Plug-and-Play MIDI Keyboard 25-Key Open-Source Instrument for Educators & Makers

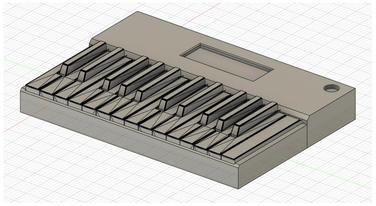
Electrical Engineering Department

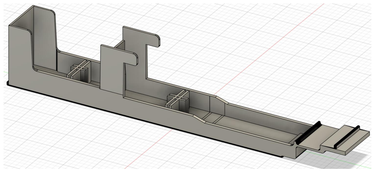
Danylchuk, Michael (BS Electrical Engineering) Hatchett, Zac (BS Electrical Engineering)

Project Advisor: Nadir Mir

Introduction

Music programs are often the first to be cut when school budgets shrink. Traditional MIDI keyboards are expensive, difficult to repair, and not built with students in mind. Our project solves these issues by creating a fully open-source, ultra-low-cost (~$50), 25-key MIDI keyboard that operates immediately when plugged into any computer.

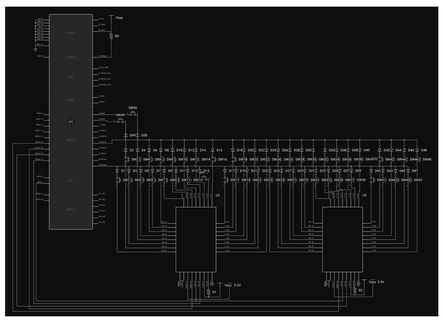




This project was motivated by the need to give students access to functional, modern instruments without burdening schools with recurring costs. We wanted to build something any classroom, club, or beginner could use and repair without relying on expensive commercial hardware.

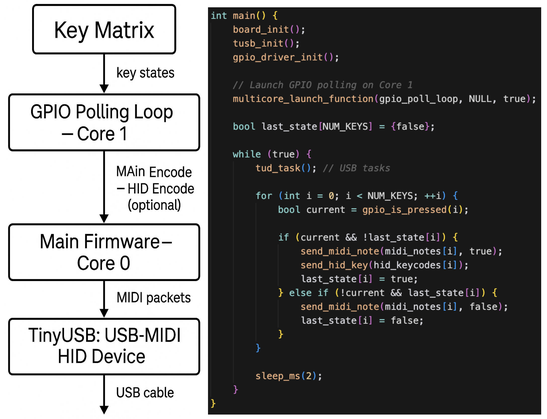
Methodology

Hardware Development:
We built the keyboard around the Raspberry Pi Pico 2W, using a dual-switch matrix for velocity sensing. Each key drives two scan lines routed through multiplexers, letting the RP2040 read press order and compute velocity in firmware. The enclosure and key bed were fully 3D-printed, designed for full-size feel, easy repairs, and efficient printing with minimal failure points.



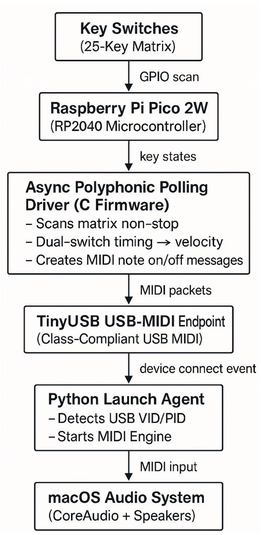
Methodology

Firmware Implementation:
The microcontroller runs custom C firmware built on TinyUSB to generate class-compliant USB MIDI messages. Timing between the two switches on each key is measured to calculate velocity for expressive playback.




```c
int main() {
    board_init();
    tusb_init();
    gpio_driver_init();

    // Launch GPIO polling on Core 1
    multicore_launch_function(gpio_poll_loop, NULL, true);

    bool last_state[NUM_KEYS] = {false};

    while (true) {
        tud_task(); // USB tasks

        for (int i = 0; i < NUM_KEYS; ++i) {
            bool current = gpio_is_pressed(i);

            if (current && !last_state[i]) {
                send_midi_note(midi_notes[i], true);
                send_hid_key(hid_keycodes[i]);
                last_state[i] = true;
            } else if (!current && last_state[i]) {
                send_midi_note(midi_notes[i], false);
                last_state[i] = false;
            }
        }

        sleep_ms(2);
    }
}
```

