riassunta dall'espressione "If you can't open it, you don't own it"[3][b7]. Il fenomeno era incentrato sulla figura dell'appassionato che poteva competere con i prodotti commerciali attraverso la propria abilità e conoscenza dei materiali, portando spesso anche a un risparmio economico.

2.3 La filosofia Open Source

Con l'avvento dei personal computer anche gli appassionati di software ebbero la possibilità di poter creare i propri applicativi e di migliorare quelli già esistenti. Questi si scontrarono

[1] Si preferisce usare il termine inglese in quanto, riferendosi alle caratteristiche dei prodotti Open Source, i relativi termini italiani possono essere fuorvianti. In seguito saranno usati altri termini nella loro lingua originale per il medesimo motivo.
[2] "Etica del Do It Yourself", o del fai-da-te.
[3] Letteralmente "Se non lo sai aprire, non lo possiedi". Con questa massima si vuole riassumere la filosofia dei Makers, secondo la quale non si è fatto proprio un oggetto se non lo si apre per conoscerlo nelle sue più piccole caratteristiche.

però con la questioni dei diritti d'autore e di sfruttamento dei proprietari, che non intendevano lasciare trapelare il funzionamento dei propri prodotti. Nacque in contrapposizione a questa mentalità la filosofia dell'Open Source: gli stessi creatori e detentori dei diritti sui software diffondevano il proprio codice per permettere ad altri programmatori di svilupparlo e di correggerlo dove necessario. Il movimento divenne ben presto ampio e supportato da organizzazioni come la FSF (Free Software Foundation), fondata da Richard Stallman, uno dei pionieri del software open. Con l'avvento di Internet la diffusione del software divenne ancora più capillare e si svilupparono grandi community i cui membri costituiscono veri e propri team di sviluppo. Tra i migliori e più affermati "prodotti" del movimento Open Source sono il sistema operativo Linux, la licenza GNU GPL[4], il compilatore GCC[5] e l'enciclopedia online Wikipedia.

2.4 Arduino

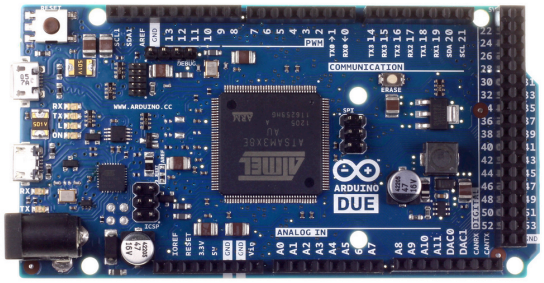
Il progetto Arduino nasce dalla necessità di Banzi e dei suoi collaboratori di mettere a disposizione dei propri studenti uno strumento semplice ed intuitivo per produrre prototipi e sfruttare la tecnologia in modo creativo. Fin dal principio l'iniziativa fu indirizzata verso l'apertura a tutte le forme di collaborazione Open Source[s2]. Il team Arduino forniva nuove versioni aggiornate di basi di prototipazione che gli utenti sfruttavano in modo diversificato, ricambiando i produttori con nuovi spunti per ulteriori migliorie al progetto, fornendo anche le caratteristiche stesse della scheda, in modo che chiunque potesse creare un "clone" di Arduino, formandolo secondo le proprie esigenze. Tuttavia non si verificò un fenomeno di dispersione, anzi più la community si allargava più si espandeva la fama del progetto Arduino. Dal 2005 fino ad oggi sono state prodotte diciannove schede Arduino riconosciute, con differenti microprocessori e diverse configurazioni per quanto riguarda gli input e gli output. Il colore blu e il classico "scalino" su uno dei lati sono i marchi di fabbrica studiati per rendere Arduino un prodotto unico e riconoscibile. Per il progetto di questo elaborato è stata scelta la scheda Arduino Due.

Il logo di Arduino

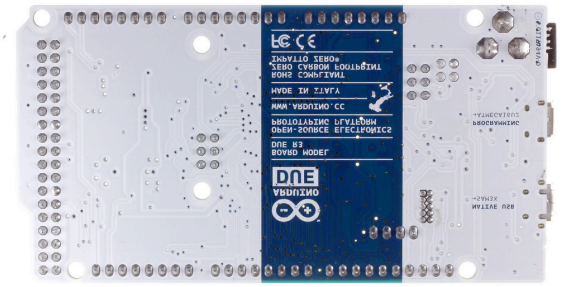
[4] GNU General Public Library, una licenza software che regola la distribuzione dei contenuti Open Source. GNU è l'acronimo ricorsivo per GNU is Not Unix, ovvero il sistema operativo open sviluppato da Richard Stallman nel 1984.
[5] GNU C Compiler, compilatore per il linguaggio C.

2.4.2 Arduino Due

La scheda Arduino Due fu immesso sul mercato nel 2012 e presenta una dotazione che si discosta leggermente da quella che si può considerare standard per i modelli precedenti. La prima caratteristica saliente di questa scheda è il processore Atmel SAM3X8e ARM[6] Cortex M3 con un clock da 84 MHz. Sfruttando un set di istruzioni a 32 bit di tipo RISC, questo prodotto è uno dei migliori in quanto a rapporto prestazioni/consumi e infatti è utilizzato in una serie di progetti embedded dove è fondamentale il consumo delle batterie, .



Arduino Due, fronte.



Arduino Due, retro

Questo processore mette a disposizione dei programmatori una memoria flash da 512 kByte, di molto superiore rispetto alle altre schede Arduino, e 96 kByte di RAM, resa ancora più efficiente da un sistema DMA[7]. La grande capacità di elaborazione del processore ha permesso al team di Arduino di potenziare la disponibilità di input e output della scheda, che propone:

- 54 pin digitali I/O, tra i quali 12 possono fornire un output in Pulse Width Modulation[8].
- 12 pin per gli input analogici con una profondità di campionamento di 12 bit.
- 2 pin per gli output analogici, anch'essi con una risoluzione di 12 bit
- 4 canali di comunicazione seriale, dividi in coppie di I/O.
- 2 porte USB, di cui una dedicata all'interscambio con il software di programmazione, mentre la seconda permette di utilizzare Arduino come un mouse o una tastiera.
- 6 porte per la comunicazione attraverso i protocolli SPI, CAN e TWI.

La scheda Due può essere alimentata in diversi modi, tra cui il jack DC, il pin "VIN" o la stessa porta USB, e può ricevere una tensione variabile tra i 7 e 16 V, mentre può fornire una tensione di 3,3 o 5 V attraverso i relativi pin. I pin digitali di I/O possono ricevere o fornire

[6] Advanced RISC Machine.
[7] Direct Memory Access, un sistema che gestisce gli accessi in memoria senza l'intervento della CPU.
[8] La PWM permette, attraverso la generazione di un'onda quadra modulata in larghezza, di generare output analogici[s3].

una variazione di potenziale di 3,3 V.

Nonostante le differenze con le altre schede offerte dal team Arduino, la scheda Due consente la connessione con la stessa varietà di sensori e attuatori che hanno fatto di Arduino uno dei prodotti preferiti dagli appassionati di elettronica e non solo. Inoltre è stata mantenuta la compatibilità con le diverse Shield preesistenti alla Due. Le Shield sono delle schede che possono essere montate a diretto contatto con Arduino per espanderne le capacità, soprattutto per quanto riguarda la comunicazione. Tra le Shield disponibili nello store ufficiale di Arduino troviamo Shield per implementare la comunicazione Wi-Fi, Bluetooth o GSM, ma esistono anche Shield specifiche per la connessione dei servomotori. Esistono inoltre numerose Shield non ufficiali e vengono forniti attraverso il sito ufficiale di Arduino tutta la documentazione necessaria per costruire la propria Shield.

2.4.3 L'IDE Arduino

Perseguendo sempre l'ideale di "openess", gli sviluppatori di Arduino basarono l'Integrated Development Enviroment sulla piattaforma Wiring, creata da Hernando Barragàn durante la sua permanenza all'Interaction Design Insitute di Ivrea nel 2003, lo stesso in cui Massimo Banzi e i suoi collaboratori, David Cuartielles, Tom Igoe, Gianluca Martino e David Mellis, concepirono due anni più tardi Arduino. A sua volta Wiring si appoggia a Processing, un linguaggio di programmazione Open Source basato su Java creato da Casey Reas e Benjamin Fry presso il Massachusetts Institute of Technology, anch'esso rilasciato con licenza Open Source.

L'aspetto puramente grafico dell'IDE di Arduino deriva da quello di Wiring e dal punto di vista degli intenti, i due progetti sono uno la naturale evoluzione dell'altro. Infatti, entrambi sono stati concepiti per fornire uno strumento facilmente utilizzabile, oltre che dal punto di vista prettamente elettronico, anche dal punto di vista informatico. Chiari indizi di questa derivazione sono presenti anche nelle caratteristiche del linguaggio di programmazione, come ad esempio la presenza della libreria "wiring.h", fondamentale per interfacciare e semplificare tutte le operazioni di input/output.

La compatibilità tra Arduino e Processing è molto alta, nonostante uno sia sviluppato in C/C++ e uno in Java, e le poche differenze sono puntualizzate all'interno della documentazione disponibile sul sito ufficiale di Arduino [s4]. Inoltre anche Arduino, come Processing, definisce "sketch" i file contenenti il codice. Questa compatibilità permette a un progetto pensato per uno dei due linguaggi di essere scambiati a seconda delle necessità: se un progetto richiede operazioni molto complesse e deve interfacciarsi con strumenti complessi può essere scritto in Processing ed essere installato sopra ad un normale computer, mentre se

la priorità è semplicità di implementazione e uso di mirato di poche risorse, allora si può propendere per utilizzare una scheda Arduino e scrivere il codice necessario nel suo linguaggio di programmazione dedicato.

2.5 Programmare con Arduino

La facilità di sviluppo di un'applicazione con Arduino è uno dei principi fondanti dell'intero progetto e lo sforzo dei creatori ha raggiunto notevoli risultati. Alla prima apertura dell'IDE di Arduino, disponibile in download gratuito per piattaforme Windows, Mac OS X e Linux[9], viene caricato uno sketch vuoto che presenta le funzioni sufficienti e necessarie per qualunque programma: `setup()` e `loop()`. La prima funzione viene eseguita una sola volta all'accensione della scheda e al suo interno possono essere dichiarate e inizializzate variabili, oppure possono essere assegnati vari funzionalità a determinati pin. La funzione `loop()` invece è una funzione ciclica, come suggerisce il nome stesso, e viene eseguita continuamente fino allo spegnimento della scheda. Questa funzione costituisce il vero e proprio "corpo" del programma e in essa sono svolte tutte le operazioni fondamentali.

Se non vi è una particolare necessità di programmazione, non è richiesta l'inclusione di alcuna libreria, poiché il compilatore provvede in automatico a includere la libreria standard di Arduino. Esiste comunque una serie di librerie comprese nell'IDE che possono essere incluse all'interno in un programma, oppure si possono includere librerie di terzi. All'interno del sito è inoltre presente un tutorial su come scrivere la propria libreria[10].

2.5.2 Blink, un esempio [b8]

Per mostrare un semplice programma sarà illustrato uno degli esempi che vengono forniti all'interno della IDE, "Blink", che può essere considerato il corrispettivo di "Hello World" in Arduino.

