USART e MIDI

Salve a tutti!

Con questo documento intendo fornire una panoramica generica su cosa sia prima di tutto il protocollo MIDI e come questo possa essere gestito da un Microcontrollore. Il documento non vuole assolutamente essere esaustivo in materia nella maniera più assoluta, d'altronde io come voi sono un appassionato che semplicemente condivide con chi ama l'elettronica le sue esperienze.

Intanto un po' di ciance su cosa sia il Protocollo MIDI. Questo porta con sé due nature:

- Un insieme di "regole" di comunicazione che rende di fatto possibile il dialogo fra apparecchi di diversi distributori
- Uno standard hardware per la sua implementazione

Dal punto di vista delle "regole di comunicazione" il MIDI è del tutto indipendente dalla sua realizzazione fisica. Le comunicazioni MIDI si basano su "pacchetti" (entità atomiche di informazioni) di 8 bit. Ogni informazione che si vuole trasmettere (che sia un cambio canale, la pressione di una nota) è composta da uno ***Status Byte*** e da uno o più ***Data Byte/s***. Cosa significa questo?

Prima di entrare in particolari facciamo un esempio semplice, spiegheremo dopo il messaggio in ogni sua singola parte; sulla nostra tastiera premiamo il DO 1, sul nostro cavo midi scorreranno questi 3 bytes:

(BIN) 10010001 00011000 01100100

(HEX) 91 18 64

Ci sono dunque 3 pacchetti da 8 bit ciascuno, ma cosa significano? Come detto il primo byte è sempre uno status byte che identifica il tipo di evento accaduto (in questo caso Note On), ma vediamo di scomporre le sue parti e di meglio capire cosa significano:

1 001 0001

- Il primo bit è settato a 1 per indicare che questo è uno status byte

- I 3 bit seguenti indicano il tipo di evento, Note On è codificato con 001

- Gli ultimi 4 bit indicano il canale (***channel***) al quale è indirizzato il messaggio

Mentre non è difficile capire i primi due punti, concentriamoci sul terzo: il MIDI channel è un sistema di "indirizzi"; in tutto possono essere 16 (infatti abbiamo a disposizione 4 bit) questo serve affinché lo stesso dispositivo possa inviare comandi a fino 16 apparecchi diverse. Immaginiamo di avere un rack di 16 synth che suonano ad un Note On; se vogliamo far suonare solo il 13 apparecchio, allora l'indirizzo (il channel) sarà il 13. Il funzionamento è il seguente: i 3 byte arrivano sempre e comunque a tutti i dispositivi (collegati a catena con MIDI IN e MIDI THRU) i quali però sono settati per ricevere SOLO quelli di un determinato canale, se l'indirizzo sul messaggio è diverso dal loro, questo verrà semplicemente ignorato.

Ma torniamo ai 2 successivi Status Byte: il primo bit di entrambi è fisso a '0' per indicare che essi sono Data Bytes, i restanti 7 bit indicano, nel primo la nota (0011000 è C1) e nel secondo la *velocity*, ovvero l'intensità con la quale è stato premuto il tasto sulla vostra tastiera.

Dunque in generale un comando MIDI è composto sempre da uno Status Byte che va a descrivere il significato dei successivi Status Byte (vale questo discorso anche per i Program Change e i Control Change).

Per una lista completa e dettagliata dei messaggi MIDI si rimanda a questo link: http://www.midi.org/techspecs/midimessages.php

