MIDI Protocol and MIDI Over USB

By Siqi Wang, ECE '22

Introduction

When electronic music started to be seen in the 1940s, people started to make musical sounds out of discrete electronic components such as oscillators, filters, and mixers. The need for a standardized protocol that communicates the devices involved in producing such music greatly increased. By the time of 1960s, synthesizers appeared as well as a popular control method called "voltage control" that allows different parts of the device to operate together. However, different manufacturers usually make devices that are not compatible with each other. In the 1970s, synthesizers using digital electronics were introduced, solving a lot of electrical problems and the complex module settings became digital parameters. This change produced the opportunity of a common interface for all manufacturers. (Lehrman, 1993)

History of MIDI

The idea of a common digital interface for synthesizers was first proposed by Dave Smith. His company introduced Universal Synthesizer Interface (USI) in 1981. This new technology was the predecessor of MIDI (Musical Instrument Digital Interface). After the idea was captured and some synthesizer manufacturers started to try it out on their product and gain success, the official MIDI Specification 1.0 was published in 1983.

MIDI is a technical standard that describes a communication protocol, digital interface, and electrical connectors that connect a large variety of musical and electronic devices including synthesizers, mixing boards, and computers. The compatibility of MIDI gives music producer the chance to work with devices and instruments from different manufacturers. For example, one can plug a Roland keyboard into a Yamaha synthesizer or connect it to a MacBook using USB and record the MIDI sequences using a Digital Audio Workstation (DAW).

The goal of MIDI is to be spread-out and universal. Therefore, the cost of adding such technology to a device is relatively low. However, the trade-offs are problems such as low-speed hardware transports. Today, the issue has been solved by the introduction of other transports such as the Universal Serial Bus (USB).

USB

When there were just too many different types of connectors for electronics devices in the 1990s, the need for a universal way to transmit data surged, just like the need of MIDI in the 1970s and 1980s. Therefore, USB 1.0 was released in January 1996. After multiple iterations, the newest USB Type C connector and the USB 4 specification are published in 2014 and 2019. (USB Spec, 1996)

Thanks to the advance in both the MIDI and USB, it has been so much easier to create and process music now than that of 30 years ago. Due to the popularity and compatibility of the USB, as well as the increasing use of computer and DAWs, the need for MIDI over USB has been rising. Therefore, it is important to

understand how exactly MIDI communication protocol works and how it could be implemented via USB connectors.

Implementation with Max

Max and Arduino

There are multiple ways to have MIDI and USB work together. One easier way is using a graphical programming language named Max.

Max/MSP is a flexible environment that has become the most popular development platform for practitioners of computer based live performance (Place, 2006). The power of Max is that it not only can read MIDI signals from a standard MIDI controller (such as an electric keyboard), but also can generate MIDI commands and send them to DAWs to create and control music. With the built-in blocks/objects, one does not even need to understand the MIDI specification to generate MIDI signals or create a MIDI music device.

Arduino is an open-source electronics platform based on easy-to-use hardware and software. It was born at the Ivrea Interaction Design Institute as an easy tool for fast prototyping, aimed at students without a background in electronics and programming (Kosky, 2021). Over the years, it has been used in thousands of projects, from personal electronics to scientific instruments. What makes Arduino and Max go along is, thanks to the USB, the serial bus—a universal method of transmitting digital information.

In Arduino, one can easy sketch up a code that send messages to the serial using the "*Serial.print()*" function. Max, similarly, can receive and write the serial as well. With the build-in MIDI modules in Max and some programming, one can easily translate the messages sent from the Arduino to MIDI commands and have it read by a DAW. One can also use a third-party module in Max called *Arduino2Max* which makes this process even easier.

MIDI Specification

The MIDI Cable

The original MIDI specification comes with the "MIDI cable", whose ends are MIDI-DIN connectors. The MIDI signal is on Pin 5 of the connector, and the voltage to drive the circuit is on
Pin 4. Pin 2 is connected to the cable shield, and Pins 1 and 3 are not normally connected (Lehrman, 1993).

