Abstract

MIDI and their associated MIDI controllers are an invaluable tool in the music production process for both the aspiring enthusiast and working professional. However, the marketplace for MIDI controllers is dominated by expensive and complicated devices that are unaffordable for many artists. The question posed is whether a cheap and personalized MIDI controller can be built using an affordable microcontroller. Using an Arduino UNO and with the help of some third-party software, we demonstrate the build for a functional MIDI controller with 3 buttons and 2 potentiometers.

DIY MIDI Controller

By Arunkarthik Archunan,

Aaron Carlisle,

Hunter Fourt

Introduction - The need for MIDI

Over the last couple of decades, one of the most exciting developments in music technology has been the lowered cost at which artists can build personalized home recording studios. As the technology has improved, it is now feasible to record professional quality sounds on a limited budget. This has resulted in many of the top-selling artists today writing and recording music in their homes. Similarly, many world-renowned audio engineers are now moving away from expensive, outboard analog gear to the much cheaper and widely available software alternatives.

Some of the most popular analog instruments, effects, EQ channels, and audio compressors are now being simulated through software driven design at a fraction of the cost. If a sought-after piece of audio gear can be digitally modeled, it is likely to be emulated. This has led to a new type of market demand in the form of computer interface devices known as MIDI controllers. MIDI controllers allow a user to interact hands-on with their recording software in a way that's far more intuitive than a mouse or keyboard.

Practical Considerations – Defining the problem

With the ability to turn electronic signals into music and the powerful software options available, MIDI has become the fastest growing standard in the music industry for both professionals and enthusiasts. But such an invention does not come without some serious downsides.

Above all, it's not unreasonable to ask: *Where should I start?*

Getting into MIDI, understanding what it is and how it works, finding a device and software that suits your needs and goals – as well as your budget, is no simple task. The cost of a traditional MIDI setup is also nothing to scoff at. Between the MIDI device itself, editing and mixing software, and additional sound libraries, you could spend hundreds or even thousands of dollars before writing a single note.

The degree of scalability, customization, and capability are also rather limited for these devices. If you were to buy a cheap MIDI keyboard and use some free software, you would quickly find yourself unable to get a specific sound that you might have been envisioning. Worse yet you might now feel obligated to get a better board with more features and functionality. Perhaps you'd even pay a subscription fee to unlock those new sounds as you find yourself digging a deeper and deeper financial hole in the pursuit of your art.

The solution for this exhausting and multifaceted problem is rather simple. We can build our own device and give it as much or as little functionality as we want. We can do this cheaply, safely, and quickly utilizing something called an Arduino UNO. With a combination of a few buttons, a few potentiometers, 14 jumper wires, and a breadboard, we can convert an Arduino into a MIDI input device that's capable of controlling virtual instruments.

Arduino – What is it?

Arduino is an open-source electronics platform for microcontrollers that are designed for ease of use when it comes to both hardware and software. Arduino is cross-platform, and the open-source nature of the software and hardware allows an ever-growing community to contribute new libraries and circuit designs that are freely available to everyone. The attraction of Arduino is that it is very cheap and user friendly while also being extremely scalable depending on the project, thus making it a perfect tool for students to learn the basics of software and hardware design.

Arduino was created in 2005 through the work of Hernando Barragán, Massimo Banzi, David Cuartielles, Gianluca Martino, and Tom Igoe. The goal was to create "an easy-to-use programmable device for interactive art design projects" (Hughes). They

utilized 8-bit microcontrollers from Atmel and “designed a self-contained circuit board with easy-to-use connections, wrote bootloader firmware for the microcontroller, and packaged it all into a simple integrated development environment (IDE) that used programs called “sketches.”” (Hughes).

An Arduino board at its most basic level is capable of reading inputs and generating outputs. For example, you could press a button on your board (input) and generate a sound from that (output). At the more advanced end, Arduino can be the brains of incredible machines from farming bots to recycling sorters and beyond.

MIDI – A brief overview

MIDI is an electronic standard which allows musical instruments to communicate with each other and to communicate with computer programs. It does not produce any sound on its own. The acronym stands for Musical Instrument Digital Interface. It consists of a sequence of messages that tell digital instruments or computer software what notes to play, the velocity for each note, the volume, and more. This lets MIDI effectively transmit musical information allowing us to create playable instruments.