Per quanto riguarda invece la parte hardware il protocollo MIDI è una trasmissione ***seriale***. Questo significa che i bit sono trasmessi uno dopo l'altro (pensate ad infilare in un tubo 8 tesserine dello scarabeo , il ricevente li prenderà uno per uno e potrà ricomporre il messaggio originale). Parlando di specifiche squisitamente tecniche, la trasmissione avviene a 31250 baud ovvero a 31250 simboli al secondo ed, essendo una trasmissione *asincrona*, necessita di bit di start e end. Cerchiamo di spiegare quanto detto, il baud è un valore molto importante, infatti identifica la velocità di trasmissione dei dati; questo parametro va correttamente impostato nel vostro PIC in maniera che i dati vengano ricevuti correttamente. Il termine "asincrono" riferito alla trasmissione indica il fatto che non c'è bisogno di una connessione che funzioni da "clock"; in una trasmissione sincrona infatti oltre alla linea dati, c'è una linea di clock che tiene "a tempo" ricevitore e trasmettitore, in questo caso la trasmissione comincia sempre a tempo, e non c'è possibilità di perdere il sincronismo. In una trasmissione asincrona, mancando questo riferimento, il trasmettitore "avverte" il ricevitore dell'imminente trasmissione con un bit, e a fine un bit di end conferma la chiusura della stessa; dunque guardando il filo, a fronte di un invio di 8 bit ciò che vedremmo sarebbe:

1 10010001 1

Ovvero un totale di 10 bit, il primo e l'ultimo in realtà non appariranno mai al programmatore del PIC poiché essi sono degli "artifici" necessari allo strato hardware per funzionare correttamente, e dunque proprio da questo eliminati (per essere precisi in realtà il bit di chiusura può essere usato dal programmatore come bit di parità per controllare l'integrità delle informazioni ricevute; questo è possibile settando appropriatamente un bit in determinati registri di sistema dei PIC).

Ora parliamo di cose pratiche, ma come riceviamo i messaggi MIDI sui nostri microcontrollori?

Premetto nuovamente che parlando ora di un procedimento che è davvero tecnico, non ho nessuna intenzione di tradurre il datasheet di un determinato PIC riportando per filo e per segno le operazioni Assembler necessarie per settare tutto in maniera corretta: sarebbe stupido e soprattutto assolutamente inutile in quanto basta prendere il datasheet del proprio PIC per poter consultare una guida passo-passo su come operare correttamente; il mio pretenzioso intento è quello di fornirvi delle informazioni molto generali che vi permettano di capire cosa vi è scritto e quindi affrontare con cognizione di causa l'argomento.

Prima di tutto è giusto dire che i modi per ricevere messaggi MIDI sono molti, uno è quello di sfruttare pin qualsiasi di un qualsiasi microcontrollore e programmare via software tutto l'algoritmo che riceve, invia e rende disponibili i dati; va da sé che questa soluzione è non solo la più difficile, ma anche priva di vantaggi se consideriamo che PIC supportanti l'interfaccia USART sono comunissimi e tutt'altro che costosi (parliamo, per capirci di PIC 16F62X).

La chiave di tutto è proprio l'interfaccia hardware USART (Universal Synchronous Asynchronous Receiver Transmitter), ovvero quell'insieme di registri e regole che permette di ricevere trasmissioni seriali Sincrone ed Asincrone. Per quello che ci interessa come detto la trasmissione dovrà essere Asincrona.

La prima cosa da fare sarà ovviamente impostare la baud rate necessaria (31250), per fare questo si dorvà settare un particolare valore in un registro di sistema; questo valore sarà usato all'interno di una formula per ricavare il giusto baud rate. Questa formula dipende oltre che dal valore impostato dall'utente, anche dalla frequenza dell'oscillatore scelto per far funzionare il microcontrollore; questo è ovvio se pensiamo che non si sta facendo altro che "insegnare" al PIC come ricavarsi un nuovo clock a partire da quello che ha già a disposizione, insomma tot cicli dell'oscillatore corrisponderanno ad un ciclio di clock per la USART.

