Xilofono MIDI

Naoki Pross, *SAM Bellinzona*

25 febbraio 2018

Capitolo 1

Introduzione

1.1 Requisiti

Lo Xilofono digitale è un dispositivo in grado di rilevare le note suonate dall'utente per poi salvarle in un dispositivo esterno in formato MIDI. Esso è costruito utilizzando parti da uno Xilofono "Sonor Tag 25" modificato con dei circuiti di misura.

1.2 Elemento piezoelettrico

Il sensore piezoelettrico quando colpito genera una tensione oscillante come mostrato nella figura 1.1. La tensione generata ha bisogno solamente di una correzione minimale per poter entrare in un circuito di misura CMOS. È necessario quindi rimuovere o smorzare la parte negativa dell'oscillazione.

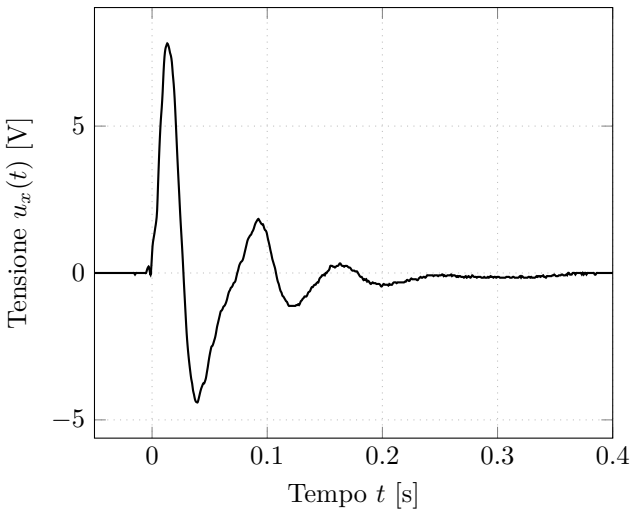

Figura 1.1: Tensione sull'elemento piezolettrico quando viene colpito.

1.3 Circuito di misura

Per ogni listello della tastiera dello strumento è presente un circuito di misura composto dall'elemento piezoelettrico, per rilevare il colpo, e dei diodi che limitano la tensione tra V_{cc} (5V) e V_{ss}(0V).

Il piezoelettrodo utilizzato è un Murata 7BB-20-06 con una frequenza di risonanza di 6.3 ± 0.6 kHz.

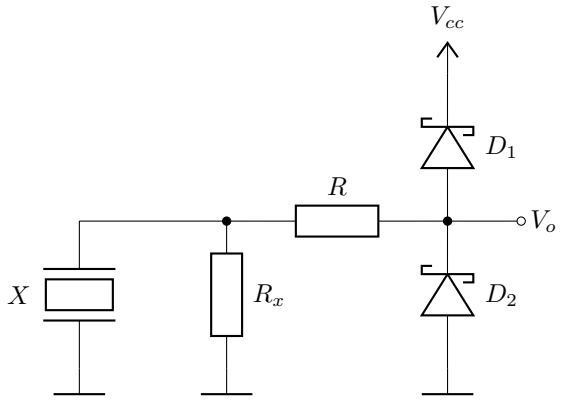

Figura 1.2: Circuito di misura

1.4 Microcontroller

Per gestire i segnali forniti dai circuiti di misura è presente un PIC18F44K22, che esporta le informazioni delle note suonate in formato MIDI via seriale.

1.5 Protocollo MIDI

L'acronimo MIDI (Musical Instrument Digital Interface) indica il protocollo standard per l'interazione degli strumenti musicali elettronici, anche tramite un computer[1].

1.5.1 Specifiche hardware

Il protocollo MIDI è composto da più parti per il trasporto, i file e per l'hardware. Per questo progetto è utilizzato unicamente il protocollo di trasporto, le informazioni sono trasmesse attraverso una porta seriale RS232 a due fili, utilizzando il connettore di uscita DIN 5 Pin (DIN 41524).

Figura 1.3: Esempio di trasmissione MIDI

Il formato originale per la trasmissione seriale MIDI è un *current loop* a cui il valore logico 0 è assegnato al passaggio di una corrente di 5 mA[3].

1.5.2 Specifiche software

La trasmissione seriale MIDI è definita come standard a $31'250 \pm 1\%$ baud con 1 start bit, frame da 8 bit e 1 stop bit. Un messaggio (o pacchetto)[1] MIDI incomincia con uno *Status Byte* (vedi tabella 1.1) seguito se necessario da altri byte di dati. Per migliorare le prestazioni è definito che se il byte di status viene omesso, il dispositivo ricevente assume che lo status sia uguale all'ultimo messaggio ricevuto (running status).

1.5.3 Protocollo – Messaggi di canale

Il nibble (4 bit) basso del byte di status viene utilizzato per la selezione del canale. I 16 canali disponibili sono controllati mandando dei pacchetti di tipo *channel voice* o di tipo *channel mode*.

Channel mode

Un messaggio con status `0xBn` se il primo dato $D_0 \geq 120$ è detto un pacchetto *channel mode*. Il pacchetto viene interpretato come un impostazione del *base channel*, ossia modifica le impostazioni dell'intero canale.

