EMID Final Project Report: The Chordian

MUS141: Electronic Musical Instrument Design
Professor Paul Lehrman

William Bix von Goeler

December 18, 2024

Figure 1. The completed revised Chordian prototype with accordion straps attached.

Overview

The Chordian is an electronic MIDI instrument designed around the classical bellows-based accordion. While drawing from the general design of acoustic accordions, the Chordian adopts a number of distinct qualities and novel features:

- We target a much smaller lightweight form factor with 8 piano style keys and 12 mechanical keyboard style buttons.
- Traditional accordion bellows are emulated in software by differentiating the velocity of a TOF sensor placed between the two halves, along with an expandable mechanical assembly and springs for better play feel.
- The traditional piano style key bed is augmented with two changes allowing further creative expression: soft potentiometers tracking fingertip position up and down keys, and force sensitive resistors placed underneath semi-flexible metal sheets in place of keys.
- Left hand buttons are designed to trigger preset chords, allowing the user to easily create rich compositions.

After completing the first iteration of the Chordian, we identified several shortcomings that hampered playability: the tangle of wires connecting the instrument to external Arduinos, the imprecise IR-based bellows sensing, and the lack of physical resistance during bellows movement. For this revision, we focused on addressing these issues through wireless communication, a Time of Flight distance sensor, and improved physical construction. The instrument now operates as a self-contained battery-powered device, communicating wirelessly to a desktop receiver that handles MIDI conversion and audio synthesis.

Planning and Design

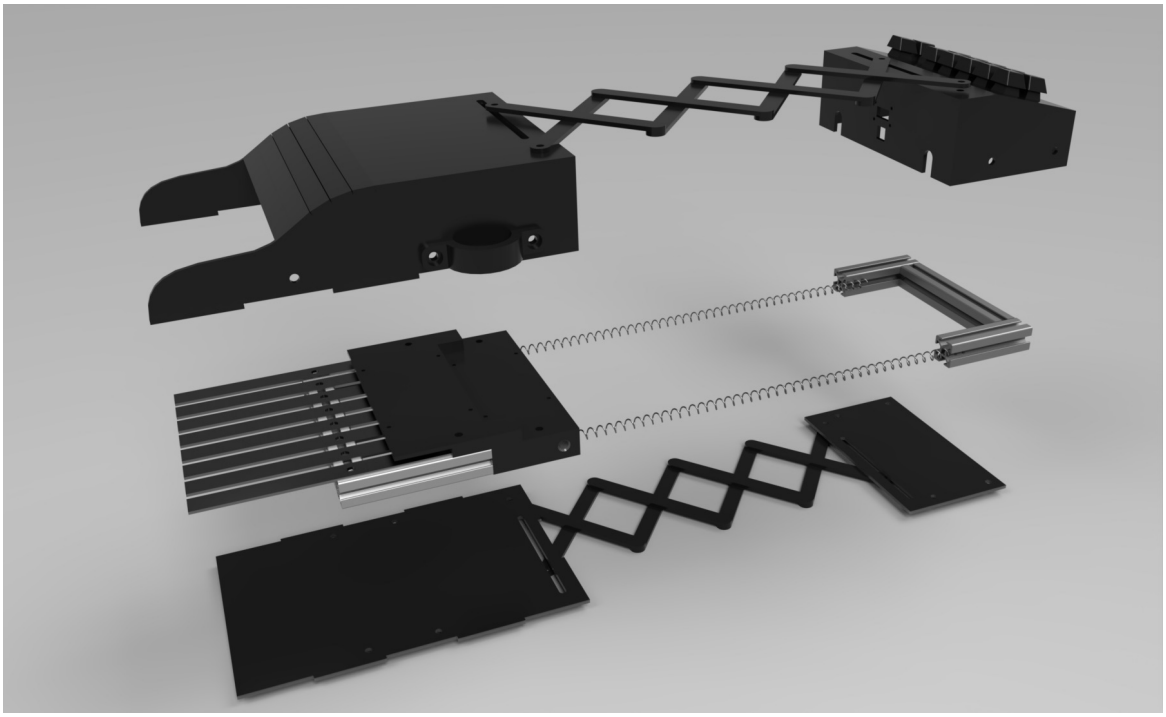

Figure 2. Exploded CAD render showing all revised Chordian components.

In the evaluation of our initial prototype, we identified the wiring as the largest problem. Long cables ran from the instrument to external Arduinos on a desk, creating a precarious situation where the player had to constantly concern themselves with wire positioning and connection status. We noted at the time that implementing wireless connectivity would completely eliminate these issues, though it would require placing the microcontrollers inside the instrument body.

We also wanted to revisit the bellows articulation system. The initial design used a SHARP IR analog sensor that required extensive filtering and had multi-centimeter accuracy at best. During that development, we had originally intended to use a Time of Flight sensor but encountered technical difficulties that forced the IR fallback. For this revision, we committed to making the TOF sensor work properly.

We retained the following parts from the initial prototype:

Part	Quantity
Arduino Mega 2560 Rev 3	1
Linear Soft Potentiometer	8
Accordion Straps	1
Leather Handle	1

We purchased the following additional parts for the revised design:

Part	Quantity
ESP32-DevKitC-V4	2

Part	Quantity
Force Sensitive Resistor	8
VL53L0X TOF Sensor	1
Mechanical Keyboard Switches	12
Rotary Encoders	2
9V Battery	1
Springs	2

Design Pivot: ESP32 ADC Limitation

Our original plan called for placing an ESP32 on each half of the instrument, enabling full wireless communication between the halves and eliminating all wired connections. However, during development we discovered that enabling WiFi on the ESP32 disables 8 of its 16 analog-to-digital converter pins. Since we needed 16 analog inputs for our soft pots and FSRs, this left us without enough ADC capacity.