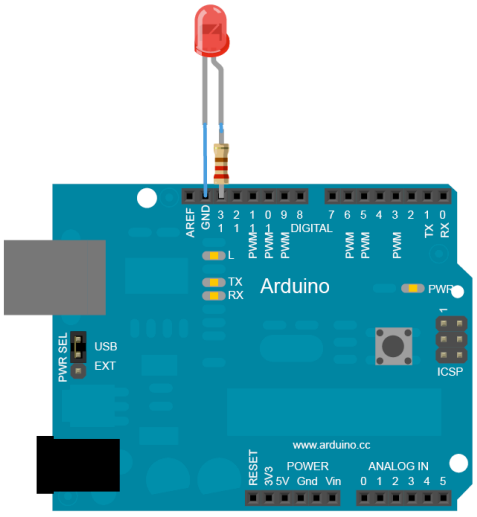
La prima operazione che viene compiuta è la dichiarazione di una variabile intera "led" e la sua inizializzazione. Questo avviene soprattutto per comodità di

```
/*
  Blink

  This example code is in the public
domain.
 */

int led = 13;

void setup() {
  pinMode(led, OUTPUT);
}

void loop() {
  digitalWrite(led, HIGH);
  delay(1000);
  digitalWrite(led, LOW);
  delay(1000);
}
```

[9] http://arduino.cc/en/Main/Software.
[10] http://arduino.cc/en/Hacking/LibraryTutorial.

programmazione: è più facile ricordarsi un nome piuttosto che un numero e in caso di correzione, si dovrà solamente modificare il valore assegnato a tale variabile. In seguito, nella funzione `setup()` viene invocata la funzione `pinMode()` [s5]: questa riceve in ingresso un numero intero e una funzione che può assumere un pin (INPUT, OUTPUT, INPUT_PULLUP) e assegna al pin identificato dal numero intero quella funzione. Quindi il pin identificato dal numero 13 è configurato come un pin di output. Viene usato questo particolare numero perché la maggior parte delle schede Arduino ha un led installato collegato al suddetto pin, quindi non è necessario nessun tipo di collegamento per testare questo breve sketch. Nella funzione `loop()` vengono iterate due operazioni: la prima è la chiamata della funzione `digitalWrite()` [s6], la seconda funzione è `delay()` [s7]. La funzione `digitalWrite()` riceve anch'essa due parametri, il primo è ancora l'identificatore di un pin, il secondo è un valore booleano, HIGH o LOW, che corrispondono al livello di tensione alto (5V o 3,3V a seconda delle schede) o basso (0V), che si traducono, in questo caso, in accensione o spegnimento del led collegato al pin 13. Questa funzione compila correttamente solo se il pin indicato è stato inizializzato come un pin di output. Dopo l'accensione o lo spegnimento del led, quindi la funzione `delay()` viene invocata e, come il nome assegnatole indica chiaramente, ritarda l'esecuzione dell'operazione seguente, in questo caso di mille millisecondi.

Una volta compilato questo sketch e caricato attraverso la porta seriale di una scheda Arduino, si vedrà il led integrato nella scheda accendersi, restare in questo stato per un secondo, spegnersi e restare spento per un secondo e così finché non si spegne la scheda. In caso il led non fosse presente si può collegare il led al pin 13 come in figura[11].



Collegamento di un led al pin 13.
La resistenza ha un valore di 220Ω.

[11] Tutte le immagini che illustrano collegamenti tra Arduino e varie componenti hardware sono tratte dal sito ufficiale e sono state realizzate tramite Fritzing (http://fritzing.org).

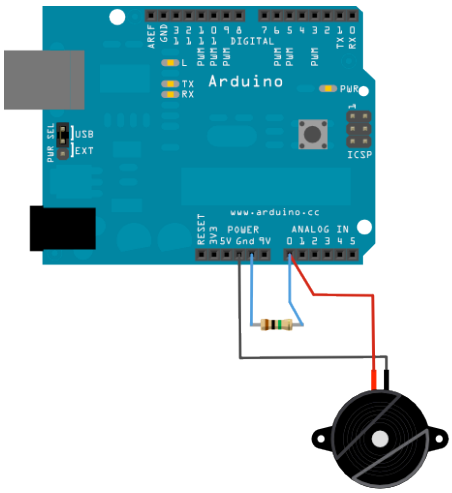
Capitolo III
Combinare MIDI e Arduino

3.1 Le componenti hardware

Il progetto relativo a questo elaborato consiste in un dispositivo capace di trasformare degli stimoli di tipo meccanico in messaggi aderenti allo standard MIDI. In particolare è stato implementato uno strumento che potesse essere usato per gestire dei suoni di batteria e inviare messaggi di Control Change. Per fare ciò sono stati impiegati dei trasduttori piezoelettrici, una scheda Arduino Due e una presa DIN a cinque poli. A corredo della struttura principale vi sono quattro pulsanti e otto led[1] a due colori che rendono più comprensibile l'uso dello strumento e ne ampliano le capacità.

3.1.2 Sensori piezoelettrici

I sensori piezoelettrici sono costituiti da cristalli che hanno la proprietà di creare una differenza di potenziale quando sottoposti a una deformazione meccanica. Questi cristalli sono molto sensibili, le variazioni che sono in grado di rilevare possono essere nell'ordine dei nanometri. La loro funzione all'interno del progetto è di trasformare una grandezza fisica, come il contatto tra questi trasduttori e un corpo esterno, in una variazione di potenziale che può essere ricevuta, interpretata e modellata da un microcontrollore.



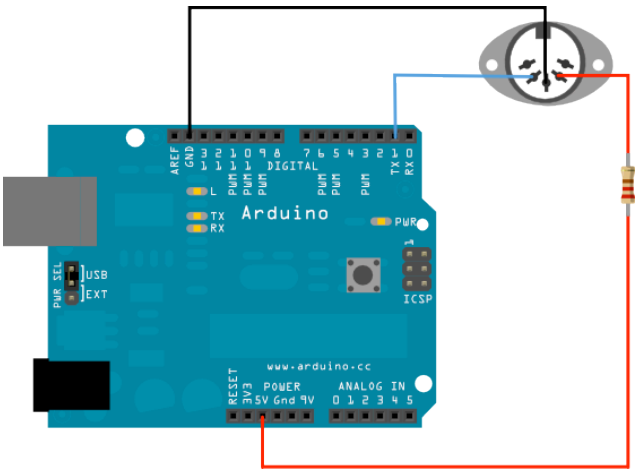
Arduino permette di utilizzare dei trasduttori piezoelettrici come ingressi analogici grazie ad un certo numero di porte, variabile da modello a modello di scheda, nel caso di Arduino Due sono dodici dedite a questo compito. Il polo positivo del sensore delle quindi essere collegato a una di queste porte, indicate con le sigle A0 – A11, mentre quello negativo deve essere collegato a massa. Viene inoltre adottato come accorgimento un collegamento tra ingresso e massa attraverso una resistenza di valore 1 MΩ per evitare che la differenza di potenziale creatasi all'interno del cristallo sia superiore a quella supportata dagli ingressi.

Gli ingressi analogici sono collegati ad un dispositivo ADC[2] a 12 bit, quindi una scheda Arduino Due può tradurre le variazioni di voltaggio da 0 a 3,3V (5V in altre schede) in valori binari da 0 a 1023. La funzione deve essere chiamata per la conversione è `analogRead()`[s8], che riceve in entrata uno dei pin di ingresso, precedentemente inizializzato come tale, e restituisce un valore intero. I produttori sottolineano come

[1] Il funzionamento dei led e la loro connessione è il medesimo illustrato nel paragrafo 2.5.2
[2] Analog to Digital Converter, convertitore analogico-digitale.

l'operazione di input analogico richieda circa cento microsecondi, quindi il numero massimo di letture al secondo è nell'ordine della decina di migliaia.

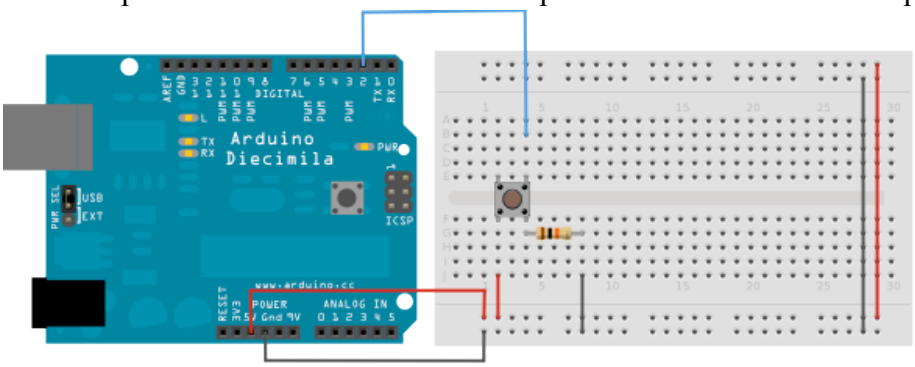
3.1.3 La comunicazione seriale



Sia Arduino che lo standard MIDI prevedono un protocollo di comunicazione seriale, per cui è molto semplice fare in modo che progetti, realizzati anche a livello amatoriale, possano comunicare con i prodotti aderenti allo standard. Tra i pin in dotazione delle schede Arduino ve ne sono due dediti alla comunicazione, indicati dalle sigle TX e RX, rispettivamente in uscita e in entrata, corrispondenti ai pin digitali 1 e 0. Per le esigenze del progetto è quindi necessario collegare il pin seriale di uscita al pin 5 della presa DIN, il pin 2 della presa alla massa e il pin 4 al pin che eroga +5V delle schede Arduino, ponendo in serie una resistenza da 220Ω, come illustrato in figura. Per allineare il flusso di dati in uscita da Arduino e la capacità di lettura dei dispositivi MIDI si imposta la frequenza di emissione dei dati del primo attraverso la funzione `Serial.begin()`[s9], che riceve in ingresso la suddetta frequenza e la imposta come velocità di trasmissione, in questo caso di 31,25 Kbaud. Per poter visualizzare il menù si utilizza la possibilità di Arduino di trasmettere dati seriali tramite la propria porta di programmazione. La scheda Arduino provvede in modo autonomo a convertire i dati seriali in uscita nel protocollo USB, che poi vengono visualizzati tramite il monitor seriale presente nell'IDE. In questo caso si può scegliere una frequenza diversa da quella del canale MIDI, in quanto non si è più legati allo standard. In questo caso la funzione utilizzata è `Serial.print()`[s10] che stampa sul monitor seriale dei caratteri ASCII leggibili dall'uomo.

3.1.4 I pulsanti

Per comunicare in modo rapido con la scheda Arduino è possibile installare dei semplici pulsanti, come rappresentato in figura. Questo collegamento praticamente



mette in stato HIGH il pin assegnato al pulsante, in quanto viene collegata il pin di corrente ad esso. Premendo il pulsante viene interrotto il flusso di corrente e questo porta lo stato del pin a LOW. Monitorando lo stato del pin attraverso la funzione `digitalRead()` [s11] è possibile programmare una serie di azioni corrispondenti alla pressione del pulsante. Per evitare una serie di errori di lettura è stata implementata una funzione che filtra i valori della suddetta funzione, ma di essa si parlerà più diffusamente nel paragrafo dedicato al software.

3.2 Il software

Il software è stato programmato per assolvere la principale funzione del progetto, generare messaggi MIDI, con particolare attenzione ai messaggi per batteria. Per questo motivo la funzione principale dello sketch esegue una continua scansione degli ingressi analogici e richiama la principale funzione per l'invio del messaggio MIDI relativo a quell'ingresso. In aggiunta vengono monitorati anche gli ingressi digitali relativi ai pulsanti, in modo da poter accedere alle altre funzioni del programma. Tra di esse vi è la possibilità di modificare il tipo di messaggio da inviare o il valore dello stesso.

3.3 L'acquisizione dei dati

Per comunicare al microcontrollore quale azione compiere è necessario interagire con esso attraverso i suoi sensori. Questi si dividono in due categorie, quelli digitali e quelli analogici. I primi sono costituiti da pulsanti, mentre i secondi sono costituiti da cristalli piezoelettrici. Pur basandosi su metodi diversi di acquisizione degli input, entrambe le tipologie necessitano di un filtraggio dei risultati per evitare problemi derivati da letture multiple o errate.

3.3.2 Debounce[3] digitale

Per evitare letture multiple, nella documentazione ufficiale è stato inserito un tutorial [s12] che, tramite un controllo dei tempi tra due rilevazioni, scarta le letture troppo vicine tra loro. Prendendo spunto da questo tutorial è stata implementata una funzione che, ricevuto in input l'alias del pin che si riferisce al pulsante, restituisce un valore diverso a seconda che la lettura sia accettata oppure scartata.

