Self Playing Harmonica

ECE 445 Final Report - Spring 2026

Group 66

Sean Jasin (sjasi3), David Zhang (dzhan6), Robert Zhu (robertz4)
TA: Wenjing Song

Table of Contents

1. Introduction

1.1 Problem

The harmonica is a versatile, simple, yet technically difficult instrument to play. There is a need for the background music of a live instrument, yet it is difficult to master the harmonica. Some lack the time to practice and learn the harmonica. For others, they may no longer be able to physically play the harmonica, or do not have access to training or musical education. Existing musical devices exist for keyboard and string instruments, but not for wind instruments. There is a need for a self-playing harmonica that can produce melodies without requiring manual lip or breath control.

1.2 Solution

We propose a device that can play the harmonica. The self-playing harmonica consists of multiple subsystems. The power supply provides power at all required voltages for the MCU, air pumps, and electronic pneumatic valves. The harmonica-computer interface connects to both the harmonica and the MCU, and is responsible for controlling the airflow through the harmonica. It consists of pneumatic tubes, air pumps, and electronic valves. The MCU is responsible for controlling the pumps and valves in the harmonica-computer interface, as well as taking a MIDI file and converting it into a sequence of pump and valve motions. Lastly, songs are uploaded to the MCU through WiFi. We will create a website where the user can upload a MIDI file, and that file will then be available to play on the device.

1.3 Visual Aid

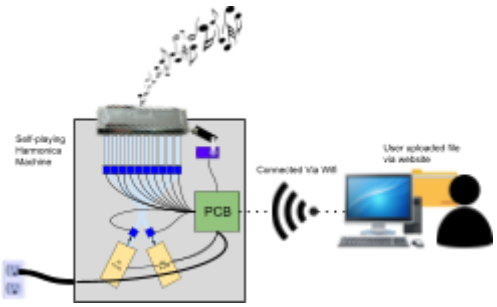

Figure 1. Visual aid for self-playing harmonica. Shows user input, harmonica control, and musical output.

1.4 High-Level requirements list

1. The device must be able to operate all basic functionality of a chromatic harmonica.
   - Playing all notes (blow in and suck out air for all holes)
   - Engage and disengage the slide.
   - The notes must be audible at 10m from a listener
2. The device must be able to receive song data wirelessly.

- User interface latency must not exceed 1s

3. The device must be able to operate the harmonica continuously for a minimum of 5 minutes

2. Design

2.1 Block Diagram

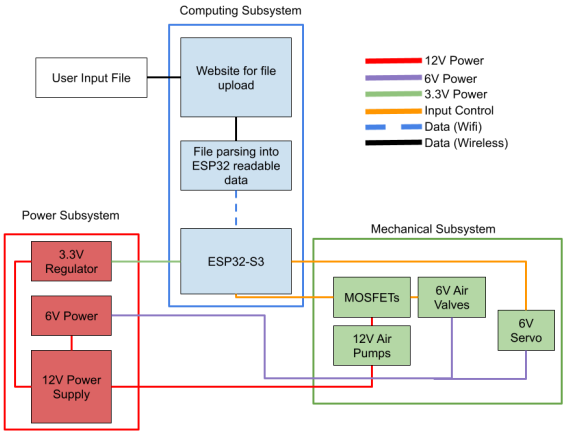

Figure 2. Block diagram of the device, showing all signal paths with color-coded signal type.

2.2 Physical Design

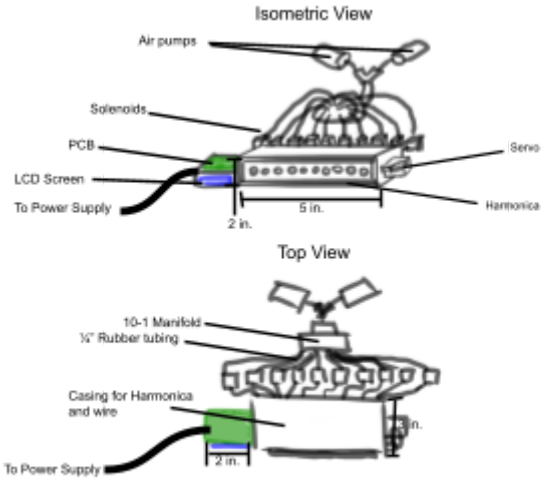

Figure 3. Physical design of Self-Playing Harmonica

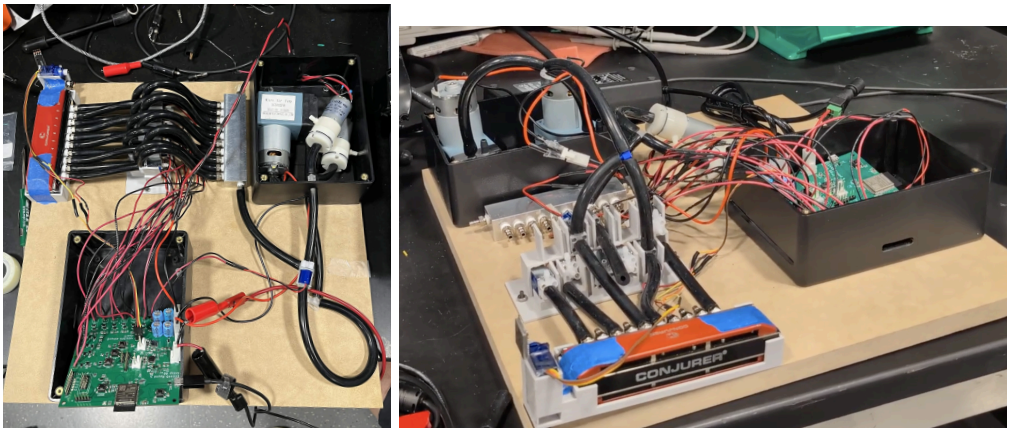

Figure 4. Original Project Configuration (left) and Final Configuration (right)

The self-playing harmonica, as well as its circuitry and the servo, will be contained within a plastic casing. The LCD screen will also fit onto the cover of the case. Outside of the case, the solenoid valves, air manifold, and the pumps will be connected to the harmonica via ¼" rubber tubing, and controlled via the PCB.

For our final design, we were unable to generate enough airflow or static pressure with 12V motors only. Because of this, we had to use two 4.5V motors in a parallel configuration to push the air, and we ended up using the 12V motors in a parallel configuration to draw the air from the harmonica. We also could not use the 2-10 port manifold either. This was due to the large volume of the inner portion of the manifold, which resulted in significant mechanical latency of the air traveling from motors to the harmonica. The manifold also resulted in leakage through all the connecting joints, as well as improperly sealed solenoids, so the manifold was ultimately replaced with a few t-joints as shown in the second picture above. In the final product, we also could only play one note at a time, and were only able to connect to three holes at a time. We could also not connect both the blowing motors and the drawing motors to the apparatus, as that would result in more than one solenoid in series which will cause problems as discussed later.

2.3 Subsystem Overview:

2.3.1 Power Subsystem

The power subsystem must be capable of supplying 3.3V, 6V, and 12V power to the device. The 3.3V power supply is for the MCU, and the 6V power supply is used by the pneumatic valves, the servo, and the 4.5V DC motors, and the 12V supplies the 12V motors. We utilized a 12V 120W power supply that plugs into a wall outlet. We then converted the 12V power supply into the 6V and 3.3V power rails using a buck converter on the PCB.