Rather than wait for new parts to ship, we adapted by keeping one Arduino Mega 2560 from the initial prototype to handle all analog sensor reads. The Mega communicates with the onboard ESP32 over I2C via a ribbon cable between the two instrument halves. The ESP32 then transmits data wirelessly to a second ESP32 receiver connected to the computer. This hybrid approach preserved most of our wireless goals while working within the hardware constraints we discovered. We discuss alternatives for achieving full wireless in the latter Future Improvements section.

Construction

Figure 3. The revised Chordian prototype

The revised prototype represents a complete physical redesign from the original wooden construction. The new housing uses laser-cut acrylic panels supported by an internal aluminum frame, resulting in a cleaner aesthetic and more rigid structure.

Figure 4. Internal frame, springs, and scissor mechanism connecting both halves.

The two halves connect through a scissor mechanism that allows for smooth extension and retraction along a set track. In our initial prototype evaluation, we noted that the connection method between the halves could benefit from tuning to make pulling smoother. For this revision, we added springs between the halves to provide resistance during push and pull motions. The springs create a more tactile, instrument-like feel and help smooth direction changes while playing, addressing the disconnected feeling we experienced with the original unresisted sliding motion.

The key bed half houses the Arduino Mega and 8 metal keys. Each key has a soft potentiometer on top for finger position tracking and a force sensitive resistor underneath for pressure detection. The aluminum frame provides stable mounting for the keys and sensors. The button half contains the ESP32, TOF sensor, 12 mechanical keyboard switches arranged for chord triggering and octave selection, and 2 rotary encoders for scale navigation. We chose mechanical keyboard switches over simple digital buttons for their tactile feedback and retro aesthetic. Power comes from an onboard 9V battery, making the instrument fully self-contained during performance.

Circuitry

The circuitry follows a straightforward design with the primary complexity being the multi-device communication chain. The diagram above shows the high-level data flow from physical sensors through to audio output. The Arduino Mega reads all 16 analog sensors: 8 soft potentiometers for finger position and 8 force sensitive resistors for key pressure. Each analog sensor uses a 10k ohm pull-down resistor to filter signal noise and provide stable readings. Without these resistors, the sensors would be difficult to work with due to floating voltage issues.

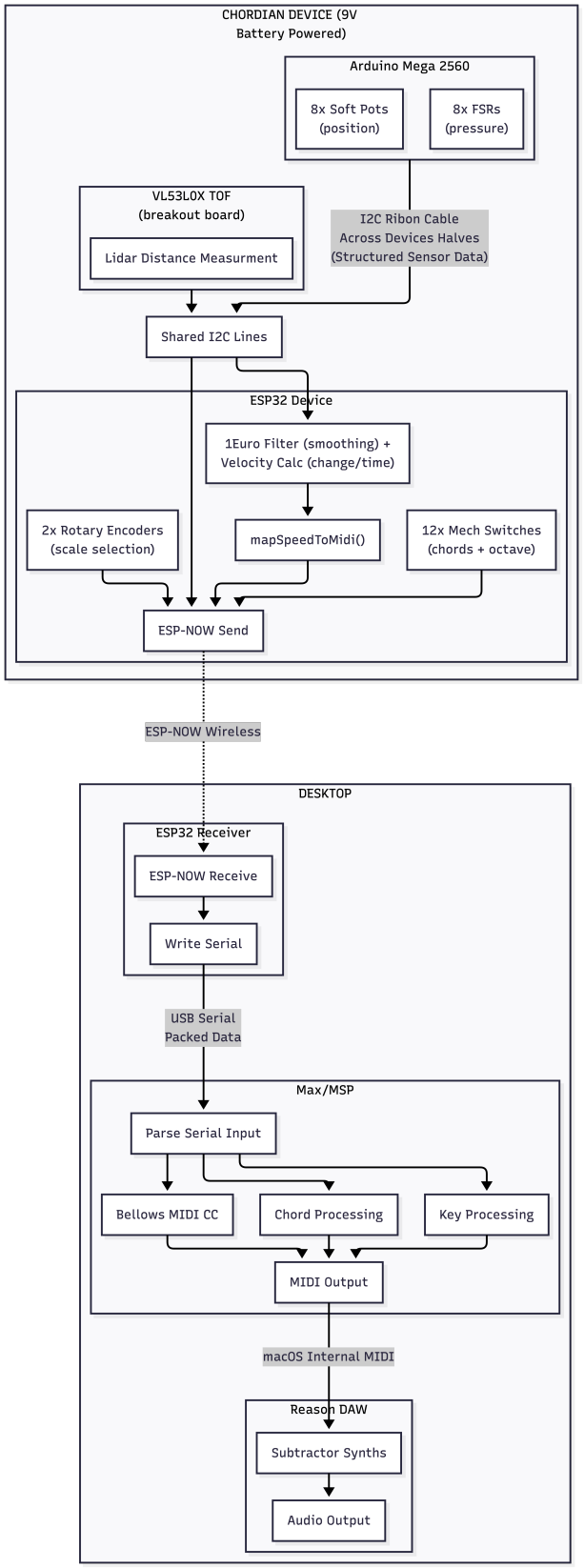

Figure 5. Full system block diagram showing data flow from sensors to audio output.