MIDI was originally created in 1983 to improve the compatibility across affordable electronic synthesizers. The standard was completed in 1983 with the cooperation of various well-known musical instrument manufacturers including Korg, Oberheim, Roland, Sequential Circuits, and Yamaha (Gibson). Today, MIDI is used across a wide range of applications, including music production and performances.

Using computer software enables musicians and producers to record MIDI messages and store them as musical performance data. This has become an essential part of the modern music recording process. Producers can send inputs to different parts of their recording software and affect the various parameters while performers can utilize MIDI instruments to orchestrate, arrange and edit their digital recordings. This is in contrast to mouse and keyboard shortcuts.

Connecting MIDI to the PC

Since MIDI is primarily used for sending and receiving musical performance data, the cables and connections between MIDI devices are interchangeable. A traditional MIDI cable consists of a 5-pin “DIN” style connector that is unidirectional. This means that two cables are needed to both transmit and receive MIDI information. Connecting to a computer requires an additional piece of hardware known as an audio interface. The interface is used to convert parallel data bytes from the PC data bus to the serial MIDI format (Heckroth).

While many audio interfaces allow for 5-pin MIDI connectivity, most manufacturers have opted to build USB ports into their MIDI devices. USB has the advantage of being bidirectional, which eliminates the need for an extra cable. The standardization of USB has also allowed for class compliant drivers. This enables users to plug their controllers directly into their computers without having to install any additional software. Although MIDI has been previously used with FireWire, Ethernet, and even Wi-Fi, USB has become the most common way to connect computers, tablets, and smartphones to MIDI devices (MIDI Association). This is the implementation we will be using.

The UNO and USB over MIDI

One setback when using the UNO is that the USB port functions a bit differently than the connection used on a typical USB peripheral. In what Arduino calls a “programming port,” the UNO uses an ATmega16U2 microcontroller as a USB-to-serial converter to receive instructions from the IDE. By comparison, the more expensive Arduino Due includes a “Native USB port” that supports the host/client configuration required for standard USB devices (Arduino).

However, we can solve this problem with the use of third-party software. Hairless MIDI, by Angus Gratton, is a free program released under the GNU Lesser General Public License 2.1 that converts serial data into MIDI commands. It is described as “the easiest way to connect serial devices (like Arduinos) to send and receive MIDI signals”

(Hairless MIDI). This program allows us to bypass the hardware limitations of the UNO while maintaining the necessary bandwidth to create a functional controller.

Virtual MIDI Cable

Our builds are assembled on Windows 10/11. Because of this, we come across the issue of needing an internal MIDI connection that's typically provided by the software component of a consumer audio interface. This "Virtual MIDI Cable" is what provides a connection between Hairless MIDI and the virtual instrument of our choice. For our design, we opted to use the software LoopBe1 by nerds.de which is free for non-commercial, personal use (Schmitt.de).

There are several free alternatives available including loopMIDI by Tobias Erichsen (Tobias Erichsen), and we make the assumption that any one of these options will suffice for our implementation. Additionally, while we have not tested a build with Mac OS, it is our understanding that Mac users have built in internal MIDI routing, and we assume that this setup will work in those scenarios as well.

The Arduino IDE and MIDI Library

After following the setup guide for the UNO, we will need to install a MIDI Library that's listed in the Library Manager for the Arduino IDE. The library is called "MIDI Library" by Francois Best and allows the Arduino to read and send MIDI messages (Best). The version we are using is 5.0.2.

Our code is based on a program written for several Arduino microcontrollers by Gustavo "Nerd Musician" Silveira. We can start by removing a lot of the code that's used for other microcontrollers. We won't be needing this since we are only using the UNO. Next, we set the constant variables N_BUTTONS and N_POTS to represent the number of buttons and potentiometers. It's also important to define the pins for each button and potentiometer that is connected to the Arduino. We can do this by setting the index values in the corresponding arrays for BUTTON_ARDUIN_PIN[N_BUTTONS] and

POT_ARDUINO_PIN[N_POTS] where values of 2-13 are used for digital button inputs, and values A0-A5 are used for analog potentiometers.