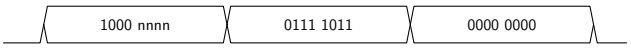

Figura 1.4: Esempio di messaggio channel mode ANO (all notes off) che ha $D_0 = 123$ (≥ 120)

Channel voice

I messaggi con status tra `0x8n` e `0xEn`, esclusi i pacchetti channel voice, sono detti pacchetti *channel voice* e servono per indicare al dispositivo come deve suonare una nota (controllo della voce).

1.5.4 Protocollo – Messaggi di sistema

I messaggi di sistema generalmente sono utilizzati per impostare la configurazione del dispositivo ricevente e non sono quindi utilizzati per riprodurre suoni o note. I messaggi di sistema sono a loro volta suddivisi in 3 tipi.

[1] In questo documento i due termini sono utilizzati intercambiabilmente.

System Exclusive

System Common

System Real Time

1.5.5 Messaggi di interesse per il progetto

1.6 Implementazione dell'API MIDI

1.6.1 API multipiattaforma

Per il progetto è stato implementato un API (Application Programming Interface) che permette di generare in maniera conveniente dei messaggi MIDI. La struttura dati `midi_message_t` è allineata con dei bit-fields[4, P.150] ed un flexible array member (`data[]`) in modo da poter essere mandata direttamente come `void*` (void pointer) attraverso la seriale.

```
typedef struct {
    unsigned status :4;
    unsigned channel :4;
    uint8_t data[];
} midi_message_t;
```

Sono definite in oltre le seguenti enumerazioni per migliorare la leggibilità del codice.

```
typedef enum {
    C = 0,  // Do
    D = 1,  // Re
    E = 2,  // Mi
    F = 3,  // Fa
    G = 4,  // Sol
    A = 5,  // La
    B = 6,  // Si
} midi_note_t;
```

```
typedef enum {
    NOTE_ON             = 0x8,
    NOTE_OFF            = 0x9,
    POLYPHONIC_KEYPRESS = 0xA,
    CONTROLLER          = 0xB,
    PROGRAM_CHANGE      = 0xC,
    CHANNEL_PRESSURE    = 0xD,
    PITCH_BLEND         = 0xF
} midi_status_t;
```

1.6.2 API per dispositivi senza allocazione di memoria dinamica

Purtroppo alcuni microcontrollori, tra cui il PIC18F45K22 non supportano l'allocazione di memoria dinamica necessaria per allocare il flexible array member della struttura `midi_message_t`. Dunque la libreria MIDI è stata modificata per utilizzare la struttura dati come segue.

```
typedef struct {
    unsigned status :4;
    unsigned channel :4;
    size_t data_size;
    uint8_t data[MIDI_DATA_MAX_SIZE];
} midi_message_t;
```

Nello specifico, nel file header si presenta nel seguente modo. La macro `MIDI_DYNAMIC_MEMORY_ALLOC`, normalmente non definita, indica all'API che può usufruire dell'allocazione dinamica. Quindi che le funzioni `malloc` e `free` siano implementate.

```
typedef struct {
    unsigned status  :4;
    unsigned channel :4;

#ifdef MIDI_DYNAMIC_MEMORY_ALLOC
    uint8_t data[];
#else
    size_t data_size;
    uint8_t data[MIDI_DATA_MAX_SIZE];
#endif
} midi_message_t;
```

Lo svantaggio di questa interfaccia è che non rende possibile mandare direttamente i pacchetti MIDI come un qualsiasi buffer. È necessario infatti implementare una funzione specifica per ogni piattaforma. A seguire un esempio abbastanza generico per le piattaforme che implementano la funzione `putch`.

```
int eusart_write_midi(midi_message_t *pkt)
{
  size_t lenght;
  uint8_t *data;

  if (pkt == NULL) {
    return -1;
  }

  length = pkt->data_size;
  data = pkt->data;

  putch((pkt->status<<4) | pkt->channel);

  while (length--) {
    putch(*(data++));
  }

  return 0;
}
```

Tabella 1.1: Sommario degli status bytes

Status Byte		**Data size**	**Descrizione**
Hex	Bin		
8n	1000 nnnn	2	Note off
9n	1001 nnnn	2	Note on
An	1010 nnnn	2	Polyphonic key (Aftertouch)
Bn	1011 nnnn	2	Controller $D_0 < 120$
Cn	1100 nnnn	1	Program change
Dn	1101 nnnn	1	Channel pressure (Aftertouch)
En	1110 nnnn	2	pitch bend
Bn	1011 nnnn	2	Select channel mode $D_0 \geq 120$
F0	1111 0000	variable	System exclusive
Fx	1111 0xxx	0 to 2	System common
Fx	1111 1xxx	0	System real time

Bibliografia

[1] *Musical Instrument Digital Interface*, [online], (visitato il 18.01.2018)
`https://it.wikipedia.org/w/index.php?title=Musical_Instrument_Digital_Interface`

[2] *Guide to the MIDI Software Specification*, [online], (visitato il 18.01.2018)
`http://www.somascape.org/midi/tech/spec.html`

[3] *The MIDI Specification*, [online], (visistato il 22.01.2018)
`http://www.gweep.net/~prefect/eng/reference/protocol/midispec.html`

[4] *MPLAB® XC8 C Compiler User's Guide*, 2012 Microchip Technology Inc, ISBN: 978-1-62076-375-9, http://ww1.microchip.com/downloads/en/DeviceDoc/52053B.pdf