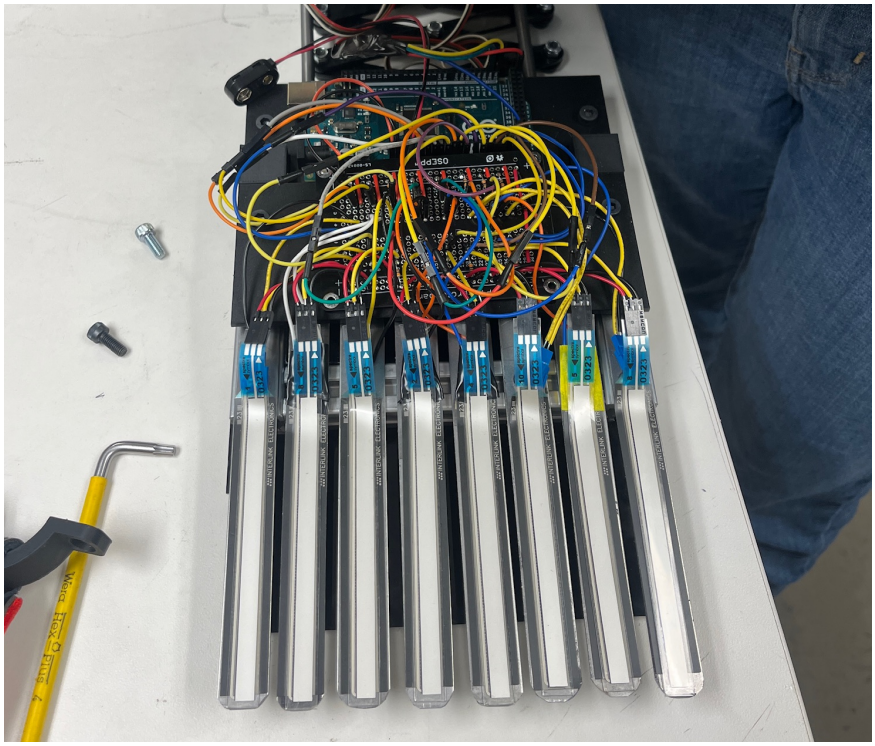

Figure 6. Internal view showing sensor wiring and connections.

The ESP32 handles the digital inputs including 12 mechanical switches and 2 rotary encoders, as well as the VL53L0X TOF sensor over the shared I2C bus. The ESP32 also performs all signal processing for the bellows articulation before transmitting the complete data packet wirelessly. A 4-wire ribbon cable connects the two halves, carrying I2C data lines (SDA/SCL) plus power and ground. This was a compromise from our original full-wireless design, but keeps the wiring minimal and internal to the instrument.

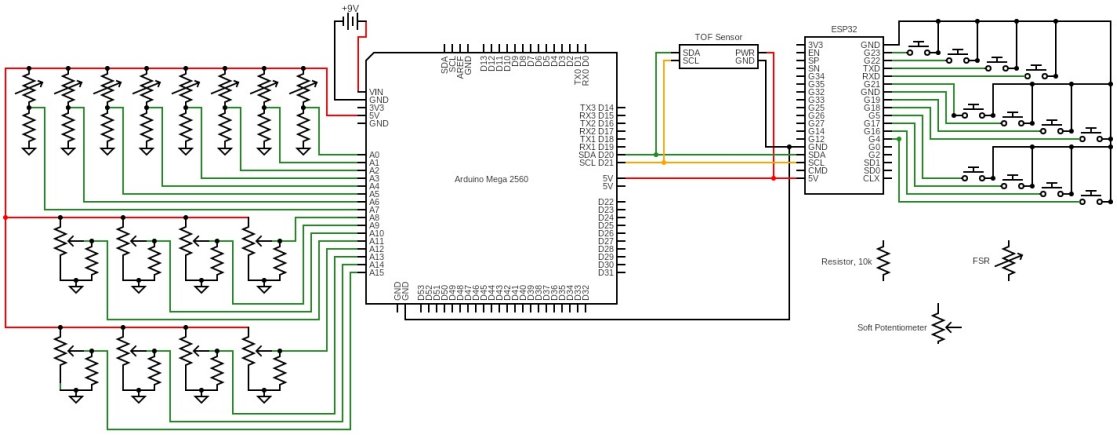

Figure 7. Onboard wiring diagram showing pull-down resistor configuration.

Embedded Hardware, Wireless Communication, and Firmware Processing

Microcontroller Architecture

The system uses three microcontrollers in a coordinated chain:

1. The Arduino Mega 2560 acts as a sensor peripheral, continuously reading all 16 analog inputs and packaging them into a structured data format. It operates as an I2C slave device, responding to data requests from the ESP32.
2. The ESP32 Device is the main controller aboard the instrument. It coordinates communication between the Mega and the wireless receiver, reads the TOF sensor and digital inputs, performs all signal filtering and MIDI mapping for the bellows, and transmits complete data packets via ESP-NOW.
3. The ESP32 Receiver sits on the desk connected via USB to the computer running Max. It receives wireless packets and writes them to serial for Max to parse.