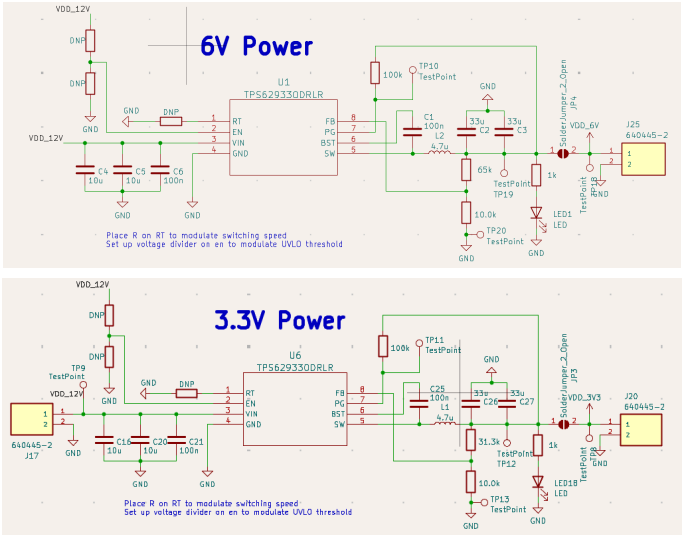

Figure 4: Power Subsystem

The TPS62933ODRLR buck converters use a reference voltage (0.8V) in order to create a negative feedback loop using a voltage divider that will result in the correct output voltage. The equation for the resistors required for the voltage divider is below in equation 1:

$$R_{FBT} = \frac{V_{OUT} - V_{REF}}{V_{REF}} * R_{FBB} \qquad (1)$$

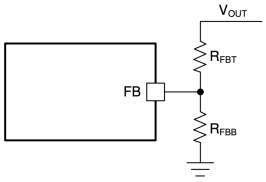

Figure 5: Buck Converter V_{OUT} Circuit Diagram

Where R_{FBT} is the top resistor of the voltage divider, V_{FBB} is the bottom resistor, V_{REF} = 0.8V, and V_{OUT} is the desired output voltage. Using these equations, we calculated our resistances to be 65 kΩ and 10 kΩ for 6V and 31.3 kΩ and 10 kΩ for 3.3V.

2.3.2 Computing Subsystem

The computing subsystem will use an ESP32-S3 WROOM-1 for its WiFi capabilities. The MCU has 2 functions: mechanical control and MIDI upload. The mechanical control will take MIDI inputs and will output the signal to control the motors (solenoids and servos) on the mechanical subsystem.

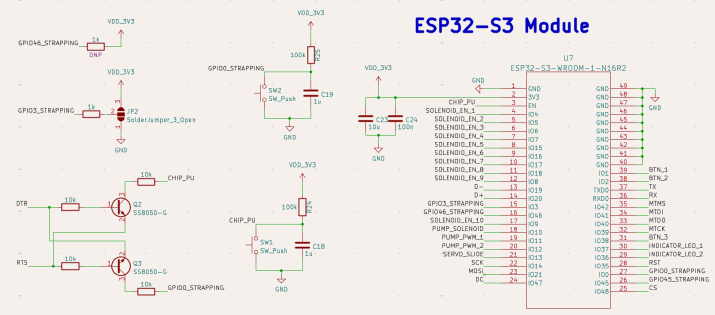

Figure 6: ESP32-S3 Microcontroller Module

On top of using an ESP32-S3 microcontroller, the computing system also contains capability for manual control. This consists of an LCD screen and buttons. These inputs are managed by the microcontroller, which will control output to the LCD screen. Our computing unit will also include a debug header in which we can connect to via PC to debug our software.

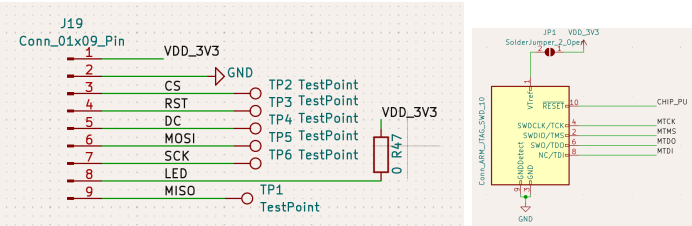

Figure 7: LCD Screen Connection Point and Debug Header

The computing subsystem also consists of an HTML controller as well as embedded software on the microcontroller itself.

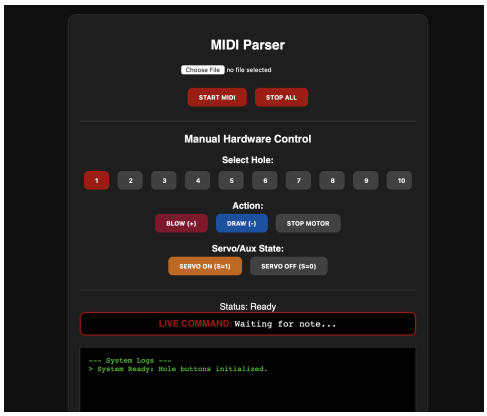

Figure 8: HTML Controller User Interface

The HTML controller consists of an HTML document to create the user interface, a .MIDI file parser, as well as a function to send data to the ESP32 using an HTTP fetch request. The HTML user interface can be seen above. The .MIDI file parser uses the Tone.js library in order to obtain the note lengths, pitches, and volume from an uploaded .MIDI file. It parses through each track, and obtains note information from all tracks in a .MIDI file. A .MIDI file can contain multiple tracks – for example, an orchestral piece may have separate tracks for a violin, a piano, a

trombone, etc. The .MIDI parser parses each note into three different values that the ESP32 can read:

- h: an integer from 1-10: it selects the hole that is opened so that air can be blown or drawn
- a: an integer from -100 to 100: this activates the motors to either blow air or draw air, and the magnitude of the number determines the PWM signal the motor should receive
- s: a boolean value: 0 is servo out/slide unactuated and 1 is servo in/slide actuated.

These values are controlled by a dictionary in which each note's hole, air direction, and slide are hard-coded into the parser's code as shown below:

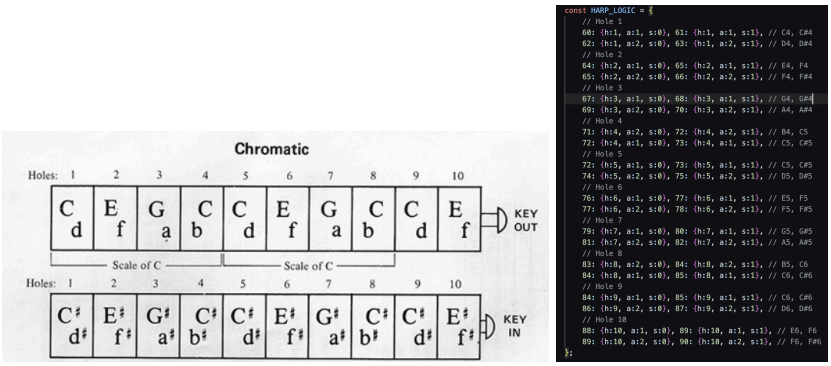

Figure 9: Chromatic Harmonica Note Chart and Parser Dictionary

Once these values are parsed, they are sent to the ESP32 via an HTTP fetch request. The user is required to connect to the ESP32's built in Wi-fi network, and the parser will send fetch requests with a modified url in real time to the ESP32, which removes almost all the calculation logic from the PCB itself, and instead uses the user's device side to create all the calculations.

After the HTML controller is the embedded system. The embedded system consists of 3 parts, WiFi connection broadcasting, HTTP request handling, and software to hardware translation. The WiFi connection broadcasting portion utilizes the esp_wifi.h library in order to set the configuration of the WiFi network (name and password). The library also helps broadcast the actual WiFi network using the configuration we make.

```
wifi_config_t wifi_config = {
    .ap = {
        .ssid = WIFI_SSID,
        .ssid_len = strlen(WIFI_SSID),
        .password = WIFI_PASS,
        .max_connection = 4,
        .authmode = WIFI_AUTH_WPA2_PSK
    },
};
```

Figure 10: WiFi Configuration Setup

```
ESP_ERROR_CHECK(esp_wifi_set_mode(WIFI_MODE_AP));
ESP_ERROR_CHECK(esp_wifi_set_config(WIFI_IF_AP, &wifi_config));
ESP_ERROR_CHECK(esp_wifi_start());
ESP_LOGI(TAG, "WiFi AP Started. SSID:%s", WIFI_SSID);
```

Figure 11: WiFi Broadcasting Initialization

The WiFi configuration sets important values which determine how many users can connect to the WiFi network, what the name of the network is, and what the password of the network is.