[3] Termine derivato dal verbo inglese "to bounce", rimbalzare, significa letteralmente "anti-rimbalzo". Ancora una volta viene evitata la traduzione in italiano per mantenere lo specifico significato del termine originale.

```
int Read(int button)
{
  int flag = 0;
  int reading = digitalRead(button);

  if (reading != lastButtonState[button - 7]) {
    // reset the debouncing timer
    lastDebounceTime[button - 7] = millis();
  }
  if ((millis() - lastDebounceTime[button - 7]) > debounceDelay) {
    if (reading != buttonState[button - 7]) {
      buttonState[button - 7] = reading;
      if(reading == HIGH) flag = 1;
    }
  }
  lastButtonState[button - 7] = reading;
  if (flag == 1) {
    return 0;
  } else return 1;
}
```

Una volta invocata, tramite il valore di input (button) viene letto lo stato del pin relativo tramite la funzione `digitalRead()` e memorizzandolo all'interno della variabile reading. Se questa è diversa dal valore della variabile lastButtonState, il valore dell'ultima lettura, viene settato il timer della lettura (lastDebounceTime) tramite la funzione `millis()` [s13]. Se la differenza tra l'istante successivo e quello in cui è stato resettato il timer è maggiore del tempo di debounce scelto (debounceDelay) e il valore della lettura è diverso da quello attuale del pin (buttonState), quest'ultimo viene impostato come il valore letto. A questo punto, dopo aver memorizzato il valore dell'ultima lettura, se questi registra HIGH come stato del pin e che questo è cambiato rispetto al precedente valore, la funzione termina restituendo il valore 0, mentre se viene registrato il valore LOW si procede con la restituzione del valore 1. In questo modo è possibile controllare tutti gli ingressi digitali e comunicare direttamente al microcontrollore di agire in un determinato modo, come sarà poi mostrato più esaustivamente nel paragrafo riservato alla gestione dei menù.

3.3.3 Debounce analogico

Il controllo degli ingressi analogici è molto più complesso in quanto, a differenza della loro controparte digitale, devono poter essere accedute in modo indipendente tra loro per quanto riguarda i tempi di debounce. Inoltre è necessario tenere in considerazione le fluttuazioni generate dalla conformazione dei materiali che costituiscono i trasduttori.

```
int checkhit(int vel, byte spad)
{

  if(vel > threshold)
    {
      if (hit[spad] == false )
        {
          if(millis() - timing[spad] > release)
            {
              timing[spad] = millis();
              hit[spad] = true;
              return 0;
             }
          else
             return 1;
        }
       else
         return 1;
    }
   else
    {
       hit[spad] = false;
       return 1;
    }
}
```

La funzione che gestisce questo processo riceve in entrata il valore fornito dalla funzione `analogRead()` invocata ciclicamente su tutti gli ingressi (vel) e l'alias del pin del quale si vuole effettuare il controllo (spad). Il primo controllo che viene effettuato richiede che il valore di ingresso sia superiore ad una soglia (threshold) impostata per evitare di interpretare come un comando dell'utente una semplice variazione di pressione dovuta a spostamenti. Se questo risulta non superato, la funzione termina e viene registrato in un array (hit) l'eliminazione della lettura e viene restituito il valore 1. Questo array permette di tenere ordinati e distinti i valori per ciascun input analogico.

Se il valore in entrata è maggiore della soglia, avviene un secondo controllo che verifica che il precedente valore letto sia stato inferiore, altrimenti la funzione termina e restituisce 1. Questo è per evitare che le pressioni continue vengano interpretate come una serie di pressioni a breve intervallo l'una dall'altra.

Il terzo controllo consiste, come avviene per gli input analogici, in un confronto tra il tempo attuale, restituito dalla funzione `millis()`, e il tempo della precedente lettura, anch'esso memorizzato in un array (timing). Se la differenza tra i due timestamp è inferiore alla soglia scelta (release) la funzione termina e restituisce 1. Se invece la condizione è soddisfatta, viene registrato nell'array timing il tempo dell'ultima lettura e nell'array hit il successo della lettura. Infine la funzione termina restituendo 0.

3.3.4 La scelta dei tempi

La scelta dei tempi per i debounce è molto differente per i due tipi di lettura. Nel caso della lettura degli input digitali i tempi possono essere anche piuttosto lunghi, poiché non è necessario che il segnale mandato dall'utente arrivi al microcontrollore con tempestività. Molto diverso e molto più complesso è il caso degli input analogici: è fondamentale ridurre qualunque tipo di ritardo, se non quelli dovuti ai tempi di elaborazione.

La variabile debouceDelay è quindi impostata a 100, in modo che tra una lettura e l'altra passi almeno un decimo di secondo. Per la variabile release è stato scelto di impostarla a un valore di 75 millisecondi, che corrisponde, in ambito musicale, ad un valore inferiore alla durata di un sedicesimo ad una frequenza di 180 battiti per minuto.

3.4 I messaggi MIDI

Il dispositivo assegna a ogni sensore uno specifico messaggio MIDI memorizzato, tranne il sensore assegnato all'input A7, che funge da switch per la scelta di due diversi valori assegnati al sensore A1. A disposizione dell'utilizzatore ci sono quindi dieci banchi in cui sono memorizzate le combinazioni di messaggi da poter usare contemporaneamente.

Ogni banco può essere modificato, così come ogni singolo messaggio, attraverso un menù. Il banco consiste di un array bidimensionale in cui si susseguono uno status byte e due data byte.

```
byte patches[10][24] = {{SB00, DB01, DB02, …, SB021, DB022, DB023}
                                                    …
                        {SB90, DB91, DB92, …, SB921, DB922, DB923}}
```

3.4.2 Inviare messaggi

```
void midimsg (int pad, int velocity)
{
      if(pad == 2 && analogRead(pedal) > threshold) pad++;

      Serial1.write(patches[patch][pad*3]);
      Serial1.write(patches[patch][(pad*3)+1]);
      if (patches[patch][pad*3] == 0x99) {
                  velocity = map(velocity, threshold, 1023, 1,   127);
                  Serial1.write(velocity);
                  delay(velocity);
                  Serial1.write(patches[patch][pad*3]);
                  Serial1.write(patches[patch][(pad*3)+1]);
                  Serial1.write(Noteoff);
                  }
            else  Serial1.write(patches[patch][(pad*3)+2]);


      return;
}
```

Quando si attiva un sensore, il programma rileva il valore della differenza di potenziale attraverso il materiale piezoelettrico e filtra, come visto nel paragrafo precedente, le letture ritenute accidentali. Una volta scartati i valori, viene invocata la funzione `midimsg()` che richiede in ingresso quale sensore è stato attivato e qual è stata l'intensità dello stimolo ricevuto.

Il primo controllo serve a determinare se il sensore A7 (pedal) è premuto in concomitanza con il sensore A1 (pad 2). Se ciò avviene il valore di pad viene portato a tre, in modo che venga mandato un messaggio diverso. Il concetto è simile a quello che regola un hi-hat in una batteria acustica: se il pedale non è premuto, il suono è di tipo aperto, mentre se il pedale è premuto il suono dell'hi-hat è chiuso.

I messaggi vengono inviati attraverso la porta di comunicazione seriale attraverso la funzione `Serial.write` [s14]. Usando come indici per l'array la patch selezionata al momento (patch) e il numero del pad e spostandosi verso destra vengono inviati in sequenza lo Status Byte e il primo Data Byte. Per il secondo Data Byte viene fatto un distinguo a seconda del tipo di messaggio, analizzando il Byte di stato. Se il messaggio è un Note On sul canale 10[4], invece del secondo Data Byte memorizzato, la funzione trasmette il valore dell'intensità passatagli, rimappandolo tramite la funzione `map()` [s15], in modo che funga da velocity per la nota eseguita. La funzione restituisce il valore passatogli (velocity), compreso tra un valore minimo (threshold) e un valore massimo (1023), scalandolo proporzionalmente tra i due nuovi valori limite, inferiore (1) e superiore (127). Dopo aver atteso un tempo proporzionale all'intensità del colpo, viene inviato un messaggio di Note On identico ma con velocity pari a 0, in modo da "spegnere" la nota precedentemente inviata[5]. Nel caso in cui, invece, il messaggio sia di tipo Control Change, viene semplicemente inviato il secondo Data Byte memorizzato, dopo di che la funzione termina.

3.5 Gestione dei messaggi

Sono possibili numerose varianti all'interno della configurazione dei messaggi. Le modifiche sono accessibili grazie ad un menù interattivo che, grazie all'utilizzo del monitor seriale e dei pulsanti, permette una gestione rapida e diretta da parte dell'utilizzatore. Il sistema è stato implementato su diversi livelli e permette di modificare ogni singola parte dei messaggi,

[4] È il canale che lo standard General MIDI assegna ai suoni percussivi, in esadecimale il messaggio viene codificato come 0x99.

[5] Nel codice viene passato alla funzione la variabile Noteoff, che è inizializzata a 0, perché passare direttamente il valore crea un conflitto all'interno della funzione stessa per il quale essa non supera la fase di compilazione.

restando però fedeli alle specifiche MIDI, GM e a quelle particolari del progetto.

3.5.2 Cambiare patch e accedere al menù

All'interno della funzione loop(), che compone il corpo principale degli sketch di Arduino e viene ripetuta ciclicamente fino allo spegnimento della scheda, si può interagire con il microcontrollore attraverso i pulsanti up, down e edit. Quando viene rilevata la pressione di uno dei primi due tasti (quando la funzione `Read()` ritorna 0, vedi Par. 3.3.2) viene modificato il valore della variabile patch, che funge da indice per la scelta del banco di memoria da cui leggere. Questo permette di poter cambiare set di messaggi in modo rapido. In seguito al cambio di banco viene invocata anche la funzione `load()`, che permette di identificare in modo visivo i tipi di messaggi, accedendo di un colore diverso i led di controllo a seconda del tipo di messaggio associato al pad a cui fa riferimento[6]. Viene inoltre mostrato sul monitor seriale il nuovo valore di patch.

```
if (Read(up) == 0) {
    patch = (++patch) % 10;
    load(patch);
    Serial.println(patch + 1, DEC);
    delay(100);
  }
  if (Read(down) == 0) {
    patch = (patch--) % 10;
    if (patch == 255) patch = 9;
    load(patch);
    Serial.println(patch + 1, DEC);
    delay(100);
  }
  if (Read(edit) == 0) {
    editmenu();
    load(patch);

  }
```

Se viene premuto il tasto edit viene invocata la funzione `editmenu()`, che blocca la funzione principale finché non sarà terminata. Questo permette di agire sulle impostazioni dei messaggi senza che il dispositivo ne invii alcuno.

3.5.3 La scelta del parametro da modificare

Una volta entrati nella funzione `editmenu()`, si potrà, sempre con l'ausilio dei pulsanti up e down, scorrere tra le voci che possono essere

```
void editmenu() {
  Serial.println("Patch Editing");
  Serial.print("Edit ");
  Serial.println(menu[g_menu_pos]);
  while (true)
  {
    if (Read(esc) == 0) break;
    if (Read(up) == 0) {
      g_menu_pos = (++g_menu_pos) % 4;
      Serial.println(menu[g_menu_pos]);
    }
    if (Read(down) == 0) {
      g_menu_pos = (g_menu_pos--) % 4;
      if (g_menu_pos == 255) g_menu_pos = 3;
      //lcd.setCursor(5,1);
      Serial.println(menu[g_menu_pos]);
    }
    if (Read(edit) == 0) {
      pad_select(g_menu_pos);
      break;
    }
  };
  return;
}
```

[6] La funzione esegue in modo ciclico e condizionale il codice mostrato nel Par. 2.5.2

modificate della patch che era selezionata al momento dell'invocazione della funzione. Il tasto edit invoca a sua volta una nuova funzione `pad_select()`, a cui viene passato il riferimento a quale parametro si vuole modificare. Una volta terminata la modifica, il comando break farà terminare il loop all'interno del quale sono racchiuse le istruzioni necessarie per spostarsi nel menù. È anche possibile uscire dal loop utilizzando il pulsante esc, senza effettuare modifiche. Una volta usciti dal ciclo la funzione terminerà.