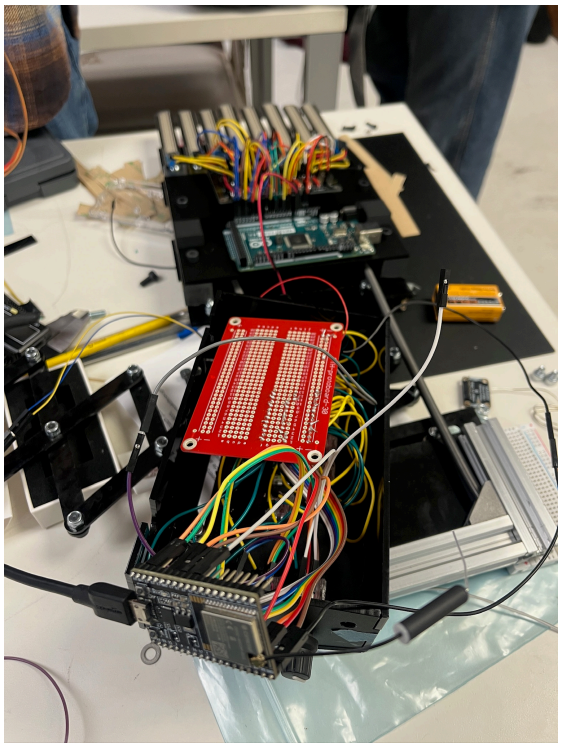

Figure 8. Development setup showing ESP32 with protoboard and Arduino Mega.

I2C Communication

The Arduino Mega and ESP32 communicate using the I2C protocol over the ribbon cable connecting the instrument halves. The Mega operates as a peripheral device at address 0x08, packaging sensor readings into a structured format:

```
struct analogData {
    uint16_t sensorForce[8];  // 8 Force Sensors
    uint16_t sensorPot[8];    // 8 Soft Pots
};
```

The ESP32 requests this data each loop cycle using the Wire library. When the request arrives, the Mega responds with the current sensor values. This approach keeps the analog reading isolated from the main processing loop, allowing each device to focus on its specific responsibilities.

ESP-NOW Wireless Communication

For wireless transmission between the instrument and receiver, we use ESP-NOW rather than traditional WiFi or Bluetooth. ESP-NOW is a connectionless protocol developed by Espressif (the designers of the ESP32 chipset) that allows for low-latency peer-to-peer communication between ESP32 devices.

We chose ESP-NOW for several reasons. It provides low latency (~1-5ms) compared with Bluetooth Low Energy (~5-20ms) or WiFi (~10-50ms), which is critical for real-time musical control. The pairing process is simple, requiring only the receiver's static MAC address.

The device ESP32 transmits a complete data packet each cycle containing all analog values, digital button states, and the processed bellows MIDI CC value. The receiver ESP32 simply unpacks this data and writes it to USB serial in a format Max can parse.

TOF Sensor and Signal Processing

The initial prototype used a SHARP IR analog sensor for bellows distance measurement. While functional, it required extensive filtering through a 50-sample median buffer and exponential moving average, and still only achieved multi-centimeter accuracy. In evaluating that design, we noted wanting to return to a Time of Flight sensor system, which would eliminate the need for the dual Arduino setup we had used and provide greater accuracy with less delay. For this revision, we switched to the VL53L0X Time of Flight sensor from Adafruit.

The VL53L0X uses laser-based distance measurement with 1-2mm accuracy and communicates digitally over I2C. This eliminates the analog noise issues we faced with the IR sensor and provides much faster, more accurate readings. The sensor updates at approximately 33Hz with a range of 30mm to 1000mm.

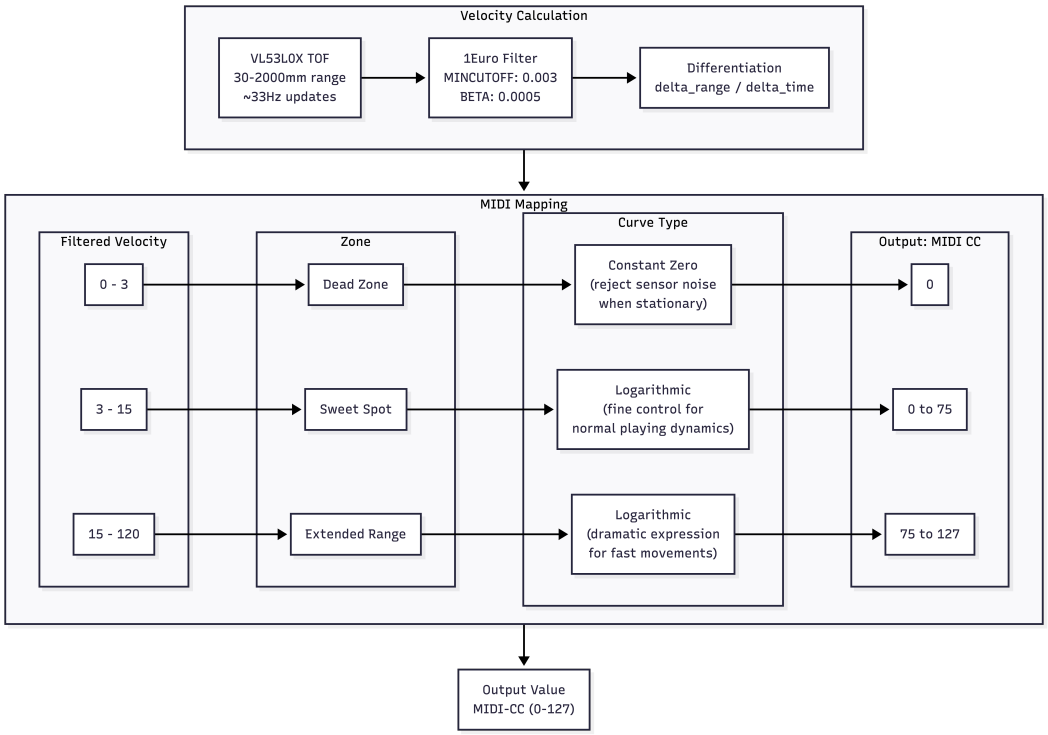

Figure 9. Signal processing pipeline for bellows articulation.