The WiFi broadcasting initialization sets the WiFi mode to broadcast a WiFi network instead of connecting to an already existing network and then starting the internal WiFi module with esp_wifi_start().
The HTTP request handling is the portion that takes an incoming request to the ESP32's IP address and a specific URL, in the case of this project http://192.168.4.1/play when connected to the ESP32's WiFi network mentioned prior. We use the esp_http_server.h library to set up the server configuration. The HTTP handler uses a similar structure to the WiFi network setup to create a webpage to respond to HTTP requests.

```
httpd_uri_t play_uri = {
    .uri      = "/play",
    .method   = HTTP_GET,
    .handler  = play_api_handler,
    .user_ctx = NULL
};
httpd_register_uri_handler(server, &play_uri);
```

Figure 12: HTTP Request Setup

The configuration consists of uri, method, handler, and user_ctx. The uri config sets the link which the ESP32 will respond to which is "/play", so the full length URL would be "http://192.168.4.1/play." The method config sets which request type the ESP32 will respond to. In this case the request type we used was GET requests which allow for URL manipulation to handle data. To elaborate further, the link "http://192.168.4.1/play?h=0&a=0&s=0" would result in the ESP32 receiving 3 values, an "h" value, an "a" value, and an "s" value. These values will later determine what should be done mechanically in the software to hardware translation portion of the embedded system.

The last part of the embedded system is the software to hardware translation portion which translates the received values "h", "a", and "s" into mechanical movement. The "h", "a", and "s" values control hole, motor action, and servo respectively. They all work similarly, the value is read and the corresponding GPIO is set. For example, the "h" snippet of code is shown below.

```
if (httpd_query_key_value(buf, "h", buf2, sizeof(buf2)) == ESP_OK &&
    httpd_query_key_value(buf, "a", buf3, sizeof(buf3)) == ESP_OK) {
  int whole_note = atoi(buf2);
  int active = atoi(buf3);
  ESP_LOGI(TAG, "%d is set: %d", whole_note, active);
  if (whole_note > 0 && whole_note <= SCNT) {
    solenoid_state[whole_note - 1] = active;
  }
  for (int i = 0; i < SCNT; i++) {
    gpio_set_level(solenoid_pins[i], solenoid_state[i]);
    ESP_LOGI(TAG, "%d is pin: %d and is set: %d", i, solenoid_pins[i],
             solenoid_state[i]);
  }

  char resp[64];
  snprintf(resp, sizeof(resp), "{\"status\":\"ok\",\"pin_state\":%d}",
           aux_state);
  httpd_resp_set_type(req, "application/json");
  httpd_resp_send(req, resp, HTTPD_RESP_USE_STRLEN);
}
```

Figure 13: "h" Snippet for Software to Hardware Translation

The httpd_query_key_value function reads a request which contains "h", "a", and "s" values and stores it "h" into buf2 and "a" into buf3. These values are then stored into a more readable variable called whole_note and active respectively. The part is a for loop which checks if the note is within the correct range and sets it to the same as active (any non-zero value is treated as 1). The next for loop iterates through each solenoid state and sets their corresponding GPIO to HI or LO as set in the solenoid_state array.

While each control generally works the same, we will be including code snippets for clarity.

```c
if (httpd_req_get_url_query_str(req, buf, sizeof(buf)) == ESP_OK) {
    char ang_str[10];
    if (httpd_query_key_value(buf, "s", ang_str, sizeof(ang_str)) == ESP_OK) {
        int angle = atoi(ang_str);
        if(angle > 0) {
          set_servo_angle(70);
          ESP_LOGI(TAG, "Servo Angle set to: %d", 70);
        } else {
          set_servo_angle(90);
          ESP_LOGI(TAG, "Servo Angle set to: %d", 90);
        }
        httpd_resp_send(req, "{\"status\":\"ok\",\"type\":\"servo\"}", HTTPD_RESP_USE_STRLEN);
    }
}
```

```c
if (httpd_query_key_value(buf, "a", buf2, sizeof(buf2)) == ESP_OK) {
    int speed = atoi(buf2);

    // Clamp speed between -100 and 100
    if (speed > 100) speed = 100;
    if (speed < -100) speed = -100;

    uint32_t duty = (abs(speed) * PWM_PERIOD) / 100;

    if (speed > 0) {
        // Forward
        solenoid_state[SOLF] = 1;
        solenoid_state[SOLR] = 0;
        mcpwm_comparator_set_compare_value(comp_a, duty);
        mcpwm_comparator_set_compare_value(comp_b, 0);
    } else if (speed < 0) {
        // Reverse
        solenoid_state[SOLR] = 1;
        solenoid_state[SOLF] = 0;
        mcpwm_comparator_set_compare_value(comp_a, 0);
        mcpwm_comparator_set_compare_value(comp_b, duty);
    } else {
        // Stop
        solenoid_state[SOLF] = 0;
        solenoid_state[SOLR] = 0;
        mcpwm_comparator_set_compare_value(comp_a, 0);
        mcpwm_comparator_set_compare_value(comp_b, 0);
    }
```

Figure 14: "a" Motor Value and "s" Servo Value Snippets

2.3.3 Mechanical Subsystem

The mechanical subsystem consists of all mechanical components of the device. This includes air pumps, solenoids, and each of their respective drivers. The drivers will allow the MCU to control the motors that dictate the airflow and the note being played.

The mechanical subsystem consists of eleven solenoids, four air pumps, and a servo motor. The roles of each component are as listed below:

- Solenoids 1-10: These open allow passages to the holes of the harmonica
- Solenoid 11: This solenoid selects between blowing and drawing motors.
- 2 4.5V DC Air pumps: These air pumps are connected mechanically in parallel to provide blowing air
- 2 12V DC Air pumps: These air pumps are connected mechanically in parallel to draw air.

The drivers are shown below:

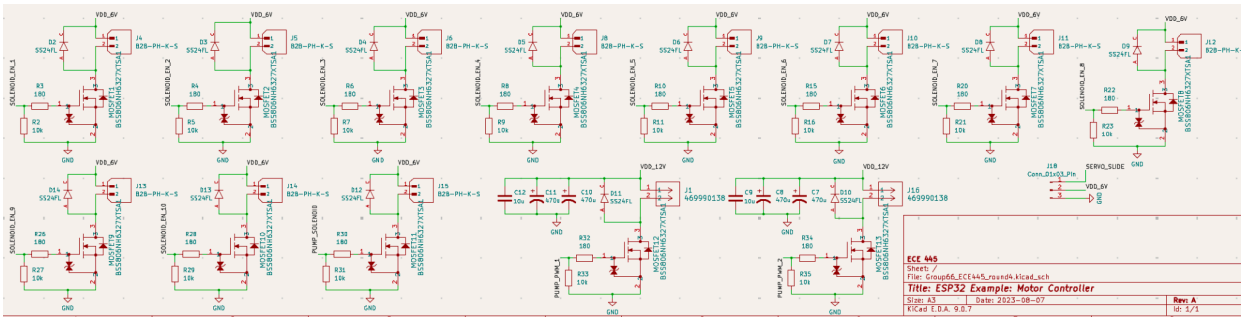

Figure 15: Mechanical Subsystem Drivers

The drivers for all the solenoids consist of a low-side MOSFET and a power-recycling diode. These are all driven by 3.3V signals by the ESP32 and control 6V reaching the solenoids. The drivers for the motors (one for blowing and one for drawing) are similar to the drivers for the

solenoids, with two main differences – capacitance was added in order to increase 12V rail stability, and it connects to 12V and not 6V. The servo is powered by the 6V rail, and controlled via a PWM signal from the ESP32.

