Design Document: USB MIDI Controller Arduino (KLP-MMX-M0X)

John Jones

March 4th, 2023

Table of Contents:

1.1 – Introduction:

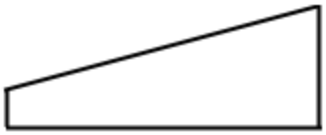
Music Instrument Digital Interface (MIDI) is a communication standard for music that can connect to electrical instruments, computers, and capable audio devices. MIDI encodes audio information as musical instructions, no actual audio is sent through MIDI but rather the instructions, such as piano note presses, are sent to the end device which then produces audio. MIDI information is very small compared to audio information. A typical CD standard quality song is encoded at a frequency of 44.1kHz and a bit depth of 16 bits, which means that every second 44,100 * 16 * # of channels (typically 2) bits are saved. In comparison, MIDI information is 3 bytes large per note, and by using a short snippet of audio (typically 1 second long) producers were able to emulate an audio recording with far less storage space. This standard provided great flexibility to music producers in the 80s and 90s by allowing less information to be stored on their computers for their songs. Storage space has become more readily available and has become less of a concern since MIDI's initial conception in 1981, however, MIDI is still used today for its convenience and flexibility. Many digital instruments have MIDI ports on them and can be controlled by a MIDI controller, sequencer, or any other MIDI-sending device. MIDI channels can easily swap out the audio samples and allowed music producers to hear how their composition would sound on different instruments. As a music producer myself I use MIDI in every track I produce and wanted to control many attributes of the music at once. This MIDI Controller allows you to control three knobs and three keys at once and allows you to swap between channels to have more control.

1.2 – Design Considerations:

I wanted this device to be easy to use. I wanted to emulate how music hardware is designed, such as AKAI and Behringer's hardware, and how they built their devices to be simple to set up with people with minimal knowledge on computers to figure out. To accomplish this, I chose a board with an on-board USB connection to allow for the device to be powered by USB as well as communicating between the MIDI end-device (a computer) using only one connecting wire.

I wanted to use as little hardware as possible. To do so I programmed the buttons to act in multiple ways. The buttons act as typical MIDI note triggers, a virtual switch to switch between modes, and a means to increase/decrease the channel selected. This cut down on the number of buttons on the board while maintaining the functionality. I was unable to cut down on the number of LEDs on board without the user not having feedback on what was happening. (see section 2.1)

I wanted it to be small, portable, and handle easily. I was able to achieve a small size by limiting the number of components in the project and I made it portable by only requiring one USB cable and the box to transport it. Having it handle easily was a question of the outer shell design in my eyes, and I wanted to design the shell to feel inviting to the people who were using it. My solution was to slant the device so the components faced the person using it, allowing all the components to be visible to the user without them looking directly down at it.



Slanting the controller to face the user 'invites' them to use it.

I wanted it to control many values. I achieved this by storing a channel offset value ranging from [0, 13] which will change the MIDI channel the knobs are linked to. The user can modify this value by entering the offset changing mode by holding down the L and R buttons until the status LED turns on.

2.1 – Hardware Design:

The KLP-MMX-M0X was built on a 5V Sparkfun Pro Micro board and uses three 100k potentiometers, three push buttons, and three LEDs. The Pro Micro board features an ATmega32u4 microcontroller with support of analog and digital pins, PWM pins, and an on-board USB type-B port. I chose this board because the ATmega32u4 processor allows for MIDI information to be sent over serial.

The buttons are connected to digital pins, potentiometers are connected to analog pins, and the LEDs are connected to PWM-enabled pins (5,6, and 9) for dimming because the LEDs were extremely bright when turned on.

I wanted to use as little hardware as possible and to do so I had the buttons work in two different ways, similar to the TD3 design and how buttons have different functions in different contexts. LEDs are important and are the only visual indicators for users. Three total LEDs were used to indicate whether the offset mode is turned on, and to indicate when the user has reached the edge of the available virtual channels. I believed that removing any more LEDs would not give enough information to the user and would affect the user experience.