The raw distance values still require filtering before they can be used to calculate a stable velocity or speed value that can be used musically. We use the 1Euro filter, an adaptive low-pass filter library designed specifically for noisy signals requiring real-time response. The defining feature of the 1Euro filter is that it dynamically adjusts its cutoff frequency based on signal speed, meaning slow movements get heavy smoothing to reduce jitter, while fast movements get lighter smoothing to preserve responsiveness.

The filter is applied twice in our pipeline. First, we filter the raw distance values to get a smooth position signal. Then we differentiate to calculate velocity, and filter the velocity signal as well. The second pass uses different perameters for base cutoff and responsivness. The filter parameters were tuned through experimentation:

```
// Distance filter parameters
#define MINCUTOFF 0.003   // Minimum cutoff - lower = more smoothing
#define BETA 0.0005       // Speed coefficient - higher = faster response
// Velocity filter parameters
#define VMINCUTOFF 0.007
#define VBETA 0.003
```

MIDI CC Mapping

The filtered velocity value needs to be mapped to the 0-127 MIDI CC range in a way that feels musical and playable. In evaluating the initial prototype, we planned to investigate various functions for mapping raw velocity values to MIDI CC range, including quadratic, logarithmic, and exponential curves. A simple linear mapping does not work well because most playing happens at lower velocities, and the relationship between physical gesture and perceived loudness is not linear.

We designed a three-zone logarithmic mapping function called mapSpeedToMidi that addresses these issues:

The first zone is a dead zone from 0 to 3, which returns 0. This eliminates false triggers from sensor noise when the instrument is stationary, allowing it to rest silently.

The second zone is the sweet spot from 3 to 15, which maps logarithmically to MIDI values 0-75. This is where normal playing happens. The logarithmic curve provides fine control at lower velocities where most expression occurs, with diminishing sensitivity at higher speeds.

The third zone is the extended range from 15 to 120, mapping logarithmically from 75 to 127. This zone is harder to reach and rewards intentional dramatic gestures with the full MIDI range.

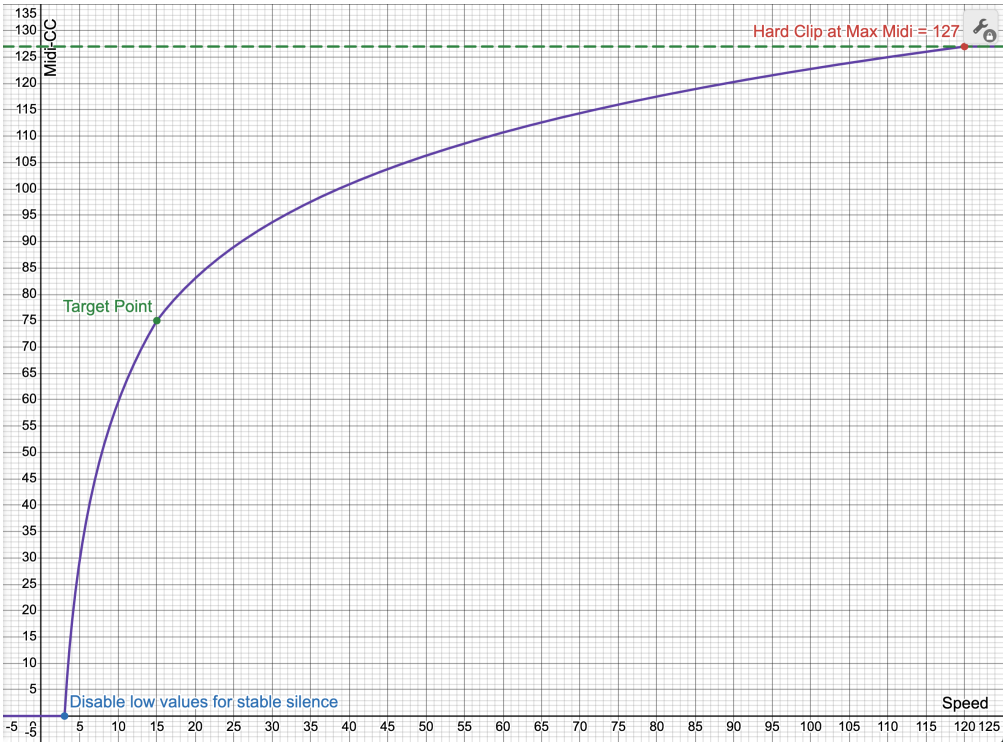

Figure 10. Graph

```
float mapSpeedToMidi(float speed) {
    const float DEAD_ZONE = 3.0;
    const float SWEET_SPOT_SPEED = 15.0;
    const float MAX_SPEED = 120.0;

    if (speed < DEAD_ZONE)
        return 0;

    if (speed < SWEET_SPOT_SPEED) {
        float t = (speed - DEAD_ZONE) / (SWEET_SPOT_SPEED - DEAD_ZONE);
        return log1p(t * 9.0) / log(10.0) * 75.0;
    } else {
        float t = (speed - SWEET_SPOT_SPEED) / (MAX_SPEED - SWEET_SPOT_SPEED);
        t = constrain(t, 0.0, 1.0);
```

```
        return 75.0 + log1p(t * 9.0) / log(10.0) * 52.0;
    }
}
```

This mapping approach matches comfortable bellow movement speed with human perception of dynamics much better than a linear scale.