The mechanical design went through various stages. Originally, the ten hole solenoids and the two sets of motors were connected to a two-to-ten hole air manifold. However, this resulted in too much leakage, as each solenoid had some leakage in the rubber hosing connection. It also caused a lot of mechanical latency, as the air had to travel through a greater volume to reach the harmonica. We solved this issue by replacing the manifold with T-joints. On top of this, the solenoid had too much resistance due to the small bore of the inside of the solenoid head. Therefore, we could only route air through one solenoid in series, not two solenoids in series. This resulted in the final design we used in the demo requiring us to manually change whether the motors that were connected to the harmonica were blowing or drawing. A labeled diagram of what we changed is shown here below:

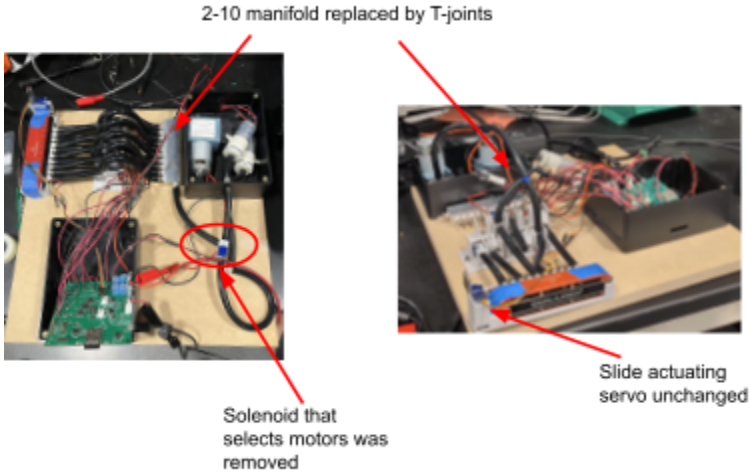

Figure 16: Mechanical Changes Made Throughout the Design Process

2.4 Design Alternatives

2.4.1 Board revision 1

Board revision 1 was the starting board. It has a 5V input support, a 3V3 bucking regulator, utilized 4V5 motors (rated for up to 6V prolonged), 5-6V solenoids, and a 4-6V servo. This design had issues with slow solenoid turn on time and the motors not pushing enough air to the harmonica. By testing different voltages with the solenoid and motor, we discovered that using 6V resulted in faster solenoid turn on times and slightly more airflow. These problems were then addressed in board revision 2 by using a 6V input rail instead of 5V.

2.4.2 Board revision 2

Board revision 2 addressed slow solenoid turn on time and weak airflow by increasing the input supply voltage from 5V to 6V. This resulted in much faster solenoid turn on times. However, the motors seemed to still not push enough air for the harmonica to be audible consistently. The hardware components, namely, the motors, solenoids, and servo did not change during this revision. To address the weak airflow, we redesigned the next revision to have 12V input instead of 6V, updated the motor to 12V motors with better on paper specifications, and added a new 6V buck for the solenoids and the servo.

2.4.3 Board revision 3

Board revision 3 addressed the weak airflow again with 12V motors and a new 12V input supply. We added a 6V bucking regulator to support the same solenoid and servo used in the previous designs which worked reliably. In addition to new hardware, the MOSFETs for each driver was updated to be one type of MOSFET instead of two. This made house keeping for materials and sourcing easier. This revision also had an overhaul of the GPIO pinouts to make the layout/routing easier. This makes the specific code pinout for board revision 1 and 2 to be incompatible with the code pinout of board revision 3. During this revision we discovered that our pinout for the servo was incorrect, the MOSFET footprints were too small, and that the board would brownout when the parallel motors were turned on. The first 2 problems were addressed by fixing their layout and specifications in the schematic. The last problem had to be tested separately. During testing, we discovered that the motors would sometimes draw upwards of 3A which our input power could not handle. To address this issue, we added capacitors to the 12V rail which act as a buffer to provide more current.

2.4.4 Board revision 4

Board revision 4 addressed the brownout issue by adding capacitors at the 12V input. This solved our brownout problem as intended. The issue regarding servo pinout and MOSFET footprint were also addressed in this board revision and were fixed accordingly in the schematic. This board revision also added more physical buttons which can be used for user input. This board revision fully worked as intended electrically and in software.

2.5 Design Description & Justification

2.5.1 Modular design

We used a modular design for our power stage which separates the 6V output from the peripherals and the 3.3V buck. We also separated the 3.3V buck from its peripherals as well. We chose to build these portions as modules because of the concern and sensitivity of the other components. The module would only be connected once thorough testing confirmed that the unit

was fully functional. This way we could avoid destroying components which could bear unnecessary cost and lost time. The method for testing each module was to inject power and check the output. If the output was within acceptable range (+/- 500mV) we would connect the module to the rest of its peripherals.

2.5.2 Justification for Electrical Design

2.5.2.1 Buck converters

For the buck converter, we used the TPS62933ODRLR which has adjustable output voltage and supports up to 3A continuous current. We chose this buck converter because of how many peripherals are attached to our board, 11 solenoids, 1 servo, and 4 motors (initially 2). We wanted enough room to make adjustments in the future which ended up being a good idea as we would increase the number of motors in our design, and therefore the current draw. This headroom is what allowed us to make these choices without changing the power system design.

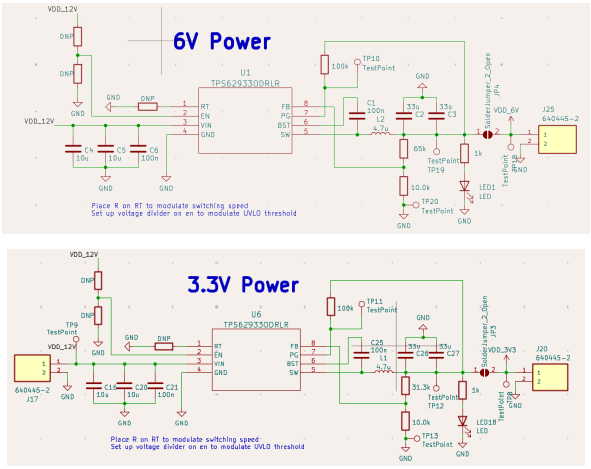

Figure 17: Schematic of Buck Converter Circuits

2.5.2.2 Microcontroller

Due to the nature of our requirements, we needed a microcontroller which had wireless capabilities, plenty of GPIO to work with, and a fast CPU for quick processing. The ESP32 satisfies each requirement with WiFi and bluetooth capabilities, 45 GPIO pins, and a 240MHz processor.

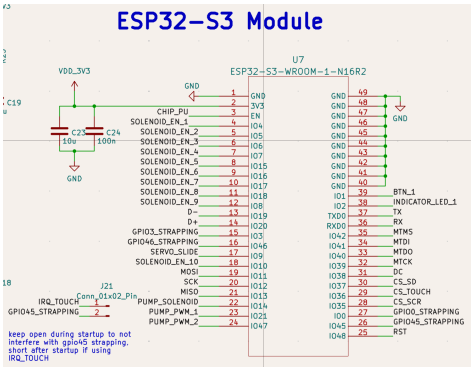

Figure 18: Microcontroller Circuit Schematic

2.5.2.3 Motor drivers

Our main requirements when choosing parts for the motor driver was that it needs to support 2A in-rush and 550mA sustained. We settled on using the SSM3K339R_LF MOSFET, a 40V 2A capable MOSFET. If pulsed, the MOSFET supports up to 4A of current. These values satisfy our requirements with extra headroom for future design changes.