We also need to decide the beginning MIDI note for the first button in our array. Middle C on a piano is typically represented as note 60 and most virtual drum kits start their percussion maps at note 36 for the kick drum. We will set our starting note to 36.

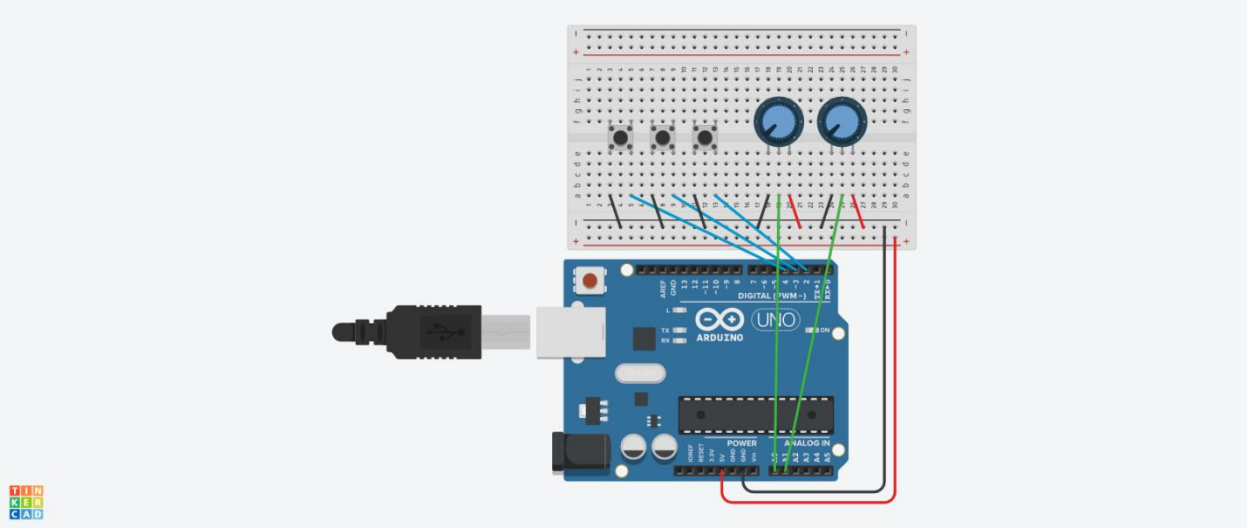
Finally, it's worth noting some of the information that we've learned during our process and through troubleshooting. MIDI uses a form of serial to communicate at a data rate of 31250 maximum bits per second with each byte made of 8 bits plus a stop and start bit. MIDI commands are most often three bytes in length but sometimes more or less.

Arduinos use a form of serial data transmission, in order to communicate data with a computer. The baud rate of a program is the maximum number of bits per second. After some experimentation, we found that a baud rate of 115200 bits per second worked best with our program. According to the Arduino documentation, the default configuration of the data is SERIAL_8N1.

On our way, we also learned that MIDI was just another form of serial data. In fact, MIDI connectors use a UART to transmit the parallel data from a computer into serial data. Therefore, it is considered a form of asynchronous data itself (Benitez).

Hardware Build

For this build we have several material requirements. First, we need a microcontroller, in our case we are using an Arduino UNO R3. Next, we will need a breadboard, three push buttons, two potentiometers, fourteen M-M wires of varying length and a single Arduino UNO USB Data Sync cable. We will be arranging these components in a configuration that looks something like this:



With this configuration, N_BUTTONS = 3 and N_POTS = 2. The corresponding arrays are BUTTON_ARDUINO_PIN[N_BUTTONS] = {2, 3, 4}; and POT_ARDUINO_PINS[N_POTS] = {A0, A1};

Putting it all together

With the Arduino connected and Hairless MIDI open, we are now processing serial data and converting it into MIDI input. In combination with virtual MIDI routing, we can now control any Digital Audio Workstation (DAW) or Virtual Studio Technology (VST) with MIDI mapping functionality. To test this, there are dozens of free-to-use VST Instruments available online that include MIDI mapping. For our demonstration we used SUB, a vintage analog drum module with a built-in frequency oscillator (Sampleson). Similar to a modern PC game, it's just a matter of keybinding the buttons on the controller to any of the functions in the VST or DAW interface. In the case of SUB, simply right-click the control you'd like to key map.