Max Processing

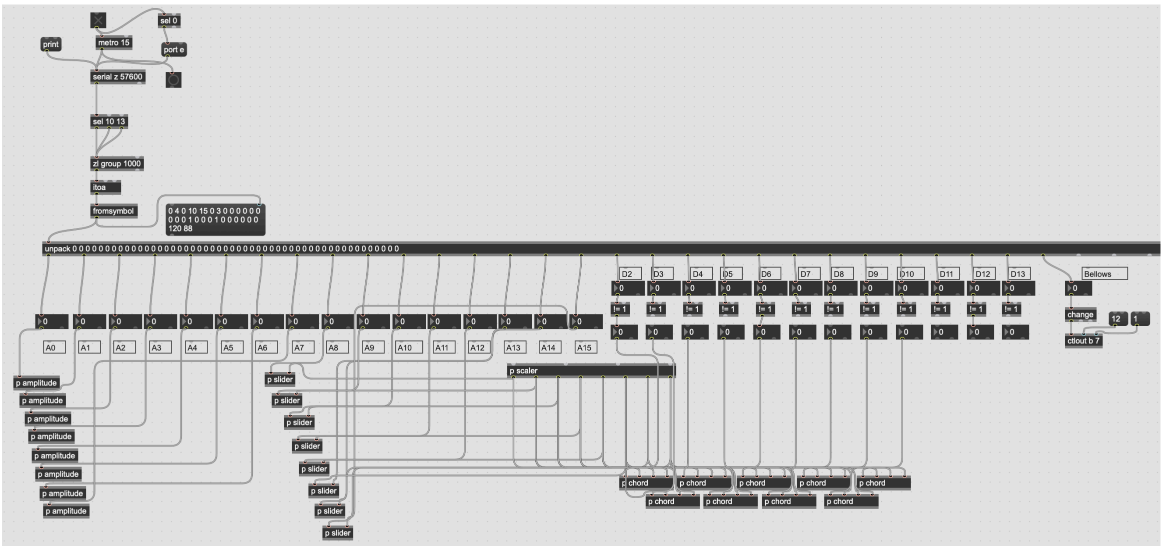

Figure 11. Full Max patch showing modular subpatch design.

The Max patch receives serial data from the ESP32 receiver and converts it to MIDI messages for Reason. In evaluating the initial prototype, we noted that the Max processing could be greatly improved by making the patch more modular. Rather than using eight copies of the key processing pipeline, we wanted to create subpatches that could be utilized at a per-key level.

The original patch duplicated the key processing logic eight times, once for each key. This made the patch difficult to maintain because any change had to be made in eight places. For the revised design, we created reusable subpatches that encapsulate the processing logic for each component type.

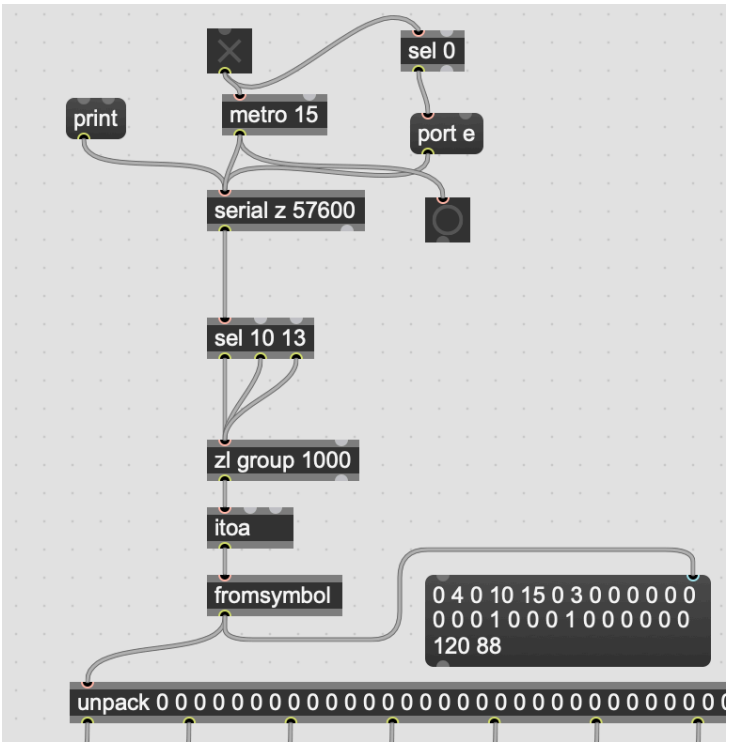

Figure 12. Serial data extraction subpatch.

The patch begins by parsing the incoming serial data from the ESP32 receiver. The data arrives as a formatted string containing all sensor values, which gets split and routed to the appropriate processing chains.