3.5.4 Selezionare i pad

Una volta scelto quale parametro si vuole modificare, è necessario indicare quale pad dovrà essere interessato dalle modifiche. Il metodo per scegliere il pad è sempre quello rappresentato dall'utilizzo dei pulsanti up e down e anche il pulsante esc permette di terminare la funzione. La particolarità è rappresentata dal tasto edit: una volta selezionato il pad, rappresentato dalla variabile npad, in base a quale parametro è stato passato in precedenza , viene invocata la funzione specifica per la modifica di quel parametro, passando alla selezionata il valore del pad interessato.

```
void pad_select(int position)
{
  byte npad = 0;
  while(true)
      {
      …
      //selezione del valore di npad
      …
      if (Read(edit) == 0){
                  switch(position){
                  case 0 : edittype(npad) ;
                  case 1 : editchannel(npad);
                  case 2 : editdata1(npad);
                  case 3 : editdata2(npad);
                  }
                  break;
      }
      };
      return;
}
```

3.5.5 Modificare il tipo di messaggio

La funzione `edittype()` permetterà di modificare il tipo di messaggio, scegliendo tra Note On su canale 10 (0x99) e Control Change sul canale 1 (0xB0). I tasti up e down serviranno a selezionare il tipo di messaggio, mentre il pulsante edit memorizzerà la nuova impostazione. Dopo quest'azione viene invocata una speciale funzione chiamata `reset()`. A questa funzione vengono passati il pad e il suo

```
void reset(byte spad, byte stype)
{
      if(stype == 0x99)
            {
                  patches[patch][spad+1] = 0x23;
            }
      else
            {
                  patches[patch][spad+1] = 0x14;
                  patches[patch][spad+2] = 0x00;
            }

      return;
}
```

nuovo tipo di messaggio. Essa provvede a settare gli altri i Data Byte al valore stabilito di default. Per i messaggi di Note On verrà impostato il timbro di Acoustic Bass Drum (0x23), mentre il secondo Data Byte verrà ignorato, mentre per i Control Change verrà salvato il valore di controllo 20 (0x14), che non è assegnato a nessuna funzione specifica nelle specifiche MIDI, e il secondo Data Byte viene settato a zero.

3.5.6 Modificare il canale di trasmissione dei messaggi

Se dal menù principale si è scelto di cambiare il canale MIDI sul quale è trasmesso il messaggio, viene invocata la funzione `editchannel()`, che prende in entrata il pad

```
void editchannel(int mpad)
{
  int j;
  byte nchannel = patches[patch][(mpad * 3)];

  for (j = 0; j < 4; j++ ) bitClear(nchannel, 7 - j);

  if (patches[patch][mpad * 3] == 0x99) {
    Serial.println("Canale non editabile per suoni di batteria");
    delay(2000);
    return;
  }

  else {

  Serial.println("Channel Editing");
  Serial.print("Channel : ");
  Serial.println(nchannel + 1, DEC);
  }

  while (true)
  {

    if (Read(esc) == 0) break;
    if (Read(up) == 0) {
      nchannel = (++nchannel) % 16;
      Serial.println(nchannel + 1, DEC);
    }
    if (Read(down) == 0) {
      nchannel = (nchannel--) % 16;
      if (nchannel == 255) nchannel = 15;
      Serial.println(nchannel + 1, DEC);
    }
    if (Read(edit) == 0) {

      patches[patch][(mpad * 3)] = 0xB0 | nchannel;
      break;
    }
  };
  return;

}
```

selezionato per la modifica. Tramite la funzione `bitClear()` [s16], si estrapola dal messaggio preso in considerazione solo l'informazione del canale, settando a 0 i bit rimanenti. Nel caso in cui il messaggio selezionato sia un Note On per batteria, la funzione mostra all'utente un messaggio di errore e termina senza effettuare modifiche.

Se il messaggio è invece un Control Change, è possibile scegliere quale valore assegnargli tramite i pulsanti up e down, mentre premendo il tasto edit, si memorizza il nuovo canale mediante un OR a bit a bit (|) tra l'identificatore dei Control Change nello Status Byte (0xB0) e il valore del canale stesso.

3.5.7 Modificare il primo Data Byte

Modificare il primo Data Byte è un'azione molto diversa secondo il tipo di messaggio cui si fa riferimento.

```
void editdata1(int mpad)
{
  if (patches[patch][(mpad * 4)] == 0x99) {
    byte mdata1 = patches[patch][(mpad * 3) + 1];
    Serial.println("Data byte 1");
    Serial.print("Value : ");
    Serial.println(mdata1, DEC);

    while (true) {
      if (Read(esc) == 0) break;
      if (Read(up) == 0) {
        mdata1 = (++mdata1) % 82;
        Serial.println(mdata1, DEC);}
      if (Read(down) == 0) {
        mdata1 = (mdata1--) % 82;
        if (mdata1 == 255) mdata1 = 81;
        Serial.println(mdata1, DEC);}
      if (Read(edit) == 0) {
        patches[patch][(mpad * 3) + 1] = mdata1;
        break;}
    };
  }
  else {
    byte mdata1 = patches[patch][(mpad * 3) + 1];
    Serial.println("Data byte 1");
    Serial.print("Value : ");
    Serial.println(mdata1, DEC);
    while (true) {
      if (Read(esc) == 0) break;
      if (Read(up) == 0) {
        mdata1 = (++mdata1) % 128;
        Serial.println(mdata1, DEC);}
      if (Read(down) == 0) {
        mdata1 = (mdata1--) % 128;
        if (mdata1 == 255) mdata1 = 127;
        Serial.println(mdata1, DEC);}
      if (Read(edit) == 0) {
        patches[patch][(mpad * 3) + 1] = mdata1;
        break;}
    };
  }
  return; }
```

Se si vuole modificare il primo Data Byte di un messaggio di Note On è stato ritenuto un approccio migliore attenersi alle specifiche General MIDI. Per questo motivo i valori possibili per Data Byte sono circoscritti a quelli compresi tra la nota B0 (Key #35, Acoustic Bass Drum) e la nota A4 (Key #81, Open Triangle).

Al contrario invece il primo Data Byte di un messaggio Control Change può assumere tutti i centoventotto valori possibili. Come già precedentemente illustrato, il valore può essere scelto tramite i tasti up e down, il salvataggio avviene tramite la pressione del pulsante edit, mentre premendo esc si esce dal menù senza modificare alcun elemento.

3.5.8 Modificare il secondo Data Byte

Anche la modifica del secondo Data Byte dipende strettamente dal tipo di messaggio. Nel caso sia esso un messaggio per batteria, però, così come avviene per il menù di scelta del canale, viene mostrato un messaggio di errore, in quanto il secondo Byte contiene la velocity della nota e viene fornita dai sensori piezoelettrici.

Nel caso invece di un messaggio di tipo Control Change è possibile assegnare un qualsivoglia valore compreso tra 0 e 127, così come per il primo Data Byte.

3.6 Scelte implementative relative ai messaggi

Per quanto riguarda le scelte relative all'invio e alla composizione dei messaggi, è stata fatta una distinzione tra i due tipi di messaggi coinvolti in questo progetto.

Per i messaggi di tipo Note On per batteria si è scelto di rispettare in modo pedissequo le specifiche General MIDI. Questo aumenta la compatibilità dello strumento realizzato con i dispositivi che sono nati per aderire a quel protocollo, che, di fatto, è nato all'inizio degli anni '90 del secolo scorso. Ciononostante, anche in prodotti di ampia diffusione, come alcune Digital Audio Workstation[7] , basano le proprie batterie elettroniche sui timbri GM. Altri casi, come il software Superior Drummer[8], usa una sua mappatura dei timbri, che si basa comunque sul GM ma non lo segue fedelmente.

Al contrario i Control Change sono implementati non per essere aderente pienamente allo standard MIDI. Questo non comporta modifiche nella sintassi dei messaggi ma non lega un particolare numero di Control Change a una funzione specifica. Questa è una scelta dovuta al fatto che, prendendo ancora in esame Ableton Live, in molte applicazioni software che coinvolgono messaggi MIDI, la mappatura dei messaggi avviene via software. Inoltre è stato

[7] Si porta come esempio Ableton Live: https://www.ableton.com/en/manual/instrument-drum-and-effect-racks/#drum-racks

[8] http://www.toontrack.com/tv.asp?channel=tutorials&item=146

deciso di non implementare la tecnica MSB + LSB, in quanto non sono presenti controlli continui che necessitano di una granularità sottile, ma di fatto tutti i pad possono essere intesi come trigger di messaggi MIDI.

CONSIDERAZIONI FINALI
E
SVILUPPI POSSIBILI

Lo strumento progettato e realizzato per questa tesi è perfettamente utilizzabile in una situazione di performance e la sua struttura potrebbe essere utilizzata come base per una possibile produzione commerciale. Esso si integra perfettamente con varie applicazioni che usano il MIDI come comunicazione, sia attraverso la porta DIN sia attraverso la comunicazione via USB.

Le possibilità di sviluppo che possono avere progetti come quello illustrato in questo elaborato sono molteplici. L'utilizzo di sensori piezoelettrici di dimensioni differenti e la loro sistemazione in strumenti dalle forme più disparate sono alcuni dei punti di forza che spingono il mercato delle periferiche MIDI. Un esempio è il progetto "Drum Pants" di Tyler Freeman[1], che ha sviluppato un prodotto che unisce le caratteristiche MIDI alla possibilità di usare il proprio corpo come un vero e proprio strumento. L'interesse del grande pubblico è dimostrato dal fatto che il progetto, proposto sulla piattaforma di crowfunding *kickstarter.com*, ha raccolto più del doppio dei finanziamenti richiesti per iniziare la produzione su grande scala.

La grande disponibilità di sensori per Arduino rende naturale pensare a strumenti che sfruttino diverse grandezze fisiche per innescare la generazione di messaggi MIDI. Si può pensare di usare un accelerometro che può per esempio sfruttare il movimento su più assi, oppure un sensore laser che utilizzi l'interruzione del raggio come trigger. Sempre tra le possibilità offerte da Arduino, si possono citare i numerosi shield che consentono di comunicare attraverso altrettanti protocolli: si può pensare ad esempio alla comunicazione via Bluetooth o Wireless, fino ad arrivare alle classiche USB.

L'utilizzo di Arduino è inoltre un ottimo punto di partenza per poi poter passare a livelli di produzione più ampia, creando dal prototipo originale una scheda personalizzata e abbattendo i tempi di sviluppo e ottimizzando poi quelli di produzione. Infatti, a differenza di molti altri prodotti più competitivi a livello produttivo, Arduino rimane notevolmente più semplice e didatticamente più intuitivo rispetto a un normale microcontrollore. I suoi punti di forza rimangono la sua duttilità e la possibilità di essere programmato direttamente in C, senza per forza conoscere il linguaggio Assembler di una particolare macchina.

[1] www.drumpants.com

Appendice

Tabella 1

Channel Voice Messages

STATUS BYTE		DATA BYTES	
1st Byte Value Binary |Hex| Dec	Function	2nd Byte	3rd Byte
10000000= 80= 128	Chan 1 Note off	Note Number (0-127)	Note Velocity (0-127)
10000001= 81= 129	Chan 2 Note off	Note Number (0-127)	Note Velocity (0-127)
10000010= 82= 130	Chan 3 Note off	Note Number (0-127)	Note Velocity (0-127)
10000011= 83= 131	Chan 4 Note off	Note Number (0-127)	Note Velocity (0-127)
10000100= 84= 132	Chan 5 Note off	Note Number (0-127)	Note Velocity (0-127)
10000101= 85= 133	Chan 6 Note off	Note Number (0-127)	Note Velocity (0-127)
10000110= 86= 134	Chan 7 Note off	Note Number (0-127)	Note Velocity (0-127)
10000111= 87= 135	Chan 8 Note off	Note Number (0-127)	Note Velocity (0-127)
10001000= 88= 136	Chan 9 Note off	Note Number (0-127)	Note Velocity (0-127)
10001001= 89= 137	Chan 10 Note off	Note Number (0-127)	Note Velocity (0-127)
10001010= 8A= 138	Chan 11 Note off	Note Number (0-127)	Note Velocity (0-127)
10001011= 8B= 139	Chan 12 Note off	Note Number (0-127)	Note Velocity (0-127)
10001100= 8C= 140	Chan 13 Note off	Note Number (0-127)	Note Velocity (0-127)
10001101= 8D= 141	Chan 14 Note off	Note Number (0-127)	Note Velocity (0-127)
10001110= 8E= 142	Chan 15 Note off	Note Number (0-127)	Note Velocity (0-127)
10001111= 8F= 143	Chan 16 Note off	Note Number (0-127)	Note Velocity (0-127)
10010000= 90= 144	Chan 1 Note on	Note Number (0-127)	Note Velocity (0-127)
10010001= 91= 145	Chan 2 Note on	Note Number (0-127)	Note Velocity (0-127)
10010010= 92= 146	Chan 3 Note on	Note Number (0-127)	Note Velocity (0-127)
10010011= 93= 147	Chan 4 Note on	Note Number (0-127)	Note Velocity (0-127)
10010100= 94= 148	Chan 5 Note on	Note Number (0-127)	Note Velocity (0-127)
10010101= 95= 149	Chan 6 Note on	Note Number (0-127)	Note Velocity (0-127)
10010110= 96= 150	Chan 7 Note on	Note Number (0-127)	Note Velocity (0-127)
10010111= 97= 151	Chan 8 Note on	Note Number (0-127)	Note Velocity (0-127)
10011000= 98= 152	Chan 9 Note on	Note Number (0-127)	Note Velocity (0-127)
10011001= 99= 153	Chan 10 Note on	Note Number (0-127)	Note Velocity (0-127)
10011010= 9A= 154	Chan 11 Note on	Note Number (0-127)	Note Velocity (0-127)
10011011= 9B= 155	Chan 12 Note on	Note Number (0-127)	Note Velocity (0-127)
10011100= 9C= 156	Chan 13 Note on	Note Number (0-127)	Note Velocity (0-127)
10011101= 9D= 157	Chan 14 Note on	Note Number (0-127)	Note Velocity (0-127)
10011110= 9E= 158	Chan 15 Note on	Note Number (0-127)	Note Velocity (0-127)
10011111= 9F= 159	Chan 16 Note on	Note Number (0-127)	Note Velocity (0-127)
10100000= A0= 160	Chan 1 Polyphonic Aftertouch	Note Number (0-127)	Pressure (0-127)
10100001= A1= 161	Chan 2 Polyphonic Aftertouch	Note Number (0-127)	Pressure (0-127)
10100010= A2= 162	Chan 3 Polyphonic Aftertouch	Note Number (0-127)	Pressure (0-127)