Conclusion

The freedom to write and record music with an interface device that artists are more familiar with aids in the creative process and leads to a more productive studio session. The tactile feedback of physical faders and potentiometers enables sound engineers to write and record precision automation. The universality of MIDI gives live

performers flexibility to sync their music with lights and effects like never before. For these reasons and more, MIDI stands at the forefront of musical innovation.

But without cheaper ways to enter into the space, many artists won't be able to take advantage of this revolution in music production. However, we have demonstrated that a low cost and easy to build MIDI controller is indeed possible with an Arduino microcontroller. It is our hope that projects like ours will offer an alternative to some of the more costly controllers and provide artists the flexibility and freedom to be more engaged in their creative processes without limitation.

Works Cited

"Basics of USB-MIDI." *The MIDI Association*, https://www.midi.org/midi-articles/basic-of-usb.

Benitez, Rylan. "Is MIDI a UART? | Trekkyrecords.Com." Www.Trekkyrecords.Com, www.trekkyrecords.com/is-midi-a-uart. Accessed 21 Apr. 2022.

Best, Francois. "FortySevenEffects/arduino_midi_library: MIDI for Arduino." *GitHub*, FortySevenEffects, https://github.com/FortySevenEffects/arduino_midi_library.

Erichsen, Tobias. "LoopMIDI | Tobias Erichsen." *Tobias Erichsen*, https://www.tobias-erichsen.de/software/loopmidi.html. Accessed 21 Apr. 2022.

"Getting Started with the Arduino Due." *Arduino*, 1 Oct. 2017, https://www.arduino.cc/en/pmwiki.php?n=Guide%2FArduinoDue.

Gibson, John. "The MIDI Standard: Introduction to MIDI and Computer Music: Center for Electronic and Computer Music: Jacobs School of Music." *INDIANA UNIVERSITY BLOOMINGTON*, cecm.indiana.edu/361/midi.html. Accessed 6 Apr. 2022.

Gratton, Angus. "The Hairless MIDI to Serial Bridge." *The Hairless MIDI<->Serial Bridge*, Free Software Foundation, https://projectgus.github.io/hairless-midiserial/.

Heckroth, Jim. "Tutorial on MIDI and Music Synthesis." *The MIDI Association*, The MIDI Manufacturers Association, Apr. 2006,

https://www.midi.org/specifications/developer-white-papers/tutorial-on-midi-and-music-synthesis-2. Accessed 22 Apr. 2022.

Hughes, J. M. (n.d.). *Arduino: A technical reference*. O'Reilly Online Learning. Retrieved April 5, 2022, from https://www.oreilly.com/library/view/arduino-a-technical/9781491934319/ch01.html

"MIDI | Definition, Meaning, and Facts." *Encyclopedia Britannica*, www.britannica.com/art/MIDI-music-technology. Accessed 6 Apr. 2022.

Silveira, Gustavo. "Silveirago/DIY-MIDI-Controller." *GitHub*, The Nerd Musician, https://github.com/silveirago/DIY-Midi-Controller. Accessed 22 Apr. 2022.

"Sub. Free Analog Drums." *Sampleson*, Sampleson, https://sampleson.com/SUB-free-analog-drums.html. Accessed 22 Apr. 2022.

Team, T. A. (n.d.). *What is Arduino?* Arduino. Retrieved April 5, 2022, from https://www.arduino.cc/en/Guide/Introduction. Accessed 22 Apr. 2022.