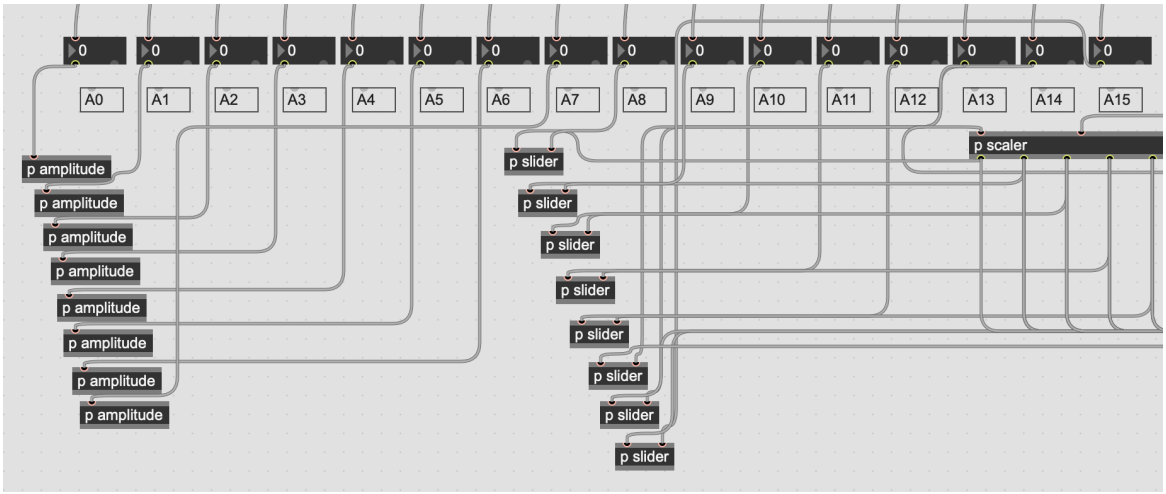

Figure 13. Key processing subpatch.

Each key uses a subpatch that handles the soft pot and FSR values for that key. The soft pot value controls a filter frequency CC, while the FSR determines note velocity and triggering. The same subpatch is instantiated eight times with different MIDI note assignments.

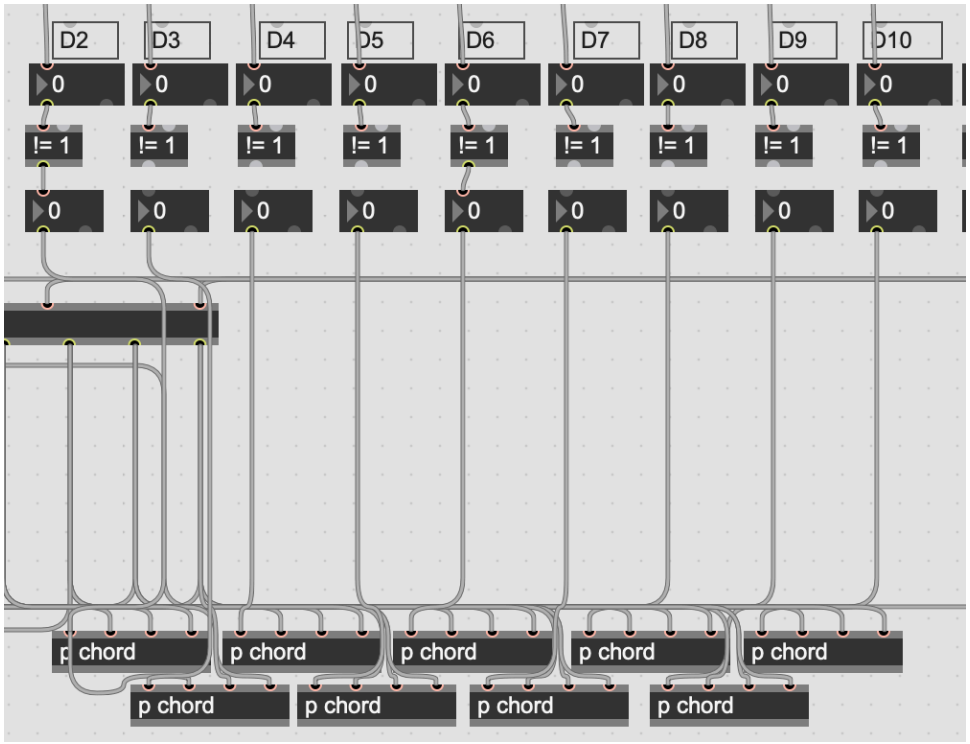

Figure 14. Chord processing subpatch.

The chord buttons use a similar approach, with predefined MIDI note sets for each button that create major chords. The octave buttons shift the base note up or down.

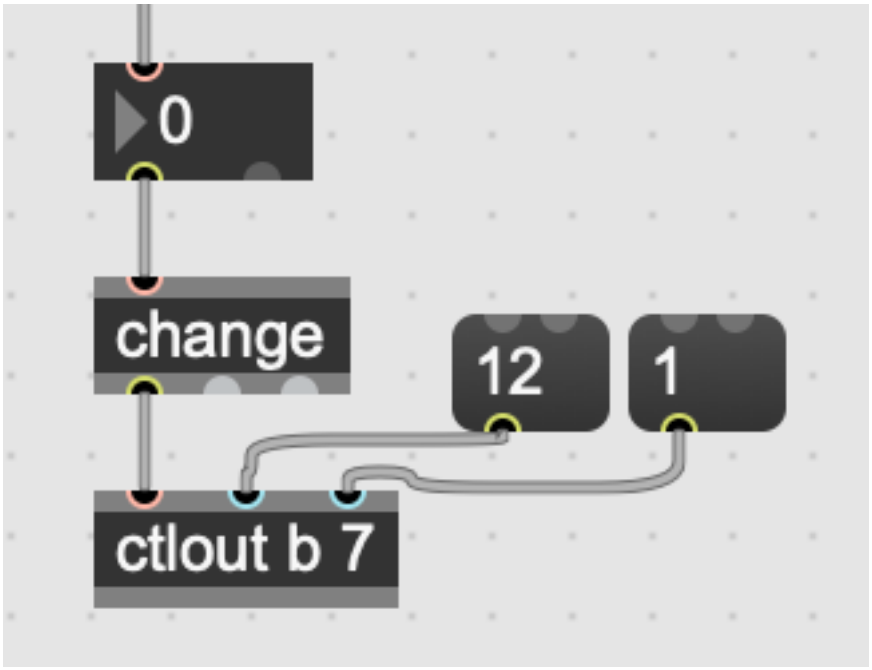

Figure 15. Bellows MIDI CC output.