2.5.2.4 Solenoid drivers

Our requirements for the solenoid drivers was that it needs to support 1A max and 300mA typical. Given the specs of the motor drivers, we settled on using the same driver.

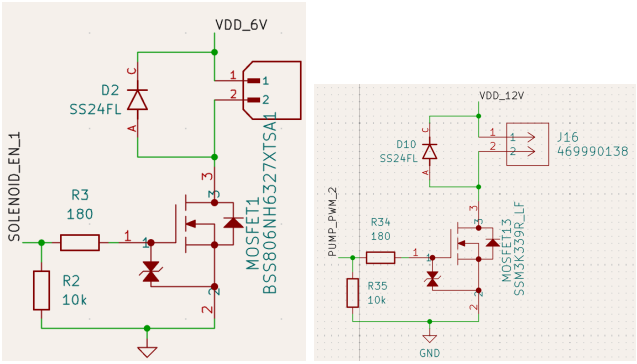

Figure 20: Motor and Solenoid Driver Schematic

2.5.3 Justification for Mechanical Design

2.5.3.1 Servo selection

For the servo selection, we had a servo on hand which we used during breadboard testing. We were able to get the servo to properly engage the harmonica slide and decided to settle on this particular servo, the 1080SF from Amazon. It is capable of 1.9kg/cm and a speed of 0.08±0.01 sec/60° at 6V. We did not have future issues regarding this component choice.

2.5.3.2 Motor selection

For the motor selection, we chose 2 different motors, a 4.5V rated motor and a 12V rated motor. We initially chose the 4.5V rated motor as a way to create an initial prototype to see what would work and what wouldn't. During the breadboard design, we were able to get sound from the 4.5V rated motor. However during the PCB design with proper sealing it no longer produced sound. The difference was that with the seal the motor is not drawing surrounding air to increase airflow. This is known as the Bernoulli Effect, more specifically entrainment. In order to address this, we used a higher airflow and higher static pressure motor, the Micro Air Pump, 12 Volt DC from skycraftsurplus. The skycraftsurplus motor is able to provide 10-12LPM at a maximum outlet pressure of 29PSI. Further down the timeline, we realized this was only one of the issues preventing the harmonica from being audible.

2.5.3.3 Solenoid selection

For the solenoid selection, we chose the DFR0866. We chose this as the solenoid because it fit our tube lining, had a normally open and normally closed direction, was cheaper for our budget, and worked during initial breadboard testing. However, this choice of solenoid would later prove to be a heavy constraint in the airflow to the harmonica dropping the airflow to nearly half of what would normally be if there was no solenoid. See the figure below.

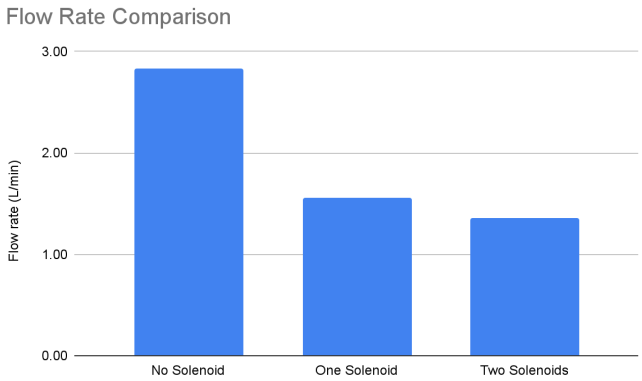

Figure 21: Flow Rate with and without Solenoids

We tried mitigating the reduction in flow rate by increasing the bore size on the head of the solenoid with some audible successes. However, the wider bore heads were inconsistent as they were 3D printed and had inconsistencies from one solenoid head to another. In addition, they increased air leakage, which ended up causing the same problem as before, where not enough air would get to the proper harmonica hole.

2.5.3.4 T-splitters

We chose to use T-splitters which we 3D printed. We opted for this route instead of the initial design 10 port manifold because the 10 port manifold's inner volume was too large, drastically

increasing propagation delay and preventing proper air flow when switching from pushing to pulling air from the harmonica. The T-splitters resulted in a much more audible sound from the harmonica in comparison to the 10 port manifold.

2.6 Cost (Bill of Materials)

Total cost of parts: $146.69 (See Appendix B 6.2).
In addition to parts, there will also be a labor cost as well. At $35/hr for 8 hours a week per employee for 3 employees over 12 weeks, that amounts to a total of 35*8*3*12 = **$10,080.**
In total, the project will cost approximately **$10,226.69** to fund the development and manufacture of the self-playing harmonica.

3. Requirements & Verification

3.1 Procedures

Each requirement, verification, and procedure for all subsystems can be found in Appendix A 6.1. Note that some requirements and verifications have changed or been outright removed since the design document due to changes in the design that have been made.

3.2 Results

3.2.1 Power Subsystem

Under load, the 6V rail was able to fall within our required tolerance of +/- 0.5V. We loaded the supplies by using the worst case scenario of our board, all peripherals active. During this test the results of the 6V power supply can be seen in the figure below.

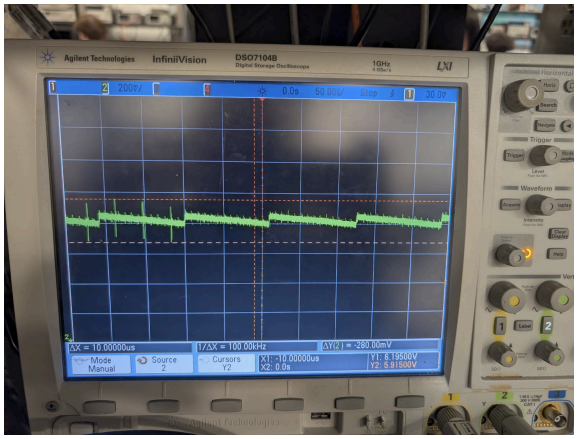

Figure 22: 6V rail ripple

The ripple shown in the oscilloscope was 280mV which is well below our +/- 0.5V tolerance limit we required.

Under the same conditions, the 3.3V rail was able to within our specified tolerance of +/- 0.5V. This is shown in our testing with the oscilloscope in the figure below.

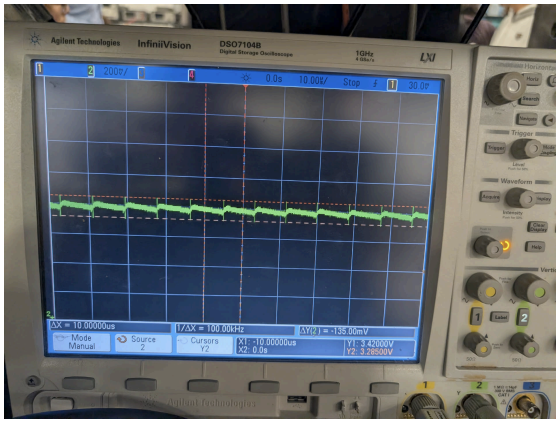

Figure 23: 3.3 rail ripple

The ripple shown in the oscilloscope was 135mV which falls well below the specified ±0.5V tolerance we required for this portion of the subsystem.

Lastly, was the inductive load tests. We tested components when loaded with a heavy inductive load, the 12V and 4.5V motors earlier revisions proved to brownout earlier. However, the final design was able to safely continue operating without the issue of browning out due to the added input supply capacitors that were added in revision 4 which are shown in the figure below.