2.2 – Software Design:

The program is relatively simple: it reads the values of the three potentiometers and the three buttons and sends MIDI data using the read inputs. Messages will only be sent if there is a change to the read value since its last read. While only using three knobs the program can control up to 16 channels utilizing a channel offset value. To change the offset value the user must hold down the L and R buttons at the same time until the status LED turns on and the user can increase/decrease the offset value by clicking the L and R buttons.

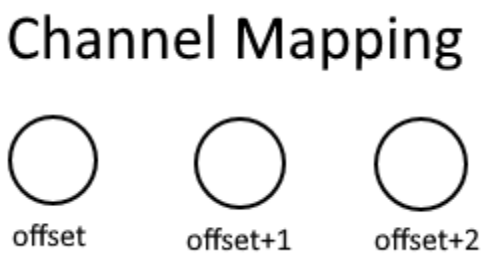


Diagram demonstrates how the knobs map to MIDI channels

2.3 – Software Considerations:

Touchy potentiometers: (See section 2.4)

Changing modes: The program will increment a variable while the L and R buttons are held down until it reaches a value and the offset mode flips on/off. All buttons are read before any action is performed.

2.4 – Issues encountered and their solutions:

Touchy potentiometers: I originally programmed the Potentiometers to only send MIDI packets if there was a change to it's value since its last check, but the Potentiometers would read different values even if it wasn't being touched which would flood the computer with MIDI packets. To remedy this, I added a change limit value (about 3) to check before sending MIDI; if the change in value is greater than the limit value then it will send MIDI. This did not affect on accuracy because the potentiometer's read in a range of 0-1023 while MIDI is limited to 0-127, so the value would be divided anyways and a change of ~3 would not affect the reading.

Ableton Connection: Ableton will not receive information from the device unless the "Remote" preference is checked in the MIDI Preferences tab of settings. Simply checking the box fixes this issue.

Buttons clicking: The buttons were the hardest thing to get right in this project, as they had to be physically touching the ground for them to click correctly. I solved this by hot gluing the buttons down to a piece of cardboard and connecting it to the top cardboard piece with a piece of cardboard measured to the bottom of the box. This allows the top to be removed easily to fix issues and the buttons can be pressed properly. I used plastic straws as button extenders.

2.5 – Future

While using this device I realized a few issues with how it handled:

1. Switching between channels would jolt the channel to the position the knob is set to, making changes to multiple channels harsh sounding