10100011= A3= 163	Chan 4 Polyphonic Aftertouch	Note Number (0-127)	Pressure (0-127)
10100100= A4= 164	Chan 5 Polyphonic Aftertouch	Note Number (0-127)	Pressure (0-127)
10100101= A5= 165	Chan 6 Polyphonic Aftertouch	Note Number (0-127)	Pressure (0-127)
10100110= A6= 166	Chan 7 Polyphonic Aftertouch	Note Number (0-127)	Pressure (0-127)
10100111= A7= 167	Chan 8 Polyphonic Aftertouch	Note Number (0-127)	Pressure (0-127)
10101000= A8= 168	Chan 9 Polyphonic Aftertouch	Note Number (0-127)	Pressure (0-127)
10101001= A9= 169	Chan 10 Polyphonic Aftertouch	Note Number (0-127)	Pressure (0-127)
10101010= AA= 170	Chan 11 Polyphonic Aftertouch	Note Number (0-127)	Pressure (0-127)
10101011= AB= 171	Chan 12 Polyphonic Aftertouch	Note Number (0-127)	Pressure (0-127)
10101100= AC= 172	Chan 13 Polyphonic Aftertouch	Note Number (0-127)	Pressure (0-127)
10101101= AD= 173	Chan 14 Polyphonic Aftertouch	Note Number (0-127)	Pressure (0-127)
10101110= AE= 174	Chan 15 Polyphonic Aftertouch	Note Number (0-127)	Pressure (0-127)
10101111= AF= 175	Chan 16 Polyphonic Aftertouch	Note Number (0-127)	Pressure (0-127)
11000000= C0= 192	Chan 1 Program Change	Program # (0-127)	none
11000001= C1= 193	Chan 2 Program Change	Program # (0-127)	none
11000010= C2= 194	Chan 3 Program Change	Program # (0-127)	none
11000011= C3= 195	Chan 4 Program Change	Program # (0-127)	none
11000100= C4= 196	Chan 5 Program Change	Program # (0-127)	none
11000101= C5= 197	Chan 6 Program Change	Program # (0-127)	none
11000110= C6= 198	Chan 7 Program Change	Program # (0-127)	none
11000111= C7= 199	Chan 8 Program Change	Program # (0-127)	none
11001000= C8= 200	Chan 9 Program Change	Program # (0-127)	none
11001001= C9= 201	Chan 10 Program Change	Program # (0-127)	none
11001010= CA= 202	Chan 11 Program Change	Program # (0-127)	none
11001011= CB= 203	Chan 12 Program Change	Program # (0-127)	none
11001100= CC= 204	Chan 13 Program Change	Program # (0-127)	none
11001101= CD= 205	Chan 14 Program Change	Program # (0-127)	none
11001110= CE= 206	Chan 15 Program Change	Program # (0-127)	none
11001111= CF= 207	Chan 16 Program Change	Program # (0-127)	none
11010000= D0= 208	Chan 1 Channel Aftertouch	Pressure (0-127)	none
11010001= D1= 209	Chan 2 Channel Aftertouch	Pressure (0-127)	none
11010010= D2= 210	Chan 3 Channel Aftertouch	Pressure (0-127)	none

11010011= D3= 211	Chan 4 Channel Aftertouch	Pressure (0-127)	none
11010100= D4= 212	Chan 5 Channel Aftertouch	Pressure (0-127)	none
11010101= D5= 213	Chan 6 Channel Aftertouch	Pressure (0-127)	none
11010110= D6= 214	Chan 7 Channel Aftertouch	Pressure (0-127)	none
11010111= D7= 215	Chan 8 Channel Aftertouch	Pressure (0-127)	none
11011000= D8= 216	Chan 9 Channel Aftertouch	Pressure (0-127)	none
11011001= D9= 217	Chan 10 Channel Aftertouch	Pressure (0-127)	none
11011010= DA= 218	Chan 11 Channel Aftertouch	Pressure (0-127)	none
11011011= DB= 219	Chan 12 Channel Aftertouch	Pressure (0-127)	none
11011100= DC= 220	Chan 13 Channel Aftertouch	Pressure (0-127)	none
11011101= DD= 221	Chan 14 Channel Aftertouch	Pressure (0-127)	none
11011110= DE= 222	Chan 15 Channel Aftertouch	Pressure (0-127)	none
11011111= DF= 223	Chan 16 Channel Aftertouch	Pressure (0-127)	none
11100000= E0= 224	Chan 1 Pitch Wheel Control	Pitch Wheel LSB (0-127)	Pitch Wheel MSB (0-127)
11100001= E1= 225	Chan 2 Pitch Wheel Control	Pitch Wheel LSB (0-127)	Pitch Wheel MSB (0-127)
11100010= E2= 226	Chan 3 Pitch Wheel Control	Pitch Wheel LSB (0-127)	Pitch Wheel MSB (0-127)
11100011= E3= 227	Chan 4 Pitch Wheel Control	Pitch Wheel LSB (0-127)	Pitch Wheel MSB (0-127)
11100100= E4= 228	Chan 5 Pitch Wheel Control	Pitch Wheel LSB (0-127)	Pitch Wheel MSB (0-127)
11100101= E5= 229	Chan 6 Pitch Wheel Control	Pitch Wheel LSB (0-127)	Pitch Wheel MSB (0-127)
11100110= E6= 230	Chan 7 Pitch Wheel Control	Pitch Wheel LSB (0-127)	Pitch Wheel MSB (0-127)
11100111= E7= 231	Chan 8 Pitch Wheel Control	Pitch Wheel LSB (0-127)	Pitch Wheel MSB (0-127)
11101000= E8= 232	Chan 9 Pitch Wheel Control	Pitch Wheel LSB (0-127)	Pitch Wheel MSB (0-127)
11101001= E9= 233	Chan 10 Pitch Wheel Control	Pitch Wheel LSB (0-127)	Pitch Wheel MSB (0-127)
11101010= EA= 234	Chan 11 Pitch Wheel Control	Pitch Wheel LSB (0-127)	Pitch Wheel MSB (0-127)
11101011= EB= 235	Chan 12 Pitch Wheel Control	Pitch Wheel LSB (0-127)	Pitch Wheel MSB (0-127)
11101100= EC= 236	Chan 13 Pitch Wheel Control	Pitch Wheel LSB (0-127)	Pitch Wheel MSB (0-127)
11101101= ED= 237	Chan 14 Pitch Wheel Control	Pitch Wheel LSB (0-127)	Pitch Wheel MSB (0-127)
11101110= EE= 238	Chan 15 Pitch Wheel Control	Pitch Wheel LSB (0-127)	Pitch Wheel MSB (0-127)
11101111= EF= 239	Chan 16 Pitch Wheel Control	Pitch Wheel LSB (0-127)	Pitch Wheel MSB (0-127)

Tabella 2.1

Channel Voice Control Change

Status Byte

STATUS BYTE		DATA BYTES	
1st Byte Value Binary |Hex| Dec	Function	2nd Byte	3rd Byte
10110000= B0= 176	Chan 1 Control/Mode Change	Table 2.2 & 3	Table 2.2 & 3
10110001= B1= 177	Chan 2 Control/Mode Change	Table 2.2 & 3	Table 2.2 & 3
10110010= B2= 178	Chan 3 Control/Mode Change	Table 2.2 & 3	Table 2.2 & 3
10110011= B3= 179	Chan 4 Control/Mode Change	Table 2.2 & 3	Table 2.2 & 3
10110100= B4= 180	Chan 5 Control/Mode Change	Table 2.2 & 3	Table 2.2 & 3
10110101= B5= 181	Chan 6 Control/Mode Change	Table 2.2 & 3	Table 2.2 & 3
10110110= B6= 182	Chan 7 Control/Mode Change	Table 2.2 & 3	Table 2.2 & 3
10110111= B7= 183	Chan 8 Control/Mode Change	Table 2.2 & 3	Table 2.2 & 3
10111000= B8= 184	Chan 9 Control/Mode Change	Table 2.2 & 3	Table 2.2 & 3
10111001= B9= 185	Chan 10 Control/Mode Change	Table 2.2 & 3	Table 2.2 & 3
10111010= BA= 186	Chan 11 Control/Mode Change	Table 2.2 & 3	Table 2.2 & 3
10111011= BB= 187	Chan 12 Control/Mode Change	Table 2.2 & 3	Table 2.2 & 3
10111100= BC= 188	Chan 13 Control/Mode Change	Table 2.2 & 3	Table 2.2 & 3
10111101= BD= 189	Chan 14 Control/Mode Change	Table 2.2 & 3	Table 2.2 & 3
10111110= BE= 190	Chan 15 Control/Mode Change	Table 2.2 & 3	Table 2.2 & 3
10111111= BF= 191	Chan 16 Control/Mode Change	Table 2.2 & 3	Table 2.2 & 3

Tabella 2.2

Channel Voice Control Change

Data Byte

Control Number (2nd Byte Value)			**Control Function**	**3rd Byte Value**	
Decimal	**Binary**	**Hex**		**Value**	**Used As**
0	00000000	00	Bank Select	0-127	MSB
1	00000001	01	Modulation Wheel or Lever	0-127	MSB
2	00000010	02	Breath Controller	0-127	MSB
3	00000011	03	Undefined	0-127	MSB
4	00000100	04	Foot Controller	0-127	MSB
5	00000101	05	Portamento Time	0-127	MSB
6	00000110	06	Data Entry MSB	0-127	MSB
7	00000111	07	Channel Volume (formerly Main Volume)	0-127	MSB
8	00001000	08	Balance	0-127	MSB
9	00001001	09	Undefined	0-127	MSB
10	00001010	0A	Pan	0-127	MSB
11	00001011	0B	Expression Controller	0-127	MSB