E' inoltre possibile determinare l'errore come differenza percentuale fra il baudrate desiderato e quello effettivo; facciamo un esempio con le formule trovate su un datasheet di un PIC 16F62X:

$$Desired\ Baud\ Rate = \frac{Fosc}{64\ (X+1)}$$

Ipotizzando di avere uno strano quarzo da 11MHz, ovvero 11000000 Hz, ed essendo il "Desired Baud Rate" di 31250 ne deduciamo che X debba essere '4,5' che approssimato ad un intero viene '4', l'errore in questo caso sarà dunque:

$$error = \frac{Calculated\ Baud\ Rate - Desired\ Baud\ Rate}{Desired\ Baud\ Rate} = \frac{34375 - 31250}{31250} = 0{,}1\%$$

La trasmissione e ricezione concettualmente è qualcosa di assolutamente semplice. Per trasmettere un byte bisogna caricarlo nel registro dedicato (è uno shift register parellel in / serial out), il dato verrà inviato in maniera seriale attraverso il PIN di solito chiamato TX, al termine della trasmissione un bit verrà settato per confermare l'avvenuta trasmissione e verrà generato un interrupt (il quale volendo può essere disabilitato).

La ricezione prevede un procedimento "speculare"; i dati ricevuti sul pin RX e memorizzati su uno shift register serial in / parellel out, al termine della ricezione, verranno aggiunti ad un buffer gestito con modalità FIFO (ovvero, il primo dato ad entrarvi è anche il primo ad uscirne) e viene generato un interrupt che dovrà gestire l'elaborazione del dato appena ricevuto. Il buffer è necessario poiché c'è il fattore tempo da considerare; se infatti vi fosse un solo registro a memorizzare i dati ricevuti pensate a cosa succederebbe nella ricezione di due dati uno dopo l'altro: se non si è abbastanza veloci nel "consumare" il primo dato, il secondo lo andrebbe a sovrascrivere!

Da un punto di vista logico, è tutto qui! Molto semplice grazie al fatto che l'intera interfaccia hardware USART "nasconde" tutti i dettagli tecnici dei quali non dovete preoccuparvi: per farvela semplice, avete a disposizione un postino al quale potete affidare un messaggio, e che vi organizza la posta in arrivo impilandola sul tavolino!

C'è un ultimo dettaglio da affrontare, e riguarda a tutto ciò che separa il cavo MIDI e i pin RX e TX del microcontrollore. Questi infatti non vengono brutalmente connessi l'uno con l'altro (questo comporterebbe una rischiosa connessione diretta fra apparecchiature: se una dovesse rompersi potrebbe portarsi dietro tutte le apparecchiature a cascata, inoltre nella connessione MIDI THRU, a lungo andare l'impedenza del cavo porterebbe perdite di segnali gravissime). Di schemi per connettere i dispositivi in rete se ne trovano a bizzeffe, prendendo questo schema come riferimento, le componenti principali sono:

- Un opto-isolatore (6N138, 6N139, 6N137, PC900 ecc..) che appunto fisicamente divide i vari dispositivi
- Buffer, i quali hanno il compito di “ricostruire” il segnale che potrebbe aver subito distorsioni dovute all’opto-isolatore

Il segnale in ingresso viene ricreato dall’opto-isolatore e trasmesso simultaneamente al pin di ricezione del PIC e, rigenerato da un buffer, al MIDI Thru (che in effetti altro non è che una copia dell’input). I pacchetti in uscita dal pin RX vengono bufferizzati e spediti attraverso la connessione MIDI Out.

Prima di chiudere vi segnalo questi programmi freeware con i quali potrete monitorare le vostre porte MIDI studiando nel dettaglio tutti i pacchetti ricevuti (e come diceva il mio professore di “Reti di Calcolatori”: per capire veramente una cosa, vi ci dovete “immergere”!):

http://www.midiox.com/

http://miosstudio.midibox.org/

Per questo brevissimo tutorial è tutto, come detto le informazioni qui contenute non hanno assolutamente la pretesa di essere esaustive, ma di gettare delle basi più o meno complesse che non solo permettano a chi vuole iniziare a giocare con il protocollo MIDI di farlo, ma soprattutto di suscitare interesse e domande nei lettori che troveranno tutto l’aiuto che vogliono sul forum di diyitalia.eu al quale tutti noi dovremmo solo che essere grati!

Ciao Ragazzi, alla Prossima!

davidefender