2. When changing modes, stray MIDI notes would be sent if both buttons were not clicked at the same time

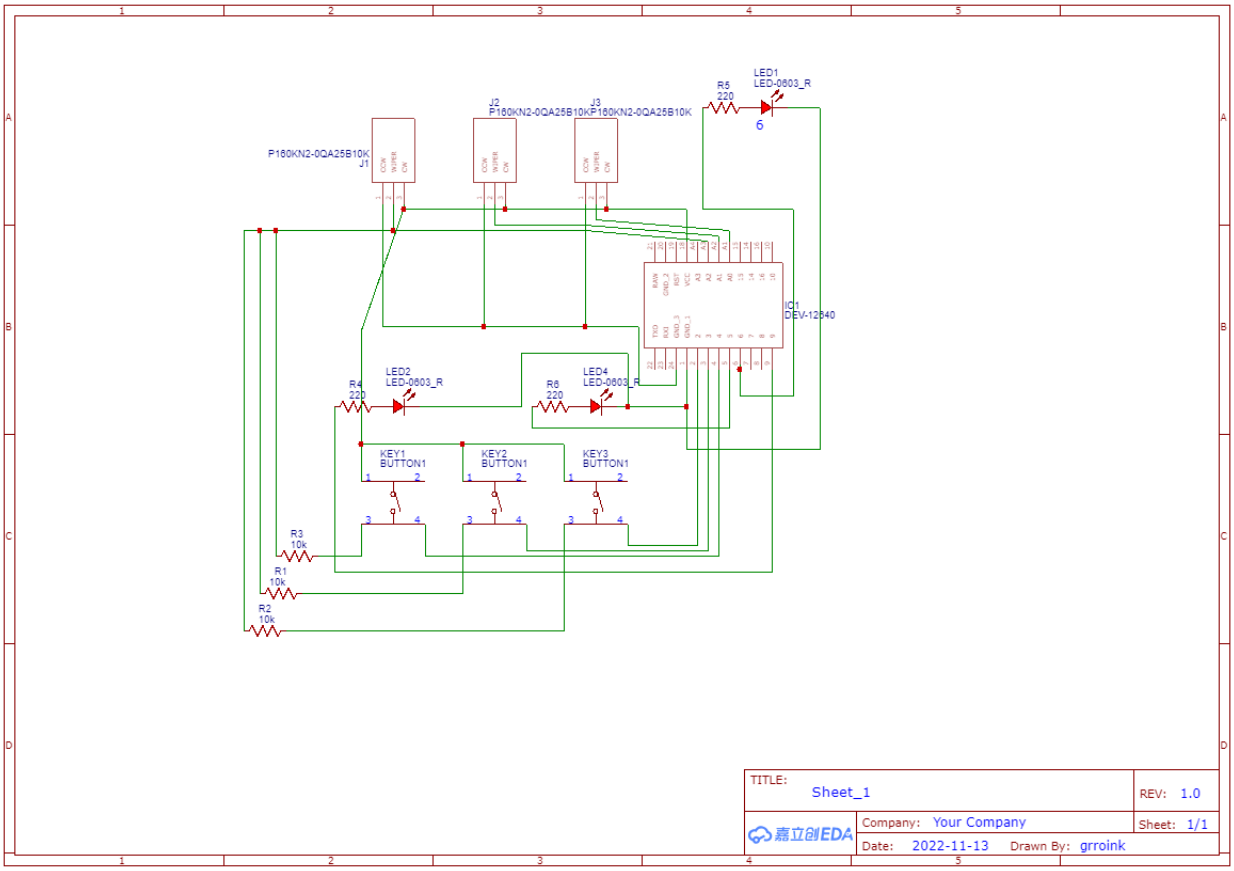
To fix these issues, the software can be reprogrammed as follows:

1. Instead of sending the value of the potentiometer as it is seen, the potentiometers can control the CHANGE to the selected channel. By storing all 16 channels and their current value we can track their current value and increase/decrease its value when the knob is turned to the right/left respectively.
2. Waiting for a few cycles when a button is clicked before sending the MIDI note from button clicks will prevent those stray notes from occurring

3.1 – Final product:

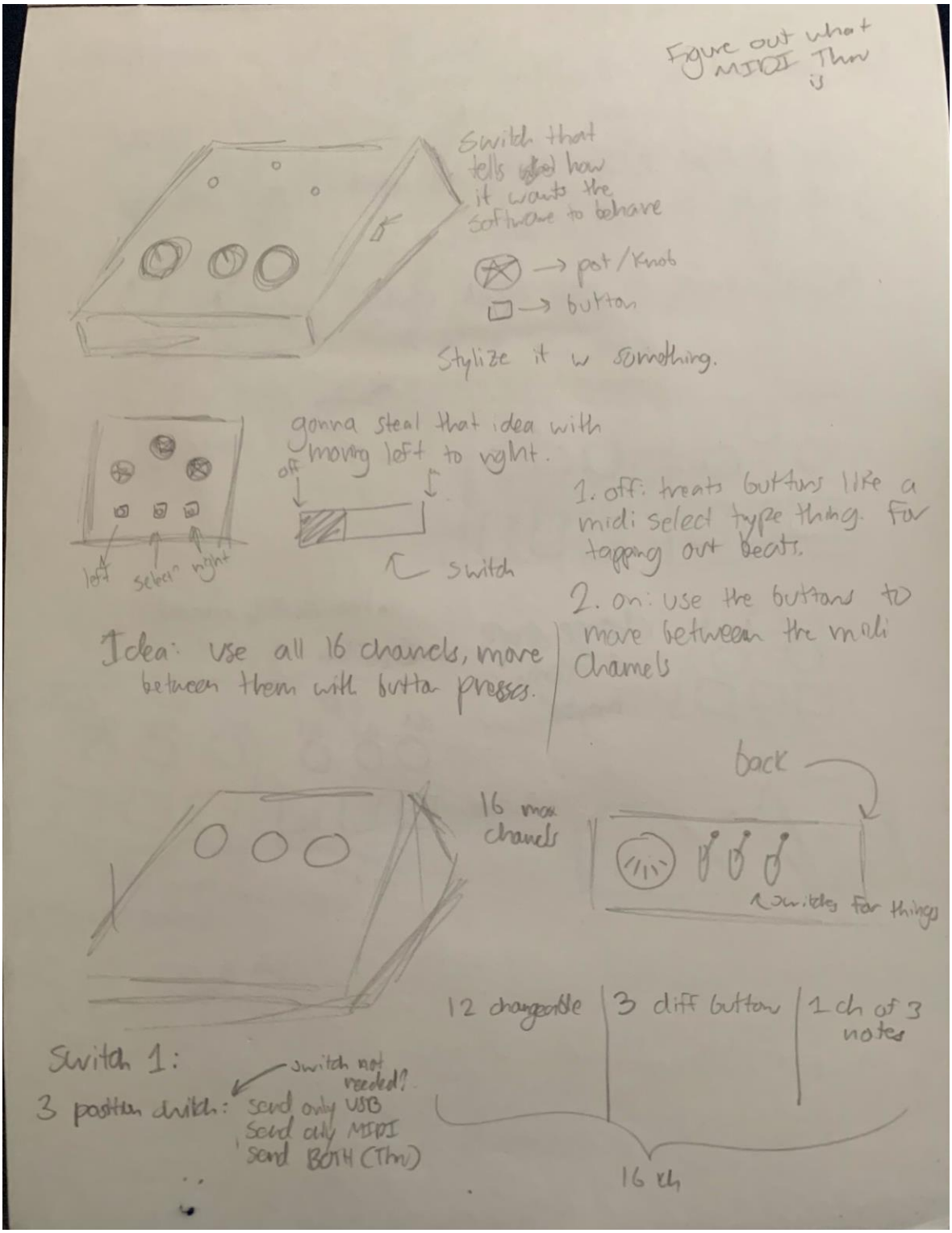
Screenshots from my YouTube video

3.2 – PCB Design:



Simply connecting the components to the Pro Micro.

3.3 – Initial Designs:

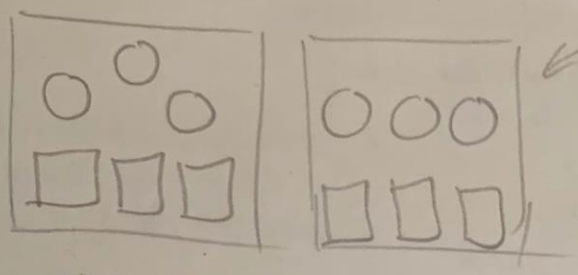


I originally designed the device to have a physical MIDI port as well as the USB

Can just send to USB without worrying. ← :)

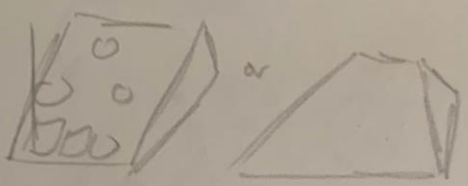
Switch for using buttons as selector / using them as midi note info stuff.

Physical memory to store which config? or not



Going but works :(

change shape of extension maybe?



? lol

When switch moved.

Switch - 1 :

OFF:

ch1 ch2 ch3

ch 4 ?

ON

x range: [5-14?]

ch x ch x+1 ch x-2

midi OFF (selects the value of x)

guh :)

4 – Conclusion

This was my first personal project with Microcontrollers and it was very fun and I learned a lot from it. I hope to continue and work on more projects in the future!

If you have any questions, suggestions, or want to talk to me feel free to reach out to me at johnwjones1337@gmail.com