The bellows MIDI CC value arrives already processed from the ESP32, so the Max patch simply routes it to the appropriate MIDI CC channel for Reason to interpret as expression control.

Audio Synthesis

The Reason rack follows a similar structure to the initial prototype, using Subtractor synthesizer modules for sound generation. The keys and chord buttons each route to their own Subtractor instance, allowing independent sound design for melodic and harmonic content.

For the revised design, we modified the synthesizer patches to use more accordion-inspired timbres. The line mixer controls overall amplitude based on the bellows MIDI CC, and a simple reverb provides spatial depth.

In our initial evaluation, we noted that sound design would play a larger role once we had the components and sensors working together. We had planned a more extensive sound design phase to tune the synthesizer parameters for optimal playability, but ran out of time before the demo due to issues getting the full assembled device working. The current patches are functional but would benefit from further refinement now that the hardware is operational.

Qualitative Analysis: Playing the Instrument

The Chordian has classic accordion style straps attached to the key bed half. These straps hold the weight of the instrument and keep the keys locked in place against the user's chest as the button half is pulled back and forth.

The spring mechanism greatly increased playability by adding a small amount of needed resistance and smoothing changes in direction while playing. This addresses a shortcoming we identified in the initial prototype, where the lack of resistance made bellows control feel disconnected from the sound.

The force sensitive resistors worked quite well, especially when combined with the reinforced aluminum key bed. We were able to get readable values that could be interpreted musically, in stark contrast to the bend sensors used in the initial design. Those bend sensors produced mostly noise due to how the keys flexed during actuation, and we noted in that evaluation that they were essentially unusable. The FSRs respond cleanly to downward pressure, providing the key velocity sensing we originally intended.

The soft pot finger position tracking feels fairly intuitive and adds nice expression to held notes. Sliding a finger along the key while holding a note is reminiscent of adding vibrato to a guitar string. The main limitation is the narrow width of the linear sensors, which means touches near the edges of keys do not register well. A future improvement would be to add silicone key covers that distribute fingertip pressure more evenly across the key surface.

The most musical aspect of the Chordian is the ability to modulate sound through multiple modes simultaneously. The bellows control volume and expression, key pressure affects velocity, and finger position modulates filter frequency. The ability to modulate sound without lifting a finger from the key felt intuitive and slightly reminiscent of the continuous expressive gestures possible on acoustic instruments.

Evaluation and Future Work

Successes

The revised prototype achieved several of its primary goals. Wireless communication now works between the instrument and the desktop receiver, eliminating the external wiring that was the largest problem with the initial design. The TOF sensor provides much better bellows accuracy than the IR sensor, and the logarithmic MIDI mapping creates a more playable response curve. The physical construction is cleaner and

more rigid, and the spring mechanism significantly improves the feel of bellows articulation. The modular Max patch design makes future modifications much easier to implement.

Challenges

We encountered several issues during development that required adaptation. The ESP32 ADC limitation discovery mid-project forced the hybrid Arduino Mega design, which works but adds complexity. At the demo, the TOF sensor failed to initialize due to faulty solder joints on the I2C connections, entering a boot loop. We worked around this by hardcoding the bellows value, then later resolved it through better management of the shared I2C lines wiring. Some mechanical keyboard buttons also did not respond in the assembled device, which we traced to digital pin mapping issues and fixed post-demo by changing the pins used for digital reads and correcting the pin assignments in firmware.

Potential Future Improvements

Full Wireless Between Halves Full wireless communication between instrument halves could be achieved by adding an analog multiplexer to handle all 16 analog reads through a single ADC pin, working around the ESP32 WiFi limitation. This would allow us to remove the Arduino Mega and ribbon cable entirely, replacing the Mega with a second ESP32 in the other half. The two ESP32 devices would communicate via ESP-NOW, and the reads are near instantaneous so this modification would not significantly impact performance while fully realizing the original wireless design.

BLE MIDI Another potential improvement of interest would be implementing full onboard MIDI handling using the Arduino BLE MIDI library. This would require an overhaul of the ESP32 firmware to process all sensor data and generate MIDI messages directly on the device, rather than sending raw sensor values to Max for conversion. The instrument would appear as a standard Bluetooth MIDI device in macOS Audio MIDI Setup, allowing direct connection to any DAW without intermediate software. This approach would eliminate the need for both the receiver ESP32 and the Max patch, significantly simplifying the system architecture and making the Chordian a fully standalone wireless MIDI controller.

Power and Battery A lithium battery with charging circuitry would replace the disposable 9V battery, making the instrument more practical for regular use. An external power switch would also help conserve battery when not in use.

Key Covers Silicone key covers would improve playability by distributing finger pressure across the full key width, addressing the edge detection issues with the narrow soft pots. The covers would also improve the visual appearance by hiding the sensor circuitry and provide a better tactile feel.

LCD Display An LCD display could show the current scale, octave, and connection status, making the instrument easier to use without a computer screen visible.

Closing Thoughts

Despite the challenges encountered, the revised Chordian represents a significant step forward from the initial prototype. The core expressive capabilities now work well enough to explore the musical potential of the design, and the remaining improvements are mostly refinements rather than fundamental changes.

The experience of iterating on the design, adapting to hardware limitations, and refining the user experience of a musical device was quite rewarding.