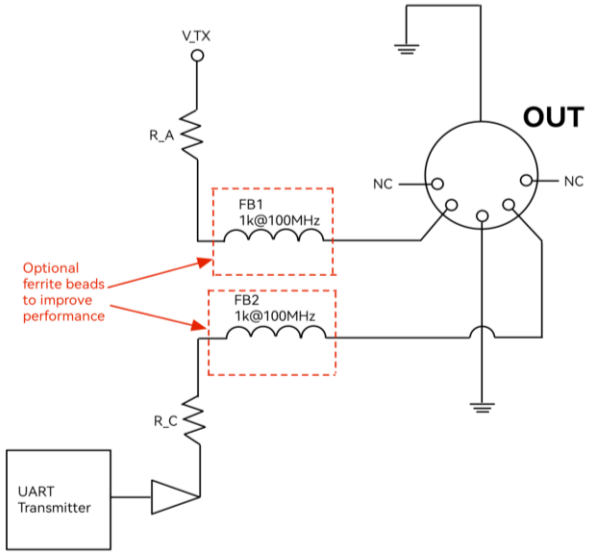

Figure 1. MIDI-DIN pin out (Lehrman, 1993)

Obviously, the MIDI-DIN connectors are to be plugged into a MIDI-IN jack. The MIDI-IN jack uses the same pin configuration as the MIDI-DIN, except Pin 2 is not connected. The MIDI-IN jacks are also not connected directly to the device's circuit. Instead, there is an opto-isolator in between that transmits the electrical signals using lights, preventing high-voltages from affecting the device.

Because of the existence of the opto-isolator, when one needs to send MIDI signals to two devices, it has to be done using the MIDI-THRU port. This is because the opto-isolator would drop the voltage into half if one is using a Y-connector, causing the failure in digital logic.

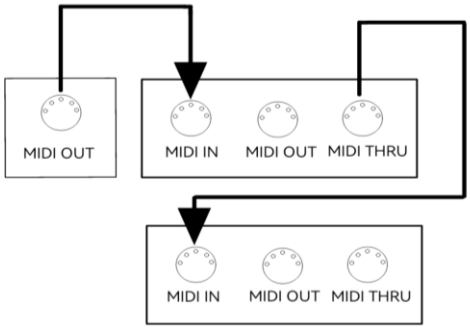

Figure 2. MIDI-THRU connection (Lehrman, 1993)

The MIDI Communication Protocol

According to Figure 1, the data transmission uses a standard protocol called UART. The UART signal usually starts with a start bit (0), followed by 8 data bits, a parity bit, then one stop bit (1). The MIDI, however, does not use the parity bit.

To get started with the 8-bit values, the most significant bit (MSB, left-most bit in our case) should be looked at first. This bit indicates the nature of the data string. If the MSB is 0, this would be a data string. If the MSB is 1, this would be a command (status) string. In a command string, the most significant nibble represents the MIDI command code (examples of which could be found in Table I); the least significant nibble represents the MIDI channel information.

Table I: MIDI Command Set

Name	Hex Value	Dec. Values	Data Bytes
Note Off	80-8F	128-143	2
Note On	90-9F	144-159	2
Key Pressure	A0-AF	160-175	2
Control Change	B0-BF	176-191	2
Program Change	C0-CF	192-207	1
…	…	…	…
System Real Time	F8-FF	248-255	0

The "Data Bytes" in Table I means the number of data bytes that comes with the command. For example, the "Note On" command is followed by two data bytes, "note number" and "velocity". As the first bit is always "0" for the data bytes, the bit depth of MIDI data is 7-bit—giving 128 steps of value.

Running Status

The value of velocity is related to something called "implicit Note Off", which means when a "Note On" command with velocity value being 0 is seen, the command is equivalent to a "Note Off". This is part of the protocol's "Running Status" notion, in which "only the first status byte is transmitted, and successive messages simply transmit new data bytes" (Diakopoulos, 2011).