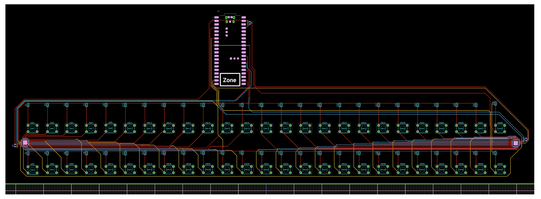
Software / Sound Engine:
On the host computer, a Python-based sound engine processes incoming MIDI data and performs real-time polyphonic audio synthesis. This approach ensures cross-platform compatibility with no drivers or setup required.



System Validation Overview:
We evaluated the keyboard's latency, polyphony, velocity accuracy, and compatibility across platforms. The results confirm stable real-time performance suitable for classroom and hobbyist use.

Analysis and Results

Hardware Performance Summary:
The custom PCB design, diode-isolated matrix, and dual-row velocity sensing achieved stable key scanning at 1 kHz with clean signal isolation across all 24 columns, validating the hardware architecture for full production feasibility.

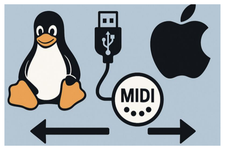


Polyphonic Note Handling:
The Python audio engine supports true polyphony, allowing multiple notes to sound simultaneously. During stress-testing with chords of 4–6 notes, the engine maintained stable playback without audio dropouts or buffer underruns.

```python
import time
import threading
import mido
from engine import play_note, stop_note

def midi_listener():
    print("🎧 Starting Pico MIDI listener thread...")
    while True:
        try:
            # Find the Pico (USB MIDI device)
            inputs = mido.get_input_names()
            pico_input = next((name for name in inputs if 'Pico' in name or 'MIDI' in name), None)

            if not pico_input:
                print("❌ Pico MIDI not found. Retrying in 2s...")
                time.sleep(2)
                continue

            print(f"🎹 Connected to: {pico_input}")
            with mido.open_input(pico_input) as inport:
                for msg in inport:
                    if msg.type == 'note_on' and msg.velocity > 0:
                        print(f"[PICO PLAY] Note {msg.note}")
                        play_note(msg.note, velocity=msg.velocity)
                    elif msg.type in ['note_off', 'note_on'] and msg.velocity == 0:
                        print(f"[PICO STOP] Note {msg.note}")
                        stop_note(msg.note)

        except Exception as e:
            print(f"⚠️ MIDI error: {e}. Reconnecting in 2s...")
            time.sleep(2)
```

Velocity Detection Accuracy:
Dual-switch timing enabled basic velocity estimation. We verified consistent separation of "soft," "medium," and "hard" presses based on Δt between the two contact closures. This provides expressive control beyond simple on/off triggering.

Cross-Platform Compatibility Results:
The device was recognized as a class-compliant MIDI controller on macOS and Linux without any driver installation. HID fallback mode was validated on macOS as a secondary input method, confirming functional compatibility across multiple host platforms.



Summary/Conclusions

This project delivers a functional, low-cost 25-key MIDI keyboard using a custom key matrix, RP2040 firmware, and a Python audio engine. It reliably scans keys, computes velocity, and outputs real-time polyphonic audio over USB-MIDI. The open and repairable design makes it practical for classrooms, hobbyists, and makers, with a clear path for future extensions like improved velocity curves or expanded key counts.

Key References

[1] MIDI Association. "MIDI 1.0 Detailed Specification." www.midi.org/specifications.

[2] TinyUSB Development Team. "TinyUSB: USB Device Stack." https://github.com/hathach/tinyusb.

[3] Raspberry Pi Foundation. "RP2040 Datasheet & Pico SDK." https://github.com/raspberrypi/pico-sdk.

[4] Mido: Python MIDI Library https://github.com/mido/mido.

[5] SoundDevice Library Documentation. https://python-sounddevice.readthedocs.io.

[6] Future Market Insights. "MIDI Controller Market Forecast (2023–2028)."

Acknowledgements

We would like to thank Professor Nadir Mir for his guidance, feedback, and support throughout the project. We also appreciate the Electrical Engineering Department for providing resources and a platform to present our work.

Special thanks to our classmates for testing early prototypes and offering valuable suggestions during development.