Appendix

```
/*

 * Credit:

 * Original code made by Gustavo Silveira, 2019.

 * http://www.musiconerd.com

 * http://www.youtube.com/musiconerd

 *

 * This code was modified by Aaron Carlisle, Arunkarthik Archunan, and Hunter
      Fourt,to be used with the Arduino UNO

 * This Sketch reads the Arduino's digital and analog ports and send midi notes and
      midi control change

*/



#define ATMEGA328 1



#include <MIDI.h> // by Francois Best

MIDI_CREATE_DEFAULT_INSTANCE();




// BUTTONS

const int N_BUTTONS = 9; //*  total numbers of buttons

const int BUTTON_ARDUINO_PIN[N_BUTTONS] = {2, 3, 4, 5, 6, 7, 8, 9, 10}; //* pins of
      each button connected straight to the Arduino



int buttonCState[N_BUTTONS] = {};        // stores the button current value

int buttonPState[N_BUTTONS] = {};        // stores the button previous value
```

```
// debounce

unsigned long lastDebounceTime[N_BUTTONS] = {0};  // the last time the output pin
      was toggled

unsigned long debounceDelay = 50;    // the debounce time; increase if the output
      flickers



// POTENTIOMETERS

const int N_POTS = 1; //* total numbers of pots (slide & rotary)

const int POT_ARDUINO_PIN[N_POTS] = {A0}; // pins of each pot connected straight to
      the Arduino



int potCState[N_POTS] = {0}; // Current state of the pot

int potPState[N_POTS] = {0}; // Previous state of the pot

int potVar = 0; // Difference between the current and previous state of the pot



int midiCState[N_POTS] = {0}; // Current state of the midi value

int midiPState[N_POTS] = {0}; // Previous state of the midi value



const int TIMEOUT = 300; // Amount of time the potentiometer will be read after it
      exceeds the varThreshold

const int varThreshold = 10; // Threshold for the potentiometer signal variation

boolean potMoving = true; // If the potentiometer is moving

unsigned long PTime[N_POTS] = {0}; // Previously stored time

unsigned long timer[N_POTS] = {0}; // Stores the time that has elapsed since the
      timer was reset



// MIDI

byte midiCh = 1; // MIDI channel to be used
```

```
byte note = 60; // Lowest note to be used

byte cc = 1; // Lowest MIDI CC to be used




void setup() {


  Serial.begin(115200);


  // Initialize buttons with pull up resistors

  for (int i = 0; i < N_BUTTONS; i++) {

    pinMode(BUTTON_ARDUINO_PIN[i], INPUT_PULLUP);

  }


}


void loop() {

  buttons();

  potentiometers();

}


void buttons() {


  for (int i = 0; i < N_BUTTONS; i++) {

    buttonCState[i] = digitalRead(BUTTON_ARDUINO_PIN[i]);  // read pins from
      arduino
```

```
    if ((millis() - lastDebounceTime[i]) > debounceDelay) {
      if (buttonPState[i] != buttonCState[i]) {
        lastDebounceTime[i] = millis();
        if (buttonCState[i] == LOW) {
          // Sends the MIDI note ON
          MIDI.sendNoteOn(note + i, 127, midiCh); // note, velocity, channel
        }
        else {
          // Sends the MIDI note OFF
          MIDI.sendNoteOn(note + i, 0, midiCh); // note, velocity, channel
        }
        buttonPState[i] = buttonCState[i];
      }
    }
  }
}

void potentiometers() {


  for (int i = 0; i < N_POTS; i++) { // Loops through all the potentiometers

    potCState[i] = analogRead(POT_ARDUINO_PIN[i]); // reads the pins from arduino

    midiCState[i] = map(potCState[i], 0, 1023, 0, 127); // Maps the reading of the
      potCState to a value usable in midi
```

```
potVar = abs(potCState[i] - potPState[i]); // Calculates the absolute value
  between the difference between the current and previous state of the pot



if (potVar > varThreshold) { // Opens the gate if the potentiometer variation
  is greater than the threshold

  PTime[i] = millis(); // Stores the previous time

}



timer[i] = millis() - PTime[i]; // Resets the timer 11000 - 11000 = 0ms



if (timer[i] < TIMEOUT) { // If the timer is less than the maximum allowed time
  it means that the potentiometer is still moving

  potMoving = true;

}

else {

  potMoving = false;

}



if (potMoving == true) { // If the potentiometer is still moving, send the
  change control

  if (midiPState[i] != midiCState[i]) {



    // Sends the MIDI CC

    MIDI.sendControlChange(cc + i, midiCState[i], midiCh); // cc number, cc
  value, midi channel



    potPState[i] = potCState[i]; // Stores the current reading of the
  potentiometer to compare with the next
```

```
          midiPState[i] = midiCState[i];

        }

      }

    }

}
```