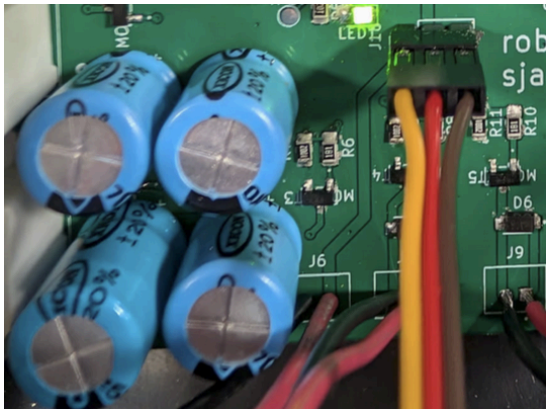

Figure 24: Bulk Capacitance on 12V Rail

Under heavy inductive loads, the board did not damage other components nor did it brownout due to inability to supply current spikes. This portion of the power subsystem requirements is satisfied.

3.2.2 Computing Subsystem

We tested the computing system by connecting to the self-hosted WiFi network hosted by the ESP32. We found the IP address to be 192.168.4.1. This IP address will not change and is used in all procedures.

For the first test, the user must be able to provide input to a website hosted by the ESP32. The website hosted by the ESP32 is an API (application programming interface) and can be accessed at http://192.168.4.1/play when connected to th ESP32's WiFi network. We were able to test this via terminal using the `curl` command and also able to test this in the browser. Both options successfully resulted in a response from the ESP32. Attached below is an example of the webserver API being accessed using `curl`

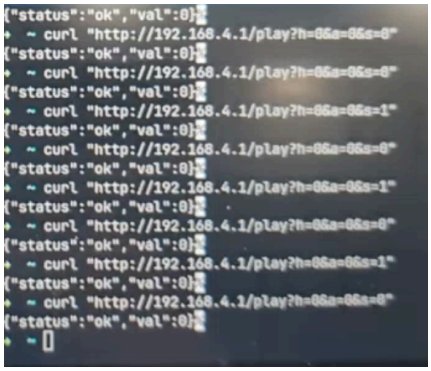

Figure 25: `curl` Terminal Access to the Webserver API

The next requirement was that user inputs must be reflected through the GPIO of the ESP32. The way we tested this was by sending a command to the servo to turn on. The servo is also connected to an LED which is on when the servo is turned on and off when the servo is turned off. By sending the servo on command, we were able to verify that the LED turned on and therefore the GPIO successfully reflects the user inputs. We were also able to verify this by the audible clicks of the solenoids when the holes were selected in the webserver.

The next requirement is, the ESP32 will control LEDs to display the state of the chip, and buttons for user control. The LED as explained previously is linked to the servo, which was able to function as intended and turned on when the servo was engaged and off when disengaged. The buttons on the MIDI parser interface were also visibly demonstrated when the servo was toggled.

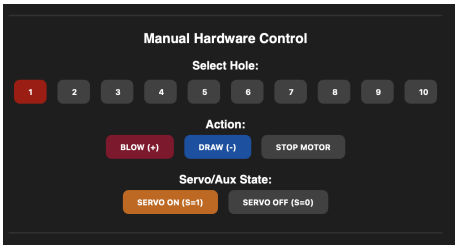

Figure 26: Manual Control Buttons on HTML Controller

The next requirement is that the ESP32 must control a servo via PWM signal. We tested this by turning the servo on and then off and verifying that the servo moved reliably when the buttons were pressed. This demonstrated that the PWM signal was being sent reliably and consistently.

Lastly, the ESP32 must control the MOSFET drivers. Two things are required for this to be verified, there is a PWM signal, and the output voltage level is shifted from ESP32 logic level to proper driver levels (12V or 6V depending on motor or solenoid driver). We tested this motor to test both PWM and voltage level shifting. This is demonstrated in the figure below.

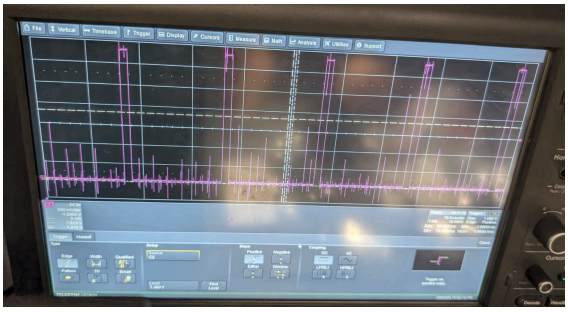

Figure 27: Driver PWM Signal on Oscilloscope

3.2.3 Mechanical Subsystem

The first requirement of the mechanical subsystem is to be able to control the motor speed using the PWM. This was verified by audibly hearing that the motor speed changed as a different PWM signal was sent to it via the HTML controller. We were also able to adjust the volume of the sound coming out of the chromatic harmonica.

For the second requirement of the mechanical subsystem, it was to ensure that all drivers successfully provided the solenoids and motors with 12V. This requirement was verified by using an oscilloscope to measure the voltage while the motors were running, resulting in the torque ripple oscilloscope screenshots in figures 22 and 23. We were also able to verify by visibly and audibly inspecting the motors, solenoids, and servo, verifying that they activated when expected to.

The third requirement of the mechanical subsystem was to be able to blow and draw air using different air pumps. We were able to verify this by physically feeling the air being drawn or pushed out of the tubes that the motors were connected to. In addition, we were able to blow and draw notes of the harmonica, successfully demonstrating blowing and drawing capabilities.

The next requirement of the mechanical subsystem was to be able to blow and draw air with resistance (such as from the tubing, solenoids, or manifold). This was only partially able to be achieved. We were able to test three different holes, holes 2, 5, and 6, and we were not able to include the solenoid. We were also not able to blow or draw air through two or more solenoids in

series, due to the significant decrease in air flow as shown in the flow rate comparison in figure 21. We were also forced to retire the two-to-ten hole manifold, as it caused too much leakage and mechanical latency for any real air to travel to the holes.

The fifth requirement was for one of the solenoids to select between the blowing and drawing motors. This solenoid was omitted in the final demo for the same reasons as shown in the previous requirement – the solenoids created too much resistance, and therefore could not be placed with more than one in series.

The final requirement for the mechanical subsystem was for the servo to actuate the harmonica lever. This was fairly easy, as it required a 27º servo actuation to push the lever, drawing about 200 mA while the servo was in the on position. This was able to change the note of the harmonica, so it was a success.

Figure : Servo Disengaged (left) vs. Engaged (right)

4. Conclusion

4.1 Accomplishments

4.1.1 Electrical Accomplishments

Our final revision of the design fully functioned electrically. The power system was able to provide 12V, 6V, and 3.3V. All motor drivers and solenoid drivers functioned as intended, turning the motor and solenoid on or off [individually] as intended. The electrical components did not overheat and were able to operate for several minutes without failure or future malfunction.

4.1.2 Software Accomplishments

The software portion fully worked as intended. The HTML and javascript code allowed for proper parsing of MIDI files into usable values for the ESP32. It provided an interface for users that is simple in design and understandable to use. The embedded code allowed for a self-hosted WiFi network for users to connect to. It also successfully created a self–hosted website which allows for communication between the user's device and the self-playing harmonica. All the electrical components were able to be controlled by the software aspect as well – GPIO being

toggled on or off as well as being PWMed. Furthermore, the software side allowed for very low latency communication between the user's device and the harmonica.

4.1.3 Mechanical Accomplishments

While the mechanical subsystem was the core issue of the design, we were still able to accomplish many aspects which are a necessity such as proper engagement and disengagement of the harmonica slider via the servo, quick activation and deactivation of the harmonica slider, fast on-off time for the motors, motor speed control, fast switching solenoid times, and the ability to play 3/10 of the holes from the harmonica.