12	00001100	0C	Effect Control 1	0-127	MSB
13	00001101	0D	Effect Control 2	0-127	MSB
14	00001110	0E	Undefined	0-127	MSB
15	00001111	0F	Undefined	0-127	MSB
16	00010000	10	General Purpose Controller 1	0-127	MSB
17	00010001	11	General Purpose Controller 2	0-127	MSB
18	00010010	12	General Purpose Controller 3	0-127	MSB
19	00010011	13	General Purpose Controller 4	0-127	MSB
20	00010100	14	Undefined	0-127	MSB
21	00010101	15	Undefined	0-127	MSB
22	00010110	16	Undefined	0-127	MSB
23	00010111	17	Undefined	0-127	MSB
24	00011000	18	Undefined	0-127	MSB
25	00011001	19	Undefined	0-127	MSB
26	00011010	1A	Undefined	0-127	MSB
27	00011011	1B	Undefined	0-127	MSB
28	00011100	1C	Undefined	0-127	MSB
29	00011101	1D	Undefined	0-127	MSB
30	00011110	1E	Undefined	0-127	MSB
31	00011111	1F	Undefined	0-127	MSB
32	00100000	20	LSB for Control 0 (Bank Select)	0-127	LSB
33	00100001	21	LSB for Control 1 (Modulation Wheel or Lever)	0-127	LSB
34	00100010	22	LSB for Control 2 (Breath Controller)	0-127	LSB
35	00100011	23	LSB for Control 3 (Undefined)	0-127	LSB
36	00100100	24	LSB for Control 4 (Foot Controller)	0-127	LSB
37	00100101	25	LSB for Control 5 (Portamento Time)	0-127	LSB
38	00100110	26	LSB for Control 6 (Data Entry)	0-127	LSB
39	00100111	27	LSB for Control 7 (Channel Volume, formerly Main Volume)	0-127	LSB
40	00101000	28	LSB for Control 8 (Balance)	0-127	LSB
41	00101001	29	LSB for Control 9 (Undefined)	0-127	LSB
42	00101010	2A	LSB for Control 10 (Pan)	0-127	LSB
43	00101011	2B	LSB for Control 11 (Expression Controller)	0-127	LSB
44	00101100	2C	LSB for Control 12 (Effect control 1)	0-127	LSB
45	00101101	2D	LSB for Control 13 (Effect control 2)	0-127	LSB
46	00101110	2E	LSB for Control 14 (Undefined)	0-127	LSB
47	00101111	2F	LSB for Control 15 (Undefined)	0-127	LSB
48	00110000	30	LSB for Control 16 (General Purpose Controller 1)	0-127	LSB
49	00110001	31	LSB for Control 17 (General Purpose Controller 2)	0-127	LSB
50	00110010	32	LSB for Control 18 (General Purpose Controller 3)	0-127	LSB
51	00110011	33	LSB for Control 19 (General Purpose Controller 4)	0-127	LSB
52	00110100	34	LSB for Control 20 (Undefined)	0-127	LSB
53	00110101	35	LSB for Control 21 (Undefined)	0-127	LSB
54	00110110	36	LSB for Control 22 (Undefined)	0-127	LSB
55	00110111	37	LSB for Control 23 (Undefined)	0-127	LSB
56	00111000	38	LSB for Control 24 (Undefined)	0-127	LSB
57	00111001	39	LSB for Control 25 (Undefined)	0-127	LSB
58	00111010	3A	LSB for Control 26 (Undefined)	0-127	LSB
59	00111011	3B	LSB for Control 27 (Undefined)	0-127	LSB
60	00111100	3C	LSB for Control 28 (Undefined)	0-127	LSB
61	00111101	3D	LSB for Control 29 (Undefined)	0-127	LSB
62	00111110	3E	LSB for Control 30 (Undefined)	0-127	LSB
63	00111111	3F	LSB for Control 31 (Undefined)	0-127	LSB
64	01000000	40	Damper Pedal on/off (Sustain)	≤63 off, ≥64 on	---
65	01000001	41	Portamento On/Off	≤63 off, ≥64 on	---
66	01000010	42	Sostenuto On/Off	≤63 off, ≥64 on	---
67	01000011	43	Soft Pedal On/Off	≤63 off, ≥64 on	---
68	01000100	44	Legato Footswitch	≤63 Normal, ≥64 Legato	---
69	01000101	45	Hold 2	≤63 off, ≥64 on	---

70	01000110	46	Sound Controller 1 (default: Sound Variation)	0-127	LSB
71	01000111	47	Sound Controller 2 (default: Timbre/Harmonic Intens.)	0-127	LSB
72	01001000	48	Sound Controller 3 (default: Release Time)	0-127	LSB
73	01001001	49	Sound Controller 4 (default: Attack Time)	0-127	LSB
74	01001010	4A	Sound Controller 5 (default: Brightness)	0-127	LSB
75	01001011	4B	Sound Controller 6 (default: Decay Time - see MMA RP-021)	0-127	LSB
76	01001100	4C	Sound Controller 7 (default: Vibrato Rate - see MMA RP-021)	0-127	LSB
77	01001101	4D	Sound Controller 8 (default: Vibrato Depth - see MMA RP-021)	0-127	LSB
78	01001110	4E	Sound Controller 9 (default: Vibrato Delay - see MMA RP-021)	0-127	LSB
79	01001111	4F	Sound Controller 10 (default undefined - see MMA RP-021)	0-127	LSB
80	01010000	50	General Purpose Controller 5	0-127	LSB
81	01010001	51	General Purpose Controller 6	0-127	LSB
82	01010010	52	General Purpose Controller 7	0-127	LSB
83	01010011	53	General Purpose Controller 8	0-127	LSB
84	01010100	54	Portamento Control	0-127	LSB
85	01010101	55	Undefined	---	---
86	01010110	56	Undefined	---	---
87	01010111	57	Undefined	---	---
88	01011000	58	High Resolution Velocity Prefix	0-127	LSB
89	01011001	59	Undefined	---	---
90	01011010	5A	Undefined	---	---
91	01011011	5B	Effects 1 Depth (default: Reverb Send Level - see MMA RP-023) (formerly External Effects Depth)	0-127	---
92	01011100	5C	Effects 2 Depth (formerly Tremolo Depth)	0-127	---
93	01011101	5D	Effects 3 Depth (default: Chorus Send Level - see MMA RP-023) (formerly Chorus Depth)	0-127	---
94	01011110	5E	Effects 4 Depth (formerly Celeste [Detune] Depth)	0-127	---
95	01011111	5F	Effects 5 Depth (formerly Phaser Depth)	0-127	---
96	01100000	60	Data Increment (Data Entry +1) (see MMA RP-018)	N/A	---
97	01100001	61	Data Decrement (Data Entry -1) (see MMA RP-018)	N/A	---
98	01100010	62	Non-Registered Parameter Number (NRPN) - LSB	0-127	LSB
99	01100011	63	Non-Registered Parameter Number (NRPN) - MSB	0-127	MSB
100	01100100	64	Registered Parameter Number (RPN) - LSB*	0-127	LSB
101	01100101	65	Registered Parameter Number (RPN) - MSB*	0-127	MSB
102	01100110	66	Undefined	---	---
103	01100111	67	Undefined	---	---
104	01101000	68	Undefined	---	---
105	01101001	69	Undefined	---	---
106	01101010	6A	Undefined	---	---
107	01101011	6B	Undefined	---	---
108	01101100	6C	Undefined	---	---
109	01101101	6D	Undefined	---	---
110	01101110	6E	Undefined	---	---
111	01101111	6F	Undefined	---	---

112	01110000	70	Undefined	---	---
113	01110001	71	Undefined	---	---
114	01110010	72	Undefined	---	---
115	01110011	73	Undefined	---	---
116	01110100	74	Undefined	---	---
117	01110101	75	Undefined	---	---
118	01110110	76	Undefined	---	---
119	01110111	77	Undefined	---	---

Tabella 3

Channel Mode Control Change

Data Byte

Control Number (2nd Byte Value)			**Control Function**	**3rd Byte Value**	
Decimal	**Binary**	**Hex**		**Value**	**Used As**
120	01111000	78	[Channel Mode Message] All Sound Off	0	---
121	01111001	79	[Channel Mode Message] Reset All Controllers (See MMA RP-015)	0	---
122	01111010	7A	[Channel Mode Message] Local Control On/Off	0 off, 127 on	---
123	01111011	7B	[Channel Mode Message] All Notes Off	0	---
124	01111100	7C	[Channel Mode Message] Omni Mode Off (+ all notes off)	0	---
125	01111101	7D	[Channel Mode Message] Omni Mode On (+ all notes off)	0	---
126	01111110	7E	[Channel Mode Message] Mono Mode On (+ poly off, + all notes off)	Note: This equals the number of channels, or zero if the number of channels equals the number of voices in the receiver.	---
127	01111111	7F	[Channel Mode Message] Poly Mode On (+ mono off, +all notes off)	0	---

Tabella 4

System Common Messages

STATUS BYTE		DATA BYTES	
1st Byte Value Binary |Hex| Dec	Function	2nd Byte	3rd Byte
11110000= F0= 240	System Exclusive	**	**
11110001= F1= 241	MIDI Time Code Qtr. Frame	-see spec-	-see spec-
11110010= F2= 242	Song Position Pointer	LSB	MSB
11110011= F3= 243	Song Select (Song #)	(0-127)	none
11110100= F4= 244	Undefined (Reserved)	---	---
11110101= F5= 245	Undefined (Reserved)	---	---
11110110= F6= 246	Tune request	none	none
11110111= F7= 247	End of SysEx (EOX)	none	none
11111000= F8= 248	Timing clock	none	none
11111001= F9= 249	Undefined (Reserved)	---	---
11111010= FA= 250	Start	none	none
11111011= FB= 251	Continue	none	none
11111100= FC= 252	Stop	none	none
11111101= FD= 253	Undefined (Reserved)	---	---
11111110= FE= 254	Active Sensing	none	none
11111111= FF= 255	System Reset	none	none

Tabella 5

Universal System Exclusive

Non-Real Time (7EH)			
SUB-ID #1	**SUB-ID #2**	**DESCRIPTION**	
00		Unused	
01		Sample Dump Header	
02		Sample Data Packet	
03		Sample Dump Request	
04	nn	MIDI Time Code	
	00		Special
	01		Punch In Points
	02		Punch Out Points
	03		Delete Punch In Point
	04		Delete Punch Out Point
	05		Event Start Point
	06		Event Stop Point
	07		Event Start Points with additional info.
	08		Event Stop Points with additional info.
	09		Delete Event Start Point
	0A		Delete Event Stop Point
	0B		Cue Points
	0C		Cue Points with additional info.
	0D		Delete Cue Point
	0E		Event Name in additional info.
05	nn	Sample Dump Extensions	
	01		Loop Points Transmission
	02		Loop Points Request
	03		Sample Name Transmission
	04		Sample Name Request
	05		Extended Dump Header
	06		Extended Loop Points Transmission
	07		Extended Loop Points Request
06	nn	General Information	
	01		Identity Request
	02		Identity Reply
07	nn	File Dump	
	01		Header
	02		Data Packet
	03		Request
08	nn	MIDI Tuning Standard (Non-Real Time)	
	00		Bulk Dump Request
	01		Bulk Dump Reply
	03		Tuning Dump Request
	04		Key-Based Tuning Dump
	05		Scale/Octave Tuning Dump, 1 byte format
	06		Scale/Octave Tuning Dump, 2 byte format
	07		Single Note Tuning Change with Bank Select
	08		Scale/Octave Tuning, 1 byte format
	09		Scale/Octave Tuning, 2 byte format
09	nn	General MIDI	
	01		General MIDI 1 System On
	02		General MIDI System Off
	03		General MIDI 2 System On
0A	nn	Downloadable Sounds	
	01		Turn DLS On
	02		Turn DLS Off
	03		Turn DLS Voice Allocation Off

	04		Turn DLS Voice Allocation On
0B	nn	File Reference Message	
	00		reserved (do not use)
	01		Open File
	02		Select or Reselect Contents
	03		Open File and Select Contents
	04		Close File
	05-7F		reserved (do not use)
0C	nn	MIDI Visual Control	
	00-7F		MVC Commands *(See MVC Documentation)*
7B	--	End of File	
7C	--	Wait	
7D	--	Cancel	
7E	--	NAK	
7F	--	ACK	
Real Time (7FH)			
SUB-ID #1	**SUB-ID #2**	**DESCRIPTION**	
00	--	Unused	
01	nn	MIDI Time Code	
	01		Full Message
	02		User Bits
02	nn	MIDI Show Control	
	00		MSC Extensions
	01-7F		MSC Commands *(see MSC Documentation)*
03	nn	Notation Information	
	01		Bar Number
	02		Time Signature (Immediate)
	42		Time Signature (Delayed)
04	nn	Device Control	
	01		Master Volume
	02		Master Balance
	03		Master Fine Tuning
	04		Master Course Tuning
	05		Global Parameter Control
05	nn	Real Time MTC Cueing	
	00		Special
	01		Punch In Points
	02		Punch Out Points
	03		(Reserved)
	04		(Reserved)
	05		Event Start points
	06		Event Stop points
	07		Event Start points with additional info.
	08		Event Stop points with additional info.
	09		(Reserved)
	0A		(Reserved)
	0B		Cue points
	0C		Cue points with additional info.
	0D		(Reserved)
	0E		Event Name in additional info.
06	nn	MIDI Machine Control Commands	
	00-7F		MMC Commands *(See MMC Documentation)*
07	nn	MIDI Machine Control Responses	
	00-7F		MMC Responses *(See MMC Documentation)*
08	nn	MIDI Tuning Standard (Real Time)	
	02		Single Note Tuning Change
	07		Single Note Tuning Change with Bank Select
	08		Scale/Octave Tuning, 1 byte format
	09		Scale/Octave Tuning, 2 byte format
09	nn	Controller Destination Setting *(See GM2 Documentation)*	
	01		Channel Pressure (Aftertouch)