Usually, when three notes are being transmitted, the message looks like:

0x90, 0x3C, 0x40
0x90, d0x3D, 0x40
0x90, 0x3E, 0x40

With "Running Status", the message then becomes:

0x90, 0x3C, 0x40, 0x3D, 0x40, 0x3E, 0x40

Advanced Message

Under some circumstances, 7-bit values are just not enough. One example would be the pitch bend (slight change of the pitch of a music note in a continuously variable manner). The format of a pitch bend message would be:

0xEc, 0xLL, 0xMM

In which c is the MIDI channel number, LL is the 7 least-significant bits of the value and MM is the 7 most-significant bits of the value.

This gives total 14-bits value for the pitch bend command, which means there could be total 16384 possible values that control the pitch bend of a note.

Control Change

Obviously, there cannot be only 11 types of controllers for a MIDI device. Therefore, "Control Change" (CC) was introduced. The message format for CC is:

0xBc, 0xcc, 0xvv

Where c is the MIDI channel number, and cc is the controller number. The controller numbers are implemented to represent some physical input (such as a knob on an electric keyboard or a slider on a mixing board). The value of those inputs is represented by vv.

Typically, controller number "0" represents the modulation wheel of an electrical instrument. Some other controller numbers are specified in the original MIDI specification. However, there are a lot of controller numbers that are not specified in the MIDI

document, which means they are completely left to the manufacturer to decide. This feature of MIDI gives one the chance to design their own controller commands. (Byron, 2015)

MIDI Over USB

Connection

The connection is pretty straightforward as it is not different from an UART-USB connection.

One important thing to notice is that most of the modern MIDI controllers use the USB-HID protocol (the one for mouses and keyboards) to communicate with a computer. This is a more ideal method than the native serial bus discussed in *Implementation with Max* (Diakopoulos, 2011).

Weakness

One weakness of MIDI over USB is that, unlike the original MIDI jacks which has both MIDI-IN and MIDI-OUT, the traditional USB requires the host-device relationship. With many MIDI devices nowadays only provide one USB device port, such as a micro-USB jack, the ability of two-way controlling is lost. This means for two USB-MIDI devices to talk with each other, a mutual host must be present. Usually, a computer with multiple USB ports is sufficient and the communication routing could be set easily using a DAW. Another possible solution is using a USB Type-C cable, but this would require both devices to support the USB 3.0 protocol and have more powerful microprocessors.

Conclusion

Both the MIDI and the USB protocols are industry standards, and they both have come a long way to bring people more efficient workflow. MIDI was designed to be low-cost and “low-tech” so that it is reachable by all manufacturers and designers. The “Control Change” command in the MIDI specification enables manufacturers and users to customize how the devices are to be controlled.

References

1. P. D. Lehrman, T. Tully and R. Moog, “What is MIDI,” in MIDI For The Professional, 1st ed. New York, NY, USA: Amsco Publications. 1993.

2. Swift, A. (2012). A brief introduction to MIDI, A brief introduction to Midi. Retrieved February 11, 2022, from https://web.archive.org/web/20120830211425/http://www.doc.ic.ac.uk/~nd/surprise_97/journal/vol1/aps2/.

3. Universal Serial Bus Specification (PDF) (Technical report). 1996. p. 29. v1.0.

4. Place, T., Lossius, T. (2006)"A modular standard for structuring patches in Max" In Proceedings of the International Computer Music Conference., Jamoma. New Orleans, US, 2006. pp. 143–146.

5. Kosky, P. G., Balmer, R. T., Keat, W., & Wise, G. (2021). Microcontrollers and Arduino. In Exploring engineering: An introduction to engineering and Design (pp. 457–458). essay, Academic Press.

6. Byron, J. (2015, October 8). MIDI Tutorial. MIDI tutorial. Retrieved April 8, 2022, from https://learn.sparkfun.com/tutorials/midi-tutorial/all

7. Diakopoulos D., Kapur, A.(2011). HIDUINO: A firmware for building driverless USB-MIDI devices using the Arduino microcontroller