4.2 Uncertainties

4.2.1 Motor

The first unsatisfactory result comes from the motor. The motor we utilize in our final design is loud and cannot force enough air through the harmonica when there are solenoids in between the motor and the harmonica. While the motor itself is capable of 10-12LPM at 29PSI, the resistance from the solenoid heavily prevents the motor from properly pushing or pulling air. Secondly, the motor is loud and audibly overpowers the sound output of the harmonica when it is not enclosed. Ideally, this is solved by creating an enclosure for the motors to reduce noise. However, the enclosure provided by the machine shop was not able to accommodate the size of the 12V motors and tubing.

4.2.2 Solenoid

The second unsatisfactory results stemmed from the solenoid. The solenoid cut down airflow by nearly half the amount compared to without the solenoid. The flow rate comparison can be found in "solenoid selection" above. This heavily reduced the air flow from the motors and was likely the largest issue in our design for the self-playing harmonica.This is the main reason for the final design being almost inaudible in comparison to the motor noise.

4.2.3 Sealing

The last unsatisfactory result came from the sealing of each component. Even this issue stems from the solenoid as the input of each solenoid could not replicate a good seal and would leak air. This was proved through the audible hiss of air when the tubing was connected to the metal side port of the solenoid and a motor activated (or manual blowing). This issue in addition to the reduced airflow from the solenoid output drastically reduced the amount of air going through the harmonica. We opted to avoid using the metal port of the solenoid. However, there are still uncertainties regarding the air leakage through the solenoids themselves when closed. The

leakage through the solenoids accumulates with more solenoids in the system. This contributed to why the final design was only able to play 3/10 holes instead of 10/10 holes.

4.3 Future Work/Alternatives

4.3.1 Fixes

The mechanical system was not functional and given future work, we would replace much of the mechanical system such that the solenoids would have wider bores which lessen the reduction in airflow and the motors have higher static pressure ones in order to better play the harmonica. We would also create a proper enclosure for the motors to reduce motor noise and ensure a seal with all of the joints (likely with plumbers tape or finding components with proper sizing), and reducing the hosing and connectors to reduce airflow latency.

4.3.2 Extra Features

While the computing system fully functioned as intended, we wanted to improve the usability of the device by implementing features such as LAN support so the user does not have to disconnect from the internet. We also wanted to integrate a touchscreen and more physical buttons for easier usability.

4.4 Ethical Considerations

4.4.1 Ethics

Creating a self-playing harmonica brings up questions about the ethical considerations surrounding accessibility as well as the role of automation in the arts. On one hand, this device could significantly expand access to music for individuals with physical impairments. On the other hand, there could possibly be concerns raised by the possibility of replacing humans with automation. As a team, we think that the ethical concerns are addressed by the intention of the design as well as its capabilities. The device we are proposing to create is not meant to replace human musicians, nor will it have the same capabilities as a musician. Rather, the self-playing harmonica is meant to increase the spread and awareness of harmonica music. The IEEE code of ethics section I.2. states that members should strive to improve the understanding of the public of intelligent systems and emerging technologies. By introducing this device, we aim to increase the awareness of smart systems. Therefore, we believe that we are operating in accordance with this code of ethics.

4.4.2 Safety

According to our analysis, our device can draw upwards of 5 amps. In order to have substantial overhead, we choose to utilize a 5V 10A power supply. This could be dangerous, as 10A is a very high current. We will follow proper electrical safety procedures in the lab safety training to ensure the safety of our team members as we proceed with this project.

5. References

[1] “Espressif IoT Development Framework,” *GitHub*, Jun. 25, 2023. https://github.com/espressif/esp-idf

[2] IEEE, “IEEE Code of Ethics | IEEE,” *Ieee.org*, 2020. https://www.ieee.org/about/corporate/governance/p7-8

[3] Super User, “Tango harmonica notes,” *Tango Harmonica*, Dec. 12, 2020. https://tangoharmonica.com/articles/chromatic-harmonica-notes [accessed May 06, 2026].

[4] C. Wilson, “Tone.js,” *Github.io*, 2025. https://tonejs.github.io

[5] Espressif Systems, "ESP32-S3 Technical Reference Manual," Version 1.8. [Online]. Available: https://documentation.espressif.com/esp32-s3_technical_reference_manual_en.pdf. [Accessed: May 06, 2026].

[6] "Micro Air Pump ZR370-02PM Specification," 4.5V ed. [Online]. Available: https://cdn-shop.adafruit.com/product-files/4699/ZR370-02PM_4.5V+eng.pdf. [Accessed: May 06, 2026].

[7] Skycraft Surplus, LLC, "Micro Air Pump, 12 Volt DC," *skycraftsurplus.com*. [Online]. Available: https://skycraftsurplus.com/products/micro-air-pump-12-volt-dc.html. [Accessed: May 06, 2026].

[8] SunFounder, "SunFounder Micro Servo 9g SF0180 Specification." [Online]. Available: http://wiki.sunfounder.cc/images/f/f0/SunFounder_Micro_Servo_9g_SF0180.pdf. [Accessed: May 06, 2026].

[9] Texas Instruments, "TPS6293x 3.8-V to 30-V, 2-A, 3-A Synchronous Buck Converters in a SOT583 Package," Datasheet TPS62933O. [Online]. Available: https://www.ti.com/lit/ds/symlink/tps62933o.pdf. [Accessed: May 06, 2026].

[10] Toshiba Semiconductor and Storage, "SSM3K339R,LF – N-Channel 40 V 2A Surface Mount SOT-23F," DigiKey Electronics. [Online]. Available: https://www.digikey.com/en/products/detail/toshiba-semiconductor-and-storage/SSM3K339R-LF/4781036. [Accessed: May 06, 2026].

[11] DFRobot, "DFR0866 6V 2-Position 3-Way Air Valve Product Overview," Mouser Electronics, Nov. 03, 2022. [Online]. Available: https://www.mouser.com/pdfDocs/Productoverview-DFRobot-DFR0866.pdf. [Accessed: May 06, 2026].

We would like to acknowledge the use of AI and large language models in our research for this project, in particular the use of Google Gemini, Claude, and ChatGPT.

6. Appendices

6.1 Appendix A (Verification and Procedure Tables)

6.1.1 Power Subsystem

Requirements	**Verification & Procedure**
Provide 6V +/- 0.5V at a continuous current of 3A	● Plug in calculated load (15W) ● Measure supply line voltage with an oscilloscope ● Supply lines should still be within the specified 6V +/- 0.5V
The power system must also be able to provide 3.3V +/- 0.5V at a continuous current of up to 0.5A.	● Plug in load (15W) ● Measure supply line voltage with an oscilloscope ● Supply lines should still be within the specified 3.3 +/- 0.5V
The power system must be capable of handling current spikes in order not to overload the components of the PCB.	● Plug in inductive/capacitive loads (max 15W/1.65W for 12V/5V/3V respectively) ● Measure supply line voltage with an oscilloscope ● Supply lines should be stable enough to power other components on the PCB without damaging them

6.1.2 Computing Subsystem

Requirements	**Verification & Procedure**
The user must be able to provide input to a website hosted by the ESP32	● Find device IP ● Connect to device webserver via IP
The user inputs must be reflected through GPIO of the ESP32	● Click button on the webserver ● ESP32 GPIO must react (ie turn LED on)
The ESP32 will control LEDs to display the state of the chip, and buttons for user control.	● LEDs should indicate the state of the chip clearly from the PCB ● Buttons should interact with the board and have a visible or audible reaction

	that the ESP32 has received the button press
The ESP32 must control a servo via PWM signal	● The user will click the servo on or off button. ● (For verification only) The ESP32 should be able to visually move a servo back and forth reliably
The ESP32 must control the MOSFET drivers	● Measure the input and output of the MOSFET driver with an oscilloscope ● The ESP32 should be able to PWM the MOSFET driver and have the output of the MOSFET reflect the PWM signal of the ESP32 at a higher voltage level