	02		Polyphonic Key Pressure (Aftertouch)
	03		Controller (Control Change)
0A	01	Key-based Instrument Control	
0B	01	Scalable Polyphony MIDI MIP Message	
0C	00	Mobile Phone Control Message	

Tabella 6.1

General MIDI Instrument Patch Map

PC#	Instrument Name	PC#	Instrument Name
1.	Acoustic Grand Piano	65.	Soprano Sax
2.	Bright Acoustic Piano	66.	Alto Sax
3.	Electric Grand Piano	67.	Tenor Sax
4.	Honky-tonk Piano	68.	Baritone Sax
5.	Electric Piano 1	69.	Oboe
6.	Electric Piano 2	70.	English Horn
7.	Harpsichord	71.	Bassoon
8.	Clavi	72.	Clarinet
9.	Celesta	73.	Piccolo
10.	Glockenspiel	74.	Flute
11.	Music Box	75.	Recorder
12.	Vibraphone	76.	Pan Flute
13.	Marimba	77.	Blown Bottle
14.	Xylophone	78.	Shakuhachi
15.	Tubular Bells	79.	Whistle
16.	Dulcimer	80.	Ocarina
17.	Drawbar Organ	81.	Lead 1 (square)
18.	Percussive Organ	82.	Lead 2 (sawtooth)
19.	Rock Organ	83.	Lead 3 (calliope)
20.	Church Organ	84.	Lead 4 (chiff)
21.	Reed Organ	85.	Lead 5 (charang)
22.	Accordion	86.	Lead 6 (voice)
23.	Harmonica	87.	Lead 7 (fifths)
24.	Tango Accordion	88.	Lead 8 (bass + lead)
25.	Acoustic Guitar (nylon)	89.	Pad 1 (new age)
26.	Acoustic Guitar (steel)	90.	Pad 2 (warm)
27.	Electric Guitar (jazz)	91.	Pad 3 (polysynth)
28.	Electric Guitar (clean)	92.	Pad 4 (choir)
29.	Electric Guitar (muted)	93.	Pad 5 (bowed)
30.	Overdriven Guitar	94.	Pad 6 (metallic)
31.	Distortion Guitar	95.	Pad 7 (halo)
32.	Guitar harmonics	96.	Pad 8 (sweep)
33.	Acoustic Bass	97.	FX 1 (rain)
34.	Electric Bass (finger)	98.	FX 2 (soundtrack)
35.	Electric Bass (pick)	99.	FX 3 (crystal)
36.	Fretless Bass	100.	FX 4 (atmosphere)
37.	Slap Bass 1	101.	FX 5 (brightness)
38.	Slap Bass 2	102.	FX 6 (goblins)
39.	Synth Bass 1	103.	FX 7 (echoes)
40.	Synth Bass 2	104.	FX 8 (sci-fi)
41.	Violin	105.	Sitar
42.	Viola	106.	Banjo
43.	Cello	107.	Shamisen
44.	Contrabass	108.	Koto
45.	Tremolo Strings	109.	Kalimba
46.	Pizzicato Strings	110.	Bag pipe
47.	Orchestral Harp	111.	Fiddle
48.	Timpani	112.	Shanai
49.	String Ensemble 1	113.	Tinkle Bell
50.	String Ensemble 2	114.	Agogo
51.	SynthStrings 1	115.	Steel Drums
52.	SynthStrings 2	116.	Woodblock
53.	Choir Aahs	117.	Taiko Drum
54.	Voice Oohs	118.	Melodic Tom
55.	Synth Voice	119.	Synth Drum
56.	Orchestra Hit	120.	Reverse Cymbal
57.	Trumpet	121.	Guitar Fret Noise
58.	Trombone	122.	Breath Noise

59.	Tuba	123.	Seashore
60.	Muted Trumpet	124.	Bird Tweet
61.	French Horn	125.	Telephone Ring
62.	Brass Section	126.	Helicopter
63.	SynthBrass 1	127.	Applause
64.	SynthBrass 2	128.	Gunshot

Tabella 6.2

General MIDI Percussion Key Map

Key #	**Drum Sound**	**Key#**	**Drum Sound**
35	Acoustic Bass Drum	59	Ride Cymbal 2
36	Bass Drum 1	60	Hi Bongo
37	Side Stick	61	Low Bongo
38	Acoustic Snare	62	Mute Hi Conga
39	Hand Clap	63	Open Hi Conga
40	Electric Snare	64	Low Conga
41	Low Floor Tom	65	High Timbale
42	Closed Hi Hat	66	Low Timbale
43	High Floor Tom	67	High Agogo
44	Pedal Hi-Hat	68	Low Agogo
45	Low Tom	69	Cabasa
46	Open Hi-Hat	70	Maracas
47	Low-Mid Tom	71	Short Whistle
48	Hi-Mid Tom	72	Long Whistle
49	Crash Cymbal 1	73	Short Guiro
50	High Tom	74	Long Guiro
51	Ride Cymbal 1	75	Claves
52	Chinese Cymbal	76	Hi Wood Block
53	Ride Bell	77	Low Wood Block
54	Tambourine	78	Mute Cuica
55	Splash Cymbal	79	Open Cuica
56	Cowbell	80	Mute Triangle
57	Crash Cymbal 2	81	Open Triangle
58	Vibraslap		

Tabella 7

SMF Meta Events

Type	Event	Type	Event
0x00	Sequence number	0x20	MIDI channel prefix assignment
0x01	Text event	0x2F	End of track
0x02	Copyright notice	0x51	Tempo setting
0x03	Sequence or track name	0x54	SMPTE offset
0x04	Instrument name	0x58	Time signature
0x05	Lyric text	0x59	Key signature
0x06	Marker text	0x7F	Sequencer specific event
0x07	Cue point		

