Music Instrument Accessibility

By Emma Pannullo, ECE '19

Introduction

Music is dynamic and complex. Changing a few notes or the way certain notes are played can transform one song into something entirely different. Three important components of music are the note itself, the loudness of the note, and the duration of the note. When creating an electronic instrument, it is important to keep these components in mind in order to capture the essence of music. The MIDI standard describes a protocol, a type of computer language, that encompasses and describes these components. MIDI stands for Musical Instrument Digital Interface and is widely used for electronic instruments because of its universality and adaptability. This paper discusses how the MIDI standard can be adapted for a real time, analog system.

The MIDI Standard

MIDI data is typically compiled into a MIDI file. This file consists of several bytes of data that define commands and parameters. These chunks of data can be compared to something we see every day, telephone numbers. Telephone number protocol has a first number to define the country, then a 3-digit number to define an area code, then 7 digits to distinguish your particular phone number from someone else's phone number in your same country and area code. In MIDI files, there are pieces of data to define the type of action to be completed, followed by the data to perform that action on.

The main type of chunks defined in MIDI files are called track chunks. Figure 1 shows the breakdown of a track chunk. The first two pieces of information are the ASCII characters defining the type of chunk and the length of the chunk. Each track chunk contains additional data called a message or an event as well as a piece of information called *delta time*. The delta time data is used to determine how much time should pass between each event. One example of this would be if two consecutive events defined a note on and a note off. The delta time would help to define the length of the note. A track chunk can contain several events. Each event contains an eight-bit command byte which corresponds to a specific action, and one to two bytes of data to be operated on depending on what is the status byte (Bello, n.d.).

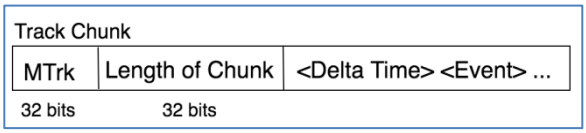

Figure 1: Organization of Track Chunk

In the decimal system, each command byte ranges in value from 128 to 255 and each data byte ranges from 0 to 127. A simple example of an event would be a *note on* command followed by two pieces of data that define the note number and at what volume the note should be played. Figure 2 shows what this event would look like in a MIDI file where the data is presented in hexadecimal number system. This event defines a note on for channel 1 of the note middle C, played at maximum volume. Velocity data corresponds to the volume at which the note is played. A faster velocity means a louder note and a slower velocity means a quieter note.

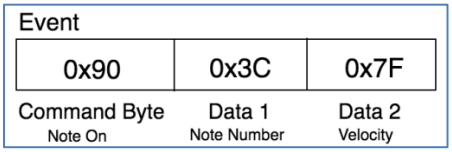

Figure 2: Event Example

The command byte in the above example reads 0x90. The MIDI "note on" format is 0x9n where n+1 is the number of the channel. In this example, n is set to 0 so this note will be played on channel 1. The second piece of information is the note number. 0x3C in hexadecimal corresponds to middle C. The third piece of information defines the velocity of the note. 0x7F in hexadecimal maps to 127 in decimal and represents the maximum velocity at which the note can be played (Lehrman & Tully, 1993).

Transmission of MIDI Data

MIDI data is transmitted as serial data. This means each individual bit is sent one at a time. Figure 3 displays the difference between serial and parallel transmission schemes. MIDI data is transmitted at a speed of 31,250 bits per second. The human ear recognizes lag in sound after about 10 milliseconds so it is important that this data is transmitted quickly.

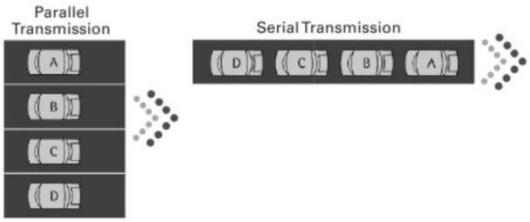

Figure 3: Parallel Vs. Serial Transmission (Guérin, 2008)

MIDI in Real Time

To construct a MIDI file, the length of the file must be known since that information must be encoded in the header chunk as well as the track chunk. MIDI is often used for composing music and playing back written music. In these situations, the length of the piece is known and the file can be populated. However, this does not account for a situation in which one would want to send live MIDI data without knowing the length of the file or how much data will be transmitted. While the standard MIDI file format is widely used and well organized, there are many situations that benefit from being able to send MIDI in real time. Earlier it was discussed that MIDI is a flexible format- this allows MIDI to be transmitted in real time through the use of additional software.

One software solution is RtMidi, a set of C++ classes that allow real-time MIDI input and output across several different popular operating systems. There is also a Python wrapper for these classes called python-rtmidi. This software solution disregards part of the MIDI file such as the header chunk and part of the track chunk. It is no longer necessary to define the length of chunks or to have chunks at all. The basis for this software is that the user can send just the events as messages. The user can link to a specified output port and send these event messages. Instead of sending delta time with each event, the user can use a sleep function, or a rest function to define the amount of time between event messages.

Mapping MIDI Ranges

When creating MIDI data in real time, the data can be manipulated according to user specification. MIDI data can be generated from an analog device using sensors to create something such as a musical instrument. These sensors create output voltages that can be read in though a microprocessor such as an Arduino. This information can be used to create MIDI messages. Output voltages from the sensors representing data such as velocity can be mapped to proper MIDI values. For example, if there is a sensor that reads values from 0 to 1024, this would need to be scaled since a MIDI data byte can only take values 0 to 127. A simple mapping would be to make the maximum value of 1024 equal to 127. Then any incoming value would be scaled by multiplying by 127 and dividing by 1024 to fit that value into the range 0 to 127.

Conclusion

MIDI is a powerful tool due to its flexibility and the way it is organized into compartmentalized pieces. These aspects allow MIDI data to be manipulated and sent in real time with the help of additional simple software solutions freely available online. Using the MIDI standard in conjunction with these software tools to create real time data, it is possible to manipulate the data to solve all sorts of problems such as making an instrument more sensitive for those that have difficulty playing. Being able to easily manipulate MIDI data can make music more accessible for many people.

References

Bello, J. (n.d.). *MIDI Code*. Retrieved from: https://www.nyu.edu/classes/bello/FMT_files/9_MIDI_code.pdf [Accessed 13 Dec. 2018].

Guérin, Robert. (2008). *MIDI power!* (2nd ed.). Boston, MA.: Thomson Course Technology.

Lehrman, P. D., Tully, T. (1993). *MIDI For The Professional.* New York, NY: Amsco Publications.