6.1.3 Mechanical Subsystem

Requirements	**Verification & Procedure**
The air pumps must receive PWM with varying duty cycles from the MOSFETs	● Measure inputs of the motors with an oscilloscope ● Duty cycle of MOSFET output is reflected in motor speed visually (can test airflow of motors)
The MOSFET drivers must convert the logic level of the ESP32 to the motor driving voltage	● Measure the inputs and outputs of the MOSFET driver with an oscilloscope ● Input voltage should be the logic level of the ESP32 (3V3) and the output voltage should reflect 6V or 12V, respectively. ●
The respective air pumps must be able to push/pull air (one push, one pull)	● Turn on the air pump with the MOSFET drivers ● One air pump should push air (test by feel if there is blowing action) ● One air pump should pull air (test by feeling if there is suction action)
The air pumps must be able to push/pull air with resistance (through the rubber tubing and 10-port air manifold)	● Turn on air pump with MOSFET drivers ● Pumps should audibly push air with

	manifold and rubber tubing attached
The valve must be able to control which air pump is accessing the rubber tubing	• Turn on respective air pump with MOSFET drivers • When the valve is off, the output should push air (test by feeling if there is blowing action) • When the valve is on, the output should pull air (test by feeling if there is suction action
The servo must be able to actuate the harmonica lever	• Turn on servo with PWM • Harmonica lever is visibly pushed • Harmonica notes are audibly changed from whole notes to accidentals

6.2 Appendix B. Bill Of Materials (Costs)

Supplier	Category	Part	Value/Description	Quantity	Unit Cost ($)	Total ($)
Digi-Key	MOSFET	SSM3K339R	N-Channel MOSFET	13	0.228	2.96
Digi-Key	IC	TPS62933ODRLR	Buck Converter	2	1.08	2.16
Digi-Key	Module	ESP32-S3-WROOM-1-N16R2	WiFi MCU Module	1	6.13	6.13
Digi-Key	Diode	SP0503BAHT	ESD Protection Diode	1	0.1	0.10
Digi-Key	Diode	SS24FL	Schottky Diode	13	0.1	1.30
Digi-Key	Connector	Molex 469990138	2-pin connector	2	0.84	1.68
Digi-Key	Connector	TE 640445-2	2-pin connector	3	0.13	0.39
Digikey	Connector	B2B-PH-K-S	JST 2-pin connector	11	0.11	1.21
E-shop	Capacitor	0805 Capacitor	100nF	5	0.1	0.50
E-shop	Capacitor	0805 Capacitor	33uF	4	0.1	0.40
E-shop	Capacitor	0805 Capacitor	10uF	7	0.1	0.70
E-shop	Capacitor	THT Electrolytic	470uF	4	0.1	0.40
E-shop	Capacitor	0805 Capacitor	1uF	2	0.1	0.20
E-shop	Resistor	0805 Resistor	10kΩ	21	0.1	2.10
E-shop	Resistor	0805 Resistor	180Ω	13	0.1	1.30
E-shop	Resistor	0805 Resistor	1kΩ	6	0.1	0.60
E-shop	Resistor	0805 Resistor	100kΩ	4	0.1	0.40

E-shop	Resistor	0805 Resistor	31.3kΩ	1	0.1	0.10
E-shop	Resistor	0805 Resistor	0Ω	1	0.1	0.10
E-shop	Resistor	0805 Resistor	5kΩ	1	0.1	0.10
E-shop	Resistor	0805 Resistor	33Ω	6	0.1	0.60
E-shop	Resistor	0805 Resistor	65kΩ	1	0.1	0.10
E-shop	Inductor	4.7uH Inductor	SMD Inductor	2	0.1	0.20
E-shop	LED	0805 LED	Generic LED	3	0.1	0.30
Digikey	Transistor	SS8050-G	BJT Transistor	2	0.1	0.20
E-shop	Switch	B3S-1000	Tactile Switch	5	0.1	0.50
E-shop	Connector	Pin Header	1x03	1	0.1	0.10
E-shop	Connector	Pin Header	1x05 Female	1	0.1	0.10
E-shop	Connector	Pin Header	1x09	1	0.1	0.10
E-shop	Connector	Pin Header	1x02	4	0.1	0.40
E-shop	Connector	Pin Header	1x05	1	0.1	0.10
E-shop	Connector	Pin Header	1x04	1	0.1	0.10
E-shop	Connector	IDC Header	2x05 SWD	1	0.1	0.10
E-shop	Connector	USB Micro-B	Connector	1	0.1	0.10
E-shop	Jumper	Solder Jumper	2-pin	3	0.1	0.30
E-shop	Jumper	Solder Jumper	3-pin	1	0.1	0.10
Machine Shop	Mechanical	Mounting Hardware	M3 Mounting	4	0.5	2.00
JLCPCB	PCB	PCB Fabrication	Board	1	1.67	1.67
Amazon	Other	Harmonica	N/A	1	30.99	30.99
Adafruit	Motor	4.5V DC Motor Pump		2	7.95	15.90
SkyCraftSurplus	Motor	12V DC Motor Pump		2	5.95	11.90
Digi-Key	Actuator	DFR0866	Solenoid Valve	10	4	40.00
Amazon	Motor	1080SF	Servo	1	2	2.00
Amazon	Material	1/8" Rubber Tubing	5ft	1	6	6
Machine Shop	Material	3D Printer Filament		1	10	10

6.3 Appendix C. Schedule

Week	**Task**	**Description**	**Roles**
02/23	1. Schematic & Board Design	Designed the first PCB and sent it out for production	1. Robert/David

03/02	1. Board revision 2 design 2. Breadboard Testing 3. CAD and 3D-print holder for harmonica and solenoids	Finished the design for board revision 2 Created a breadboard prototype for early design proof of concept and breadboard demo Holder was CAD'ed and 3D printed to hold servo to harmonica and organize the 10 hole solenoids	1. Robert 2. Everyone 3. David
03/09	1. Board 3 Design 2. Preliminary Software 3. Motor Ringing Issues	Finished board revision 3 Tested early software to make sure hardware is also properly setup Fixed motor ringing which was present on the breadboard design	1. Robert/David 2. Sean 3. Everyone
03/16	Spring Break	N/A	N/A
03/23	1. Board rev. 2 soldering 2. Board testing 3. Work with Machine Shop to finish Mechanical Layout	Finished soldering board revision 2 Tested board revision 2 power supply, ESP32 programming, and peripherals, Worked with ECE machine shop to develop boxes to hold PCBs and motors as well as create manifold	1. Everyone 2. Everyone 3. David/Robert
03/26	1. Board rev. 4 design	Changed MOSFET model for lower on resistance, changed MOSFET footprint for increased hand solderability, removed zener diodes	1. Robert

		after testing showed no benefits, added bulk capacitance to 12V rail, increased user IO.	
03/30	1. Create .MIDI parser and HTML Controller User Interface 2. Soldered Revision 3 PCV	Coded the HTML file to upload and parse .MIDI to send to the ESP32 – thought of HTTP request method for controlling project Soldered David's version of revision 3	1. David 2. Everyone
04/06	1. Software fixes	Fixed issues regarding solenoids not properly turning off during progress demo	1. Sean
04/13	1. Board rev. 4 soldering 2. Hardware testing	Finished board revision 4 Tested board revision 3 power supply, ESP32 programming, and peripherals	1. Everyone 2. Sean
04/20	1. Full testing 2. CAD for final demo 3. Solenoid head replacement	Did full testing of all components working together CADed solenoid head models CADed new base for harmonica and solenoid holders	1. Everyone 2. Robert 3. David/Robert
04/27	1. Final Demo, Report and Final Presentation	Worked to document all work and prepare final demo, presentation, and report	1. Everyone