Codice dello sketch Arduino

```
//LIBRARIES

#include <Math.h>

//DECLARATION

const int pad1 = A0;
const int pad2 = A1;
const int pad3 = A2;
const int pad4 = A3;
const int pad5 = A4;
const int pad6 = A5;
const int pad7 = A6;

const int pedal = A7

const int up = 7;
const int down = 8;
const int edit = 9;
const int esc = 10;

const int led11 = 23;
const int led12 = 25;
const int led21 = 27;
const int led22 = 29;
const int led31 = 31;
const int led32 = 33;
const int led41 = 35;
const int led42 = 37;
const int led51 = 39;
const int led52 = 41;
const int led61 = 43;
const int led62 = 45;
const int led71 = 47;
const int led72 = 49;
const int led81 = 51;
const int led82 = 53;

long lastDebounceTime[4] = {0};
long debounceDelay = 100;
```

```
int buttonState[4];
int lastButtonState[4] = {LOW};

int threshold = 150;
boolean hit[8] = {false};
int Noteoff = 0;

int v1 = 0;
int v2 = 0;
int v3 = 0;
int v4 = 0;
int v5 = 0;
int v6 = 0;
int v7 = 0;

long timing[8] = {0};
const long release = 75;

char* menu[] = {"type", "channel", "databyte1", "databyte2"};

byte g_menu_pos = 0;
byte patch;

//PATCH

byte patches[10][24] = {
{0x99, 0x24, 0x00, 0x99, 0x2E, 0x00, 0x99, 0x2A, 0x00, 0x99, 0x26, 0x00,
0x99, 0x31, 0x00, 0x99, 0x32, 0x00, 0x99, 0x2D, 0x00, 0x99, 0x33, 0x00},

{0x99, 0x24, 0x00, 0x99, 0x27, 0x00, 0x99, 0x38, 0x00, 0x99, 0x28, 0x00,
0x99, 0x30, 0x00, 0x99, 0x32, 0x00, 0x99, 0x35, 0x00, 0x99, 0x34, 0x00},

{0x99, 0x3C, 0x00, 0x99, 0x3D, 0x00, 0x99, 0x3E, 0x00, 0x99, 0x3F, 0x00,
0x99, 0x40, 0x00, 0x99, 0x41, 0x00, 0x99, 0x42, 0x00, 0x99, 0x43, 0x00},

{0xB0, 0x14, 0x00, 0xB0, 0x15, 0x7F, 0xB0, 0x16, 0x7F, 0xB0, 0x17, 0x00,
0xB0, 0x18, 0x00, 0xB0, 0x19, 0x7F, 0xB0, 0x1A, 0x00, 0xB0, 0x1B, 0x7F},

{0xB4, 0x1C, 0x7F, 0xB4, 0x1D, 0x00, 0xB4, 0x1E, 0x00, 0xB4, 0x1F, 0x7F,
0xB4, 0x66, 0x00, 0xB4, 0x67, 0x00, 0xB4, 0x68, 0x00, 0xB4, 0x69, 0x7F},
```

```
{0xB0, 0x14, 0x00, 0xB0, 0x15, 0x7F, 0xB0, 0x16, 0x7F, 0xB0, 0x17, 0x00,
0xB0, 0x18, 0x00, 0xB0, 0x19, 0x7F, 0xB0, 0x1A, 0x00, 0xB0, 0x1B, 0x7F},

{0xB5, 0x1C, 0x7F, 0xB5, 0x1D, 0x00, 0xB5, 0x1E, 0x00, 0xB5, 0x1F, 0x7F,
0xB5, 0x66, 0x00, 0xB5, 0x67, 0x00, 0xB5, 0x68, 0x00, 0xB5, 0x69, 0x7F},

{0xB8, 0x14, 0x00, 0xB8, 0x15, 0x7F, 0xB8, 0x16, 0x7F, 0xB8, 0x17, 0x00,
0xB8, 0x18, 0x00, 0xBF, 0x19, 0x7F, 0xBF, 0x1A, 0x00, 0xBF, 0x1B, 0x7F},

{0x99, 0x23, 0x00, 0x99, 0x2E, 0x00, 0x99, 0x2A, 0x00, 0xB4, 0x1F, 0x7F,
0xB4, 0x66, 0x00, 0xB4, 0x67, 0x00, 0xB4, 0x68, 0x00, 0xB4, 0x69, 0x7F},

{0x99, 0x24, 0x00, 0x99, 0x27, 0x00, 0x99, 0x38, 0x00, 0xB0, 0x17, 0x00,
0xB0, 0x18, 0x00, 0xB0, 0x19, 0x7F, 0xB0, 0x1A, 0x00, 0xB0, 0x1B, 0x7F}
};

//SETUP

void setup() {
      pinMode(up, INPUT);
      pinMode(down, INPUT);
      pinMode(edit, INPUT);
      pinMode(esc, INPUT);
      pinMode(led11, OUTPUT);
      pinMode(led12, OUTPUT);
      pinMode(led21, OUTPUT);
      pinMode(led22, OUTPUT);
      pinMode(led31, OUTPUT);
      pinMode(led32, OUTPUT);
      pinMode(led41, OUTPUT);
      pinMode(led42, OUTPUT);
      pinMode(led51, OUTPUT);
      pinMode(led52, OUTPUT);
      pinMode(led61, OUTPUT);
      pinMode(led62, OUTPUT);
      pinMode(led71, OUTPUT);
      pinMode(led72, OUTPUT);
      pinMode(led81, OUTPUT);
      pinMode(led82, OUTPUT);

      patch = 0;
```

```
      load(patch);

      Serial.begin(57600);
      Serial1.begin(31250);
}

//LOOP

void loop() {

      v1 = analogRead(pad1);
      if (checkhit(v1, 0) == 0) midimsg(0, v1);
      v2 = analogRead(pad2);
      if (checkhit(v2, 0) == 0) midimsg(1, v2);
      v3 = analogRead(pad3);
      if (checkhit(v3, 0) == 0) midimsg(2, v3);
      v4 = analogRead(pad4);
      if (checkhit(v4, 0) == 0) midimsg(3, v4);
      v5 = analogRead(pad5);
      if (checkhit(v5, 0) == 0) midimsg(4, v5);
      v6 = analogRead(pad6);
      if (checkhit(v6, 0) == 0) midimsg(5, v6);
      v7 = analogRead(pad7);
      if (checkhit(v7, 0) == 0) midimsg(6, v7);

      if (Read(up) == 0) {
            patch = (++patch) % 10;
            load(patch);
            Serial.println(patch + 1, DEC);
            delay(100);
      }
      if (Read(down) == 0) {
            patch = (patch--) % 10;
            if (patch == 255) patch = 9;
            load(patch);
            Serial.println(patch + 1, DEC);
            delay(100);
      }
      if (Read(edit) == 0) {
            editmenu();
            load(patch);
```

```
	}

}

//MIDI MESSAGE

void midimsg (int pad, int velocity)
{
	if (pad == 2 && analogRead(pedal) > threshold) pad++;

	Serial1.write(patches[patch][pad * 3]);
	Serial1.write(patches[patch][(pad * 3) + 1]);
	if (patches[patch][pad * 3] == 0x99) {
		velocity = map(velocity, threshold, 1023, 1, 127);
		Serial1.write(velocity);
		delay(velocity);
		Serial1.write(patches[patch][pad * 3]);
		Serial1.write(patches[patch][(pad * 3) + 1]);
		Serial1.write(Noteoff);
	}
	else  Serial1.write(patches[patch][(pad * 3) + 2]);

	return;
}

//ANALOG DEBOUNCE

int checkhit(int vel, byte spad){
	if (vel > threshold){
		if (hit[spad] == false ){
			if (millis() - timing[spad] > release){
				timing[spad] = millis();
				hit[spad] = true;
				return 0;
			}
			else
			return 1;
		}
		else return 1;
	}
```

```
      else {
            hit[spad] = false;
            return 1;
      }
}

//LED PATCH

void load (int selected) {
      int i;
      for (i = 0; i < 8; i++)
      if (patches[selected][i * 3] == 0x99) {
            digitalWrite((23 + (i * 4)), HIGH);
            digitalWrite((25 + (i * 4)), LOW);
      }
      else {
            digitalWrite((23 + (i * 4)), LOW);
            digitalWrite((25 + (i * 4)), HIGH);
      }
}

//MAIN MENU


void editmenu() {

      Serial.println("Patch Editing");
      Serial.print("Edit ");
      Serial.println(menu[g_menu_pos]);
      while (true){

            if (Read(esc) == 0) break;
            if (Read(up) == 0) {
                  g_menu_pos = (++g_menu_pos) % 4;
                  Serial.println(menu[g_menu_pos]);
            }
            if (Read(down) == 0) {
                  g_menu_pos = (g_menu_pos--) % 4;
                  if (g_menu_pos == 255) g_menu_pos = 3;
                  Serial.println(menu[g_menu_pos]);
            }
```

```
            if (Read(edit) == 0) {
                  pad_select(g_menu_pos);
                  break;
            }
      };
return;
}

//PAD SELECT

void pad_select(int position)
{
      byte npad = 0;

      Serial.println("Pad Select Menu");
      Serial.print("Edit Pad ");
      Serial.println(npad + 1, DEC);

      while (true) {
            if (Read(esc) == 0) break;
            if (Read(up) == 0) {
                  npad = (++npad) % 8;
                  Serial.println(npad + 1, DEC);
            }
            if (Read(down) == 0) {
                  npad = (npad--) % 8;
                  if (npad == 255) npad = 7;
                  Serial.println(npad + 1, DEC);
            }
            if (Read(edit) == 0) {
                  switch (position) {
                        case 0 : edittype(npad); break;
                        case 1 : editchannel(npad); break;
                        case 2 : editdata1(npad); break;
                        case 3 : editdata2(npad); break;
                  }
            break;
            }
      };
return;
}
```

```
//TYPE EDITING MENU

void edittype(int mpad) {
      byte mtype = patches[patch][mpad * 3];

      Serial.println("Pad Type Select");
      Serial.print("Pad type : ");
      if (mtype == 0x99) Serial.println("Note");
      else Serial.println("CC");
      while (true) {

            if (Read(esc) == 0) break;
            if (Read(up) == 0) {
                  if (mtype != 0x99) {
                        mtype = 0x99;
                        Serial.println("Drum Note");
                        delay(200);
                  }
                  else {
                        mtype = 0xB0;
                        Serial.println("Control Change");
                        delay(200);
                  }
            }

            if (Read(down) == 0) {
                  if (mtype != 0x99) {
                  mtype = 0x99;
                  Serial.println("Drum Note");
                  delay(200);
            }
            else {
                  mtype = 0xB0;
                  Serial.println("Control Change");
                  delay(200);
            }
      }

      if (Read(edit) == 0){
            patches[patch][mpad * 3] = mtype;
```

```
            reset(mpad, mtype);
            break;
      }
};
return;
}

//CHANNEL EDITING MENU

void editchannel(int mpad) {
      int j;
      byte nchannel = patches[patch][(mpad * 3)];

      for (j = 0; j < 4; j++ ) bitClear(nchannel, 7 - j);


      if (patches[patch][mpad * 3] == 0x99) {
            Serial.println("Canale non editabile per suoni di batteria");
            delay(2000);
            return;
      }

      else {
            Serial.println("Channel Editing");
            Serial.print("Channel : ");
            Serial.println(nchannel + 1, DEC);
      }

      while (true){
            if (Read(esc) == 0) break;
            if (Read(up) == 0) {
                  nchannel = (++nchannel) % 16;
                  Serial.println(nchannel + 1, DEC);
            }
            if (Read(down) == 0) {
                  nchannel = (nchannel--) % 16;
                  if (nchannel == 255) nchannel = 15;
                  Serial.println(nchannel + 1, DEC);
            }
            if (Read(edit) == 0) {
```

```
                patches[patch][(mpad * 3)] = 0xB0 | nchannel;
                break;
            }
      };
return;
}

//DATA BYTE 1 EDITING MENU

void editdata1(int mpad){

      if (patches[patch][(mpad * 4)] == 0x99){
            byte mdata1 = patches[patch][(mpad * 3) + 1];

            Serial.println("Data byte 1");
            Serial.print("Value : ");
            Serial.println(mdata1, DEC);

            while (true) {
                  if (Read(esc) == 0) break;
                  if (Read(up) == 0) {
                        mdata1 = (++mdata1) % 82;
                        Serial.println(mdata1, DEC);
                  }
                  if (Read(down) == 0) {
                        mdata1 = (mdata1--) % 82;
                        if (mdata1 == 255) mdata1 = 81;
                        Serial.println(mdata1, DEC);
                  }
                  if (Read(edit) == 0) {
                  patches[patch][(mpad * 3) + 1] = mdata1;
                  break;
                  }
            };
      }
      else {
            byte mdata1 = patches[patch][(mpad * 3) + 1];
            Serial.println("Data byte 1");
            Serial.print("Value : ");
            Serial.println(mdata1, DEC);
            while (true) {
```

```
                  if (Read(esc) == 0) break;
                  if (Read(up) == 0) {
                        mdata1 = (++mdata1) % 128;
                        Serial.println(mdata1, DEC);
                  }
                  if (Read(down) == 0) {
                        mdata1 = (mdata1--) % 128;
                        if (mdata1 == 255) mdata1 = 127;
                        Serial.println(mdata1, DEC);
                  }
                  if (Read(edit) == 0) {
                  patches[patch][(mpad * 3) + 1] = mdata1;
                  break;
                  }
            };
      }
return;
}

//DATA BYTE 2 EDITING MENU

void editdata2(int mpad) {

      if (patches[patch][mpad * 3] == 0x99) {
            Serial.println("Data byte 2 non editabile per suoni di
            batteria");
            delay(2000);
      return;
      }

      byte mdata2 = patches[patch][(mpad * 3) + 2];

      Serial.println("Data byte 2");
      Serial.print("Value : ");
      Serial.println(mdata2, DEC);
      while (true){
            if (Read(esc) == 0) break;
            if (Read(up) == 0) {
                  mdata2 = (++mdata2) % 128;
                  Serial.println(mdata2, DEC);
```

```
            }
            if (Read(down) == 0) {
                  mdata2 = (mdata2--) % 128;
                  if (mdata2 == 255) mdata2 = 127;
                  Serial.println(mdata2, DEC);
            }
            if (Read(edit) == 0) {
                  patches[patch][mpad * 4 + 2] = mdata2;
                  break;
            }
      };
return;
}

//RESET

void reset(byte spad, byte stype){

      if (stype == 0x99) patches[patch][spad + 1] = 0x23;
      else {
            patches[patch][spad + 1] = 0x14;
            patches[patch][spad + 2] = 0x00;
      }
return;
}

//DIGITAL DEBOUNCE

int Read(int button){
      int flag = 0;
      int reading = digitalRead(button);

      if (reading != lastButtonState[button - 7]) {
            lastDebounceTime[button - 7] = millis();
      }
      if ((millis() - lastDebounceTime[button - 7]) > debounceDelay) {
            if (reading != buttonState[button - 7]) {
                  buttonState[button - 7] = reading;
                  if(reading == HIGH) flag = 1;
            }
      }
```

```
        lastButtonState[button - 7] = reading;
        if (flag == 1) {
              return 0;
        }
        else return 1;
}
```

Bibliografia e Sitografia

Bibliografia

[1] Joseph Rothstein, "*MIDI : A Comprehensive Introduction*", A-R Editions, 1995.

[2] Simon Milward, "*Fast Guide to Cubase 4*", PC Publishing, 2007.

[3] MMA, "*Detailed MIDI Specs*", 1995.

[4] MMA, "*Standard MIDI-File Format Specs 1.1*", 1990.

[5] Jeffrey Rona, "*The MIDI Companion*", Hal Leonard Corporation, 1987.

[6] Andrea Nepori, "*Arduino, la rivoluzione dell'open hardware*", 2013.

[7] Mr Jalopy, "*Owner's Manifesto*". Make Magazine Volume 4, 2006.

[8] Matteo Banzi, "*Getting Started with Arduino*", O'Reily, 2009.

[9] Max Burnet and Bob Supnik,"*Preserving Computing's Past: Restoration and Simulation*", Digital Technical Journal, Volume 8, Number 3, 1996.

Sitografia

[1] NAMM 2013, "Panel discussion: Past, present ad future of MIDI", 2013
http://www.youtube.com/watch?v=SFIZc7IMzyA

[2] Wired Magazine, "Arduino: creare è un gioco da ragazzi", 2012
http://tv.wired.it/entertainment/2012/12/06/arduino-creare-e-un-gioco-da-ragazzi-eng-sub.html

[3] PWM on Arduino
http://arduino.cc/en/Tutorial/PWM

[4] Arduino and Processing
http://playground.arduino.cc/interfacing/processing

[5] pinMode() function reference
http://arduino.cc/en/Reference/PinMode

[6] digitalWrite() functiond reference
http://arduino.cc/en/Reference/DigitalWrite

[7] delay() function reference
http://arduino.cc/en/Reference/Delay

[8] analogRead() function reference
http://arduino.cc/en/Reference/AnalogRead

[9] Serial.begin() function reference
http://arduino.cc/en/Serial/Begin

[10] Serial.print() function reference
http://arduino.cc/en/Serial/Print

[11] digitalRead() function reference
http://arduino.cc/en/Reference/DigitalRead

[12] Debounce tutorial
http://arduino.cc/en/Tutorial/Debounce

[13] millis() function reference
http://arduino.cc/en/Reference/Millis

[14] Serial.write() function reference
http://arduino.cc/en/Serial/Write

[15] map() function reference
http://arduino.cc/en/Reference/Map

[16] bitClear() function reference
http://arduino.cc/en/Reference/BitClear